



Ifnu Bima

POS

desktop

**Aplikasi POS Berarsitektur Three Tier
Menggunakan Swing, Hibernate, dan Spring**

Java Desktop

**Aplikasi POS Berarsitektur Three Tier
Menggunakan Swing, Hibernate dan
Spring**

Ifnu Bima
ifnubima.org
@ifnubima

Kata Pengantar

Menulis adalah kegiatan paling saya suka di sela-sela kesibukan sehari-hari. Menulis, terutama pengetahuan, akan memberikan efek berantai kepada seluruh pembacanya. Menulis juga membuka gerbang peradaban yang bernama Sejarah. Tidak ada sejarah tanpa budaya menulis.

Java Desktop ini adalah buku pertama saya, dan semoga menjadi yang pertama dari sekian banyak buku yang bisa saya tulis. Isi dari buku ini berasal dari pengalaman saya membuat aplikasi desktop java selama kurang lebih tiga tahun. Banyak pelajaran berharga yang saya petik dari pengalaman tersebut, baik pada saat menulis maupun pada saat membuat aplikasi. Seiring dengan waktu saya ingin menuangkan dalam sebuah buku dan membaginya dengan teman-teman pembaca.

Waktu yang saya habiskan untuk menulis buku ini ternyata lebih panjang dari yang saya perkirakan, tutorial Swing dan JDBC, cikal bakal dari buku ini, saya tulis awal tahun 2007. Kemudian semenjak pertengahan tahun 2010 saya meluangkan setiap waktu luang saya, termasuk hari libur, menyelesaikan buku ini hingga pada bulan Februari 2011 buku ini saya rampungkan.

Harapan saya buku ini bisa memberikan pengetahuan yang berguna bagi karir maupun pekerjaan teman-teman pembaca. Saya juga berharap pembaca dari buku ini bisa meningkatkan taraf hidupnya, baik dalam meniti karir maupun memulai karir sebagai developer. Pengajar seperti guru SMK atau Dosen dapat menggunakan buku ini sebagai panduan mata kuliah Rekayasa Perangkat Lunak.

Kalau anda dan institusi anda mendapatkan manfaat dari buku ini dan ingin berkontribusi balik, saya menerima donasi yang nantinya akan saya sumbangkan ke panti asuhan Permata Hati yang beralamatkan di :

Jl. Roda No. 29, Kelurahan Babakan Pasar, Kecamatan Bogor Tengah, Bogor Telepon: (0251) 8312 730. email : permatahatibogor@gmail.com, url : <http://permatahatibogor.wordpress.com>, <http://saveorphan.dagdigdug.com>

Donasi bisa anda salurkan melalui rekening BCA 2630478831 atas nama Ifnu Bima Fatkhan. Atau anda bisa mengirim pulsa ke nomer IM3 saya dengan mengirim sms TP 085692118587 [nominal] ke 151, saya akan mengganti pulsa yang anda kirimkan dan menyumbangkan ke panti asuhan Permata Hati. Mohon kirimkan email konfirmasi donasi sebagai catatan, saya akan mempublikasikan donasi anda di blog saya ifnubima.org. Kalau donasi anda belum tercatat mohon ingatkan saya dengan mengirim email notifikasi. Donasi anda adalah masa depan anak-anak ini :



Versi print buku Java Desktop ini bisa dibeli dari www.nulisbuku.com. Hasil penjualan buku ini juga akan disumbangkan ke panti asuhan Permata Hati Bogor.

Artivisi Intermedia adalah partner resmi dari buku Java Desktop. Artivisi menyediakan training inhouse maupun regular dengan materi yang ada di buku ini. Trainer di Artivisi sudah sangat berpengalaman menggunakan teknik-teknik yang diterangkan dalam buku ini selama bertahun-tahun, selain itu Artivisi juga berpengalaman menyelenggarakan training Java untuk berbagai macam instansi, mulai dari pemerintahan, swasta hingga perorangan. Banyak pengetahuan tentang best practice yang tidak bisa dicakup dalam buku ini dan hanya bisa disampaikan secara verbal dan praktek lewat training profesional yang Artivisi sediakan.

Training profesional dapat mempercepat proses belajar hanya dalam 5 hari training, sedangkan membaca buku ini hingga selesai memerlukan waktu yang jauh lebih panjang. Training profesional juga memastikan bahwa peserta benar-benar belajar langsung dengan bantuan trainer profesional sehingga pengetahuan yang diperoleh cepat dimengerti. Peserta juga dapat mengajukan studi kasus yang sedang dihadapi di perusahaan, sehingga setelah training selesai, peserta mempunyai project template yang bisa diteruskan sekembalinya dari training. Setelah training selesai, peserta langsung bisa membuat aplikasi tanpa membutuhkan waktu berbulan-bulan belajar, cukup 5 hari training saja.

Artivisi Intermedia dapat dikontak lewat email info@artivisi.com, atau telp : 021-86611859 dengan Meliawati.

Terimakasih sebesar-besarnya saya ucapkan untuk teman saya Robi Kurniawan <kurniawanz@gmail.com>, <http://kurniawanz.com> yang telah berbaik hati mendesign sampul depan buku ini. Rekomendasi saya sepenuhnya kalau ada pembaca buku ini membutuhkan jasa designer. ;).

Lisensi

Buku ini mempunyai dua lisensi untuk institusi pendidikan dan untuk institusi non pendidikan/perorangan/komunitas.

Jika anda mewakili institusi pendidikan seperti sekolah, akademi dan universitas lisensinya adalah “Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License” artinya adalah:

- Tidak diperkenankan digunakan untuk tujuan komersial. Misalnya digunakan sebagai materi training profesional atau dicetak menjadi buku untuk tujuan mencari untung.
- Dianjurkan untuk digunakan sebagai bahan ajar di kelas, terutama Rekayasa Perangkat Lunak atau Sistem Berorientasi Objek.
- Diperkenankan untuk membuat produk turunan dari buku ini asal produk turunannya juga dilisensikan sebagai Creative Commons dan saya sebagai penulis diakui sebagai pembuat awal materi ini. Hal ini agar memungkinkan institusi pendidikan mengganti sampul, header, footer dan sebagian teks dari buku ini untuk mencerminkan institusi pendidikan tersebut.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Jika anda mewakili institusi non pendidikan, perorangan atau komunitas, lisensinya adalah “Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License” artinya adalah :

- Tidak diperkenankan digunakan untuk tujuan komersial. Misalnya digunakan sebagai materi training profesional atau dicetak menjadi buku untuk tujuan mencari untung.
- Dianjurkan menyebar luaskan buku ini baik secara soft copy (pdf) maupun hard copy.
- Diperkenankan meletakkan file pdf buku ini di internet untuk didownload orang lain, tetapi lebih dianjurkan untuk menyebar link download resmi buku ini :
<http://tanyajava.com/download/javadesktop>
- Tidak diperkenankan melakukan perubahan terhadap isi buku ini. Format dokumen yang diperkenankan hanya pdf asli yang didownload dari link download resmi.

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/3.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Pendahuluan

Bertahun-tahun yang lalu saya melihat sebuah film, judulnya “The Count of Monte Cristo” yang bercerita tentang seorang anak pelayan yang bisa menjadi seorang bangsawan berkat bantuan temanya. Menjadi seorang bangsawan (menaikkan status sosial) seseorang tidak cukup hanya dengan mempunyai harta, karena harta selalu bisa habis. Quote dari film tersebut yang selalu terngiang di kepala saya adalah “I’ll give you something that nobody can take away from you, knowledge”. Saya ingin selalu bisa menjadi orang yang mengucapkan kata-kata tersebut ke semua orang. Buku ini adalah salah satu yang bisa saya lakukan, saya juga meng-host podcast java berbahasa indonesia: Indo Java Podcast (<http://ifnubima.org/indo-java-podcast/>) bersama rekan saya Dito Subandono. Selain itu saya juga aktif di berbagai milis Java seperti JUG, NetBeans-Indonesia dan JLinux.

Belajar membuat aplikasi tidak bisa dilakukan tanpa proses melihat-mencontek-mencoba, buku ini ditujukan sebagai tempat anda melihat dan mencontek bagaimana sebuah aplikasi Java Desktop dibuat. Buku ini juga berusaha untuk mengajak anda mencoba menulis kode yang disediakan sehingga siklus melihat-mencontek-mencoba menjadi lengkap.

Bab 1 kita akan membahas tentang Java Fundamental, bab ini dibagi menjadi dua bagian besar: belajar sintaks java dan belajar OOP menggunakan java. Di dalam bab ini juga dibahas tentang java 5 language enhancement yang mencakup beberapa perubahan fundamental di dalam sintaks java.

Bab 2 membahas tentang tools yang kita gunakan, NetBeans. Bagaimana membuat project, menambahkan library, menambahkan library ke pallete, menggunakan editor dan debugger, dan seterusnya. Penguasaan akan IDE diperlukan untuk menaikkan produktifitas, tanpa penguasaan IDE yang baik, produktifitas tinggi susah untuk dicapai.

Bab 3 membahas tentang koneksi ke database menggunakan JDBC. Di bab ini mulai dibahas tentang design-pattern dalam membuat kode. DAO dan Service adalah design-pattern sangat penting dalam akses database. Dilanjutkan dengan membahas ORM, Hibernate dan Spring untuk akses data. Dengan menggunakan Hibernate, produktifitas programmer menjadi meningkat drastis dibanding menggunakan JDBC murni. Spring digunakan sebagai “lem” untuk merekatkan berbagai macam komponen aplikasi, termasuk nanti digunakan sebagai tulang punggung arsitektur three tier

Bab 4 membahas tentang Swing dan komponen-komponennya. Dibahas juga pattern MVC yang digunakan komponen Swing dalam mengolah dan menampilkan data.

Bab 5 membahas bagaimana membuat aplikasi POS. Dimulai dari membuat halaman master, dilanjutkan dengan membuat halaman pembelian dan penjualan.

Bab 6 membahas bagaimana membuat report dengan JasperReport. Di dalamnya termasuk juga teknik mengcompile report yang dibuat secara otomatis menggunakan ant script bawaan dari NetBeans.

Bab 7 membahas bagaimana mengimplmentasikan arsitektur three tier menggunakan Spring Remoting.

Bab 8 merupakan bab terakhir yang membahas bagaimana membuat installer menggunakan IzPack, sehingga aplikasi mudah didistribusikan dengan adanya file installer.

Pembaca bisa langsung meloncat ke Bab 3 kalau merasa sudah mempunyai dasar-dasar pengetahuan sintaks java maupun OOP. Pembaca juga bisa melewati bab 4 kalau sudah mengetahui dasar-dasar komponen Swing.

Buku ini mempunyai format penulisan untuk memudahkan pembaca. Kode akan ditampilkan dalam format yang berbeda dengan text biasa. Kode akan ditampilkan dalam sebuah kotak dengan font Monaco seperti contoh di bawah ini :

```
public class HelloWorld{
    public static void main(String[] args){
        System.out.println("Hello World!!");
    }
}
```

Command line ditampilkan dalam format yang berbeda dan diawali dengan tanda \$ mewakili

prompt. Contohnya seperti ini :

```
$ javac HelloWorld.java
```

Output dari command line di atas ditampilkan dalam format yang berbeda, seperti contoh di bawah ini.

```
Hello World!!
```

Kalau anda melakukan copy-paste dari buku ke NetBeans perlu diperhatikan bahwa beberapa karakter yang ada di dalam buku dikategorikan sebagai karakter illegal oleh NetBeans / Java Compiler. Perbedaan ini terjadi karena encoding yang digunakan kemungkinan berbeda. perbedaan paling sering terjadi ada pada penggunaan karakter " (petik dua) yang berbeda dengan text editor ", periksa terlebih dahulu perbedaan ini sebelum memeriksa apakah kodenya benar atau salah.

Source code buku ini bisa didownload dari URL berikut ini :

<http://project-template.googlecode.com/files/java-desktop-book.zip>

Kritik, saran, ide maupun laporan kesalahan ketik / narasi bisa dialamatkan ke ifnubima@gmail.com atau ke akun twitter @ifnubima. Akhir kata, selamat belajar semoga ilmu yang ada dalam buku ini bermanfaat untuk pembaca.

Singapore, Februari 2011.

Daftar Isi

Kata Pengantar.....	iii
Lisensi.....	v
Pendahuluan.....	vi
Pengenalan Java.....	2
Instalasi JDK.....	3
HelloWorld.....	3
Keyword, Identifiers dan Akses kontrol.....	4
Keyword Bahasa Pemrograman Java	4
Identifiers	5
Access Modifier.....	6
Anatomi Class Java.....	6
Deklarasi Class.....	7
Deklarasi Interface	9
Class vs Object.....	11
Method.....	11
Constructor.....	12
Property.....	14
Konstanta.....	16
Struktur Aplikasi Java.....	16
Package.....	16
Import.....	18
Jar.....	18
Classpath.....	19
Variabel dan Memory Management.....	20
Variabel	20
Passing by value dan Passing by reference	22
Tipe Data Primitif.....	23
Wrapper Class.....	24
Array.....	25
Garbage Collector.....	28
Operator.....	30
Operator Assignment	30
Operator Relasi.....	31
Operator Instanceof.....	32
Operator Aritmatik.....	33
Operator Kondisi.....	35
Operator Bitwise.....	35
Operator Bit Shift.....	37
Operator Logika.....	39
Flow Control.....	40
If	41
Switch.....	41
Exception.....	43
Sintaks Exception.....	43
Call Stack, Stack Trace dan Uncaught Exception	44
Class Exception.....	46

Hirarki Class Exception.....	47
Throws dan Throw.....	50
Perulangan / Iterasi.....	52
for.....	52
while	53
do-while.....	54
OOP dengan Java.....	54
Enkapsulasi.....	55
Inheritance, IS-A dan HAS-A.....	55
Polimorfisme (Polymorphism).....	57
Overriding dan Overloading.....	58
Casting Variabel Reference.....	59
Interface.....	59
Collection dan Generics.....	60
toString.....	60
equals dan hashCode.....	61
Java Collection Framework.....	64
List.....	64
Set.....	65
Map.....	67
Sorting.....	69
Class Collections dan Class Arrays.....	71
Class Penting.....	76
String, StringBuilder, StringBuffer.....	76
Date, Calendar, DateFormat.....	78
Joda Time.....	81
BigDecimal dan Currency.....	87
I/O.....	90
File.....	90
Reader.....	91
Writer.....	91
InputStream.....	92
OutputStream.....	92
ImageIO.....	93
Socket.....	94
Java 5 Update.....	97
Feel of Java.....	97
Enhanced for Loop.....	97
For Loop Sebelum Java 5.....	97
For Loop di Java 5.....	98
Autoboxing/Unboxing.....	99
Primitif dan Wrapper.....	99
Static Import.....	100
Utility Class.....	100
Varargs.....	100
Fungsi Dengan Jumlah Parameter Tidak Tetap.....	100
TypeSafe Enum.....	101
Tipe Data Enum.....	101
Generics.....	102
Tipe Data di Dalam Collection.....	102

Annotations.....	103
Metadata.....	103
Kesimpulan.....	104
Instalasi.....	106
Membuat Project.....	106
Menambahkan Library.....	107
Menggunakan Editor.....	109
Menggunakan Visual Designer.....	116
Membuat Visual Component.....	116
Bekerja dengan jendela pallet.....	117
Bekerja dengan jendela properties.....	117
Bekerja dengan jendela Inspector.....	117
Debugging	117
Akses Database dengan JDBC.....	121
Menenal JDBC.....	121
Database Driver.....	121
Membuat Koneksi.....	122
Menyiapkan Table	123
Mengambil dan Memanipulasi Data dari Database.....	123
Menggunakan PreparedStatement.....	125
Batch Execution.....	127
Menangani Transaction.....	127
Mendapatkan ID yang Digenerate Otomatis.....	128
JDBC-ODBC Bridge.....	128
Memanggil Function dan StoredProcedure.....	129
Model, Dao dan Service Pattern.....	131
Entity Class / Model.....	131
DAO Pattern.....	132
Service Pattern.....	133
ORM, Hibernate dan Spring.....	137
Object Relational Mapping.....	137
Hibernate	137
Object persistence.....	138
Object-relational mismatch.....	138
Granularity	138
Subtypes	138
Identity	138
Association	138
Navigasi data	139
Impelementasi ORM untuk Mengatasi Masalah Ketidaksesuaian.....	139
Identitas.....	139
Asosiasi.....	139
Navigasi data.....	139
Mapping Sederhana.....	140
Konfigurasi Hibernate	141
Menjalankan Hibernate	143
Class-class Penting dalam Hibernate	144
Menggunakan Hibernate dengan Spring ORM.....	144
Hibernate DAO menggunakan Spring.....	145
Generic DAO.....	146

Spring Service.....	147
Declarative Transaction vs Programatic Transaction.....	149
Spring Configuration dan Spring Application Context.....	150
Utility class untuk mengenerate table dan initial data.....	154
Hibernate Mapping.....	154
Entity dan Basic mapping.....	155
Id Generation.....	156
Class Diagram.....	158
One-To-One.....	159
One-To-Many dan Master-Detail.....	160
Many-to-Many.....	162
Component.....	163
CollectionOfElement.....	164
Inheritance.....	164
Database Index	165
HQL.....	166
Projection.....	167
Condition.....	167
Parameter Binding.....	168
Order By.....	169
Agregat.....	169
Subquery.....	169
Join.....	169
Masalah LazyInitializationException, N+1 Select, Lazy fetch dan Eager fetch.....	169
Transformation.....	170
Hibernate Cache.....	171
First level cache.....	171
Second level cache.....	171
Swing.....	175
Java Foundation Class	175
Feature JFC.....	175
Swing Package.....	176
Swing HelloWorld.....	176
Install Java Development Kit.....	176
Membuat program HelloWorld	177
Melakukan kompilasi program HelloWorld.....	177
Menjalankan program HelloWorld.....	177
Membuat Swing HelloWorld dengan NetBeans	177
Komponen Swing	179
Struktur Komponen Swing.....	179
Bekerja dengan JLabel, JTextField dan JButton.....	179
Bekerja dengan JCheckBox dan JRadioButton.....	182
Bekerja dengan JList dan JComboBox	184
Bekerja dengan Menu, Popup Menu dan Toolbar.....	186
Membuat Menu.....	187
Membuat Popup Menu	190
Membuat Toolbar.....	191
Membuat Dialog dan JFileChooser.....	193
Membuat pre-defined dialog dengan JOptionPane.....	193
Membuat JFileChooser.....	196

Konsep MVC.....	199
Model dalam Komponen Swing	200
TableModel.....	201
ListModel	202
Renderer	202
Editor	203
Aplikasi POS.....	207
Data Master.....	208
Mendesign Screen	209
Membuat class Main dan Inisialisasi Spring Application Context.....	212
Melengkapi kode PersonPanel.....	215
Data Transaksi.....	226
Mempersiapkan Screen Transaksi Penjualan.....	228
Membuat Service dan DAO untuk Sales dan Product.....	232
Melengkapi Kode di Class ProductLookupDialog.....	235
Melengkapi Kode Class SalesPanel.....	237
JasperReports.....	249
Pengenalan	249
Persiapan.....	249
Laporan Transaksi Harian.....	250
Membuat JasperReports Menggunakan iReport Visual Designer.....	251
Membuat Field.....	252
Mencompile File jrxml Menjadi jasper Menggunakan Ant Task	254
Mempersiapkan ReportService.....	256
Membuat UI Report.....	259
Arsitektur Three Tier.....	263
Matriks Perbandingan antar protokol remoting.....	264
Implementasi Arsitektur Three Tier.....	264
Membuat Server dan Client Class.....	268
Izpack.....	270
Membuat Installer dari Sample di IzPack.....	270
XML Konfigurasi IzPack.....	275
Tag Intsallation<installation>.....	277
Tag Information <info>.....	277
Variabel, Tag Variabel <variable> dan Tag dynamic variabel <dynamicvariable>.....	277
Kondisi dan Tag Kondisi <conditions>	278
Tag GUI Preference <guiprefs>.....	279
Tag Localization <locale>.....	280
Tag Resources <resources>.....	280
Panel <panels>.....	280
Panel-Panel yang Disediakan IzPack.....	282
HelloPanel dan HTMLHelloPanel.....	283
CheckedHelloPanel.....	283
InfoPanel dan HTMLInfoPanel.....	283
LicencePanel dan HTMLLicencePanel.....	283
PacksPanel.....	283
TargetPanel.....	283
InstallPanel.....	283
FinishPanel.....	283
Membuat Installer Aplikasi POS.....	283

Persiapan.....	284
Membuat Startup Script.....	284
Membuat Ant Script Installer.....	287
Penutup.....	289
Referensi dan Bacaan Lebih Lanjut.....	290

BAGIAN 1

JAVA FUNDAMENTAL

Pengenalan Java

Berbicara mengenai Java, kita sebenarnya membicarakan tentang dua hal yang saling berkaitan. Yang pertama adalah Java sebagai bahasa pemrograman dan Java sebagai platform pengembangan aplikasi. Di bab Java Fundamental ini kita akan belajar mengenai Java sebagai bahasa pemrograman, kita akan belajar bagaimana menulis kode Java dengan benar tanpa ada kesalahan sintaks. Setelah melewati bab Java Fundamental kita akan belajar Java sebagai platform pengembangan aplikasi.

Bahasa pemrograman Java pada awalnya dibuat oleh James Gosling pada tahun 1995 sebagai bagian dari Sun Microsystem Java Platform. Sintaks Java banyak diturunkan dari C dan C++ tetapi lebih sederhana, ketat dan mempunyai akses ke OS yang lebih terbatas. Hal ini karena Java ditujukan sebagai bahasa pemrograman yang cukup sederhana untuk dipelajari dan mudah dibaca.

Aplikasi Java ditulis sebagai file berekstensi .java yang compile menjadi file .class. File .class ini adalah bytecode yang bisa dijalankan di semua Java Virtual Machine, tidak peduli apapun OS-nya ataupun arsitektur processornya. Java adalah bahasa yang ditujukan untuk semua kebutuhan, concurrent, berbasis class, object oriented serta didesain agar tidak tergantung terhadap lingkungan dimana aplikasi dijalankan (OS dan processor).

Java ditujukan agar bisa “ditulis sekali, bisa jalan di manapun”. Sekarang ini Java adalah bahasa pemrograman paling populer dan paling banyak digunakan untuk membuat aplikasi baik aplikasi di embedded system, mobile, desktop hingga web application.

Java mempunyai empat prinsip penting yang dijadikan sebagai tujuannya, keempat prinsip ini adalah :

1. Java harus “sederhana, object oriented dan mudah dimengerti”
2. Java harus “kuat dan aman”
3. Java harus “netral terhadap arsitektur system (OS,processor) dan bisa jalan di manapun”
4. Java harus bisa dijalankan dengan “kinerja yang tinggi”
5. Java harus “interpreted, threaded dan dinamis”

Dengan kelima prinsip di atas, aplikasi java mempunyai popularitas yang sangat tinggi terutama di dunia enterprise application. Dimana semua prinsip di atas sangat cocok untuk jenis aplikasi ini. Industri yang mempunyai budget tinggi untuk IT seperti perbankan dan telekomunikasi menggunakan Java secara ekstensif. Banyak aplikasi dengan skala raksasa dibangun menggunakan platform Java.

Java Platform terdiri dari tiga buah profile : Java ME (Java Micro Edition) adalah java yang bisa berjalan di dalam embedded system seperti Java Card dan Handphone. Java SE (Java Standard Edition) adalah java yang bisa berjalan di dalam PC maupun server sebagai aplikasi standalone maupun aplikasi desktop. Java EE (Java Enterprise Edition) adalah profile java yang ditujukan untuk membuat aplikasi Enterprise seperti Web Application (Servlet) dan Enterprise Java Bean (EJB).

Instalasi platform Java terdiri dari dua paket aplikasi. Paket yang pertama adalah JRE (Java Runtime Environment), paket ini terdiri dari semua aplikasi yang dibutuhkan agar sebuah aplikasi Java bisa berjalan, seperti library dan JVM (Java Virtual Machine). Paket kedua adalah JDK (Java Development Kit), paket ini terdiri dari JRE dan ditambah dengan perkakas untuk membuat aplikasi Java seperti java compiler (javac), java documentation (javadoc) dan java archive (jar).

Buku ini membahas tentang bagaimana membuat aplikasi Java, sehingga diperlukan JDK terinstall terlebih dahulu di system anda sebelum bisa menjalankan contoh-contoh program yang ada di sini. Selama kita membahas Java Fundamental, cukup install JDK saja dan gunakan text editor sederhana seperti notepad, vi, mcedit, textedit, notepad++, maupun emacs. Setelah melewati bab ini, kita akan menggunakan NetBeans untuk membuat aplikasi yang sebenarnya.

Buku ini mengasumsikan pembacanya sudah pernah belajar dasar-dasar Algoritma

pemrograman sehingga cukup mengerti konsep-konsep dasar seperti variabel, struktur data, tipe data, iterasi, kondisi, operator dan logika matematika. Dengan asumsi ini, buku ini tidak lagi membahas pengertian apa itu variabel atau apa itu tipe data, kita langsung menerangkan bagaimana variabel di Java, bagaimana tipe data di Java dan seterusnya. Kalau anda belum mengerti mengenai konsep-konsep algoritma pemrograman sebaiknya baca dahulu buku Algoritma pemrograman yang cukup banyak tersedia di toko buku.

Bagian pertama bab ini akan membahas bagaimana menyiapkan system anda agar bisa membuat kode sederhana dengan java, mengcompile dan menjalankan kode yang sudah dicompile.

Instalasi JDK

Instalasi JDK diawali dengan mendownload JDK dari website oracle :

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Setelah proses download selesai, lanjutkan dengan instalasi JDK. Proses instalasi sangat sederhana, klik dua kali file yang sudah didownload dan anda akan diarahkan melalui langkah demi langkah proses instalasi. Setelah selesai, java akan diinstall ke folder instalasi, kalau di windows, instalasi java ada di C:\Program Files\Java\jdk1.6.0_23 dimana 1.6.0_23 adalah versi dari jdk. Pastikan bahwa instalasi sukses dengan menjalankan perintah ini di command prompt :

```
$ java -version
```

Langkah berikutnya adalah memastikan perkakas development seperti java compiler (javac) dapat diakses dari command prompt. Caranya adalah dengan memasukkan folder instalasi java ke dalam path. Ikuti langkah berikut ini untuk menambahkan folder instalasi java ke dalam path

1. klik kanan my computer, pilih properties
2. setelah terbuka jendela properties, pilih tab advance
3. di dalam path path advance klik tombol system variables
4. di dalam jendela system variables pilih baris yang terdapat path, klik tombol edit
5. tambahkan folder C:\Program Files\Java\1.6.0_23\bin diakhir dari pathnya, jangan lupa menambahkan ; sebagai pemisah
6. test langkah-langkah di atas dengan menjalankan perintah berikut ini dari command prompt :

```
$ javac -version
javac 1.6.0_22
```

Setelah langkah-langkah di atas berhasil dijalankan, kita siap untuk membuat kode pertama menggunakan Java.

Langkah instalasi java untuk Mac dan Linux tidak saya sertakan dalam buku ini, saya menganggap pengguna Mac dan Linux cukup cerdas untuk melakukannya sendiri #eaaa.

HelloWorld

Kode HelloWorld selalu menjadi kode pertama yang coba dibuat di berbagai macam bahasa pemrograman. Tradisi menulis kode HelloWorld sudah dilakukan selama bertahun-tahun. Menulis kode HelloWorld di Java sangat mudah, mari kita coba.

Buat sebuah folder kosong untuk menyimpan file .java yang akan kita buat, misalnya di c:\sample-code atau /home/user/sample-code setelah itu buka text editor dan tulis kode berikut ini :

```
public class HelloWorld{
    public static void main(String[] args){
        System.out.println("HelloWorld!");
    }
}
```

Simpan dengan nama HelloWorld.java, ingat nama class harus sama persis dengan nama file,

jadi kalau nama class-nya adalah HelloWorld maka nama filenya adalah HelloWorld.java. Setelah disimpan, compile file HelloWorld.java menggunakan javac. Jangan lupa untuk memastikan bahwa sekarang berada di dalam folder yang sama dengan folder yang digunakan untuk menyimpan file HelloWorld.java tersebut.

```
$ javac HelloWorld.java
```

Kalau proses kompilasi berjalan dengan baik, maka tidak ada pesan sama sekali dan di folder yang sama akan dibuat file HelloWorld.class, setelah kompilasi selesai sekarang kita jalankan class HelloWorld.

```
$ java HelloWorld
HelloWorld!
$
```

Perhatikan bahwa yang dijalankan dalam kode di atas adalah class HelloWorld bukan file HelloWorld.class.

Sampai di sini kita sudah bisa melakukan kompilasi file .java dengan menggunakan java compiler (javac) dan menjalankan class menggunakan JVM (java). Setelah bisa membuat kode HelloWorld, sekarang kita akan memeriksa anatomi class java dan memahami bagian-bagian dari class java.

Keyword, Identifiers dan Akses kontrol

Pada dasarnya aplikasi java adalah sekumpulan class-class yang saling berbicara satu dengan yang lain dengan cara mengeksekusi method class lain dan mengirimkan pesan dengan memasukkan parameter ke dalam method. Dalam bagian ini kita akan belajar mengenai anatomi class dalam java, keyword, kemudian identifiers dalam java dan konsep akses kontrol.

Keyword Bahasa Pemrograman Java

Pada bab pengenalan java ini kita akan membahas mengenai dasar-dasar bahasa java. Kalau kita belajar bahasa Indonesia pasti yang pertama kita bahas adalah kosa-kata dan pembentukan kalimat seperti SPOK. Bahasa pemrograman Java tidak serumit bahasa indonesia yang mempunyai jutaan kosa kata, Java hanya mempunyai 44 buah kosa kata (Keyword). Semua Keywords ini adalah kepunyaanya bahasa Java, kita harus menggunakannya dengan benar dan tidak boleh digunakan untuk tujuan lain, misalnya sebagai nama variabel atau nama class.

Berikut ini adalah daftar 44 buah Keyword java :

abstract	Boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	If	implements	import	instanceof
int	interface	long	native	new	package
private	protected	Public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert	enum				

Kita akan membahas Keyword di atas seiring dengan bab-bab dalam buku ini. Beberapa Keyword sudah cukup familiar di semua bahasa pemrograman, sehingga tidak perlu lagi diterangkan secara mendetail.

Identifiers

Identifiers adalah nama-nama yang bisa dideklarasikan dalam java tetapi bukan bagian keyword java, yang termasuk dalam identifiers antara lain: class, interface, variabel/property dan method. Tata cara penamaan identifiers di java diatur oleh beberapa aturan:

- Aturan dari compiler untuk menentukan apakah nama identifiers diperbolehkan atau tidak.
- Konvensi penamaan identifiers dari Sun yang biasa disebut sebagai "Java Code Convention".
- Standard penamaan JavaBean.

Kita akan bahas satu per satu aturan di atas.

Aturan dari compiler tentang penamaan identifiers sangat jelas, karena akan ada error pada waktu kompilasi kalau aturan ini dilanggar. Berikut ini aturan penamaan identifiers yang digunakan oleh compiler

- Aturan pertama sudah kita bahas sebelumnya adalah semua keyword java tidak boleh digunakan sebagai identifiers.
- Identifiers harus diawali oleh huruf, simbol mata uang dolar(\$) atau karakter penghubung underscore (_). Angka tidak boleh digunakan sebagai karakter pertama identifiers.
- Setelah karakter pertama, berikutnya boleh diikuti oleh huruf, simbol mata uang dolar, karakter penghubung, dan angka.
- Tidak ada pembatasan panjang identifiers
- Identifiers di java bersifat case-sensitif, foo dengan Foo adalah dua buah identifiers berbeda.
- Nama public class harus sama persis dengan nama file .java

Berikut ini adalah contoh identifiers yang diijinkan :

```
int _x;  
int $y;  
int _____17_r;  
int _$;  
int ini_adalah_nama_identifiers_yang_panjang_sekali_bertele_tele_dan_alay;
```

Berikut ini adalah contoh identifiers yang tidak diijinkan oleh compiler java

```
int 123test_test;  
int x#;  
int x:;  
int x:;  
int .titik;
```

Java Code Convention adalah sekumpulan aturan "tidak resmi" yang dibuat oleh Sun. Salah satu bagian dari Code Convention itu membahas bagaimana menamakan identifiers yang seragam. Latar belakang dibuatnya Java Code Convention ini berasal dari penelitian yang menyebutkan bahwa usaha untuk menulis kode (development) hanya berkisar 20% saja, sedangkan 80% usaha dikerahkan untuk memelihara kode dan menambahkan feature baru ke dalam aplikasi. Hal ini mendorong Sun untuk menyusun Java Code Convention agar usaha untuk membaca kode lebih mudah dan pada akhirnya kode menjadi lebih mudah untuk dipelihara dan dipahami.

Berikut ini beberapa konvensi yang digunakan dalam Java Code Convention

- Class dan Interface selalu diawali dengan huruf besar. Setiap kata selalu diawali dengan huruf besar untuk memudahkan pembacaan. Gaya ini biasa disebut dengan "Camel Case". Contohnya: Runnable, HashMap, ArrayList dan seterusnya. Selain itu, class haruslah merupakan kata benda, bukan kata sifat atau kata kerja.
- Method selalu diawali dengan huruf kecil. Setiap kata setelah huruf pertama diawali dengan huruf besar. Method haruslah kata kerja untuk menandakan bahwa method ini melakukan suatu kegiatan / aksi. Contohnya : getIndex, setIndex, println, paint, dan seterusnya.
- Variabel menggunakan camel case yang diawali dengan huruf kecil, seperti method. Variabel

sebaiknya pendek, jelas, terdengar enak dan kata benda. Contohnya : index, panjang, lebar, indexPertama dan seterusnya.

- Konstanta di java dibuat dengan mendeklarasikan sebuah variabel sebagai statik dan final, semua hurufnya adalah huruf besar yang antar kata dipisahkan oleh simbol underscore (_). Contohnya : `FRAME_WIDTH`, `ERROR_MESSAGE` dan seterusnya.

Konsep JavaBean dibuat oleh Sun sebagai dasar dari komponen dalam aplikasi java, salah satu kegunaan praktisnya adalah penggunaan JavaBean oleh IDE seperti NetBeans agar komponen-komponen Swing bisa dimanipulasi secara visual.

Framework modern seperti Spring dan EJB juga sudah mengadopsi konsep JavaBean, sehingga istilah JavaBean sangat sering muncul di dokumentasi framework ini, Spring menggunakan istilah bean saja bukan JavaBean, tapi secara teknis keduanya sama persis.

Untuk memahami konsep JavaBean, ada satu istilah yang disebut dengan Properties. Pada dasarnya properties adalah sebuah instance variabel, yaitu variabel yang berada tepat di bawah class, yang access modifier-nya private. Karena bersifat private maka harus dibuat method untuk mengakses properties dari luar class tersebut. Method untuk mengakses properties biasa disebut sebagai getter dan method untuk mengubah nilai properties disebut sebagai setter.

Berikut ini aturan penamaan method yang digunakan untuk mengakses properties (getter setter) dari JavaBean:

- Kalau tipe data properties bukan boolean maka method untuk mengakses properties diawali dengan get. misalnya `getWidth`, `getSize`, `getIndex` dan seterusnya.
- Kalau tipe data properties adalah boolean maka method untuk mengakses properties diawali dengan is. Misalnya `isEmpty`, `isRunning` dan seterusnya.
- Semua method setter harus diawali dengan set. Misalnya `setSize`, `setIndex`, `setWidth` dan seterusnya
- Nama method diturunkan dari nama variabel yang diberi awalan get, set atau is. Aturan penulisan camel case berlaku untuk method getter dan setter.
- Method setter harus public, return void dengan satu parameter yang tipe datanya sama persis dengan tipe data variabel.
- Method setter harus public, return tipe data yang sama dengan tipe data variabel, dan tanpa parameter.
- JavaBean harus mempunyai default constructor, yaitu constructor yang tidak mempunyai parameter sama sekali.

Access Modifier

public, protected, default dan private adalah empat buah level access modifier, fungsi dari access modifier adalah mengatur bagaimana bagian-bagian kode java diakses dari bagian yang lain. Ada bagian yang boleh diakses oleh siapapun karena kode di dalamnya sudah dirancang untuk itu, ada juga bagian yang hanya boleh diakses oleh class yang sama karena memang kodenya tergantung dengan bagian lain dari class tersebut dan tidak bisa digunakan oleh bagian lain.

Access modifier public menandakan bisa diakses oleh siapapun tanpa batasan. Access modifier protected bisa diakses oleh class turunannya dan class-class lain yang berada dalam package yang sama. Access modifier default tidak memerlukan keyword, kalau tidak ada salah satu dari tiga access modifier lain maka yang digunakan adalah access modifier default. Kalau access modifier default digunakan, maka hanya class dari package yang sama saja yang bisa mengakses, termasuk class itu sendiri. Yang terakhir adalah access modifier private yang hanya mengijinkan diakses oleh class yang sama.

Anatomi Class Java

Bab ini akan menjelaskan tentang anatomi class Java. Pertama kita akan belajar tentang

bagaimana mendeklarasikan class dan interface, kemudian kita akan belajar tentang class member. Method, constructor dan variabel adalah tiga buah class member, dan ketiganya juga akan kita bahas dalam bagian terpisah. Memahami bagaimana mendeklarasikan class dan class membernya adalah pengetahuan paling dasar dalam membuat aplikasi java.

Perhatikan baik-baik aturan apa saja yang diperbolehkan dan dilarang dalam membuat class Java. Aturan ini cukup banyak dan mengikat, awalnya akan sedikit terasa merepotkan dan membutuhkan waktu untuk menghafal aturan-aturan tersebut. Setelah beberapa lama, aturan itu menjadi mudah dipahami dan dihafal. Tidak ada yang lebih baik dibanding belajar dari praktek, oleh karena itu kita akan praktek menulis kode. Jangan dilewati sesi menulis kode, usahakan tulis semua kode yang ada dalam bab ini, compile dan jalankan kodenya. Dengan begitu anda akan lebih cepat menghafal aturan dalam bahasa Java. Tanpa latihan berulang-ulang maka akan membutuhkan waktu lebih lama lagi menguasai bahasa Java.

Deklarasi Class

Class di dalam java dideklarasikan menggunakan keyword class diikuti dengan nama class. Setelah nama class ada kurung kurawal buka ({) menandai awal dari class dan kurung kurawal tutup (}) yang menandai akhir dari class.

Object adalah instansiasi dari class. Kita bisa membayangkan bahwa class adalah sebuah cetakan dan object adalah materi hasil cetakan dari class. Setiap object akan mempunyai state (instance variabel/properties) yang membedakan satu object dengan object lain, kemudian object juga mempunyai behaviour (method) dimana logic dari class disimpan.

Class adalah jantungnya Java, class adalah bagian terkecil dari kode di java yang bisa berdiri sendiri. Setiap kali kita akan membuat sebuah kode java, yang pertama harus dideklarasikan adalah class. Java mengijinkan class didefinisikan dalam beberapa cara berbeda. Cara pertama adalah mendefinisikan sebuah public class di dalam file .java. Contohnya bisa kita lihat di bawah ini :

```
public class Person { }
```

Simpan di dalam file Person.java, kemudian compile file .java di atas menggunakan java compiler (javac).

```
$ javac Person.java
```

Setelah proses kompilasi berhasil, lihat isi folder dengan menggunakan perintah dir (windows) atau ls (*nix). Terlihat bahwa class Person dibuatkan satu buah file .class, dan nama dari file .class-nya sama persis dengan nama class-nya.

Kode di atas adalah deklarasi minimal dari sebuah class Java, ada keyword class diikuti oleh nama class-nya kemudian diikuti oleh kurung kurawal buka dan tutup.

Cara kedua adalah beberapa class dideklarasikan di dalam satu file .java, tetapi dalam satu file .java hanya boleh dideklarasikan satu buah public class yang namanya sama dengan nama file .java. Contohnya sebagai berikut :

```
public class Person{}  
class Person2{}  
class Person3{}  
class Person4{}
```

Simpan di dalam file Person.java, kemudian compile file .java di atas menggunakan java compiler (javac).

```
$ javac Person.java
```

Setelah proses kompilasi berhasil, lihat isi folder dengan menggunakan perintah dir (windows) atau ls (*nix). Terlihat bahwa setiap class dibuatkan satu buah file .class, dan nama dari file .class-nya sama persis dengan nama class-nya.

```
Person.class  
Person2.class  
Person3.class  
Person4.class
```

Cara ketiga adalah mendeklarasikan class sebagai inner class. Bisa kita lihat bahwa cara kedua di

atas, deklarasi class kedua dan seterusnya berada di luar dari class pertama, kemudian file .class yang dihasilkan berbeda-beda, dan penamaan class-nya tidak menyertakan nama class public-nya, semuanya adalah class berbeda tetapi diletakkan dalam file yang sama.

Konsep inner class berbeda, karena sekarang class yang dideklarasikan di dalam class lain (inner class). Inner class sangat terikat dengan class dimana inner class itu berada. Misalnya dari sisi penamaan, nama inner class harus diawali dengan nama class dimana inner class berada kemudian diikuti dengan nama inner class itu sendiri. Misalnya contoh di bawah ini :

```
public class Person{
    private class Person2{}
    private static class Person3{}
    class Person4{}
}
```

Compile file .java di atas menggunakan java compiler (javac).

```
$ javac Person.java
```

Setelah proses kompilasi berhasil, lihat isi folder dengan menggunakan perintah dir (windows) atau ls (*nix). Terlihat bahwa setiap class dibuatkan satu buah file .class, nama file .class untuk inner class diawali dengan nama class di mana inner class berada kemudian ada tanda \$ dan diakhiri dengan nama inner class-nya.

```
Person.class
Person$Person2.class
Person$Person3.class
Person$Person4.class
```

Cara deklarasi terakhir adalah anonymous inner class, yaitu inner class yang tidak mempunyai nama, loh kok bisa? feature ini sering digunakan kalau kita ingin mengimplementasikan interface di satu tempat dan implementasi itu tidak pernah digunakan di tempat lain. Contohnya sebagai berikut ini :

```
public class Person{
    private Runnable thread = new Runnable(){
        public void run(){
            System.out.println("HelloWorld from Thread");
        }
    };
}
```

Perhatikan kode di atas, ada sebuah variabel dengan nama thread, variabel tersebut adalah object dari class Runnable. Karena Runnable adalah interface, maka kita perlu membuat class yang implement interface, tetapi implementasinya tidak perlu membuat class baru dengan nama tertentu, cukup langsung new Runnable dan implementasikan method run langsung di situ.

Simpan ke file Person.java dan coba compile menggunakan javac, setelah kompilasi berhasil lihat isi dari direktori, akan ada dua buah file class yaitu Person.class dan Person\$1.class, nah class yang terakhir inilah anonymous inner class. Kalau ada dua inner class atau lebih, penamaanya menjadi Person\$2.class, Person\$3.class dan seterusnya.

Setelah kita membahas cara deklarasi java, sekarang kita bahas aturan-aturan yang harus dipatuhi pada waktu pendeklarasian class di Java. Aturan-aturan tersebut antara lain :

- Hanya boleh ada satu class public dalam satu file .java, non public class boleh lebih dari satu di dalam satu file .java
- Nama class public harus sama dengan nama file .java
- Komentar bisa diletakkan di mana saja
- Jika class berada dalam sebuah package, maka harus ada deklarasi package di bagian paling atas dari file .java
- Import berada antara deklarasi package dan deklarasi class

- Deklarasi import dan package berlaku untuk semua class dalam file .java, tidak mungkin mendefinisikan dua buah class yang mempunyai package berbeda di dalam satu file .java

Dalam aturan di atas, ada poin yang menyebutkan tentang package. Package adalah feature yang sangat penting dalam Java, pada dasarnya package adalah sebuah folder yang memisah-misahkan class. Class dengan fungsi yang mirip akan dikelompokkan dalam satu package yang sama, hal ini dimaksudkan untuk memudahkan pengelolaan class agar mudah dipahami.

Kita bisa meletakkan beberapa keyword pada waktu pendeklarasian class. Jenis keyword pertama yang bisa digunakan adalah access modifier, keyword access modifier terdiri dari empat level : public, default, protected dan private. Hanya tiga dari empat level tersebut yang benar-benar merupakan keyword, access modifier default bukan merupakan keyword karena kalau tidak ada satupun dari ketiga access modifier digunakan di dalam deklarasi class maka kita menyebutnya default. Kita akan bahas satu per satu efek dari penggunaan access modifier pada deklarasi sebuah class.

Jika sebuah class dideklarasikan sebagai public, maka semua class yang lain dapat melihat class tersebut. Melihat di sini dalam arti bisa mengimport, menginstansiasi, mengextends dan memanggil method yang ada dalam class. Jika sebuah class dideklarasikan sebagai default atau tidak ada access modifier sama sekali, maka hanya class dari package yang sama atau class turunannya yang dapat melihat class tersebut. Class tidak bisa dideklarasikan sebagai protected. Private hanya bisa digunakan oleh inner class saja, sedangkan class lain tidak bisa ditandai sebagai private.

Selain access modifier, class bisa dideklarasikan menggunakan keyword final. Jika class dideklarasikan dengan keyword final maka class ini tidak bisa diextends oleh class lain, salah satu alasan kenapa class ditandai final agar tidak ada implementasi lain selain class ini. Semua class wrapper seperti String ditandai sebagai final agar tidak ada yang mengextends class String ini. Keyword abstract bisa digunakan untuk mendeklarasikan class, hal ini akan menyebabkan abstract class tidak dapat diinstansiasi atau dibuat objectnya.

Deklarasi Interface

Interface pada dasarnya adalah sebuah class, hanya saja method-method di dalamnya hanya berupa deklarasi saja, tidak ada implementasi dari method-method tersebut. Secara teknis bisa dikatakan bahwa interface adalah class yang bersifat abstract, semua methodnya adalah public dan abstract, serta semua variabel yang ada dalam interface adalah static final atau biasa disebut sebagai konstanta. Deklarasi interface menggunakan keyword interface diikuti dengan nama interface. Contoh paling sederhana dari sebuah interface bisa kita lihat dalam kode di bawah ini:

```
interface PersonDao{}
```

Aturan penamaan interface sama dengan penamaan class, kemudian aturan pendeklarasian interface juga sama dengan pendeklarasian interface. Hanya saja tidak ada istilah inner interface layaknya inner class.

Sebagai contoh kita akan membuat class PersonDaoImpl yang mengimplementasikan interface PersonDao, di dalam interface PersonDao ada tiga buah method: save, delete dan getById sehingga class PersonDaoImpl harus mengimplementasikan ketiga method ini.

```
public class Person{
    private Long id;
    private String nama;
    public String getNama(){
        return nama;
    }
    public void setNama(String nm){
        nama = nm;
    }
    public Long getId(){
        return id;
    }
    public void setId(Long i){
        id = i;
    }
}
```

```
}  
}
```

Interface dideklarasikan dengan menggunakan keyword interface. Di dalamnya ada method-method yang dideklarasikan tetapi belum ada implementasinya, ditandai dengan adanya titik koma (;) setelah deklarasi method. Kalau method sudah diimplementasikan maka setelah deklarasi method harus ada pasangan kurung kurawal buka tutup ({}) yang menandai blok kode yang akan dieksekusi kalau method tersebut dipanggil.

```
public interface PersonDao{  
    void save(Person p);  
    void delete(Person p);  
    Person getById(Long id);  
}
```

Class mengimplementasikan interface dengan menggunakan keyword implements. Semua method dalam interface bersifat public sehingga walaupun dalam deklarasi interface tidak ada public tetapi di class yang mengimplementasikan interface harus meletakkan access identifiers public pada deklarasi methodnya.

```
public class PersonDaoImpl implements PersonDao{  
    public void save(Person p){  
        System.out.println("menyimpan Person");  
    }  
    public void delete(Person p){  
        System.out.println("menghapus person");  
    }  
    public Person getById(Long id){  
        Person p = new Person();  
        p.setId(id);  
        p.setNama("abc");  
        return p;  
    }  
}
```

Sebuah class bisa mengimplementasikan lebih dari satu interface, antara satu interface dengan interface yang lain dipisahkan dengan tanda koma, seperti contoh sederhana di bawah ini :

```
public interface PersonDaoImpl implements PersonDao, Serializable{}
```

Interface biasanya digunakan sebagai "kontrak" agar sebuah class yang mengimplementasikan interface mempunyai semua method yang ada dalam interface itu, tetapi kita tidak ingin mengekspose implementasinya. Salah satu contoh penggunaan interface adalah JDBC API. di dalam JDBC API terdapat banyak interface seperti Connection, ResultSet, Statement, PreparedStatement dan seterusnya. Setiap database yang ingin bisa diakses dari Java akan mengimplementasikan JDBC API ini dengan cara membuat class yang mengimplementasikan semua interface dalam JDBC API. Sehingga kalau kita ingin melakukan koneksi ke MySQL ataupun ke Oracle, interfacenya sama, yang berbeda hanya implementasinya saja. Implementasi dari JDBC API ini sering dikenal sebagai JDBC Driver. Mengenai JDBC kan kita bahas lebih lanjut di bagian akses data ke database, sekarang kita fokuskan ke Interface.

Interface juga digunakan dalam best practice yang sering disebut dengan "code to interface", best practice ini menganjurkan developer untuk menggunakan interface antar layer dalam aplikasi. Misalnya contoh di atas kita membuat sebuah interface dengan nama PersonDao untuk melakukan operasi penyimpanan object person, dengan membuat interface seperti ini kita bisa mempunyai implementasi berbeda-beda untuk proses penyimpanan object person. Di contoh di atas implementasi PersonDao sangat sederhana hanya menulis string ke stdout, kita juga bisa membuat misalnya PersonDaoJdbc untuk menyimpan object person ke database dengan menggunakan JDBC, atau bisa juga dibuat PersonDaoHibernate untuk menyimpan object person ke database menggunakan Hibernate.

Kalau anda masih sulit memahami dua paragraf terakhir tentang interface, tidak masalah, tujuan utama bab ini adalah menjelaskan tentang bagaimana membuat interface, seiring

dengan bab-bab selanjutnya kita akan belajar lebih jauh mengenai beberapa teknologi yang sempat disinggung di atas.

Class vs Object

Object adalah instansiasi dari sebuah Class. Kalau kita analogikan, class itu sebuah cetakan sedangkan object itu adalah barang dari hasil cetakan. Class juga bisa dikatakan sebagai kategori, sedangkan object adalah sesuatu yang memenuhi syarat-syarat yang harus dipenuhi agar masuk dalam kategori tersebut. Jadi bisa dibilang satu class bisa mempunyai banyak object, setiap object mempunyai sifat yang sama persis seperti yang didefinisikan dalam class tersebut.

Kita ambil contoh adalah class Person, kemudian kita buat sebuah instance dari class Person yaitu ifnu. Kalimat di atas kalau diwujudkan dalam kode menjadi seperti di bawah ini :

```
Person ifnu = new Person();
```

Dua kata paling kiri adalah proses deklarasi sebuah object dengan tipe Person, kemudian di sebelah kanan tanda sama dengan (=) terjadi proses instansiasi object dari class Person menggunakan keyword new. Setiap kali kita bertemu keyword new artinya hanya satu, yaitu instansiasi object dari sebuah class. Keyword new secara detail dalam level yang paling bawah, menyebabkan JVM akan membuat object di dalam memory.

Method

Method adalah sekumpulan kode yang diberi nama, untuk merujuk ke sekumpulan kode tersebut digunakan sebuah nama yang disebut dengan nama method. Method mempunyai parameter sebagai input dan nilai kembalian sebagai output. Kita bisa juga membayangkan method itu adalah sebuah mesin, ada input yang dibutuhkan dan output yang dihasilkan.

Deklarasi method terdiri dari beberapa bagian, bagian pertama adalah access modifier dari method, apakah public, private, protected atau default. Bagian berikutnya adalah tipe kembalian dari method, kalau method tidak mengembalikan apa-apa maka keyword void yang digunakan. Bagian ketiga adalah nama method, sesuai dengan aturan java code convention, nama method diawali dengan huruf kecil dan setiap kata setelahnya diawali dengan huruf besar (camel case).

Setelah nama method ada parameter. Sebuah method bisa juga tidak mempunyai parameter, punya satu, dua dan seterusnya. Setelah java 5 ada feature yang disebut dengan varargs, feature ini memungkinkan method untuk mempunyai jumlah parameter yang bervariasi. Varargs akan kita bahas di bab tentang Java 5 Language enhancement, jadi tidak akan dibahas dalam bab ini.

Bagian terakhir dari method adalah deklarasi throws exception, dimana kita bisa mendeklarasikan tipe exception yang akan dithrows oleh method. Bab tentang exception akan membahas lebih lanjut tentang throws exception.

Nama method dan parameter adalah pembeda antara satu method dengan method yang lainya. Kalau dua method namanya beda ya pasti dianggap dua buah method berbeda. Kalau dua method namanya sama dianggap sebagai method yang berbeda kalau parameternya berbeda, method dengan nama sama dan parameter berbeda ini disebut dengan overloading dalam konsep OOP.

Kalau method mempunyai nama yang sama dan parameter yang sama tetapi tipe return atau throws exceptionnya berbeda maka akan menyebabkan error pada waktu kompilasi. Jadi kalau mendefinisikan method baru, pastikan bahwa namanya tidak sama dengan method lain atau setidaknya parameternya berbeda baik dari sisi jumlahnya, tipe parameter atau posisi parameter.

Dari penjelasan di atas, struktur method seperti di bawah ini adalah benar:

```
public void main(String[] args){ }
public String methodReturnString(){ return "ini string"; }
private void methodBerparameter(String parameter1, Integer parameter2) {}
public void methodThrowsException() throws IOException {}
protected String protectedMethod(String parameter1, Integer parameter2)
    throws IOException { return "ini string"; }
public void methodBerbeda() {}
public void methodBerbeda(String parameter1) {}
public void methodBerbeda(String parameter1, Integer parameter2) {}
public void methodBerbeda(Integer parameter1, String parameter2) {}
```



```
public void methodBerbeda(Double parameter1, Double parameter2) {}
```

Ada beberapa keyword yang bisa ditambahkan dalam deklarasi method. Keyword yang paling sering digunakan adalah static. Bagian kode Java yang dideklarasikan dengan menggunakan static akan menjadi anggota dari class, bukan anggota dari object, silahkan kembali ke bab berikutnya kalau masih belum bisa membedakan mana class dan mana object.

Karena method yang ditandai dengan static adalah bagian dari class, maka bisa diakses langsung dari nama Class itu sendiri, tidak perlu membuat object. Method static hanya bisa memanggil method lain dalam satu class yang juga ditandai static. Method main yang biasa kita gunakan untuk menjalankan aplikasi java juga ditandai dengan static, misalnya kita akan memanggil method lain dari method static, maka method lainnya ini juga harus ditandai dengan static. Kita lihat contoh berikutnya :

```
public class StaticTest {
    public static void main(String[] args) {
        //static method memanggil static method lain dalam class yang sama
        contohMethodStatic();
        //method static juga bisa dipanggil dari nama classnya
        StaticTest.contohMethodStatic();
    }
    public static void contohMethodStatic() {
        System.out.println("method static dipanggil");
    }
}
```

Kalau kode di atas dicompile, kemudian dijalankan maka hasilnya seperti di bawah ini :

```
$ javac StaticTest.java
$ java StaticTest
method static dipanggil
method static dipanggil
$
```

terlihat bahwa method contohMethodStatic dipanggil dua kali, baik menggunakan nama class maupun tidak.

Keyword lain yang bisa digunakan oleh method adalah final, synchronize dan native. Keyword final akan menyebabkan method tidak bisa dioverride, kita bahas topik ini di bab OOP. Keyword synchronize akan menyebabkan hanya satu thread yang bisa mengeksekusi method ini dalam satu waktu, kalau ada thread lain maka harus mengantri sampai thread sebelumnya selesai menjalankan method tersebut. Keyword native menandai implementasi method akan diletakkan dalam kode native, misalnya ditulis menggunakan C/C++, kemudian menggunakan Java Native Interface (JNI) untuk mengakses implementasi method tersebut. Topik mengenai JNI dan native tidak kita bahas di dalam buku ini dan diserahkan kepada anda, pembaca buku ini, untuk mempelajari topik advance ini dari sumber lain.

Constructor

Constructor adalah method yang spesial, karena mempunyai aturan-aturan sebagai berikut:

- mempunyai nama yang sama persis dengan nama class
- tidak mempunyai tipe return
- digunakan untuk menginstansiasi object
- hanya mempunyai access modifier, tidak ada keyword lain yang diletakkan sebelum nama method pada deklarasi constructor.

Seperti halnya method pada umumnya, constructor bisa mempunyai parameter serta melempar (throws) exception. Constructor yang tidak mempunyai parameter disebut dengan default constructor. Setiap class pasti mempunyai setidaknya satu constructor, kalau dalam deklarasi class tidak ada constructor sama sekali, Java secara default akan mempunyai default constructor ini. Kalau ada satu saja constructor dengan parameter, maka default constructor tidak akan dibuat, kalau masih mau ada default constructor maka harus dideklarasikan secara

eksplisit. Mari kita lihat contoh kodenya :

```
public ConstructorTest {
    public void methodSederhana(){
        System.out.println("method sederhana dipanggil");
    }
    public static void main(String[] args){
        ConstructorTest test = new ConstructorTest();
        test.methodSederhana();
    }
}
```

Kode di atas memperlihatkan bahwa class ConstructorTest tidak mendefinisikan constructor sama sekali, tetapi constructor `new ConstructorTest()` dapat dipanggil tanpa menyebabkan adanya error. Hal ini disebabkan karena Java akan membuatkan default constructor kalau class tidak mendefinisikan constructor sama sekali.

```
public class ConstructorNonDefaultTest {
    public ConstructorNonDefaultTest(String text) {
        methodSederhana(text);
    }
    public void methodSederhana(String text){
        System.out.println("method sederhana dipanggil dengan text : " + text);
    }
    public static void main(String[] args){
        //error pada waktu compile karena ada constructor yang dideklarasikan
        //sehingga default constructor menjadi wajib dideklarasikan
        ConstructorNonDefaultTest test = new ConstructorNonDefaultTest();

        //constructor non default dengan satu parameter bertipe string
        ConstructorNonDefaultTest test1 =
            new ConstructorNonDefaultTest("ini test");
    }
}
```

kalau kode di atas coba dicompile maka terdapat satu buah error seperti di bawah ini :

```
$ javac ConstructorNonDefaultTest.java
ConstructorNonDefaultTest.java:11: cannot find symbol
symbol : constructor ConstructorNonDefaultTest()
location: class ConstructorNonDefaultTest
    ConstructorNonDefaultTest test = new ConstructorNonDefaultTest();
                                      ^
1 error
$
```

Constructor dapat memanggil constructor lain dalam class yang sama menggunakan keyword `this`. Kode untuk memanggil constructor lain ini **harus** berada di baris pertama dari constructor, kalau tidak maka akan ada error pada waktu kompilasi. Berikut ini contohnya :

```
public class ConstructorCallConstructorTest {
    public ConstructorCallConstructor(){
        this("constructor memanggil constructor");
        //kode lain di sini, tidak bisa diletakkan di atas keyword this
    }
    public ConstructorCallConstructor(String text) {
        methodSederhana(text);
    }
    public void methodSederhana(String text){
        System.out.println("method sederhana dipanggil dengan text : " + text);
    }
    public static void main(String[] args){
```

```

        ConstructorCallConstructorTest test = new ConstructorCallConstructorTest();

        ConstructorCallConstructorTest test =
            new ConstructorCallConstructorTest torTest("ini test");
    }
}

```

Constructor mana yang dipanggil pada waktu menggunakan keyword this ditentukan dari parameternya.

Property

Property adalah variabel yang dideklarasikan di dalam class sejajar dengan method. Variabel yang berada di dalam method bukan merupakan property, tetapi disebut sebagai local variabel. Layaknya variabel yang lain, property mempunyai tipe data dan nama. Berdasarkan aturan java bean, property akan dibuatkan method getter dan setter untuk membungkus property. Secara umum, disarankan untuk memberikan access modifier private kepada semua property, kalau ingin mengakses property tersebut maka buatlah method getter dan setter.

Deklarasi property diawali dengan access modifier kemudian diikuti dengan tipe data dan akhirnya adalah nama dari property. Property bisa langsung diinisialisasi nilainya atau tidak, kalau tidak diberi nilai, maka nilai default yang akan diset sebagai nilai awal property. Contohnya seperti di bawah ini :

```
private String stringProperty;
```

Beberapa keyword lain juga bisa digunakan untuk mendeklarasikan property. Keyword static bisa digunakan untuk mendeklarasikan property, static akan membuat property menjadi milik class bukan milik object. Property static bisa diakses dari nama classnya dan tidak perlu ada object yang diinstansiasi. Karena property static menjadi milik class, maka kalau property static ini diganti isinya maka semua kode yang mengakses property static ini akan melihat nilai yang sama. Contohnya seperti di bawah ini :

```

public class PropertyStaticTest {
    public static String nilaiStatic;
    public static void main(String[] args) {
        //property static dipanggil menggunakan nama class
        PropertyStaticTest.nilaiStatic = "nilai dari main";
        //property static juga bisa dipanggil tanpa nama class dari class yang sama
        System.out.println(nilaiStatic);
        //method main adalah method static, hanya bisa memanggil method static juga
        methodUbahPropertyStatic();
        //nilai property static berubah setelah method methodUbahPropertyStatic
        //dipanggil
        System.out.println(nilaiStatic);
    }
    public static void methodUbahPropertyStatic() {
        PropertyStaticTest.nilaiStatic = "nilai dari methodUbahPropertyStatic";
    }
}

```

Kalau class di atas dicompile dan dijalankan, hasilnya adalah seperti di bawah ini :

```

$ javac PropertyStaticTest.java
$ java PropertyStaticTest
nilai dari main
nilai dari methodUbahPropertyStatic
$

```

variabel static bisa sangat berguna untuk memudahkan mengakses suatu variabel karena tinggal menggunakan nama class saja, jadi tidak perlu membawa-bawa nilai variabel ke dalam object yang memerlukannya. Tetapi pada skenario berbeda property static ini malah merepotkan, terutama di lingkungan aplikasi web yang berjalan dalam cluster. Antara satu cluster satu dan cluster yang lain nilai static ini bisa berbeda dan menyebabkan aplikasi

menjadi tidak konsisten. Karena dalam buku ini kita tidak bekerja dalam lingkungan cluster, kita bisa menganggap property static sebagai praktek yang aman. Nantinya di dalam aplikasi contoh, kita akan cukup banyak menggunakan property static untuk menyederhanakan kode.

Keyword berikutnya yang bisa diletakkan dalam property static adalah final. Adanya keyword final dalam deklarasi property menyebabkan property hanya bisa diinisialisasi (diberi nilai) sekali saja, setelah itu nilainya tidak bisa dirubah, perilaku ini biasa kita sebut dengan konstanta: sekali diberi nilai tidak bisa berubah nilainya. Kalau kita memaksa merubah nilainya maka akan terjadi error pada waktu kompilasi atau kalau java compiler tidak sanggup menentukan errornya akan terjadi pada waktu aplikasi berjalan.

Contohnya kita akan merubah property nilaiStatic di contoh kode sebelumnya dengan menambahkan keyword final, seperti di bawah ini :

```
public class PropertyStaticFinalTest {
    public static final String nilaiStatic;
    public static void main(String[] args) {
        PropertyStaticFinalTest.nilaiStatic = "nilai dari main";
        System.out.println(nilaiStatic);
        methodUbahPropertyStatic();
        System.out.println(nilaiStatic);
    }
    public static void methodUbahPropertyStatic() {
        PropertyStaticFinalTest.nilaiStatic =
            "nilai dari methodUbahPropertyStatic";
    }
}
```

Ketika kode dikompilasi maka akan terjadi error yang menerangkan bahwa “tidak bisa mengubah nilai variabel yang ditandai dengan final”. Outputnya seperti di bawah ini :

```
$ javac PropertyStaticFinalTest.java
PropertyStaticFinalTest.java:4: cannot assign a value to final variable nilaiStatic
    PropertyStaticFinalTest.nilaiStatic = "nilai dari main";
                                ^
PropertyStaticFinalTest.java:10: cannot assign a value to final variable
nilaiStatic
    PropertyStaticFinalTest.nilaiStatic =
                                ^
2 errors
$
```

Jadi kalau begitu kapan property yang ditandai dengan final bisa diberi nilai? Pada waktu mendeklarasikan property. Contoh di bawah ini adalah cara yang benar menginisialisasi nilai ke property yang ditandai dengan final :

```
public class PropertyFinalTest {
    public final String nilaiFinal="inisialisasi";
    public static void main(String[] args) {
        PropertyFinalTest finalTest = new PropertyFinalTest();
        System.out.println(finalTest.nilaiFinal);
    }
}
```

Perhatikan kode di atas, property nilaiFinal langsung diberi nilai pada waktu deklarasi. Karena sekarang property nilaiFinal kita harus membuat object dari class PropertyFinalTest agar bisa mengakses property nilaiFinal, berbeda dengan property yang ditandai static dimana kita bisa langsung mengaksesnya tanpa membuat object.

Masih ada keyword lain yang bisa kita letakkan dalam deklarasi property, yaitu volatile. Keyword ini menyebabkan property tidak akan ikut disimpan ketika object diserialize. Proses serialize adalah proses mengubah object ke dalam bentuk yang bisa ditransfer lewat media I/O, misalnya object ini disimpan ke hardisk atau dikirim ke komputer lain lewat jaringan. Proses sebaliknya adalah deserialize dimana dari bentuk ini diubah lagi menjadi bentuk object di dalam JVM. Topik

serialize dan deserialize akan sedikit kita bahas nanti pada waktu membahas arsitektur three tier.

Konstanta

Java mempunyai keyword `const` yang bisa digunakan untuk mendeklarasikan konstanta. Tetapi dalam prakteknya, dan ini adalah praktek yang disarankan oleh Sun Microsystem, konstanta sebaiknya dideklarasikan menggunakan gabungan keyword `static` dan `final`. Lebih baik lagi, kalau konstanta diletakkan di dalam interface, karena semua property dalam interface secara default akan bersifat `public static final`, walaupun dideklarasikan tanpa ketiga keyword tersebut.

Konstanta dalam java juga mempunyai aturan penamaan yang diterangkan dalam Java Code Convention. Nama konstanta semuanya huruf besar dan dipisahkan dengan underscore (`_`) kalau terdiri dari dua kata atau lebih. Contoh pembuatan konstanta yang baik adalah seperti berikut ini :

```
public interface Constants{
    Integer MAX_USER = 10;
    String APPLICATION_NAME = "POS";
    Stiring LAKI_LAKI = "L";
    String PEREMPUAN = "P";
}
```

Perhatikan kode di atas menggunakan interface, bukan class. Kemudian semua property tidak perlu dideklarasikan sebagai `public static final` karena secara default semua property dalam interface sudah mempunyai ketiga keyword tersebut. Kalau kita deklarasikan `Constants` di atas sebagai class maka ketiga keyword tersebut harus disertakan agar property dianggap sebagai konstanta.

```
public class Constants{
    public static final Integer MAX_USER = 10;
    public static final String APPLICATION_NAME = "POS";
    public static final Stiring LAKI_LAKI = "L";
    public static final String PEREMPUAN = "P";
}
```

Sampai di sini kita sudah belajar tentang perbedaan class dan object, kemudian bagaimana antomi dari class java. Berikutnya kita akan belajar mengenai struktur aplikasi java, bagaimana kode diorganisasi dalam package, kemudian bagaimana mengimport class lain yang berada dalam package yang berbeda. Kemudian kita juga membahas apa itu jar (java archive) dan bagaimana membuatnya, sekaligus belajar bagaimana konsep classpath bekerja untuk menemukan class yang dibutuhkan aplikasi.

Struktur Aplikasi Java

Dalam bab ini kita akan mempelajari struktur aplikasi java, di dalam aplikasi java kita akan menggunakan package untuk mengorganisasi class-class dalam package agar rapi dan dipisahkan berdasarkan fungsinya. Import digunakan untuk mengimport class yang berada dalam package yang berbeda. Class-class dalam modul yang sama biasanya diletakkan dalam satu jar agar mudah didistribusikan, class-class dalam jar ini biasa juga disebut sebagai library. Kalau aplikasi memerlukan class dari jar lain, kita harus meletakkan jar tersebut di dalam classpath agar bisa ditemukan oleh JVM.

Semua konsep package, import, jar dan classpath sangat penting untuk mengatur struktur aplikasi Java. Dengan pengaturan yang rapi aplikasi bisa mudah dipelihara dan tidak memerlukan waktu yang lama untuk mempelajari kodenya. Kita akan bahas satu per satu konsep-konsep tersebut di bab berikutnya.

Package

Pakage dalam java adalah sebuah mekanisme untuk mengorganisasi penamaan class ke dalam modul-modul. Class yang mempunyai fungsionalitas serupa dan kemiripan cukup tinggi biasanya diletakkan dalam satu package yang sama. Kalau ingin menggunakan class lain yang

berada dalam package yang berbeda harus diimport terlebih dahulu menggunakan keyword `import`. Class-class dalam package agar mudah didistribusikan biasanya diletakkan dalam satu buah jar yang pada dasarnya adalah sebuah file zip saja.

Paragraf di atas menerangkan hubungan antara package, import dan jar dalam aplikasi java. Selanjutnya kita akan belajar bagaimana membuat package dan mengimport class dari package lain, kemudian membuat file jar dari class yang sudah dicompile.

Selain bertujuan untuk mengorganisasi class, package juga digunakan untuk menghindari penamaan class yang bisa bertubrukan dalam aplikasi Java. Kalau kita membuat sebuah class dengan nama yang sangat umum, misalnya class `User`, kemungkinan besar developer lain akan membuat class dengan nama yang sama, nah bagaimana kalau kita menggunakan library yang didalamnya terdapat nama class yang sama dengan class yang kita buat? class manakah yang akan dipilih oleh Java? masalah penamaan ini dipecahkan dengan menggunakan package.

Package dimana sebuah class berada akan menjadi bagian dari nama lengkap sebuah class, misalnya class `String` sebenarnya nama lengkapnya adalah `java.lang.String` karena class `String` berada dalam package `lang.util`. Untuk menghindari penamaan class yang sama, setiap developer disarankan untuk menggunakan package yang unique untuk aplikasi yang digunakan. Misalnya ada 2 buah class dengan nama `ClassA`, yang satu berada di dalam package `a.b.c` sehingga nama lengkapnya adalah `a.b.c.ClassA` sedangkan satu lagi berada di dalam package `d.e.f` sehingga nama classnya adalah `d.e.f.ClassA`.

Bagaimana menjamin nama package yang unique? gunakan nama domain website institusi anda, maka anda akan mendapatkan nama package yang unique. Ada sebuah aturan tidak tertulis dari Sun untuk menggunakan nama domain institusi yang dibalik untuk digunakan sebagai package diikuti dengan nama aplikasi.

Misalnya kita bekerja untuk perusahaan PT coding sejahtera yang mempunyai website `codings.com`, kemudian kita membuat aplikasi keuangan yang disingkat dengan AKCS (aplikasi keuangan coding sejahtera) maka kita akan membuat package dengan `com.codings.akcs`. Bagaimana kalau kita membuat aplikasi opensource? gunakan nama domain dimana project tersebut dihosting. Misalnya untuk class-class yang digunakan di buku ini akan menggunakan package `com.googlecode.projecttemplate.pos`, hal ini karena kode dalam buku ini dihosting di `project-template.googlecode.com` dan nama aplikasinya adalah `pos` (point of sales).

Package pada dasarnya adalah struktur folder untuk meletakkan kode file java, tetapi tidak bisa sembarangan menyusun struktur folder ini, hal ini dimaksudkan agar kode lebih rapi, teratur dan tidak bercampur campur.

Untuk membuat package kita akan menggunakan contoh kode class `Person` di atas, tetapi kita letakkan di dalam package `com.googlecode.projecttemplate.pos.model`. Langkah pertama kita buat struktur folder `com\googlecode\projecttemplate\pos\model` :

```
$ mkdir com
$ mkdir com/googlecode
$ mkdir com/googlecode/projecttemplate
$ mkdir com/googlecode/projecttemplate/pos
$ mkdir com/googlecode/projecttemplate/pos/model
```

Setelah itu buat file `Person.java` dengan kode di bawah ini

```
package com.googlecode.projecttemplate.pos.model;
public class Person{
    private Long id;
    private String nama;
    public String getNama(){
        return nama;
    }
    public void setNama(String nm){
        nama = nm;
    }
    public Long getId(){
        return id;
    }
}
```

```
public void setId(Long i){  
    id = i;  
}  
}
```

Perbedaan class Person di atas dengan class Person di contoh sebelumnya berada pada baris pertama dimana ada keyword package untuk mendeklarasikan di package apa class Person ini berada. Cara mengcompile class yang berada dalam package di atas seperti di bawah ini :

```
$ javac com/googlecode/projecttemplate/pos/model/Person.java
```

Setelah mengenal package, kita akan belajar bagaimana menggunakan import untuk mendeklarasikan class-class yang berada di dalam package yang berbeda.

Import

Import digunakan untuk menyederhanakan penulisan class. Tanpa menggunakan import kita harus menuliskan nama lengkap class beserta packagenya. Dengan menggunakan import, kita mendeklarasikan di mana class yang digunakan tersebut berada sehingga selanjutnya tidak perlu lagi menuliskan nama package dari sebuah class.

Ada dua pengecualian di mana import tidak diperlukan, pertama untuk class-class yang berada dalam package yang sama dan kedua adalah class-class yang berada dalam package java.lang. Kita akan membuat interface yaitu PersonDao yang di dalamnya ada class Person, nah kedua class ini akan berada dalam package yang berbeda sehingga di dalam interface PersonDao harus mengimport class Person.

Pertama kita buat dulu struktur folder untuk PersonDao :

```
$ mkdir com/googlecode/projecttemplate/pos/dao
```

Setelah itu buat interface PersonDao dengan menyertakan package dan import

```
package com.googlecode.projecttemplate.pos.dao;  
import com.googlecode.projecttemplate.pos.model.Person;  
public interface PersonDao{  
    void save(Person p);  
    void delete(Person p);  
    Person getById(Long id);  
}
```

Kalau menulis kode dengan menggunakan text editor biasa rasanya cukup repot menangani import ini, misalnya ada 20 class yang digunakan dalam satu class, maka harus ada 20 baris import untuk setiap class. Untuk sedikit memudahkan proses import ini bisa menggunakan wildcard (*), jadi kita bisa import sekaligus semua class dalam satu package dengan menggunakan wildcard ini. Contohnya di bawah ini :

```
import com.googlecode.projecttemplate.pos.model.*;
```

Import di atas akan menyertakan semua class di dalam package model, tetapi kelemahannya adalah proses pencarian class di dalam package menjadi sedikit lebih lama pada waktu eksekusi program, kalau tidak terpaksa sebaiknya import menggunakan wildcard dihindari. Nantinya kalau sudah menggunakan IDE seperti NetBeans proses import ini menjadi sangat gampang karena dibantu oleh feature dari NetBeans tidak perlu lagi menggunakan wildcard.

Jar

Sampai di sini kita sudah tahu bagaimana class, package dan import bekerja. Nah kalau class-nya sudah banyak, kita bisa mengumpulkannya menjadi satu agar mudah untuk didistribusikan. Kumpulan class yang disimpan dalam satu file disebut jar (Java Archive). Jar sebenarnya hanyalah file zip semata, kalau punya aplikasi yang bisa membuka file zip kita juga bisa membuka file jar. Kalau di windows bisa klik kanan file jar kemudian open with winzip, isi dari file jar bisa dilihat dengan gampang.

Jar bisa dibuat dengan menggunakan command line tools dari JDK yaitu jar. Di bagian sebelumnya kita sudah membuat dua class yaitu Person dan PersonDao, kedua class ini akan diletakkan di dalam file jar dengan nama pos.jar, caranya sangat gampang, gunakan tools jar

dari JDK seperti di bawah ini :

```
$ jar cvf pos.jar .
```

Di dalam file jar terdapat meta data untuk menerangkan isi dari file jar tersebut yang disebut sebagai manifest. File manifest.mf berisi keterangan tentang jar, misalnya kapan dibuatnya, dibuat oleh siapa, apa nama main classnya dan seterusnya. File manifest.mf harus diletakkan di dalam folder META-INF. Jar yang dibuat oleh tool jar dari JDK akan dibuatkan file manifest default yang berisi keterangan sederhana.

Setelah mengerti konsep dari jar, berikutnya kita belajar tentang classpath. Classpath digunakan untuk mendefinisikan di mana saja java bisa menemukan class yang diperlukan oleh aplikasi.

Classpath

Classpath adalah cara java untuk menemukan di mana file jar yang berisi library class yang digunakan dalam aplikasi. Kalau class yang digunakan tidak ditemukan dalam classpath, maka java akan mengeluarkan `ClassNotFoundException`. Kesalahan ini sering kali terjadi dan kita bisa langsung menyimpulkan bahwa class yang diperlukan tidak berada dalam classpath.

Newbie sering kebingungan dengan error ini, maka saya merasa perlu untuk menerangkan konsep classpath dengan sedikit panjang lebar agar tidak terjadi kebingungan. Newbie yang belajar membuat aplikasi dari awal menggunakan IDE seperti NetBeans sering kebingungan ketika akan menginstall aplikasi, biasanya pertanyaan yang muncul adalah bagaimana membuat versi exe dari aplikasi java yang sudah dibuat. Saya bisa menjamin bahwa membuat exe dari program java yang sudah dibuat adalah tindakan sia sia, pertama karena malah akan menghilangkan feature utama java : run anywhere, ingat exe tidak bisa jalan di OS selain windows. Kedua karena convert java ke exe tidak gampang, perlu tools yang mahal harganya dan tidak ada jaminan bahwa hasil convert ke exe akan bisa jalan mulus seperti aslinya.

Nah kalau tidak diconvert ke dalam exe bagaimana cara membuat launcher agar user tinggal klik dua kali dan aplikasi jalan? Jawabannya adalah dengan membuat file bat di windows atau file sh di linux. Di dalam file bat / sh tersebut terdapat command line untuk menjalankan aplikasinya. Untuk membuatnya diperlukan sedikit pengetahuan tentang classpath ini.

Di bagian sebelumnya kita sudah belajar membuat sebuah file jar yang berisi dua buah class : `Person` dan `PersonDao`, nah kita sekarang akan membuat sebuah class lagi yang akan mengimport class `Person` dan mempunyai main method agar bisa dijalankan. Namakan saja classnya adalah `ClasspathExercise`, kemudian simpan class ini di folder yang berbeda dengan folder sebelumnya. Misalnya folder sebelumnya ada di `c:\sample-code`, buat lagi sebuah folder kosong di `c:\sample-code-classpath`.

```
import com.googlecode.projecttemplate.pos.model.Person;
public class ClasspathExercise{
    public void main(String[] args){
        Person p = new Person();
        p.setName("ini nama person");
        System.out.println(p.getName());
    }
}
```

Compile class di atas menggunakan `javac`

```
$ javac ClasspathExercise.java
```

kalau class di atas dijalankan seperti di bawah ini tanpa classpath maka akan keluar `ClassNotFoundException` :

```
$ java ClasspathExercise
Exception in thread "main" java.lang.NoClassDefFoundError:
com/googlecode/projecttemplate/pos/model/Person
    at ClasspathExercise.main(ClasspathExercise.java:4)
Caused by: java.lang.ClassNotFoundException:
com.googlecode.projecttemplate.pos.model.Person
    at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
    at java.security.AccessController.doPrivileged(Native Method)
```



```
at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
at java.lang.ClassLoader.loadClass(ClassLoader.java:307)
at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
at java.lang.ClassLoader.loadClass(ClassLoader.java:248)
... 1 more
```

Classpath seperti yang sudah kita bahas di atas, digunakan untuk memberitahu java di mana bisa menemukan class-class yang dibutuhkan. Nah kalau cara menjalankan kodenya diganti dengan seperti di bawah ini :

```
$ java -cp c:\sample-code\pos.jar;. ClasspathExercise
ini nama person
$
```

Perhatikan command line di atas, terdapat argumen -cp yang menandakan bahwa kita akan mendefinisikan daftar classpath. File jar dan folder bisa dimasukkan sebagai daftar classpath, seperti contoh di atas kita mendaftarkan file jar c:\sample-code\pos.jar dan folder yang sedang aktif, diwakili dengan tanda titik (.).

Kalau anda bekerja di linux / unix terdapat perbedaan di tanda pemisah antar classpath, kalau windows menggunakan titik koma(;), OS selainya menggunakan titik dua(:) seperti di bawah ini

```
$ java -cp /Users/ifnu/sample-code/pos.jar:. ClasspathExercise
ini nama person
$
```

Kita juga bisa menggunakan relative path supaya tidak terjadi error kalau filenya dipindah-pindah ke folder lain :

```
$ java -cp ../sample-code/pos.jar:. ClasspathExercise
ini nama person
$
```

Nah kembali lagi ke masalah file bat / sh yang sudah kita bahas sebelumnya. Kita bisa meletakkan perintah di atas ke dalam file ini dan mengeksekusi kedua file tersebut layaknya executable file.

Setelah java 6 terdapat feature wildcard di classpath, dimana kita bisa menginclude semua file jar yang ada dalam sebuah folder. Sebelum feature ini ada, kita harus mendaftarkan semua file jar satu per satu ke dalam classpath. Perintah di atas bisa sedikit diubah menjadi seperti di bawah ini :

```
$ java -cp ../sample-code/*.jar:. ClasspathExercise
ini nama person
$
```

Sampai di sini kita sudah belajar hal-hal dasar tentang class, package, import, jar dan classpath. Berikutnya kita akan belajar lebih banyak tentang bahasa java dan OOP.

Variabel dan Memory Management

Di bagian ini kita akan banyak membahas tipe-tipe data yang bisa berada dalam JVM dan bagaimana JVM meletakkannya dalam memory. Bagian terakhir akan membahas bagaimana Garbage Collector bekerja membersihkan JVM dari data yang sudah tidak digunakan lagi.

Variabel

Setelah belajar dasar-dasar bagaimana membuat kode java, kita akan membahas dasar-dasar bahasa java mulai dari bab ini. Hal pertama yang akan kita pelajari adalah memory management di java, untuk memahami bagaimana variabel disimpan dalam memory oleh JVM kita akan belajar tentang stack dan heap. Program java akan berada dalam stack atau heap selama daur hidupnya, instance variabel dan object selalu berada di heap sedangkan local variabel berada di dalam stack.

Mari kita lihat kode java di bawah ini, dan bagaimana bagian-bagian itu disimpan dalam heap atau stack:

```

public class Person{
    private Long id;                //instance variable
    private String nama;            //instance variable
    public static void main(String[] args){
        Person ifnu;                //local variable/reference variabel ifnu
        ifnu = new Person();         //object Person diinstansiasi
        ifnu.setId(171);
        ifnu.setNama("ifnu bima");

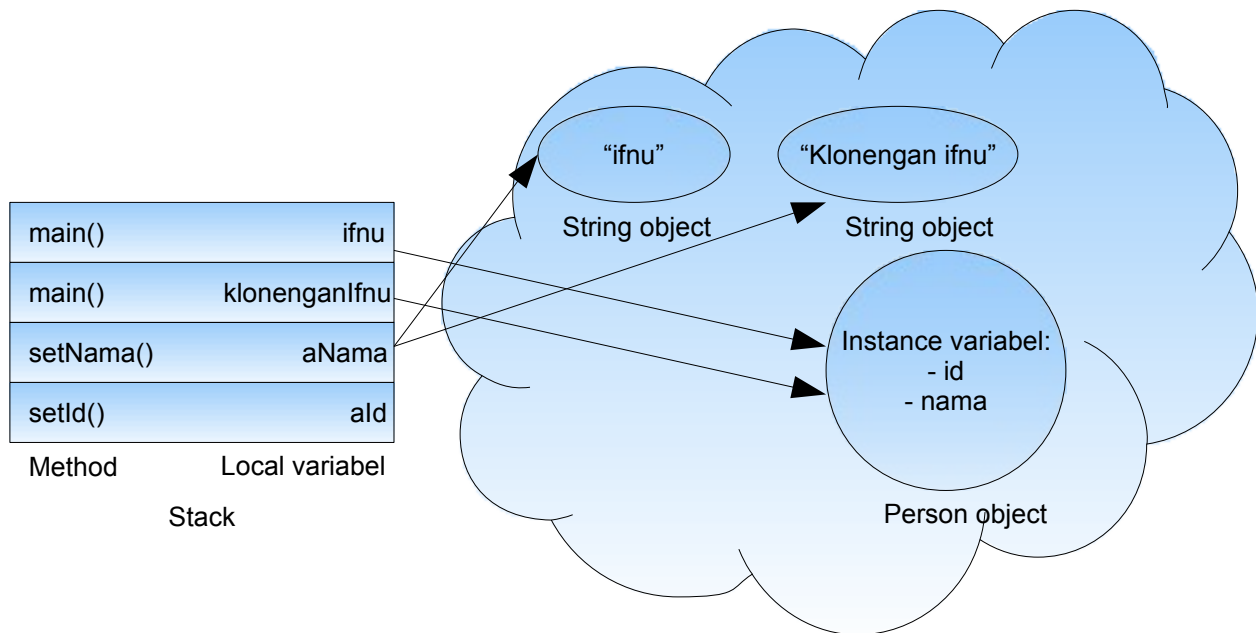
        Person klonenganIfnu = ifnu; //local variable yang menunjuk ke object
        klonenganIfnu.setNama("klonengan ifnu"); // yang sama
    }
    public void setId(Long aId){     //local variable aId
        this.id = aId;
    }
    public Long getId(){
        return this.id;
    }
    public void setNama(Long aNama){ //local variable aNama
        this.nama = aNama;
    }
    public String getNama(){
        return this.nama;
    }
}

```

Local variabel adalah variabel yang berada di dalam method, variabel ini hanya hidup selama method dieksekusi, begitu program keluar dari method maka variabel ini akan diserahkan ke garbage collector untuk dibersihkan. Variabel di dalam parameter method adalah salah satu contoh local variabel. Reference variabel adalah variabel yang menunjuk ke object, seperti ifnu dan klonenganIfnu, keduanya juga merupakan local variabel karena di deklarasi di dalam method main. id dan nama adalah instance variabel, keduanya dideklarasikan di dalam class Person.

Reference variabel ifnu dan klonenganIfnu menunjuk ke object yang sama, maka ketika klonenganIfnu mengganti nilai nama maka object yang dipegang oleh variabel ifnu juga akan ikut terganti namanya. Analoginya misalnya begini, reference variabel itu adalah alamat, sedangkan object adalah sebuah rumah. Misalnya sebuah rumah mempunyai dua alamat, yang satu alamat menggunakan nama jalan satu lagi alamat menggunakan PO BOX. Kita bayangkan misalnya pak pos mengirim barang ke alamat PO BOX ataupun alamat jalan, maka barang akan sampai ke rumah yang sama. Begitu juga dengan instance variable ifnu dan klonenganIfnu, jika satu dirubah maka yang lain ikut berubah karena keduanya menunjuk ke alamat object yang sama. Konsep reference ini penting untuk selalu diingat ketika bekerja dengan object, banyak sekali masalah yang bisa dipecahkan hanya dengan menggunakan pemahaman tentang yang baik tentang reference ini.

Berikut ini adalah gambar yang memvisualisasikan keadaan heap setelah kode di atas selesai dieksekusi :



Setelah mengetahui bagaimana reference variabel bekerja, kita akan membahas bagaimana perilaku reference variabel ini jika digunakan sebagai parameter dalam method.

Passing by value dan Passing by reference

Method dalam java bisa mempunyai parameter, setiap parameter pada dasarnya adalah local variabel, tipe dari variabel itu bisa berupa reference (object) atau bisa juga berupa tipe data bawaan dari java. Kedua jenis variabel ini akan diperlakukan berbeda oleh java, jika kita menggunakan tipe variabel dengan tipe data bawaan java maka yang terjadi adalah passing by value, artinya data yang dilempar dari kode pemanggil ke dalam method, yang dibawa adalah nilai dari variabelnya, jika nilai variabel di dalam method berubah maka nilai variabel di pemanggil tidak ikut berubah. Mari kita lihat contoh kodenya :

```
public class PassByValue{
    public static void ubahNama(String nama){
        nama = "dian";
        System.out.println("nama baru di dalam method: " + nama);
    }
    public static void main(String[] arg){
        String nama = "ifnu";
        System.out.println("nama lama dari method main: " + nama);
        ubahNama(nama);
        System.out.println("nama lama dari method main: " + nama);
    }
}
```

Terdapat variabel nama di dalam method main yang dilempar ke method ubahNama, karena tipe datanya String (tipe data bawaan dari java), maka ketika variabel nama diubah di dalam method ubahNama maka tidak akan mempengaruhi nilai variabel nama di dalam method main. Kalau kita compile dan jalankan kodenya akan keluar output seperti di bawah ini :

```
$ javac PassByValue.java
$ java PassByValue
nama lama dari method main: ifnu
nama baru di dalam method: dian
nama lama dari method main: ifnu
```

Variabel dengan tipe reference seperti : object atau tipe data array, jika digunakan sebagai parameter dalam method maka akan terjadi proses passing by reference. Artinya data yang dilempar dari kode pemanggil ke dalam method, yang dibawa adalah reference (alamat) dari

variabelnya. Jika variabel di dalam method berubah maka nilai variabel di pemanggil juga ikut berubah. Mari kita modifikasi sedikit kode di atas dengan mengganti tipe data nama dari String menjadi array String.

```
public class PassByReference{
    public static void ubahNama(String[] nama){
        nama[0] = "dian";
        System.out.println("nama baru di dalam method: " + nama[0]);
    }
    public static void main(String[] arg){
        String[] nama = new String[1];
        nama[0] = "ifnu";
        System.out.println("nilai lama dari method main: " + nama[0]);
        ubahNama(nama);
        System.out.println("nilai lama dari method main: " + nama[0]);
    }
}
```

Setelah kita ganti variabel nama dari tipe data String menjadi String[], maka method ubahNama akan menggunakan pass by reference, sehingga kalau nilai di dalam variabel nama diubah didalam method ubahNama maka variabel yang sama di dalam method main juga ikut berubah. Eksekusi kode di atas akan menghasilkan output seperti di bawah ini :

```
$ javac PassByReference.java
$ java PassByReference
nilai lama dari method main: ifnu
nama baru di dalam method: dian
nilai lama dari method main: dian
$
```

Kita sudah membahas apa itu pass by value dan apa itu pass by reference, di dalam penjelasan di atas disebutkan tentang “tipe data bawaan java”, nah tipe data jenis ini ada dua, tipe data primitif dan wrapper class. Kita akan bahas kedua jenis tipe data ini di bagian selanjutnya.

Tipe Data Primitif

Kalau java disebut tidak full OOP, hal ini dikarenakan adanya tipe data primitif ini. Jadi tipe data primitif bukanlah object, tipe ini tidak mempunyai method dan hanya mempunyai data saja. Tipe data primitif ada delapan (8) macam :

Tipe data	Ukuran	Min	Max	Keterangan
byte	8 bit	-128	127	Gunakan tipe data ini kalau range datanya kecil untuk menghemat memori dibanding dengan int, terutama kalau datanya berupa array
short	16 bit	-32768	32767	Sama dengan byte, gunakan kalau perlu optimisasi penggunaan memory, terutama kalau datanya berupa array
int	32 bit	-2147483648	2147483647	int adalah tipe data yang paling sering digunakan untuk representasi angka bulat. int adalah pilihan default jika tidak diperlukan optimisasi memori. Kalau jangkauan angkanya kurang lebar gunakan long.
long	64 bit	-9.22E+018	9.22E+018	Gunakan tipe data ini jika jangkauan angkanya lebih lebar daripada int
float	32 bit	-	-	float adalah tipe data pecahan yang didefinisikan oleh standard IEEE 754. Digit di belakang koma tidak bisa diatur, jika ingin merepresentasikan angka yang bisa diatur jumlah digit di depan dan di belakang koma, gunakan BigDecimal
double	64 bit			double adalah tipe data pecahan yang didefinisikan oleh standard IEEE 754. Digit di belakang koma tidak bisa diatur, jika ingin merepresentasikan angka yang bisa diatur jumlah digit di depan dan di belakang koma, gunakan BigDecimal.
boolean	-	-	-	Tipe data boolean hanya mempunyai dua nilai, true atau false.

char	16	"\u0000" atau 0	"\uffff" atau 65535	char adalah tipe data yang bisa menyimpan nilai unicode. Digunakan untuk merepresentasikan karakter unicode yang didalamnya terdapat abjad, angka, simbol dan karakter lainnya.
------	----	-----------------	---------------------	---

Untuk setiap tipe data primitif, kita bisa memberikan nilai literal. Notasi nilai literal untuk tiap tipe data di atas berbeda-beda, berikut ini adalah bentuk literal untuk setiap tipe data :

```
byte b = 100;
short s = 100;
int i = 26;           //decimal
int i = 032;          //octal sama dengan 26 decimal
int i = 0x1a;          //hexadecimal sama dengan 26 decimal
long l = 100l;
float f = 100.0f;
double d = 100.0;
double d = 1.00e2;
boolean bo = true;
char c = 'c';
```

Untuk tipe data primitif yang berupa angka, kita bisa melakukan pertukaran data. Proses pertukaran data antar tipe data yang berbeda disebut dengan casting. Ada casting yang dilakukan secara implisit, ada juga casting yang dilakukan secara eksplisit. Casting secara implisit dilakukan tanpa adanya kode tambahan, biasanya terjadi kalau kita memasukkan tipe data dengan ukuran yang lebih kecil ke tipe data yang lebih besar, misalnya dari byte ke int.

```
byte b = 100;
int i = b;
```

Casting secara eksplisit dilakukan jika kita memasukkan data dengan ukuran lebih besar ke ukuran lebih kecil, misalnya dari int ke byte. Perlu diperhatikan bahwa presisi data akan berubah dari ukuran ke besar ke ukuran lebih kecil, sehingga kalau nilainya melebihi jangkauan, maka terjadi pemotongan nilai dan hasilnya bisa tidak terduga. Misalnya di bawah ini :

```
int i = 100;
byte b = (byte) i;
```

Contoh di atas tidak terjadi pemotongan nilai karena 100 masih di dalam jangkauan byte, kalau contoh di bawah ini akan terjadi pemotongan nilai :

```
int i = 1000;
byte b = (byte) i;
```

nilai variabel b tidak akan 1000, karena jangkauan byte hanya sampai 127 saja. Silahkan mencoba menjalankan operasi di atas untuk mengetahui berapa nilai akhir dari variabel b setelah terjadi pemotongan, apakah dugaan anda benar?

Wrapper Class

Wrapper class adalah tipe data bawaan java yang berupa object, setiap tipe data primitif mempunyai padanan di wrapper class ini. Walaupun wrapper ini berupa class, variabel yang memegang objectnya bukanlah variabel reference. Artinya kalau ada dua buah variabel yang memegang nilai sama, satu variabel nilainya diganti maka variabel yang lain tidak akan ikut berubah nilainya. Sifat ini disebut dengan immutable.

Primitif	Wrapper class
byte	Integer
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

char	Character
	String

Sebelum java 5, selain String, semua class wrapper tidak bisa langsung diberi nilai literal. Perlu proses konversi dari nilai primitif ke wrapper classnya dan sebaliknya. Berikut ini contoh bagaimana memberi nilai ke dalam variabel dengan tipe data wrapper class sebelum java 5.

```
Integer x = new Integer(10);
int i = x.intValue();
long l = 100l;
Long y = Long.valueOf(l);
int i = Integer.parseInt("100");
```

Konversi dari satu tipe data wrapper class ke tipe data wrapper class yang lain tidak bisa menggunakan mekanisme casting seperti di dalam tipe data primitif. Hal ini dikarenakan pada dasarnya wrapper class adalah class, sedangkan casting dalam class hanya bisa dilakukan kalau ada hubungan inheritance antara kedua class yang sedang akan dicasting. Kalau proses casting antara dua class dilakukan secara ceroboh akan menyebabkan ClassCastException pada waktu kode dijalankan.

Array

Array adalah object di java yang dapat menyimpan kumpulan data dengan tipe yang sama. Array dapat menyimpan data dengan tipe primitif, wrapper class maupun reference. Walaupun array bisa menyimpan tipe data primitif, tetapi array selalu berada di heap, atau bisa dibilang apapun tipe arraynya, array itu sendiri adalah object.

Untuk bisa bekerja dengan array, kita perlu melakukan tiga langkah :

- Mendeklarasikan variabel array
- Menginisialisasi object array
- Mengisi array dengan data

Ada beberapa cara berbeda untuk melakukan ketiga hal tersebut, kita akan membahas satu per satu.

Array dideklarasikan dengan mendefinisikan tipe datanya diikuti dengan tanda kurung kotak buka dan tutup kemudian diikuti nama variabelnya. Contohnya sebagai berikut :

```
int[] data;
```

Cara deklarasi array di atas adalah cara yang dianjurkan. Semua variabel yang diletakkan setelah int[] akan bertipe array, misalnya kita modifikasi sedikit deklarasi di atas agar variabel yang dideklarasikan lebih dari satu, seperti di bawah ini :

```
int[] data, key, values;
```

Cara kedua mendeklarasikan array adalah dengan menulis tipe data kemudian diikuti oleh nama variabel serta kurung kotak buka dan tutup setelah nama variabel. Contohnya sebagai berikut :

```
int data[];
```

Cara ini tidak direkomendasikan, karena kalau ingin mendeklarasikan lebih dari satu variabel array, maka kurung kotak buka tutup harus ditulis di setiap variabel. Jika kurung kotak buka tutup tidak ditulis, maka variabel tersebut bertipe int, bukan array. Contohnya adalah :

```
int data[], key[], values;
```

variabel data dan key bertipe array int sedangkan variabel values bertipe int, bukan array.

Jumlah kurung kotak menentukan dimensi array, misalnya kita ingin membuat array dua dimensi (di dalam array ada array) dan tiga dimensi (di dalam array ada array dari array). Contohnya :

```
String[][] arrayDuaDimensi;
String[][][] arrayTigaDimensi;
```

Setelah berhasil mendeklarasikan array, sekarang waktunya menginisialisasi object array. Caranya sangat sederhana, menggunakan keyword new dan mendefinisikan panjang arraynya,

contohnya seperti berikut ini :

```
int[] data;  
data = new data[10];
```

Kode di atas artinya : buat variabel data dengan tipe array int, inisialisasi arraynya dengan panjang 10 dan isi dengan nilai 0; Kenapa diisi dengan nilai 0? hal ini dikarenakan tipe data primitif tidak bisa bernilai null, harus ada nilai defaultnya. Dalam hal ini tipe data int akan selalu mempunyai nilai default 0.

Kalau tipe data arraynya adalah wrapper class atau object, array akan berisi null, seperti contoh di bawah ini :

```
String[] data;  
data = new String[10];
```

Ukuran dari array bersifat wajib, kalau kita menginisialisasi array tanpa disertai ukuran panjang array, maka kodenya akan gagal dicompile. Contohnya seperti di bawah ini :

```
String[] data = new String[];
```

Inisialisasi array multidimensi sedikit berbeda dengan array satu dimensi. Kita bisa mendeklarasikan array dua dimensi sekaligus dengan isinya, misalnya contoh di bawah ini akan membuat matriks 10x10 sehingga bisa menampung 100 data.

```
String[] arrayDuaDimensi = new String[10][10];
```

Ketika menginisialisasi array multidimensi, ukuran array hanya wajib diisi di dimensi pertama, sedangkan dimensi kedua tidak wajib disebutkan ukurannya. Hal ini mengakibatkan isi dari array yang berupa array masih bernilai null dan harus diinisialisasi dahulu sebelum diisi dengan nilai.

```
String[] arrayDuaDimensi = new String[10][];
```

Memasukkan data ke dalam array dilakukan dengan menyebutkan index dimana data akan diletakkan. Seperti contoh di bawah ini :

```
int data = new int[4];  
data[0] = 1;  
data[1] = 12;  
data[3] = 19;
```

Perhatikan bahwa index dari array dimulai dari 0 dan selalu bernilai positif. Kalau index yang dimasukkan diluar jangkauan yang benar, akan muncul `ArrayIndexOutOfBoundsException`. Mari kita simulasikan bagaimana error `ArrayIndexOutOfBoundsException` terjadi :

```
public class ArrayError {  
    public static void main(String[] args) {  
        int[] data = new int[4];  
        data[1] = 10;  
        data[3] = 8;  
        data[4] = 9;  
    }  
}
```

Simpan dalam file `ArrayError.java`, kompilasi file java tersebut kemudian jalankan programnya:

```
$ javac ArrayError.java  
$ java ArrayError  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4  
    at ArrayError.main(ArrayError.java:6)  
$
```

Terlihat bahwa array mempunyai panjang 4, index terbesar adalah 3 karena index array dimulai dari 0, kemudian kode `data[4]` berusaha mengakses index 4 yang berada di luar jangkauan.

Sekarang kita lihat bagaimana cara memasukkan data ke dalam array multidimensi. Seperti yang sudah kita bahas sebelumnya, cara menginisialisasi array multidimensi ada dua, yang pertama adalah mengisi ukuran semua dimensi array, dan yang kedua adalah mengisi dimensi

pertama saja. Kalau cara pertama yang kita pilih untuk menginisialisasi array, maka data bisa langsung dimasukkan ke dalam array seperti contoh di bawah ini :

```
int[][] data = new int[3][3];
data[0][0] = 1;
data[0][1] = 1;
data[0][2] = 1;
data[1][0] = 1;
```

Kalau cara kedua yang dipilih, maka kita harus menginisialisasi array untuk dimasukkan ke dalam array dimensi pertama, bingung? Hmm mari kita lihat contoh saja agar lebih mudah.

```
Int[][] data = new int[3][];
data[0] = new int[5]; //init array yang dimasukkan ke array dimensi pertama
data[0][0] = 10;
data[0][2] = 10;
data[0][3] = 10;
data[1] = new int[2]; //init array yang dimasukkan ke array dimensi pertama
data[1][0] = 100;
data[1][1] = 200;
```

Nah bisa kita lihat bahwa sebelum mengisi nilai ke dalam array dimensi yang kedua, maka array dimensi pertamanya harus diisi dengan array. Dengan cara ini, maka ukuran dimensi array menjadi tidak seragam.

Dari materi array yang sudah kita bahas di atas, sepertinya bekerja dengan array cukup repot dan tidak praktis, terlalu banyak yang harus ditulis. Menyadari hal ini, Java menyediakan cara cepat untuk mendeklarasikan, menginisialisasi dan mengisi nilai array hanya dalam satu baris, baik untuk array satu dimensi maupun array dua dimensi. Berikut ini contoh sintaksnya :

```
int[] data = {1,2,3,4,6};
int[] duaDimensi = {{1,2,3},{2,3},{3}};
String[][] arrayDuaDimensi = {"satu","dua","tiga"}, {"empat","lima"};
```

Cara di atas sangat rapi dan mempersingkat banyak kode, tetapi ada satu kelemahan cara di atas, bisa dilihat bahwa tidak ada tipe data yang eksplisit untuk operand di sebelah kanan. Kelemahan di atas di atasi dengan adanya feature "anonymous array" dengan menyertakan tipe data dari array ketika diinisialisasi. Contohnya :

```
int[] data = new int[]{1,2,3,4,6};
int[] duaDimensi = new int[][]{{1,2,3},{2,3},{3}};
String[][] arrayDuaDimensi = new String[][]{"satu","dua","tiga"}, {"empat","lima"};
```

Walaupun ada keyword new, tidak perlu didefinisikan berapa panjang dari arraynya, karena panjang dari array ditentukan dari nilai yang diletakkan di dalam kurung kurawal.

Tipe data array sangat penting ketika array akan digunakan sebagai parameter sebuah fungsi, mari kita lihat kode di bawah ini :

```
public class AnonymousArray{
    public static void printArray(String[] data){
        System.out.println(data);
    }
    public static void main(String[] args){
        printArray(new String[]{"satu","dua","tiga"});
    }
}
```

Karena pada dasarnya array adalah object, maka array juga mempunyai method dan property. Salah satu yang paling berguna adalah length. Properti ini digunakan untuk mengetahui berapa panjang dari array tersebut. Contohnya sebagai berikut :

```
int[] data = new int[Integer.MAX_VALUE];
System.out.println("panjang data : " + data.length);
```

Java juga mempunyai sebuah class yang berisi perkakas untuk mengolah data, misalnya

melakukan sorting, melakukan pencarian binary bahkan hingga mengeprint isi array agar bagus tampilannya. Class tersebut adalah `java.util.Arrays`. Silahkan lihat Java Documentation bagaimana cara menggunakan perkakas yang ada dalam class `java.util.Arrays` ini.

Garbage Collector

Garbage collector adalah mekanisme JVM untuk menghapus object yang ada di memory kalau sudah tidak dibutuhkan. Garbage collector adalah salah satu feature penting JVM, karena developer tidak lagi perlu secara eksplisit menghapus object dari memory di dalam kode aplikasi. Developer tidak perlu lagi membuat kode untuk menghapus object dari memory seperti di C/C++. Di C/C++ kode yang diperlukan untuk membuat object dan menghapus object dari memory cukup menyita waktu developer, Java menghilangkan kewajiban untuk developer menulis kode penghapusan object dari memory dengan menyerahkan sepenuhnya proses tersebut ke garbage collector. Walaupun begitu, developer tidak terbebas dari praktek yang baik untuk menghindari aplikasi mengalami kehabisan memory (memory leak), error ini ditandai dengan adanya `OutOfMemoryException`.

Garbage Collector bisa juga digunakan sebagai istilah untuk merujuk ke feature manajemen memory terotomasi di Java. Pada aplikasi bisnis, setiap kali user melakukan kegiatan di dalam aplikasi, pasti ada object yang dibuat, kemudian dimanipulasi, disimpan di database dan pada akhirnya tidak diperlukan lagi. Menulis kode untuk mengelola memory secara manual bukanlah pekerjaan yang mudah, walaupun tidak bisa diukur secara pasti, usaha yang diperlukan developer bisa dua kali kalau pengelolaan memory dilakukan secara otomatis.

Di dalam java memory digunakan oleh heap, stack, tempat kumpulan variabel java dan tempat hidupnya method. Heap adalah tempat dimana semua object java berada, di tempat inilah Garbage Collector akan melakukan tugasnya, jadi tugas utama dari Garbage Collector adalah memastikan heap mempunyai cukup ruang selama eksekusi aplikasi. Ketika Garbage Collector berjalan, tujuannya hanya satu yaitu menghapus object dalam heap yang sudah tidak digunakan lagi.

Nah, kemudian pertanyaannya adalah: kapan Garbage Collector berjalan? Garbage Collector dikontrol sepenuhnya oleh JVM. Hanya JVM yang bisa memutuskan kapan Garbage Collector akan berjalan. Dari kode yang kita tulis, ada cara untuk “menganjurkan” Garbage Collector, tetapi tidak ada jaminan bahwa JVM akan langsung mengerjakan anjuran untuk menjalankan Garbage Collector ini. Pada satu waktu bisa saja JVM mematuhi anjuran ini, bisa juga di waktu yang lain tidak dijalankan. Dalam spesifikasi JVM tidak ada jaminan bahwa kalau kita menganjurkan Garbage Collector untuk dijalankan, maka pasti JVM mematuhi anjuran ini. Kita tidak bisa menggantungkan keberhasilan aplikasi kita terhadap kepatuhan JVM atas anjuran kita untuk menjalankan Garbage Collector.

Pertanyaan berikutnya adalah : definisi dari “sudah tidak digunakan lagi” itu apa? Jika suatu object tidak dapat lagi diakses oleh thread yang masih hidup, maka bisa dipastikan bahwa object tersebut sudah tidak bisa digunakan lagi. Object dalam keadaan inilah yang akan dihapus Garbage Collector dari dalam heap. Java akan kehabisan memory kalau aplikasi terus membuat object baru dan object-object tersebut masih bisa diakses oleh thread yang sedang hidup sehingga Garbage Collector tidak bisa menghapus dari heap, sampai pada akhirnya heap penuh.

Kita bisa secara sengaja membuat sebuah object tidak bisa lagi diakses dari thread yang masih hidup, cara yang paling mudah adalah dengan mengeset variabel yang memegang reference ke object tersebut menjadi null. Contohnya seperti berikut ini :

```
Date d = new Date();  
d = null;
```

Setelah variabel d di set ke null maka object Date di atas tidak bisa lagi diakses oleh variabel lainnya karena alamat ke object tersebut tidak ada lagi yang tahu. Cara lainnya adalah dengan mengganti nilai dari variabel dengan object lain, sehingga object sebelumnya tidak lagi bisa diakses. Contohnya seperti berikut :

```
Date d = new Date();  
d = new Date();
```

Kode di atas membuat dua buah object dari class Date, object pertama akan dihapus dari heap ketika Garbage Collector berjalan.

Variabel lokal yang dideklarasikan di dalam sebuah method boleh dihapus oleh Garbage Collector kalau eksekusi method selesai. Kalau tidak diperlukan, sebaiknya variabel dideklarasikan di dalam method agar mudah dihapus dari memory. Perhatikan contoh berikut ini :

```
import java.util.Date;
public class GarbageCollector{
    public static void main(String[] args){
        Date d = getDate();
        //lakukan sesuatu di sini
        System.out.println(d);
    }
    public static Date getDate() {
        StringBuffer buffer = new StringBuffer("garbage collectable");
        Date date = new Date();
        return date;
    }
}
```

Kode di atas terdapat dua buah object di dalam method getDate, object pertama adalah StringBuffer dan object kedua adalah Date. Object StringBuffer bisa dihapus oleh Garbage Collector setelah eksekusi method getDate selesai dilaksanakan, sedangkan object Date tidak bisa dihapus karena referencenya dipegang oleh variabel d di dalam method main.

Seperti yang sudah kita bahas, kita bisa “menganjurkan” Garbage Collector untuk berjalan dari dalam kode aplikasi kita. Caranya adalah memanggil method gc dari class System seperti di bawah ini :

```
System.gc();
```

Di dalam JVM tidak ada garansi bahwa setelah method gc dipanggil maka Garbage Collector akan segera berjalan. Ada kalanya Garbage Collector langsung berjalan ada kalanya tidak. Secara praktek, kita tidak perlu memanggil kode ini, kecuali kalau anda membuat aplikasi Java ME. Pemanggilan System.gc() lazim dilakukan dalam aplikasi Java ME, walaupun sebenarnya tidak dianjurkan, terutama setelah operasi pembacaan data dari server atau dari media penyimpanan.

Mari kita coba sedikit eksperimen untuk mengetes apakah setelah method gc dipanggil Garbage Collector langsung berjalan apa tidak, eksperimen ini melibatkan kode untuk menghitung berapa banyak memory yang tersedia dalam JVM.

```
import java.util.Date;
public class EksperimenGarbageCollector{
    public static void main(String[] args){
        Runtime rt = Runtime.getRuntime();
        System.out.println("jumlah memory awal : " + rt.totalMemory());
        for(int i=0;i < 1000000;i++) {
            Date d = new Date();
            d = null;
        }
        System.out.println("jumlah memory tersedia sebelum gc: " + rt.freeMemory());
        System.gc();
        System.out.println("jumlah memory tersedia setelah gc: " + rt.freeMemory());
    }
}
```

Hasil eksekusi kode di atas sebagai berikut :

```
$ javac EksperimenGarbageCollector.java
$ java EksperimenGarbageCollector
jumlah memory awal : 85000192
jumlah memory sebelum gc : 76517408
jumlah memory setelah gc : 84240832
$
```

Dalam kasus di atas setelah eksekusi method gc, Garbage Collector segera berjalan sehingga

memory yang digunakan aplikasi menjadi berkurang, ditandai dengan naiknya jumlah memory yang tersedia.

Ada istilah stop-the-world ketika kita bicara tentang Garbage Collector, istilah ini digunakan untuk menggambarkan keadaan di dalam JVM dimana semua thread akan berhenti bekerja ketika Garbage Collector berjalan. Gejala yang terlihat adalah aplikasi tiba-tiba berhenti sebentar (pause) dan tidak dapat melayani request dari user. Setelah java 6 diperkenalkan parallel Garbage Collector yang bisa berjalan parallel dan tidak menyebabkan aplikasi mengalami keadaan stop-the-world.

Sun JDK mempunyai beberapa algoritma Garbage Collector, kita bisa memilih algoritma apa yang digunakan dengan menggunakan parameter JVM. Kita tidak membahas lebih lanjut tentang Garbage Collector, karena topik ini adalah topik yang sangat lanjut dan seharusnya dikerjakan oleh developer berpengalaman. Bahkan, sebaiknya anda tidak mengutak-utik algoritma Garbage Collector kalau aplikasi masih berjalan dengan baik, hanya dalam keadaan yang cukup terpaksa maka kita bisa mencoba memilih algoritma apa yang cocok digunakan.

Sampai di sini kita sudah belajar tentang tipe data di java dan bagaimana memory management untuk setiap tipe data tersebut. Di bab berikutnya kita akan belajar bagaimana tipe data ini dioperasikan menggunakan operator.

Operator

Operator digunakan untuk mengubah nilai variabel, operator inilah yang menentukan bagaimana aplikasi memanipulasi data. Di bab operator ini tidak akan terlalu banyak penjelasan, hanya sekedar referensi untuk mengetahui bagaimana operator digunakan dalam Java. Jenis operator sendiri tidak terlalu berbeda dengan bahasa pemrograman yang lain. Berbeda dengan C++ yang dapat mengoverload operator, di Java operator tidak dapat dioverload.

Operator Assignment

Operator Assignment menggunakan simbol sama dengan (=), kita sudah melihat bagaimana operator ini digunakan di bab-bab sebelumnya. Operator ini mempunyai dua buah operand, sebelah kiri sama dengan dan sebelah kanan sama dengan. Operand sebelah kiri harus berupa variabel, sedangkan sebelah kanan bisa berupa nilai literal atau variabel yang lain. Arah operasinya adalah dari kanan ke kiri, artinya nilai sebelah kanan akan dicopy ke sebelah kiri. Setelah operasi sukses dilakukan nilai variabel di sebelah kiri akan sama dengan nilai operand di sebelah kanan.

Ada beberapa hal yang harus diperhatikan dalam menggunakan operator assignment ini, antara lain:

- Jika tipe data variabel adalah primitif, ukuran data sangat berpengaruh terhadap hasil operasi. Pastikan anda mengetahui proses casting secara implisit atau eksplisit, kalau proses casting terjadi secara eksplisit pastikan bahwa nilai datanya berada dalam rentang nilai variabel di sebelah kiri, karena casting secara eksplisit akan memotong data sesuai ukuran variabel di sebelah kiri.
- Ingat bahwa tipe data reference bukan merupakan object, tetapi alamat dimana object sebenarnya berada. Ketika kita melakukan assignment ke variabel dengan tipe data reference, yang terjadi adalah proses pengcopian alamat object dari operand di sebelah kanan ke operand di sebelah kiri, sedangkan objectnya sendiri tetap satu, setelah operasi selesai maka kedua operand akan merujuk ke alamat object yang sama. Kalau anda sudah belajar C/C++ konsep ini disebut dengan pointer, tetapi jangan salah, di java konsep pointer tidak dikenal, yang dikenal adalah konsep reference.
- Ketika mengassign nilai ke variabel dengan tipe data reference, perlu diingat aturan tentang supertypes, subtypes dan arrays. Ketiga konsep itu akan dibahas lebih lanjut ketika kita belajar tentang konsep OOP di java di bab selanjutnya.

Operator assignment bisa digabungkan dengan operator lain menghasilkan operator campuran, misalnya +=, -=, *= dan /=, operator campuran ini digunakan untuk menyingkat penulisan kode, contohnya kode di bawah ini :

```
x = x + 100;  
x = x - 1;  
y = y * 3;  
y = y / 10;
```

bisa disingkat menggunakan operator campuran menjadi :

```
x += 100;  
x -= 1;  
y *= 3;  
y /= 10;
```

Sebenarnya operator campuran ada sebelas (11) buah, tetapi hanya empat di atas saja yang sering digunakan.

Operator Relasi

Operator relasi selalu menghasilkan data boolean (true atau false), operator ini sering digunakan untuk mengecek sebuah kondisi dan diletakkan di dalam percabangan (if). Operator relasi ada enam jenis, antara lain :

- < lebih kecil
- > lebih besar
- <= lebih kecil sama dengan
- >= lebih besar sama dengan
- == perbandingan
- != tidak sebanding

Contoh penggunaan operator relasi :

```
int x = 300;  
if (x > 100){  
    //lakukan sesuatu  
}
```

Hasil dari operasi relasi bisa diassign ke variabel dengan tipe boolean, misalnya :

```
int x = 300;  
boolean b = x < 10;
```

Operator >, <, >= dan <= bisa digunakan untuk membandingkan semua tipe data bilangan bulat, bilangan pecahan dan karakter (char), baik dalam bentuk variabel ataupun dalam bentuk literal. Sedangkan operator perbandingan (== dan !=) bisa digunakan untuk mengoperasikan semua tipe data, dari primitif, wrapper class hingga tipe data reference.

Operator perbandingan cukup intuitif jika digunakan untuk membandingkan nilai data dengan tipe primitif, yang perlu diwaspadai adalah membandingkan dua buah variabel dengan tipe data pecahan, hal ini dikarenakan tipe data pecahan double atau float tidak bisa dikendalikan nilai di belakang koma, sehingga terkadang kita harus berkompromi dengan hanya membandingkan hanya sampai sekian angka di belakang koma saja.

Operator perbandingan menjadi cukup kompleks kalau digunakan untuk membandingkan nilai dari tipe data reference. Dua buah data dengan tipe data reference disebut sebanding jika keduanya memiliki alamat yang sama, atau bisa juga diartikan keduanya menunjuk ke object yang sama di dalam memory. Hati-hati menggunakan operator perbandingan terhadap tipe data wrapper class, karena hasilnya bisa diluar dugaan. Mari kita lihat kode di bawah ini :

```
public class CompareWrapperClass {  
    public static void main(String[] args){  
        Integer i = new Integer(10);  
        Integer x = new Integer(10);  
        System.out.println("new Integer(10) == new Integer(10)?" + (i==x));  
    }  
}
```

```
}
```

Tebak apa hasilnya? Ya, kalau kita compile class di atas dan dijalankan kodenya akan menghasilkan output :

```
$ javac CompareWrapperClass.java
$ java CompareWrapperClass
new Integer(10) == new Integer(10)?false
$
```

Kalau kita modifikasi sedikit kode di atas seperti di bawah ini,

```
public class CompareWrapperClass {
    public static void main(String[] args){
        Integer i = 10;
        Integer x = 10;
        System.out.println("10 == 10?" + (i==x));
    }
}
```

Ternyata hasilnya adalah sebagai berikut :

```
$ javac CompareWrapperClass.java
$ java CompareWrapperClass
10 == 10?true
$
```

Apakah anda terkejut?

Perhatikan di kode pertama, variabel `i` dan `x` diassign dengan sebuah object baru menggunakan keyword `new`, artinya antara variabel `i` dan `x` akan menunjuk ke dua buah object Integer yang berbeda di memory, walaupun sebenarnya nilai kedua object tersebut adalah sama: 10. Sedangkan kode ke-dua, variabel `i` dan `x` diassign dengan nilai literal 10, kode ini menggunakan feature autoboxing/unboxing dari java 5 untuk mengassign nilai literal 10 ke dalam wrapper class, sehingga kedua object ini sama-sama menunjuk ke nilai yang sama, yaitu literal 10.

Anda harus benar-benar jeli untuk menggunakan operator perbandingan ini kalau tipe datanya reference atau wrapper class. Gunakan method `equals` yang ada di wrapper class atau di tipe data reference untuk membandingkan apakah kedua data mempunyai kesamaan secara logika, karena operator perbandingan sebenarnya membandingkan alamat memory bukan membandingkan apakah kedua object secara logika sama atau tidak. Kita akan membahas topik "object equality" lebih lanjut di bab OOP tentang method `equals` dan `hashCode`, jadi kalau sekarang masih belum terlalu ngeh tentang topik ini, masih ada satu sesi lagi yang akan membahas lebih dalam di bab selanjutnya.

Operator instanceof

Operator `instanceof` hanya dapat digunakan untuk mengoperasikan dua buah tipe data reference. Operator ini digunakan untuk mengecek tipe dari sebuah variabel, berikut ini contohnya :

```
String s = "ini string";
if(s instanceof String){
    //lakukan sesuatu di sini
}
Long l = 10L;
Number x = l;
if(x instanceof Long){
    //lakukan sesuatu di sini
}
```

Operator `instanceof` dapat menyebabkan error ketika kompilasi kalau tipe data variabel reference tersebut tidak berhubungan. Misalnya kita mempunyai tipe data Integer tetapi dioperasikan dengan class String seperti di bawah ini :

```
Integer i = 10;
if(i instanceof String){           //menyebabkan error pada waktu kompilasi
    //lakukan sesuatu di sini
}
```

Operator instanceof biasanya digunakan sebelum melakukan downcast dari subtype ke supertype agar tidak terjadi ClassCastException. Lebih lanjut mengenai topik ini akan kita bahas dalam bab OOP di java, karena perlu mengetahui konsep inheritance sebelum bisa memahami apa itu subtype dan supertype.

Operator Aritmatik

Operator aritmatik pasti sudah sangat familiar, karena operator ini tidak jauh berbeda dengan operator aritmatik yang ada di pelajaran matematika. Simbol yang digunakan dalam java adalah :

- + penambahan
- - pengurangan
- * perkalian
- / pembagian

Penggunaan keempat operator di atas sangat standard, sama persis seperti di pelajaran matematika, misalnya :

```
int x = 10 + 100;
int y = 100 - 7;
int z = 100 * 5;
int a = 100 / 2;
```

Java juga mengenal operator modulus (%) atau biasa dikenal sebagai pembagi sisa. Misalnya 10 % 5 hasilnya adalah 0 (10 dibagi 5 sisa 0), atau 10 % 3 hasilnya adalah 1 (10 dibagi 3 sisa 1). Contohnya sebagai berikut :

```
int x = 100 % 17;
```

Operator + (plus) selain digunakan sebagai penambahan juga digunakan sebagai penggabungan String (String concatenation). Contohnya :

```
String s = "ini " + " String";
```

String concatenation menjadi sedikit kompleks kalau dirangkai dengan angka, contohnya:

```
String s = "ini " + " String";
int a = 10;
long b = 100;
System.out.println(s + a + b);
```

Ketika salah satu operand bertipe String, maka seluruh operasi + akan menjadi String concatenation, variabel a dan b juga dianggap sebagai String, output dari kode di atas adalah :

```
ini String10100
```

Kalau kita ingin menjumlahkan variabel a dan b terlebih dahulu sebelum digabungkan dengan String s, maka kita bisa meletakkan kurung di antara variabel a dan b, seperti kode di bawah ini :

```
String s = "ini " + " String";
int a = 10;
long b = 100;
System.out.println(s + (a + b));
```

Hasil kode di atas adalah :

```
ini String110
```

Java, seperti halnya C/C++ juga mengenal operator increment dan decrement :

- ++ increment (postfix dan prefix)
- -- decrement (postfix dan prefix)

Kedua operator ini disebut juga unary operator karena hanya membutuhkan satu operand saja, sedikit berbeda dengan operator aritmatik lain yang membutuhkan dua buah operand. Operator increment akan menaikkan nilai dari operand sebesar satu satuan, sedangkan decrement akan menurunkan nilai dari operand sebanyak satu satuan. Kedua operator ini adalah bentuk lebih pendek lagi dari operator gabungan (`+=` dan `-=`). Berikut ini contoh bentuk paling panjang hingga paling pendek :

```
int x = 10;
x = x + 1;           //bentuk paling panjang
x += 1;             //menggunakan operator gabungan +=
x++;               //menggunakan operator increment
int y = 100;
y = y - 1;          //bentuk paling panjang
y -= 1;            //menggunakan operator gabungan -=
y--;              //menggunakan operator decrement
```

Operator increment dan decrement bisa diletakkan setelah operand (postfix) maupun sebelum operand (prefix), kedua bentuk ini sama-sama menaikkan atau menurunkan nilai operand sebanyak satu satuan, tetapi evaluasi terhadap operasinya mempunyai nilai berbeda. Hasil evaluasi operator increment yang diletakkan setelah operand sama dengan nilai operand sebelum dievaluasi, sedangkan hasil evaluasi operator increment yang diletakkan sebelum operand adalah lebih banyak satu satuan daripada nilai operand sebelum dievaluasi, hmm sepertinya agak susah dimengerti ya? baiklah, mari kita lihat contoh di bawah ini agar lebih jelas :

```
public class IncrementTest{
    public static void main(String[] args){
        int x = 10;
        System.println("x++ = " + x++);
        System.println("Setelah evaluasi, x = " + x);
        x = 10;
        System.println("++x = " + ++x);
        System.println("Setelah evaluasi, x = " + x);
    }
}
```

Kalau dicompile dan dijalankan kode di atas, hasilnya adalah sebagai berikut

```
$ javac IncrementTest.java
$ java IncrementTest
x++ = 10
Setelah evaluasi, x = 11
++x = 11
Setelah evaluasi, x = 11
$
```

Nah bisa kita lihat bahwa hasil setelah evaluasi adalah sama (11), tetapi ketika operator increment sedang dievaluasi hasilnya berbeda. Kalau postfix hasilnya masih sama dengan nilai operand, kalau prefix hasilnya lebih banyak satu satuan dibanding nilai operand. Aturan ini juga berlaku untuk operator decrement

```
public class DecrementTest{
    public static void main(String[] args){
        int x = 10;
        System.println("x-- = " + x--);
        System.println("Setelah evaluasi, x = " + x);
        x = 10;
        System.println("--x = " + --x);
        System.println("Setelah evaluasi, x = " + x);
    }
}
```

Kalau dicompile dan dijalankan kode di atas, hasilnya adalah sebagai berikut

```
$ javac IncrementTest.java
$ java IncrementTest
x-- = 10
Setelah evaluasi, x = 9
--x = 9
Setelah evaluasi, x = 9
$
```

Untuk bahan latihan, coba tebak berapa output dari kode berikut ini?

```
public class LatihanIncrement{
    public static void main(String[] args){
        int x = 10;
        System.out.println(--x);
        System.out.println(x++ + 21 + ++x);
    }
}
```

tebakkan saya salah :(#abaikan #gapenting.

Operator Kondisi

Operator Kondisi adalah ternary operator, artinya operator ini mempunyai tiga buah operand. Operator kondisi akan mengevaluasi suatu kondisi yang nilainya benar (true) atau salah (false), kemudian mengassign suatu nilai ke dalam variabel. Dengan kata lain, operator ini mirip dengan if tetapi bertujuan untuk mengassign nilai ke sebuah variabel berdasarkan suatu kondisi. Operator kondisi menggunakan simbol ? (tanda tanya) dan : (titik dua). Contohnya sebagai berikut :

```
int x = 100;
String s = (x < 10) ? "kurang dari sepuluh" : "lebih dari sama dengan sepuluh";
```

Kode di atas artinya: kalau variabel x kurang dari sepuluh maka assign String "kurang dari sepuluh" ke dalam variabel s, sebaliknya: kalau x lebih dari sama dengan sepuluh maka assign String "lebih dari sama dengan sepuluh" ke dalam variabel s.

Operator kondisi bisa dirangkai untuk lebih dari satu kondisi, misalnya kalau kondisi pertama benar assign suatu nilai ke variabel, kalau salah tes lagi kondisi kedua, kalau benar assign nilai yang lain ke variabel dan akhirnya kalau kedua kondisi juga salah assign nilai terakhir ke variabel. Struktur kodenya seperti di bawah ini :

```
int x = 100;
String s = (x < 10) ? "kurang dari sepuluh" : (x > 100)
    ? "lebih dari seratus"
    : "lebih besar sama dengan sepuluh dan kurang dari sama dengan seratus";
```

Kode di atas artinya: kalau variabel x kurang dari sepuluh assign String "kurang dari sepuluh" ke variabel s, selainya: kalau x lebih dari seratus assign String "lebih dari seratus" ke dalam variabel s, kalau kedua kondisi salah: assign String "lebih besar sama dengan sepuluh dan kurang dari sama dengan seratus" ke dalam variabel s.

Operator Bitwise

Operator bitwise digunakan untuk melakukan operasi binary terhadap variabel maupun literal dengan tipe angka. Operator bitwise dan operator bit shift (dibahas di bab berikutnya) sangat jarang digunakan dalam aplikasi berbasis database, kalau anda merasa tidak punya banyak waktu untuk belajar manipulasi bit sebaiknya segera loncat ke bab berikutnya. Kalau anda mempunyai rasa keingintahuan tinggi dan ingin melihat low level manipulasi bit, silahkan melanjutkan membaca bab ini dan bab bit shift.

Ada beberapa operator bitwise :

- & (dan) membandingkan dua buah rangkaian binary dan mengembalikan nilai 1 jika keduanya bernilai 1
- | (atau) membandingkan dua buah rangkaian binary dan mengembalikan nilai 1 jika keduanya bernilai 1 atau salah satu bernilai 1

- \wedge (xor) membandingkan dua buah rangkaian binary dan mengembalikan nilai 1 jika hanya salah satu bernilai 1
- \sim negasi, merubah nilai nol menjadi satu dan sebaliknya

Mari kita lihat contoh kode untuk lebih mengerti bagaimana operator ini bekerja :

```
int x = 11;
int y = 13;
int z = x & y;
```

Setelah operasi selesai nilai variabel z adalah 9. Bagaimana cara perhitungannya? kita harus menyusun angka 11 dan 13 dalam bentuk binary, kemudian melakukan operasi & (dan) ke setiap binary di posisi yang sama, hanya kalau keduanya bernilai 1 maka hasilnya 1, sisanya 0. Perhatikan diagram di bawah ini untuk memperjelas operasi & :

```
1 0 1 1 => 8 + 0 + 2 + 1 = 11
1 1 0 1 => 8 + 4 + 0 + 1 = 13
----- &
1 0 0 1 => 8 + 0 + 0 + 1 = 9
```

Sekarang kita lihat contoh untuk operator | (atau) :

```
int x = 11;
int y = 13;
int z = x | y;
```

Setelah operasi selesai nilai variabel z adalah 15. Susun angka 11 dan 13 dalam bentuk binary, kemudian lihat susunan angka binarynya, kalau salah satu atau kedua angka binary pada posisi yang sama bernilai 1 maka hasilnya adalah 1, kalau keduanya 0 maka hasilnya 0. Perhatikan diagram di bawah ini untuk memperjelas operasi | :

```
1 0 1 1 => 8 + 0 + 2 + 1 = 11
1 1 0 1 => 8 + 4 + 0 + 1 = 13
----- |
1 1 1 1 => 8 + 4 + 2 + 1 = 15
```

Berikutnya kita lihat contoh untuk operator \wedge (xor) :

```
int x = 11;
int y = 13;
int z = x ^ y;
```

Setelah operasi selesai nilai variabel z adalah 6. Susun angka 11 dan 13 dalam bentuk binary, hanya kalau salah satu dari kedua angka binary bernilai 1 maka hasil operasinya 1, selain itu hasilnya 0. Perhatikan diagram di bawah ini untuk memperjelas operasi \wedge :

```
1 0 1 1 => 8 + 0 + 2 + 1 = 11
1 1 0 1 => 8 + 4 + 0 + 1 = 13
----- ^
0 1 1 0 => 0 + 4 + 2 + 0 = 6
```

Berikutnya kita lihat contoh untuk operator \sim (negasi) :

```
int x = 11;
int z = ~x;
```

Setelah operasi selesai nilai variabel z adalah -12. Kok bisa? Sebelum kita bahas kenapa hasilnya bisa negatif, perlu kita bahas dahulu bagaimana komputer merepresentasikan bilangan negatif dan positif.

Processor dan memory pada dasarnya hanya mengenal bilangan binari, angka desimal yang kita lihat juga pada dasarnya disimpan sebagai binari. Angka binari tidak mempunyai notasi negatif atau positif, untuk menandakan angka negatif atau positif dalam memory, digunakanlah teori yang disebut dengan 2's complement. Untuk mendapatkan representasi binari nilai negatif dilakukan tiga langkah, pertama ubah nilai positifnya menjadi binary, lakukan negasi dan terakhir tambahkan satu ke hasil negasi.

Mari kita lihat bagaimana membuat representasi binari dari nilai negatif -11. Pertama kita buat

representasi binari dari angka 11, tipe data yang dioperasikan adalah integer, maka panjang datanya adalah 32 bit, seperti di bawah ini :

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1
```

Setelah proses negasi, dimana nilai 0 menjadi 1 dan sebaliknya, maka representasi binarynya menjadi :

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0
```

Langkah terakhir dari proses 2's complement ini adalah menambahkan 1 ke binari hasil negasi, sehingga hasil akhir representasi binari dari -11 adalah :

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1
```

Satu hal yang pasti, kalau angka digit sebelah paling kiri 0 maka angkanya pasti positif, sebaliknya jika digit paling kiri bernilai 1 maka angkanya adalah negatif. Kalau kita mendapatkan representasi binari dari sebuah angka negatif, maka untuk mengetahui berapa nilainya perlu dilakukan operasi kebalikan dari 2's complement : kurangi satu kemudian negasikan menjadi angka positifnya. Nah, kita coba kembalikan bentuk -11 di atas menjadi bentuk positifnya :

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1
----- -1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0
----- ~
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1    => 11 positif
```

Kembali lagi ke operasi negasi dari integer di atas, sekarang kita bisa menjawab kenapa negasi 11 hasilnya adalah -12. Nah kita lihat Operasi berikut ini :

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1    => 11
----- ~
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0
```

kita kembalikan ke bentuk positifnya :

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0
----- -1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0
----- ~
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1    => 12
```

Bingung? Saya juga pas nulis buku ini juga baru ingat bagaimana membuat representasi negatif ke dalam binari, jadi tidak perlu berkecil hati, dalam prakteknya, operasi bit seperti ini jarang sekali digunakan.

Operator Bit Shift

Selain operator bitwise yang mengoperasikan bit secara logika, ada operator bit shift yang menggeser-geser posisi bit. Di java operator bit shift ada 3 :

- >> geser bit ke kanan secara aritmatik
- << geser bit ke kiri secara aritmatik
- >>> geser bit ke kanan secara logika

Sebenarnya ada juga konsep geser bit ke kanan secara logika, tetapi karena hasilnya sama persis dengan geser bit ke kiri secara aritmatik, tidak disediakan operatornya di Java. Untuk memperjelas bagaimana operasi geser-geser bit ini, mari kita lihat contoh di bawah ini:

```
int x = 11;
int y = x >> 2;           //x digeser ke kanan dua langkah
```

Hasilnya adalah y bernilai 2. Perhatikan diagram di bawah ini :

```
1 0 1 1 => 8 + 0 + 2 + 1 = 11
```

>> 2 geser bit di atas ke kanan dua kali dan tambahkan 0 di depan, sehingga 2 buah nilai 1 di sebelah kanan menjadi hilang, hasilnya :

$$0\ 0\ 1\ 0 \Rightarrow 0 + 0 + 2 + 0 = 2$$

Contoh berikutnya adalah geser bit ke kiri :

```
int x = 11;  
int y = x << 2;           //x digeser ke kiri dua langkah
```

Hasilnya y bernilai 12. Perhatikan diagram di bawah ini :

$$1\ 0\ 1\ 1 \Rightarrow 8 + 0 + 2 + 1 = 11$$

>> 2 geser bit di atas ke kiri dua kali dan tambahkan 0 di belakang, sehingga 2 buah nilai 1 di sebelah kanan bergeser ke kiri, hasilnya :

$$1\ 1\ 0\ 0 \Rightarrow 8 + 4 + 0 + 0 = 12$$

Bagaimana kalau operasi bit shift ini dilaksanakan terhadap angka negatif? Operasi penggeseran bit menggunakan tipe data int yang mempunyai panjang 32 bit, kita sudah belajar bagaimana caranya merepresentasikan nilai negatif dalam notasi binari. Misalnya kita akan menggeser -11 ke kanan dua langkah seperti kode di bawah ini :

```
int x = -11;  
int y = x >> 2;           //x digeser ke kiri dua langkah
```

Hasilnya adalah -3, kita lihat bagaimana operasi binarinya

[illegible]

Dapat dilihat di atas, bahwa pergeseran ke kanan angka negatif akan menambahkan angka 1 di sebelah kiri dan menggeser sisanya ke kanan, berbeda dengan pergeseran ke bit ke kanan pada angka positif yang menambahkan angka 0 di sebelah kiri dan menggeser sisanya ke kanan. Hal ini menyebabkan pergeseran ke kanan tidak akan merubah angka positif menjadi negatif atau sebaliknya.

Pergeseran bit ke kiri untuk angka negatif dilakukan dengan menambahkan angka 0 di bit paling kanan, jadi berbeda dengan pergeseran ke kanan yang tergantung angkanya negatif atau positif, untuk pergeseran ke kiri selalu angka 0 yang ditambahkan di posisi paling kanan. Hal ini menyebabkan pergeseran ke sebelah kiri bisa mengakibatkan angka berganti tanda dari positif menjadi negatif atau sebaliknya, tergantung apakah setelah operasi pergeseran nilai digit paling kiri 0 (positif) atau 1 (negatif).

Misalnya kita akan menggeser nilai maksimum positif integer ke kiri dua langkah seperti di kode berikut ini :

```
int x = Integer.MAX_VALUE;  
int y = x << 2;           //x digeser ke kiri dua langkah
```

Nilai y akan menjadi -4 setelah operasi di atas selesai dilaksanakan, mari kita lihat operasi binarinya:

```
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 => MAX_VALUE
----- << 2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 => -4
```

Penggeseran bit ke kanan secara logika sedikit berbeda dengan pergeseran ke kanan secara aritmatik, hal ini terjadi karena angka yang ditambahkan di bit paling kiri selalu 0 tidak tergantung angka yang akan digeser negatif atau positif. Hasil pergeseran bit ke kanan secara logika berbeda dengan pergeseran bit ke kanan secara aritmatik kalau angka yang akan digeser bernilai negatif.

Misalnya kita akan menggeser ke kanan secara logika dua langkah angka -11, seperti kode berikut ini :

```
int x = -11;  
int y = x >>> 2;    //x digeser ke kiri dua langkah
```

Hasilnya adalah y bernilai 1073741821. Perhatikan diagram di bawah ini :

[illegible]

[illegible]

Baiklah, mari kita akhiri bermain-main dengan bit, kita lanjutkan ke operator yang sangat penting yaitu operator logika.

Operator Logika

Di kurikulum ilmu komputer, operasi logika diajarkan dalam satu semester penuh di mata kuliah Logika Matematika, kenapa sampai satu matakuliah sendiri untuk mempelajari logika matematika? Hal ini dikarenakan operasi logika sangat penting dalam sebuah aplikasi, membuat rangkaian logika yang rapi dan mudah dimengerti untuk aplikasi yang rumit adalah pekerjaan yang cukup sulit. Terkadang kita akan mendapati kode dengan banyak sekali kondisi sehingga yang membaca kode menjadi sangat bingung bagaimana alur aplikasi berjalan. Pengalaman saya juga sering mendapati cukup banyak bug aplikasi diakibatkan oleh operasi logika yang salah. Jadi mari kita bahas operator logika di Java dengan keseriusan yang tinggi.

Ada enam logical operator yang bisa digunakan dalam java :

- & operator dan, di bab sebelumnya operator ini juga digunakan sebagai operator bitwise kalau operandnya bertipe angka
- | operator or, di bab sebelumnya operator ini juga digunakan sebagai operator bitwise kalau operandnya bertipe angka
- ^ operator xor, di bab sebelumnya operator ini juga digunakan sebagai operator bitwise kalau operandnya bertipe angka
- ! operator logika negasi
- && operator singkat dan
- || operator singkat or

Mari kita bahas satu per satu operator di atas. Dimulai dari operator `&`, seperti sudah dibahas di atas, operator ini juga digunakan sebagai bitwise kalau operandnya bertipe angka. Kalau operandnya bertipe boolean maka operator `&` akan menghasilkan nilai `true` kalau kedua operand bernilai `true`, selainya akan menghasilkan nilai `false`. Perhatikan contoh kode di bawah ini :

```
int x = 10;
int y = 11;
boolean z = (x > 10) & (y < 100);
```

Variabel z akan bernilai false karena operand sebelah kiri ($x > 10$) bernilai false dan operand sebelah kanan ($y \geq 10$) bernilai true. Kedua operasi di sisi kiri dan kanan akan tetap dieksekusi walaupun sudah dipastikan bahwa operand sebelah kiri bernilai false dan apapapun nilai operand di sebelah kanan tetap akan menghasilkan nilai false.

Berbeda dengan operator `&&`, dimana operator `&&` tidak akan mengeksekusi operand di sebelah kanan kalau sudah tahu bahwa operand sebelah kiri bernilai false. Hal inilah alasan kenapa operator `&&` dinamakan operator singkat dan. Perilaku operator `&&` ini penting kalau kita ingin memastikan bahwa suatu variabel object tidak bernilai null sebelum memanggil method dari object tersebut, perhatikan contoh kode di bawah ini:

```
public class LogicalOperator{
    public static void main(String[] args){
        String nama = null;
        boolean z = (nama != null) & nama.equals("ifnu");
        System.out.println(z);
    }
}
```

Kode di atas akan mengakibatkan `NullPointerException` karena walaupun sudah tahu bahwa operan sebelah kiri bernilai `false`, operand sebelah kanan tetap dieksekusi. Lihat hasil eksekusi kode di atas :

```
$ javac LogicalOperator.java
```

```
$ java LogicalOperator
Exception in thread "main" java.lang.NullPointerException
    at LogicalOperator.main(LogicalOperator.java:4)
$
```

Nah sekarang kita ubah sedikit kode di atas dengan mengganti operator & menjadi operator && seperti kode di bawah ini :

```
public class LogicalOperator{
    public static void main(String[] args){
        String nama = null;
        boolean z = (nama != null) && nama.equals("ifnu");
        System.out.println(z);
    }
}
```

Hasil eksekusinya akan lancar tidak mengakibatkan adanya NullPointerException.

```
$ javac LogicalOperator.java
$ java LogicalOperator
false
$
```

Operator | juga digunakan dalam operasi bitwise jika operandnya adalah angka. Jika operandnya berupa boolean, operator | akan menghasilkan nilai true kalau salah satu atau kedua operand bernilai true. Operand | hanya akan menghasilkan nilai false kalau kedua operand bernilai false. Aturan ini berlaku juga untuk operator ||, perbedaannya adalah kalau operator | kedua operand akan tetap dieksekusi walaupun operand sebelah kiri bernilai true. Padahal kalau operand di kiri bernilai true, apapun nilai operand sebelah kanan pasti hasilnya adalah true. Sedangkan untuk operator ||, kalau operand sebelah kiri bernilai true maka operand sebelah kanan tidak akan dieksekusi dan nilai true dikembalikan sebagai hasil operasinya. Silahkan edit sedikit kode di atas dan ganti operator & menjadi | untuk mengetahui bagaimana hasil evaluasi kode di atas kalau menggunakan operator |.

Operator ^ disebut operator xor, hasil operasi xor akan bernilai true kalau nilai operand di sebelah kiri berbeda dengan operand di sebelah kanan. Misalnya operand sebelah kiri bernilai true dan sebelah kanan bernilai false atau sebaliknya. Kalau kedua operand sama sama false atau sama sama true maka hasilnya adalah false. Perhatikan contoh di bawah ini :

```
int x = 10;
int y = 11;
boolean z = (x > 10) ^ (y < 100);
```

variabel z akan bernilai true karena (x > 10) bernilai false dan (y < 100) bernilai true.

Operator ! disebut sebagai inversi atau negasi, operator ini akan membalik nilai boolean true menjadi false dan false menjadi true. Misalnya kode di bawah ini :

```
int x = 10;
int y = 11;
boolean z = !((x > 10) ^ (y < 100));
```

variabel z akan bernilai false karena operasi (x > 10) ^ (y < 100) bernilai true.

Secara umum, hanya operator &&, || dan ! yang akan kita gunakan dalam contoh aplikasi di buku ini, sisanya jarang sekali digunakan dalam aplikasi sebenarnya.

Flow Control

Kontrol terhadap aliran eksekusi aplikasi adalah salah satu feature utama dari sebuah bahasa pemrograman. Java menyediakan beberapa cara untuk melakukan kontrol aliran eksekusi aplikasi. if dan switch adalah dua jenis yang paling sering digunakan, selain itu ada mekanisme exception yang bisa mempengaruhi aliran eksekusi program. Dalam bab ini kita akan membahas ketiganya secara intensif, terutama bagaimana best practice penggunaan exception.

If

Statement if digunakan dalam aplikasi untuk mengatur aliran eksekusi program berdasarkan kondisi tertentu. Sintaks dasarnya adalah sebagai berikut :

```
int x = 10;
if(x == 10 ) {
    System.out.println("lakukan sesuatu di sini kalau kondisi di dalam if bernilai true");
}
```

Kode di dalam kurung kurawal buka dan tutup akan dieksekusi kalau kondisi di dalam statement if bernilai true. Statement if bisa digabungkan dengan else, kode di dalam else akan dieksekusi kalau kondisi di dalam if bernilai false, contohnya :

```
int x = 11;
if(x == 10 ) {
    System.out.println("lakukan sesuatu di sini kalau kondisi di dalam if bernilai true");
} else {
    System.out.println("lakukan sesuatu di sini kalau kondisi di dalam if bernilai false");
}
```

Ada juga bentuk else if yang dirangkai untuk mengetes beberapa kondisi, seperti di bawah ini

```
int nilai = 78;
if(nilai < 40 ) {
    System.out.println("nilai < 40 = D");
} else if (nilai >= 40 && nilai < 60) {
    System.out.println("nilai >= 40 && nilai < 60 = C");
} else if (nilai >= 60 && nilai < 70) {
    System.out.println("nilai >= 60 && nilai < 80 = B");
} else if (nilai >= 80 && nilai < 100) {
    System.out.println("nilai >= 80 && nilai <=100 = A");
} else {
    System.out.println("range nilai harus antara 0 - 100");
}
```

Tanda kurung kurawal tidak diperlukan kalau blok kode yang dieksekusi di dalam if hanya satu baris saja, tetapi praktek ini tidak dianjurkan karena menimbulkan kebingungan, misalnya :

```
int x = 0;
if(x >= 0)
    System.out.println("nilai positif");
```

Perhatikan bahwa kalau if tidak diikuti dengan tanda kurung kurawal maka yang dianggap sebagai blok kode di dalam if adalah satu statement tepat di setelah if, kalau ada beberapa statement setelah if, maka sisanya tidak dianggap bagian dari blok kode if. Contohnya :

```
int x = 0;
if(x >= 0)
    System.out.println("nilai positif");
    System.out.println("selalu dieksekusi. Bukan bagian dari blok kode if");
```

Kalau kode di atas dieksekusi, println kedua akan selalu dieksekusi apapun nilai kondisi di dalam if, hal ini karena println kedua bukan bagian dari blok kode untuk if.

Switch

Switch biasanya digunakan kalau kita ingin mengevaluasi nilai dari sebuah variabel yang mempunyai banyak kemungkinan. Bentuk sintaks switch lebih singkat dan bersih dibanding kalau menggunakan if-else if-else. Hingga Java 6 switch hanya bisa digunakan kalau variabelnya bertipe angka atau enum. Berikut ini bentuk dasar dari switch :

```
int x = 10;
```

```

switch(x){
    case 1 :
        System.out.println("satu");
        break;
    case 2 :
        System.out.println("dua");
        break;
    default :
        System.out.println("bukan satu atau dua");
}

```

Dalam switch terdapat konsep yang disebut dengan fall-through, konsep ini kalau kita bayangkan mirip dengan efek domino, kalau kartu pertama rubuh akan menimpa kartu berikutnya menimpa kartu berikutnya lagi seterusnya hingga ada yang menghentikan efek dominonya. Lihat kode di atas, terdapat keyword break yang digunakan untuk menghentikan fall-through. Kalau kita hilangkan break dari kode di atas, maka kalau ada satu kondisi dari case yang dipenuhi maka kode di dalam case di bawahnya akan terus dieksekusi hingga ketemu break atau blok kode switch berakhir.

Agar memudahkan pemahaman tentang fall-through ini, coba buat kode seperti di bawah ini dan coba eksekusi kodenya:

```

public class FallThrough{
    public static void main(String[] args){
        int x = 1;
        switch(x){
            case 1 :
                System.out.println("satu");
            case 2 :
                System.out.println("dua fall-through");
            case 3 :
                System.out.println("tiga fall-through");
            case 4 :
                System.out.println("empat fall-through");
            default :
                System.out.println("default fall-through");
        }
    }
}

```

Hasil eksekusinya :

```

$ javac FallThrough.java
$ java FallThrough
satu
dua fall-through
tiga fall-through
empat fall-through
default fall-through
$

```

Perhatikan bahwa setelah kondisi di case benar, maka semua kode di dalam case akan dieksekusi hingga ketemu break atau blok kode switch berakhir. Kalau kita ubah sedikit kode di atas dengan meletakkan break di dalam case 1 seperti di bawah ini :

```

public class FallThrough{
    public static void main(String[] args){
        int x = 1;
        switch(x){
            case 1 :
                System.out.println("satu");
                break; //stop fall-through dan keluar dari switch

```

```

        case 2 :
            System.out.println("dua fall-through");
        case 3 :
            System.out.println("tiga fall-through");
        case 4 :
            System.out.println("empat fall-through");
        default :
            System.out.println("default fall-through");
    }
}
}

```

Maka hasil eksekusinya akan berbeda :

```

$ javac FallThrough.java
$ java FallThrough
satu
$

```

if dan switch adalah flow control yang digunakan kalau aplikasi berjalan dengan baik, artinya flow control oleh if dan switch digunakan untuk mengendalikan aliran eksekusi aplikasi kalau semesta berjalan seperti yang dikehendaki. Java mempunyai cara untuk mengontrol aliran aplikasi kalau terjadi error di dalam aplikasi, namanya adalah exception. Kita bahas apa itu exception di bab berikutnya.

Exception

Aplikasi biasanya tidak pernah berjalan mulus 100% setiap waktu, justru banyak sekali kejadian dimana aplikasi mengalami kondisi yang tidak diinginkan, entah ada masalah dengan data yang tidak benar, input/output yang tiba-tiba terputus, koneksi ke database yang tidak lancar hingga error yang sengaja dibuat oleh user untuk mencari celah keamanan aplikasi. Kode yang ditulis untuk handle masalah dan membuat aplikasi tangguh dalam segala medan sangat penting perannya, bahkan menurut penelitian jumlahnya mencapai 80% dari total kode yang dibuat developer.

Mengingat pentingnya penanganan kesalahan, Java menyediakan feature penanganan kesalahan yang elegan, mudah digunakan dan tepat sasaran. Feature ini disebut dengan exception handling, dengan feature ini kita tidak perlu mengetest hasil kembalian dari suatu fungsi untuk mengetahui apakah ada error apa tidak, dengan begitu pengecekan kesalahan bisa lebih elegan dan jelas.

Sintaks Exception

Keyword yang digunakan dalam exception handling ada tiga : try, catch, finally, throw dan throws. Kita akan belajar bagaimana menggunakan try dan catch terlebih dahulu baru kemudian kita belajar tentang finally. Contoh-contoh kode akan menggunakan beberapa class input/output untuk membaca file, tidak perlu bingung bagaimana caranya menggunakan I/O, kita akan membahas I/O dalam satu bab tersendiri, yang penting pahami dahulu bagaimana sintaks exception handling serta best practicenya. Karena pentingnya konsep ini, sebaiknya ulangi membaca bab ini jika belum paham benar bagaimana konsepnya.

Sintaks penggunaan try dan catch adalah sebagai berikut ini :

```

try {
    //di sini adalah baris yang dilindungi oleh try,
    //letakkan kode yang potensial mengalami exception di sini
} catch(ClassException variabelException){
    //kalau terjadi exception lakukan sesuatu
    //setidaknya lakukan logging
    //blok catch yang kosong bisa dikategorikan sebagai kode konyol dan mengerikan
    //pastikan ada kode untuk memberi keterangan tentang exception apa yang terjadi
} catch(ExceptionLain variabelException){

```



```
//idem  
}
```

Kode di dalam try{} adalah kode yang melempar exception (throw), kalau kode di dalamnya berjalan normal dan tidak ada exception yang terjadi, eksekusi akan sampai ke baris terakhir. Sebaliknya, kalau ada exception dengan tipe ClassException maka eksekusi akan loncat dalam blok catch pertama, kalau ada exception dengan tipe ExceptionLain maka eksekusi akan loncat ke dalam blok catch kedua.

Misalnya kita ingin membaca file dari harddisk, kira-kira struktur kodenya akan seperti di bawah ini :

```
try{  
    bukaFileDariHarddisk  
    bacaFileBarisPerBaris  
    tampilkanIsiFileKeConsole  
} catch(IOException ex){  
    tampilkanErrorDiLogDanConsole  
}  
}
```

Kalau kode untuk buka file dari harddisk gagal, maka eksekusi langsung loncat ke catch, kode untuk baca baris dan menampilkan isi file tidak akan dieksekusi. Dengan begitu, aplikasi tetap bisa berjalan dan tidak langsung keluar.

Sintaks exception berikutnya adalah try-catch-finally. Ada tambahan satu blok kode lagi, yaitu finally. Kalau kode di dalam try menimbulkan exception maka kode akan loncat ke dalam blok kode di catch, sedangkan kode di dalam finally akan selalu dieksekusi baik exception terjadi di dalam try atau tidak. Kode yang ada dalam finally biasanya adalah kode untuk melepaskan resource, membersihkan variabel yang sudah tidak dipake lagi, bisa dibilang kode di finally itu digunakan untuk bersih-bersih aplikasi agar tidak menggunakan resource yang sudah tidak diperlukan lagi. Sintaksnya adalah seperti di bawah ini :

```
try {  
    //kode yang ada exception  
} catch (ExceptionPertama ex){  
    //handle exception dengan tipe ExceptionPertama  
} catch (ExceptionKedua ex){  
    //handle exception dengan tipe ExceptionKedua  
} finally{  
    //bersihkan resource yang dipakai, baik terjadi exception ataupun tidak  
}  
}
```

try bisa langsung diikuti oleh finally tanpa adanya catch sama sekali, hal ini bisa dilakukan kalau di dalam try exception yang terjadi adalah unchecked exception. Artinya, exceptionnya tidak wajib ditangkap (catch), loh kalau tidak ditangkap apa yang terjadi? Yang terjadi adalah konsep yang disebut dengan uncaught exception atau exception yang tidak tertangkap yang pada akhirnya menyebabkan aplikasi berhenti berjalan. Untuk mengetahui bagaimana mekanismenya saya akan menjelaskan tentang stack trace.

Call Stack, Stack Trace dan Uncaught Exception

Call Stack adalah urutan pemanggilan method di dalam aplikasi java. Awal dari pemanggilan method tentu saja adalah method main, kemudian di dalam method main tersebut pasti ada kode untuk memanggil method lain dalam suatu class, kemudian method tersebut memanggil lagi method yang lain dan seterusnya sehingga proses pemanggilan methodnya bertumpuk-tumpuk. Nah tumpukan pemanggilan method ini yang kita sebut dengan call stack.

Bayangkan call stack itu layaknya sebuah gedung bertingkat, fondasi gedung bertingkat tersebut adalah method main. Setiap lantai di atasnya adalah method lain yang dipanggil oleh method main dan seterusnya sampai ke atas. Uncaught exception menyebabkan sebuah method keluar dari eksekusinya untuk dilanjutkan ke method berikutnya sambil berharap bahwa ada kode untuk menangkap exception itu. Dalam ilustrasi gedung bertingkat, uncaught exception adalah kejadian dimana sebuah lantai dalam gedung runtuh, runtuhnya lantai tersebut berharap ada yang menahan (menangkap) di lantai bawahnya, kalau tidak ada

satupun lantai yang menangkap runtuhannya ini maka secara keseluruhan gedung akan kolaps, atau aplikasi akan berhenti bekerja.

Stack trace adalah rekaman jejak call stack ketika exception terjadi, kita bisa melihat urutan pemanggilan method dari awal sampai akhir dimana exception terjadi. Dengan adanya stack trace kita bisa menganalisis root cause (penyebab utama) terjadinya exception. Menemukan root cause dari sebuah exception sangat penting dalam proses memperbaiki aplikasi, selama root cause belum ditemukan, biasanya kita hanya bisa menebak-nebak apa yang terjadi dengan aplikasi tetapi tidak bisa memastikan bagian mana yang bermasalah. Stack trace dengan gamblang akan menunjukkan root cause terjadinya exception, penting sekali untuk melakukan log terhadap stack trace ini agar mempercepat pencarian kesalahan dalam aplikasi.

Mari kita lihat ilustrasi di atas dengan menggunakan kode. Kita akan membuat sebuah class dengan beberapa method, di dalam method ini akan memanggil method lain sampai pada method terakhir kita sengaja membuat kode yang menghasilkan exception, kemudian kita akan bisa melihat stack trace dari exception tersebut dan menentukan root cause-nya.

```
public class StackTraceRootCause {
    public void methodPertama() {
        System.out.println("method pertama dipanggil");
        methodKedua();
    }
    public void methodKedua() {
        System.out.println("method kedua dipanggil");
        methodKetiga();
    }
    public void methodKetiga() {
        System.out.println("method ketiga dipanggil");
        methodKeempat();
    }
    public void methodKeempat() {
        System.out.println("method keempat dipanggil");
        methodKelima();
    }
    public void methodKelima() {
        System.out.println("method kelima dipanggil");
        String abc = null;
        abc.toString(); //kode ini akan menyebabkan NullPointerException
    }
    public static void main(String[] args) {
        StackTraceRootCause strc = new StackTraceRootCause();
        strc.methodPertama();
        System.out.println("kode ini tidak akan dieksekusi " +
            "karena program sudah keluar " +
            "ketika exception di methodKelima tidak ditangkap");
    }
}
```

Kita lihat kode di atas, terdapat enam buah method: methodPertama sampai methodKelima dan method main. Method main akan memanggil methodPertama, methodPertama akan memanggil methodKedua dan seterusnya hingga pada akhirnya methodKelima sengaja dibuat agar melempar NullPointerException tanpa satupun method sebelumnya menangkapnya.

Perhatikan ilustrasi call stack di bawah ini :

6 methodKelima()
5 methodKeempat()
4 methodKetiga()
3 methodKedua()
2 methodPertama()
1 main()

Call Stack

Method main sebagai fondasi dari call stack adalah method pertama yang dipanggil, berikutnya methodPertama hingga methodKelima. Di dalam methodKelima terjadi exception yang tidak ditangkap oleh method di bawahnya sehingga aplikasi keluar.

Kode di atas kalau dieksekusi akan menghasilkan keluaran seperti berikut ini :

```
$ javac StackTraceRootCause.java
$ java StackTraceRootCause
method pertama dipanggil
method kedua dipanggil
method ketiga dipanggil
method keempat dipanggil
method kelima dipanggil
Exception in thread "main" java.lang.NullPointerException
    at StackTraceRootCause.methodKelima(StackTraceRootCause.java:21)
    at StackTraceRootCause.methodKeempat(StackTraceRootCause.java:16)
    at StackTraceRootCause.methodKetiga(StackTraceRootCause.java:12)
    at StackTraceRootCause.methodKedua(StackTraceRootCause.java:8)
    at StackTraceRootCause.methodPertama(StackTraceRootCause.java:4)
    at StackTraceRootCause.main(StackTraceRootCause.java:25)
$
```

Stack trace adalah output yang dimulai dari kalimat "Exception in thread "main"..." hingga ke bawah. Dari output stack trace di atas kita bisa melihat dengan jelas apa jenis exception yang terjadi, dari mana awal mulanya hingga di class apa dan di baris berapa kejadian exception itu terjadi. Pemahaman yang sangat baik terhadap stack trace ini sangat menentukan kemampuan developer dalam mencari kesalahan aplikasi untuk berikutnya membuat solusi dari kesalahan (bug) tersebut.

Ada pepatah kuno developer bilang: "kemampuan developer junior dan senior dalam menulis program tidak akan jauh berbeda karena dibatasi kemampuan manusia mengetik yang terbatas, tetapi kemampuan junior developer dan senior dalam menganalisa kesalahan plus mencari solusi kesalahan itu bisa berbeda hingga puluhan ribu kali. Junior developer bisa berhari hari tidak bisa menemukan kesalahan dan solusinya, sedangkan senior developer bisa hanya dengan sekali lihat langsung tahu dimana salahnya dan bagaimana mengatasi kesalahan tersebut".

Class Exception

Exception bisa berasal dari dua tempat: JVM dan Aplikasi. Exception yang berasal dari JVM bersifat umum dan bisa terjadi di semua aplikasi, seperti di contoh sebelumnya kita mendemonstrasikan terjadinya NullPointerException, ClassNotFoundException dan IndexOutOfBoundsException. Kita juga bisa membuat class exception sendiri yang spesifik terhadap aplikasi.

Membuat class exception yang spesifik terhadap aplikasi kita sangat disarankan, dan merupakan best practice yang sering disarankan. Mari kita lihat bagaimana caranya membuat class exception untuk aplikasi kita sendiri :

```
public class ExceptionPertama extends Exception{
    public ExceptionPertama() {}
    public ExceptionPertama(String pesan) {
        super(pesan);
    }
}
```

Perhatikan bahwa class `ExceptionPertama` di atas harus menextends class `Exception` yang sudah disediakan oleh JDK. Dalam kode di atas ada dua buah keyword yang belum kita bahas yaitu `super` dan `extends`, tidak perlu khawatir tentang dua buah keyword tersebut, kita akan membahasnya nanti di bab tentang OOP, sekarang fokuskan perhatian ke bagaimana cara membuat class exception.

Setelah class exception dibuat seperti di atas, kita akan membuat kode sederhana yang menggunakan class exception di atas. Perhatikan kode di bawah ini :

```
public class ExceptionPertamaTest {
    public static void main(String[] args) {
        try{
            System.out.println("eksekusi exception pertama");
            throw new ExceptionPertama("ini pesan exception pertama");
        } catch(ExceptionPertama ex){
            ex.printStackTrace();
        }
    }
}
```

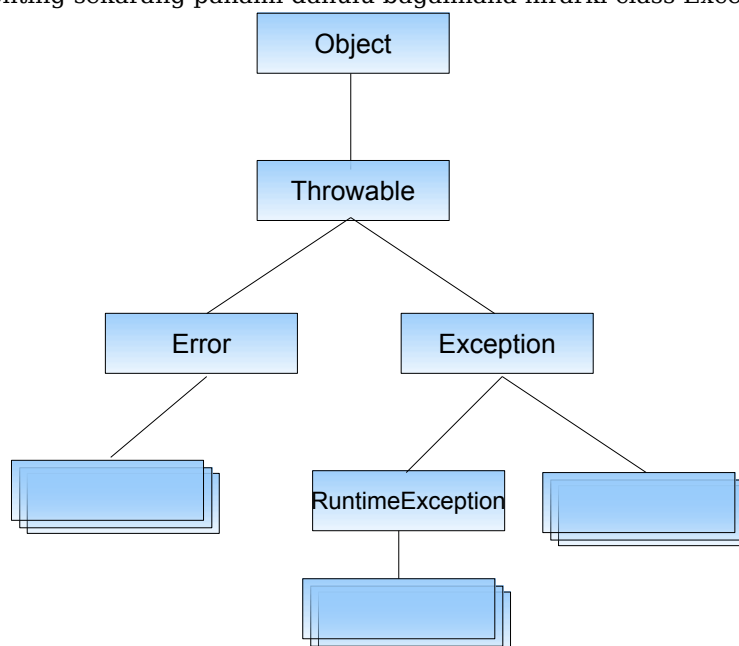
Class di atas mendemonstrasikan bagaimana class exception yang sudah kita buat digunakan. Terlihat ada sintaks `try-catch` yang digunakan untuk `handle` exception yang kita `throw`. Tidak perlu khawatir tentang apa itu keyword `throw`, kita akan bahas di bab selanjutnya.

Class exception secara default akan mempunyai semua method yang dimiliki oleh class `Exception`, salah satu method yang sering digunakan adalah `printStackTrace`. Fungsi ini gunanya adalah untuk menampilkan stack trace ke console. Kalau sudah lupa apa itu stack trace, silahkan buka-buka lagi bab sebelumnya yang membahas stack trace.

Seperti yang kita lihat, cara membuat class exception yang spesifik untuk aplikasi kita sangat mudah, silahkan buat class exception yang mendeskripsikan error yang spesifik, hal ini memudahkan analisa stack trace kalau terjadi exception dalam aplikasi.

Hirarki Class Exception

Dalam bab ini kita akan membahas hirarki class exception, kita belum membahas konsep OOP cukup banyak hingga bab ini, jadi kalau belum terlalu mengerti apa itu inheritance, jangan terlalu khawatir, yang penting sekarang pahami dahulu bagaimana hirarki class `Exception`.



Seperti yang anda bisa lihat, hirarki kelas exception dimulai dari Object, kemudian ada class Throwable yang merupakan parent class dari semua jenis Exception. Ada dua buah class turunan langsung dari class Throwable yaitu class Error dan class Exception. Class Exception sendiri mempunyai satu turunan yaitu Runtime Exception.

Error biasanya tidak disebabkan karena kesalahan yang terjadi dalam aplikasi, tetapi lebih karena keadaan JVM yang tidak normal, misalnya error karena kehabisan memory. Error tidak perlu dihandle dalam try-catch karena tidak ada cara untuk mengatasi keadaan Error. Secara teknis sebenarnya error bukanlah exception karena bukan merupakan turunan dari Exception.

Berbeda dengan Error, Exception merepresentasikan kesalahan yang terjadi dalam aplikasi. Kita bisa melakukan recovery untuk mengatasi Exception. JDK mempunyai banyak sekali class turunan dari Exception, nama class-class ini sangat descriptive sehingga hanya dengan melihat nama class Exceptionnya kita sudah bisa menebak kesalahan apa yang terjadi dalam aplikasi.

Class RuntimeException adalah class spesial yang disebut juga dengan unchecked exception. Nama unchecked exception digunakan karena RuntimeException tidak wajib dicatch. Contoh turunan dari RuntimeException adalah NullPointerException, ClassCastException, IndexOutOfBoundsException dan masih banyak lagi.

Mari kita lihat contoh kode yang menyebabkan RuntimeException :

```
public class RuntimeExceptionTest {
    public static void main(String[] args){
        int i = Integer.parseInt("abc");
        System.out.println("kode setelah exception");
    }
}
```

Kode di atas akan menyebabkan RuntimeException karena berusaha mengubah string "abc" menjadi angka. Perhatikan bahwa tidak wajib ada statement try-catch untuk menghandle RuntimeException. Tanpa ada try-catch dalam kode di atas, maka kalau terjadi exception kode akan segera keluar dari method, nah karena ini adalah method main dan tidak ada lagi method yang lain (ingat analogi call stack dengan gedung tinggi), aplikasi akan langsung keluar, dan string "kode setelah exception tidak akan ditampilkan".

```
$ javac RuntimeExceptionTest.java
$ java RuntimeExceptionTest
Exception in thread "main" java.lang.NumberFormatException: For input string:
"abc"
    at
java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
    at java.lang.Integer.parseInt(Integer.java:449)
    at java.lang.Integer.parseInt(Integer.java:499)
    at RuntimeExceptionTest.main(RuntimeExceptionTest.java:3)
$
```

Misalnya kita tidak yakin dengan String yang akan diubah menjadi integer berhasil atau tidak, kita bisa meletakkan statement try-catch, kalau ada error nilai integer kita set menjadi 0, lanjutkan program ke baris berikutnya, modifikasi kodenya seperti di bawah ini :

```
public class RuntimeExceptionTest {
    public static void main(String[] args){
        int i = 0;
        try{
            i = Integer.parseInt("abc");
        } catch(NumberFormatException ex) {
            ex.printStackTrace();
        }
        System.out.println("kode setelah exception");
    }
}
```

Kalau kita jalankan kode di atas, maka setelah terjadi exception pada waktu mengeksekusi

method `parseInt`, eksekusi akan meloncat ke dalam `catch`, mencetak stack trace ke console kemudian melanjutkan eksekusi untuk mencetak string "kode setelah exception" ke console. Dengan modifikasi di atas, kode menjadi lebih handal dan tidak menyebabkan program keluar sebelum eksekusi selesai semuanya.

```
$ javac RuntimeExceptionTest.java
$ java RuntimeExceptionTest
java.lang.NumberFormatException: For input string: "abc"
    at
    java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
        at java.lang.Integer.parseInt(Integer.java:449)
        at java.lang.Integer.parseInt(Integer.java:499)
        at RuntimeExceptionTest.main(RuntimeExceptionTest.java:5)
kode setelah exception
$
```

Class `Exception` yang bukan turunan dari `RuntimeException` disebut dengan checked exception. Kode yang didalamnya ada exception dengan tipe ini harus diletakkan dalam blok `try`, kalau tidak, maka akan terjadi kompilasi error. Sebagai contoh kita akan menulis kode untuk membuka file, kode ini mengandung checked exception yaitu `IOException`. Pertama kita lihat dulu kode tanpa `try`, kode ini akan menyebabkan adanya kompilasi error :

```
import java.io.FileInputStream;
public class CheckedExceptionTest {
    public static void main(String[] args) {
        FileInputStream inputStream = new FileInputStream("buka-file.txt");
        System.out.println("kode setelah buka file");
    }
}
```

hasil kompilasi kode di atas adalah :

```
$ javac CheckedExceptionTest.java
CheckedExceptionTest.java:4: unreported exception java.io.FileNotFoundException;
must be caught or declared to be thrown
    FileInputStream inputStream = new FileInputStream("buka-file.txt");
                                ^
1 error
$
```

terlihat pada output proses kompilasi di atas bahwa konstruktor `new FileInputStream("buka-file.txt");` melempar exception yang harus ditangkap (`catch`) atau dideklarasikan untuk dilempar (`throw`) lagi. Di bab ini kita baru belajar bagaimana menangkap exception, sedangkan bagaimana caranya melempar lagi exception kita bahas di bab berikutnya.

Untuk mengatasi error kompilasi di atas, kita akan menambahkan `try-catch` di kode di atas.

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class CheckedExceptionTest {
    public static void main(String[] args) {
        try{
            FileInputStream inputStream = new FileInputStream("buka-file.txt");
        } catch(FileNotFoundException ex){
            ex.printStackTrace();
        } catch(IOException ex) {
            ex.printStackTrace();
        }
        System.out.println("kode setelah buka file");
    }
}
```

Perhatikan bahwa konstruktor `new FileInputStream("buka-file.txt");` menyebabkan dua buah exception yaitu `FileNotFoundException` dan `IOException`. Kedua exception ini ditangkap dan dihandle dalam catch yang berbeda, kita bisa juga menangkap hanya `IOException` karena pada dasarnya `FileNotFoundException` adalah turunan `IOException`, sehingga cukup ditangkap sekali saja maka kedua tipe sudah dihandle. Contohnya seperti di bawah ini :

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class CheckedExceptionTest {
    public static void main(String[] args) {
        try{
            FileInputStream inputStream = new FileInputStream("buka-file.txt");
        } catch(IOException ex) {
            ex.printStackTrace();
        }
        System.out.println("kode setelah buka file");
    }
}
```

`IOException` dan `FileNotFoundException` adalah turunan `Exception`, sehingga bisa juga kita tangkap dengan tipe `Exception`, kalau ini yang kita lakukan maka semua class exception turunan dari `Exception` akan ikut ditangkap juga. Praktek “tangkap semua exception tanpa pandang bulu” seperti ini dikategorikan sebagai praktek kurang baik, karena semua jenis exception akan dihandle dengan cara yang sama, padahal kita bisa menghandle exception dengan lebih baik kalau bisa memilah-milah cara menghandle exception berdasar tipenya. Menangkap exception terbalik dengan menangkap koruptor, kalau koruptor harus ditangkap tanpa pandang bulu, sedangkan exception harus ditangkap dengan cara dipilah-pilah berdasarkan tipenya.

Throws dan Throw

Bab sebelumnya sudah membahas bahwa untuk menghandle exception caranya ada dua : cara pertama yaitu dengan menggunakan try-catch yang sudah kita bahas, cara kedua adalah dengan melempar exception ke method pemanggilnya. Solusi kedua ini terbalik dengan solusi pertama yang langsung menghandle exception di tempat, solusi kedua ini malah melempar tanggung jawab ke method pemanggilnya untuk menghandle exception, jadi methodnya cuci tangan dan tidak bertanggung jawab. Nah sekarang pertanyaanya, bagaimana kalau tidak ada satupun method pemanggilnya yang menangkap exception ini? Kalau hal ini terjadi, maka aplikasi akan keluar dari eksekusi (ingat analogi call stack dengan gedung tinggi).

Sintaks untuk melempar tanggung jawab menangkap exception menggunakan keyword `throws` yang diletakkan di deklarasi method. Hmm, bingung? Mari kita lihat contohnya agar tidak bingung. Misalnya kita tidak ingin menangkap exception `IOException` dan `FileNotFoundException` seperti dalam contoh sebelumnya, kodenya seperti berikut ini :

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class ThrowsExceptionTest {
    public static void main(String[] args) throws FileNotFoundException,
        IOException {
        FileInputStream inputStream = new FileInputStream("buka-file.txt");
        System.out.println("kode setelah buka file");
    }
}
```

Perhatikan kode di atas, tanpa ada try-catch kodenya tidak akan menyebabkan error pada waktu kompilasi. Kuncinya ada pada keyword `throws` di sebelah method `main`, dengan adanya `throws` ini maka method `main` tidak wajib melakukan try-catch terhadap kedua tipe exception. Nah karena method `main` adalah method paling bawah dari call stack, maka kalau ada exception yang tidak dihandle oleh method `main` ini aplikasi akan langsung keluar. Ingat, aturan ini hanya berlaku untuk checked exception, sedangkan unchecked exception tidak perlu

try-catch ataupun throws.

Kita lihat lagi contoh bagaimana kalau exception dilempar di satu method dan pada akhirnya exception tersebut ditangkap di method pemanggilnya. :

```
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
public class ThrowsExceptionTest {
    public static void main(String[] args) {
        try {
            methodTidakBertanggungJawab();
        } catch (FileNotFoundException ex){
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        System.out.println("kode di dalam method main");
    }
    public static void methodTidakBertanggungJawab() throws FileNotFoundException,
        IOException {
        FileInputStream inputStream = new FileInputStream("buka-file.txt");
        System.out.println("kode setelah buka file");
    }
}
```

di dalam method `methodTidakBertanggungJawab`, exception tidak dihandle tetapi dilempar ke method pemanggilnya dengan menggunakan sintaks `throws`. Sedangkan method `main` yang memanggil method `methodTidakBertanggungJawab`, akhirnya menghandle exception sehingga eksekusi aplikasi tidak langsung keluar dan dilanjutkan dengan mencetak string di bawahnya baru kemudian keluar.

Beberapa bab terakhir kita terus membahas bagaimana caranya menghandle exception dengan cara menangkap atau melempar exception ke method pemanggilnya. Nah, kalau kita menangkap sesuatu pasti pada awalnya kan ada yang melempar exception ini. Hukum sebab-akibat juga berlaku untuk exception, dimana ada sebab (throw exception) ada juga akibat (handle exception dengan try-catch atau throws). Keyword `throw` (tanpa s) adalah awal dimana exception dibuat untuk kemudian dilempar pertama kalinya.

Sintaks `throws` sangat sederhana, pertama kita harus membuat object dari class `Exception` atau turunannya, kemudian letakkan keyword `throw` diikuti object dari class `Exception` tersebut, mirip-mirip dengan penggunaan `return`. Contoh bagaimana menggunakan `throw` bisa dilihat dalam bab `Class Exception`.

Keyword `throw` banyak digunakan dalam class-library dalam java untuk memberitahu kode dalam aplikasi kita bahwa telah terjadi error. Sebagian besar library java yang berhubungan dengan I/O (Input/Output) akan `throw` exception kalau ada masalah dalam I/O, misalnya dalam contoh pembacaan file di atas ada exception yang menerangkan bahwa file tidak ketemu (`FileNotFoundException`) atau ada masalah pada saat pembacaan file (`IOException`).

Keyword `throw` jarang sekali digunakan dalam kode aplikasi, kecuali dalam keadaan dimana kita ingin memberitahu bagian lain dalam aplikasi bahwa sudah terjadi error. Arsitektur aplikasi modern biasanya mengikuti pola "separation of concern", artinya pemisahan kode-kode tertentu dalam lapisan berbeda. Misalnya kode untuk mengakses database dan kode untuk user interface (UI) diletakkan dalam lapisan yang berbeda, tidak boleh bercampur. Class-class yang digunakan untuk mengakses database tidak boleh bocor ke lapisan UI dan sebaliknya class-class UI tidak boleh sampai merembes ke class untuk mengakses database. Nah, kalau misalnya class yang mengakses database ingin melempar exception ke class UI, maka kita harus membuat class exception sendiri dan class inilah yang akan kita lempar ke class UI.

Bagaimana implementasi "separation of concern" ini akan kita bahas lagi lebih lanjut di bab setelah bab `Java Fundamental`.

Perulangan / Iterasi

Perulangan atau Iterasi adalah konsep yang selalu ada dalam semua bahasa pemrograman, begitu juga dengan java. Bentuk iterasi dalam Java ada beberapa, yang pertama adalah for, kemudian ada while dan do-while. Kita akan membahas satu per satu setiap bentuk perulangan di bagian berikutnya.

for

Keyword for digunakan untuk melakukan iterasi kalau jumlah iterasinya tertentu, misalnya kita ingin melakukan iterasi sebanyak 10x atau 100x. Iterasi menggunakan for memerlukan tiga buah parameter, parameter pertama digunakan sebagai batas bawah iterasi, parameter kedua adalah kondisi kapan iterasi berhenti dan parameter terakhir digunakan untuk menaikkan atau menurunkan counter. Kita lihat sintaks dasar dari for seperti di bawah ini :

```
for( parameter1 ; parameter2 : parameter3) {  
}
```

sintaks iterasi for diawali dengan keyword for, kemudian diikuti dengan kurung buka, setelahnya ada tiga buah parameter yang dipisahkan dengan titik dua (:), kurung buka tutup dan diakhiri dengan pasangan kurung kurawal buka-tutup untuk menandakan blok kode yang akan dieksekusi berulang-ulang hingga iterasi berhenti. Berikut ini contoh konkrit penggunaan iterasi menggunakan for :

```
public class ForTest {  
    public static void main(String[] args) {  
        for(int i = 0; i < 10; i++) {  
            System.out.println("iterasi ke-" + i);  
        }  
    }  
}
```

Perhatikan sintaks for di atas, parameter pertama diisi dengan `int i = 0`, ini artinya adalah kita akan menggunakan variabel `i` sebagai counter (penghitung) untuk for diawali dari 0. Parameter kedua diisi dengan `i < 10`, artinya iterasi for akan terus berlangsung selama `i` bernilai kurang dari 10 dan berhenti ketika `i` sudah mencapai nilai sepuluh. Parameter ketiga diisi dengan `i++` artinya setiap kali iterasi maka counter `i` akan naik satu nilai. Ketiga parameter ini mengindikasikan bahwa iterasi dilakukan dari `i` bernilai 0, berakhir ketika `i` mencapai nilai 10 dan setiap kali iterasi `i` naik satu-satu, total akan terjadi 10x perulangan.

Hasil eksekusi dari kode di atas adalah sebagai berikut :

```
$ javac ForTest.java  
$ java ForTest  
iterasi ke-0  
iterasi ke-1  
iterasi ke-2  
iterasi ke-3  
iterasi ke-4  
iterasi ke-5  
iterasi ke-6  
iterasi ke-7  
iterasi ke-8  
iterasi ke-9  
$
```

Iterasi for juga bisa berlangsung mundur, contohnya seperti di bawah ini :

```
public class ForMundurTest {  
    public static void main(String[] args) {  
        for(int i = 10; i > 0; i--) {  
            System.out.println("iterasi ke-" + i);  
        }  
    }  
}
```

```
}  
}
```

Hasil eksekusinya akan sedikit berbeda, seperti di bawah ini :

```
$ javac ForMundurTest.java  
$ java ForMundurTest  
iterasi ke-10  
iterasi ke-9  
iterasi ke-8  
iterasi ke-7  
iterasi ke-6  
iterasi ke-5  
iterasi ke-4  
iterasi ke-3  
iterasi ke-2  
iterasi ke-1  
$
```

Ingat, iterasi for digunakan kalau kita tahu berapa kali iterasi dilaksanakan, hal ini dikarenakan iterasi for memerlukan parameter awal, kondisi berhenti dan berapa banyak counter bertambah/berkurang dalam setiap iterasi. Kalau iterasi tidak diketahui berapa kali dilakukan dan hanya tahu bahwa selama suatu kondisi tertentu masih benar iterasi akan terus berlangsung maka kita bisa menggunakan while.

while

Seperti yang sudah kita bahas sedikit di atas, bentuk iterasi while digunakan kalau kita tidak tahu berapa kali iterasi akan berlangsung, tetapi tahu kapan iterasi berhenti dari sebuah kondisi tertentu. Bentuk sintaks while sangat sederhana dan hanya memerlukan satu saja parameter kondisi yang bernilai boolean true atau false, selama kondisi ini masih bernilai true maka iterasi akan terus dilaksanakan. Kalau nilai kondisi adalah false maka iterasi while akan berhenti. Bentuk sintaksnya adalah seperti di bawah ini :

```
while( kondisi ) {  
}
```

Sintaks while diawali dengan keyword while kemudian diikuti dengan kurung buka, di dalam kurung buka ada satu kondisi, dan diakhiri kurung tutup, setelah itu ada pasangan kurung kurawal buka tutup.

Kita akan membuat contoh kecil yang akan mengambil nilai waktu sekarang, mengetes apakah waktu sekarang ini kalau dibagi 3 sisanya 0, selama tidak memenuhi kondisi di atas maka iterasi dijalankan terus menerus. Kode untuk contoh di atas adalah seperti berikut ini :

```
public class WhileTest {  
    public static void main(String[] args) {  
        while(System.currentTimeMillis() % 3 != 0) {  
            System.out.println("waktu sekarang dibagi 3 masih ada sisanya");  
        }  
    }  
}
```

Jumlah iterasi dalam kode di atas tidak menentu, terkadang iterasi bisa terjadi banyak kali, sedikit kali atau bahkan tidak terjadi iterasi sama sekali.

```
$ javac WhileTest.java  
$ java WhileTest  
waktu sekarang dibagi 3 masih ada sisanya  
waktu sekarang dibagi 3 masih ada sisanya  
$
```

Kalau kita ingin agar iterasi terjadi setidaknya sekali saja maka kita bisa menggunakan do-while yang akan kita bahas di bab berikutnya.

do-while

Bentuk iterasi do-while digunakan kalau kita ingin agar setidaknya satu iterasi terjadi baru kemudian dites kondisi apakah iterasi berikutnya dilaksanakan atau tidak. Sama dengan bentuk iterasi while, bentuk iterasi do-while juga memerlukan satu parameter kondisi di dalam while-nya. Sintaks do-while adalah seperti berikut ini :

```
do {  
} while( kondisi );
```

Kita modifikasi sedikit kode di atas dengan menggunakan bentuk iterasi do-while, seperti di bawah ini :

```
public class WhileTest {  
    public static void main(String[] args) {  
        do {  
            System.out.println("waktu sekarang dibagi 3 masih ada sisanya");  
            while(System.currentTimeMillis() % 3 != 0);  
        }  
    }  
}
```

kalau kita eksekusi kode di atas, maka setidaknya string di atas diprint sekali, kalau misalnya kondisi di dalam while masih benar maka string-nya akan diprint lagi sampai kondisi di dalam while bernilai false.

Sampai di sini pembahasan sintaks java sudah mencukupi, di bagian selanjutnya kita akan membahas OOP dengan java. Pembahasan OOP akan lebih banyak berikisar tentang teori dan praktek pemrograman yang baik. Walaupun Java adalah bahasa pemrograman OOP, tetapi kalau kita tidak mempraktekkan OOP dengan baik, bentuk kodenya malah akan mirip bahasa prosedural. Jadi praktek penggunaan OOP memerlukan disiplin dari perogrammernya.

OOP dengan Java

Menjadi developer Java, mengharuskan anda untuk menyelami konsep OOP dengan sangat dalam. Tiap hari anda harus bermimpi tentang inheritance dan hirarki class Java, kekuatan polymorphism menjadi senjata pamungkas kode anda, kohesi dan loosely coupling menjadi napas anda pertama bangun tidur, dan composition menjadi sarapan anda di pagi hari. #eaa #lebay.

Sudah banyak terjadi kasus dimana seorang developer java yang sudah berpengalaman bertahun-tahun tetapi konsep OOPnya sangat lemah, sehingga kode java malah mirip kode C dimana banyak method static dan tidak ada design pattern yang diimplementasikan sama sekali. Atau programmer java yang awalnya developer Visual Basic menulis kode java sama persis dengan biasanya menggunakan VB, yaitu dengan meletakkan semua kode yang diperlukan dalam satu blok kode action dari sebuah tombol. Hasilnya tentu saja spaghetti code, enak tapi kelihatan campur aduk, ruwet, berceceran dan berantakan.

Maka dari itu, OOP menjadi topik yang sangat penting dalam buku ini. Konsep OOP harus benar-benar dikuasai agar kode yang dihasilkan menjadi rapi, self explained: hanya dengan melihat kode sudah paham apa maksudnya tanpa membaca dokumentasi dan logis. Penerapan konsep OOP membuat kode menjadi sangat berkualitas dan harganya mahal karena kualitas yang baik tersebut.

Untuk menambah motivasi membuat kode berkualitas baik, bayangkan bahwa anda akan menjadi developer selama beberapa bulan saja, kemudian ada seseorang yang akan memantain kode anda selama berpuluh-puluh tahun berikutnya. Orang ini adalah megalomaniac madosastic berdarah dingin tanpa perasaan yang tidak pernah mandi berbulan-bulan. Setiap kali maintainer ini naik emosinya karena harus membaca kode anda yang berantakan, dia akan segera menuju rumah dimana anda tinggal, kemudian menyiksa anda dengan benda tumpul, menjepit anda di antara keteknya yang bau dan berusaha menyayat anda dengan golok berkarat. #lebaylagi #horor.

Konsep OOP yang baik juga dapat membuat aplikasi kita menjadi cukup modular, dalam beberapa kasus kita bisa mengganti satu class saja untuk melakukan enhancement terhadap

aplikasi tanpa perlu mengotak-atik class lainya. Misalnya suatu perhitungan diletakkan dalam sebuah interface, ketika dalam development kita mengimplementasikan sebuah class dengan perhitungan tertentu. Ketika beberapa saat aplikasi berjalan, ternyata ada perhitungan yang berbeda, nah kita tinggal mengimplementasikan interface yang sama dan menginisialisasi perhitungan dengan menggunakan class yang kedua. Hoplaaa, maka aplikasi akan menghitung dengan menggunakan class kedua ini.

Implementasi OOP yang baik akan menawarkan dua kualitas ke dalam aplikasi kita : modular (modularity) dan mudah dipelihara (maintenability). Tetapi kualitas tersebut tidak datang begitu saja, anda harus berusaha keras dengan disiplin tinggi untuk mencapainya. Java memberi anda landasan yang kuat dengan feature OOP, tetapi Java tidak bisa mencegah anda menulis kode yang tidak berkualitas baik, anda masih tetap bisa menulis kode Java dengan gaya prosedural ataupun gaya spaghetti.

Nah kita akan membahas satu per satu apa itu OOP, siapkan mental anda karena konsep OOP cukup abstract dan perlu waktu untuk dicerna. Silahkan baca bab per bab secara berulang-ulang hingga anda mengerti esensi dari teori OOP. Sekalian juga hafalkan bagaimana konsep tersebut kalau diimplementasikan ke dalam bentuk kode.

Enkapsulasi

Enkapsulasi adalah konsep dimana sebuah logic kode dibungkus dalam satu bentuk kode yang menggambarkan dunia nyata. Class adalah sebuah enkapsulasi dari perilaku yang diwakilinya dalam dunia nyata. Setiap entitas dalam dunia nyata mempunyai karakteristik dan perilaku. Karakteristik ini diwujudkan sebagai property, sedangkan perilaku diwujudkan dalam sebuah method. Kedua hal ini : karakteristik dan perilaku dienkapsulasi menjadi satu entitas dalam class.

Bahasa prosedural tidak mempunyai feature enkapsulasi ini, sehingga kalau kita melihat kode kita tidak bisa melihat wujud abstraksi kode tersebut dengan dunia nyata. Semua hanya berbentuk fungsi / method, sedangkan datanya berceceran di sana sini. OOP memberikan konsep yang sangat praktis tentang bagaimana menjembatani logic yang ada di dunia nyata dengan wujudnya dalam kode. Sehingga ketika kita bicara dengan user aplikasi yang tidak mengetahui kode, kita tetap bisa bicara dengan istilah yang sama. Misalnya kita berbicara tentang customer, kita bicara customer yang nyata dan juga sekaligus berbicara tentang class Customer yang ada dalam kode.

Dengan adanya enkapsulasi, programmer yang pada awalnya tidak terlibat dalam analisis aplikasi, dan hanya bisa melihat kode bisa dengan mudah mengasosiasikan kode tersebut dengan keadaan nyata. Enkapsulasi menyebabkan kode menjadi self explained, artinya hanya dengan melihat kode kita dapat mengetahui apa logic di balik kode ini, atau kita bisa mengetahui apa maksud dari kode yang sedang kita baca.

Java dari awal didesain sebagai bahasa untuk membuat aplikasi Enterprise. Sebenarnya antara aplikasi enterprise dan aplikasi main-main tugas kuliah kodenya tidak jauh berbeda, yang jauh berbeda adalah ukurannya. Kalau aplikasi main-main tugas kuliah, ukuran kodenya paling masih beberapa kilobyte saja, tetapi aplikasi dengan skala enterprise bisa sampai bergiga-giga. Developer yang mengerjakan aplikasi juga jauh sekali perbandingannya, karena aplikasi enterprise bisa dikerjakan hingga ratusan orang. Dengan ukuran sebesar itu, tidak mungkin membuat dokumentasi yang mendetail tentang kodenya, oleh karena itu sebisa mungkin setiap kode yang ditulis oleh developer sudah self explained.

Inheritance, IS-A dan HAS-A

Inheritance (turunan) ada di mana-mana dalam Java. Sangat tidak mungkin membuat aplikasi tanpa melibatkan konsep Inheritance, faktanya setiap class yang kita tulis adalah turunan dari class Object. Anda ingin bukti? Ingat-ingat lagi bab tentang operator, di sana ada satu operator instanceof, operator ini digunakan untuk mengetes apakah sebuah object merupakan tipe dari suatu class. Operator ini akan mengembalikan true kalau tipe dari object tersebut adalah class yang ada di operand sebelah kanan dari operator instanceof, sebaliknya nilainya adalah false. Kita lihat contohnya di bawah ini :

```
public class InheritanceTest {
    public static void main(String[] args){
        InheritanceTest test = new InheritanceTest();
        if(test instanceof Object) {
```

```

        System.out.println("test tipenya Object");
    } else {
        System.out.println("test tipenya bukan Object");
    }
}
}

```

Kalau kita compile kode di atas dan dijalankan, outputnya adalah seperti di bawah ini :

```

$ javac InheritanceTest.java
$ java InheritanceTest
test tipenya Object
$

```

Terbukti bahwa walaupun tidak secara eksplisit kita definisikan InheritanceTest sebagai turunan dari Object, faktanya InheritanceTest adalah turunan dari Object karena operator instanceof mengembalikan nilai true ketika object dari InheritanceTest dioperasikan dengan class Object.

Implementasi inheritance dalam kode menggunakan keyword extends. Kalau kita ingin membuat sebuah class turunan dari class yang lain, kita gunakan keyword extends setelah deklarasi nama class diikuti dengan class yang ingin digunakan sebagai orang tua dari class tersebut. Misalnya kita ingin akan membuat dua buah class, class pertama adalah Customer sedangkan class kedua adalah MemberCustomer, dimana class MemberCustomer adalah turunan dari class Customer. Kode class Customer adalah seperti di bawah ini :

```
public class Customer {}
```

kode class MemberCustomer adalah seperti di bawah ini

```
public class MemberCustomer extends Customer {}
```

Class MemberCustomer dalam hal ini mempunyai hubungan IS-A dengan class Customer. Kalimat lengkapnya adalah "MemberCustomer IS-A Customer". Hal ini dikarenakan MemberCustomer adalah turunan Customer, analoginya kita bisa menggunakan kucing dan hewan. Kucing adalah turunan dari hewan, maka kucing adalah hewan. Sama juga seperti MemberCustomer adalah turunan dari Customer, maka MemberCustomer adalah Customer.

Apa tujuan dari inheritance dilihat dari sisi kode? Yang pertama adalah mempromosikan code reuse, yaitu penggunaan kembali kode yang sudah ditulis. Hal ini dikarenakan, sebagai turunan dari Customer, MemberCustomer akan mempunyai semua sifat-sifat (kode) dari class Customer. Misalnya kita ubah sedikit kode di atas, kita tambahkan property dan method di dalam class Customer :

```

public class Customer {
    private Long id;
    public void setId(Long aId){
        this.id = aId;
    }
    public Long getId() {
        return this.id;
    }
}

```

kemudian class MemberCustomer tetap seperti semula :

```
public class MemberCustomer extends Customer {}
```

Buat lagi satu class untuk mengetes konsep inheritance :

```

public class CustomerTest {
    public static void main(String[] args) {
        MemberCustomer member = new MemberCustomer();
        member.setId(1001);
        System.out.println("id customer : " + member.getId());
    }
}

```

Kalau kita compile ketiga class di atas dan dijalankan outputnya adalah sebagai berikut ini :

```
$ javac MemberCustomer.java Customer.java CustomerTest.java
$ java CustomerTest
id customer : 100
$
```

Terlihat bahwa tidak ada error dalam proses kompilasi, padahal class MemberCustomer tidak mendefinisikan apapun, tetapi method setId dan getId dapat digunakan. Hal ini dikarenakan MemberCustomer mengextends Customer, sehingga method setId dan getId diturunkan. Catat juga bahwa property id sendiri tidak diturunkan ke dalam MemberCustomer, karena property id menggunakan private sebagai access modifier.

Hubungan HAS-A lebih sederhana penjelasannya dibanding hubungan IS-A. Hubungan HAS-A atau biasa disebut dengan komposisi (composition) terjadi kalau sebuah class mempunyai sebuah property. Misalnya kita bisa melihat dalam kode di atas Customer HAS-A Long dengan nama id.

Kita akan membuat sebuah satu buah class lagi dengan nama Address, class ini akan digunakan oleh class Customer untuk mendefinisikan alamat Customer. Kodenya seperti di bawah ini :

```
public class Address {
    private String street;
    private String city;
    private String postCode;
}
```

Hubungan antara Customer dan Address adalah HAS-A, diimplementasikan dengan membuat property di dalam class Customer dengan tipe Address. Kita ubah sedikit kode dari class Customer di atas menjadi seperti di bawah ini :

```
public class Customer {
    private Long id;
    private Address address;
    public void setId(Long aId){
        this.id = aId;
    }
    public Long getId() {
        return this.id;
    }
    public void setAddress(Address aAddress) {
        this.address = aAddress;
    }
    public Address getAddress() {
        return this.address;
    }
}
```

Kode di atas memperlihatkan hubungan Customer HAS-A Address.

Manfaat kedua dari inheritance adalah polimorfisme, kita akan membahas tentang polimorfisme di bab di bawah ini.

Polimorfisme (Polymorphism)

Arti polimorfisme secara harfiah (makna kata) adalah banyak bentuk. Sebuah object bisa diassign ke dalam tipe yang berbeda-beda. Contohnya MemberCustomer, class MemberCustomer IS-A :

- Object, karena semua class pasti extends Object
- Customer, karena MemberCustomer extends Customer
- MemberCustomer

Jadi kalau ada variabel dengan tipe Object, Customer atau MemberCustomer bisa diassign dengan instance dari MemberCustomer. Contohnya seperti di bawah ini :

```
Object o = new MemberCustomer();
```

```
Customer c = new MemberCustomer();
MemberCustomer mc = new MemberCustomer();
Object object = mc;
Customer cust = mc;
```

Interface juga bisa digunakan dalam skenario IS-A ini, misalnya untuk class MemberCustomer bisa mengimplementasikan sebuah interface, kemudian interface ini bisa digunakan sebagai tipe variabel yang bisa diassign dengan instance dari MemberCustomer. Class MemberCustomer akan kita ubah sedikit seperti berikut ini :

```
public class MemberCustomer extends Customer implements Serializable {}
```

Sekarang kita bisa mendeklarasikan variabel dengan tipe Serializable dan mengassign instance dari MemberCustomer ke dalam variabel tersebut. Atau dengan kata lain MemberCustomer IS-A Serializable, seperti contoh di bawah ini :

```
Serializable s = new MemberCustomer();
```

Jadi MemberCustomer mempunyai banyak bentuk (polimorfisme). Kita akan sering melihat kode dengan pola seperti ini, variabel akan dideklarasikan dengan tipe interface tetapi diinstansiasi dengan class yang mengimplementasikan interface tersebut. Praktek seperti ini dikategorikan sebagai praktek yang baik sehingga banyak diikuti oleh developer Java.

Overriding dan Overloading

Setiap kali sebuah class mengextends class lain, class tersebut bisa mengoverride method yang ada di dalam class orang tuanya. Alasannya karena di class anaknya ingin mengimplementasikan method tersebut dengan cara yang berbeda. Misalnya dalam class MemberCustomer akan mengoverride method setId untuk memeriksa apakah id yang dimasukkan bernilai null atau tidak, kalau nilainya null maka sebuah pesan dicetak ke dalam console. Setelah pesan dicetak, method setId yang dimiliki oleh orang tua dari class Customer, hal ini bisa dilakukan dengan menggunakan keyword super. Contohnya adalah seperti ini :

```
import java.io.Serializable;
public MemberCustomer extends Customer implements Serializable {
    public void setId(Long aId) {
        if(id == null) {
            System.out.println("nilai id tidak boleh null");
        } else {
            super.setId(aId);
        }
    }
}
```

Overriding method dilakukan dengan cara mendeklarasikan method yang sama persis dengan method yang ada di class orang tuanya. Deklarasi dari method harus sama persis, mulai dari access modifiernya, keyword yang digunakan, nama method, jumlah parameter dan letak parameter, hingga deklarasi throws exception.

Apa yang terjadi kalau misalnya kita mendeklarasikan sebuah variabel dengan tipe Customer tetapi diinstansiasi dengan object MemberCustomer, kemudian kita panggil method setId? Apakah method setId yang ada dalam Customer atau MemberCustomer yang akan dieksekusi? Method yang dioverride akan terikat dengan instance dari variabel, bukan tipe dari variabel, jadi jawaban dari pertanyaan di atas adalah : yang dieksekusi adalah method setId dari class MemberCustomer bukan dari class Customer.

Mari kita tulis kode untuk melihat bagaimana aturan ini diimplementasikan dalam kode :

```
public OverridingTest{
    public static void main(String[] args){
        Customer c = new Customer();
        Customer mc = new MemberCustomer();
        Long id = null;
        //method setId yang dipanggil adalah yang dari class Customer karena c
        //dideklarasikan dengan tipe Customer dan diinstansiasi dengan class
```

```

//Customer
c.setId(id);
//method setId yang dipanggil adalah yang dari class MemberCustomer
//walaupun mc dideklarasikan dengan tipe Customer
mc.setId(id);
}
}

```

Method overloading adalah salah satu feature dalam bahasa pemrograman Java, dimana dua buah method bisa dideklarasikan dengan nama yang sama asal argumennya berbeda, baik dari jumlahnya, tipenya atau urutan dari parameternya. Berikut ini adalah method yang berbeda walaupun namanya sama :

```

public void setId(Long aId) {}
public void setId(Integer aId) {}
public void setId(Long aId, boolean checkNull) {}
public void setId(boolean checkNull, Long aId) {}

```

Overloading dan overriding juga berlaku terhadap constructor, karena pada dasarnya constructor adalah method.

Casting Variabel Reference

Polimorfisme memungkinkan sebuah variabel yang dideklarasikan dengan menggunakan class orang tuanya diinstansiasi dengan object dari class anaknya. Hal ini sesuai dengan kaidah hubungan IS-A. Nah, bagaimana kalau misalnya variabel yang sudah dideklarasikan dengan class orang tua kemudian diinstansiasi dengan menggunakan class anaknya akan diassign lagi ke variabel lain dengan tipe class anaknya? Hal ini bisa dilakukan dengan menggunakan casting. Sintaks casting sudah kita pelajari sedikit di bab tipe data primitif, kita akan ulang sedikit di bab ini untuk tipe data reference.

Kita lihat bagaimana casting terhadap tipe data reference :

```

Customer c = new MemberCustomer();
//casting berhasil karena c diinstansiasikan dengan class MemberCustomer
MemberCustomer mc = (MemberCustomer) c;
c = new Customer();
//casting gagal dan terjadi ClassCastException karena c diinstansiasikan
//dengan class Customer
mc = (MemberCustomer) c;
String s = "ini string";
//kode ini gagal dicompile karena String tidak mempunyai hubungan sama sekali
//dengan MemberCustomer
mc = (MemberCustomer) s;

```

Interface

Interface sudah dibahas sedikit di dalam bab deklarasi interface. Dalam bab ini akan kita bahas lebih lanjut tentang aturan-aturan mengimplementasikan interface. Ketika kita mendeklarasikan class yang mengimplementasikan interface, sebenarnya kita sudah menandatangani kontrak untuk mengimplementasikan semua method di dalam interface.

Kita ambil contoh lagi interface PersonDao yang sudah kita gunakan di bab sebelumnya. Kode interface PersonDao adalah seperti di bawah ini :

```

public interface PersonDao{
    void save(Person p);
    void delete(Person p);
    Person getById(Long id);
}

```

Misalnya kita buat class PersonDaoImpl yang akan mengimplementasikan interface PersonDao di atas, maka semua method PersonDao harus diimplementasikan juga, seperti di bawah ini :

```

public class PersonDaoImpl implements PersonDao{

```



```

public void save(Person p){
    System.out.println("menyimpan Person");
}
public void delete(Person p){
    System.out.println("menghapus person");
}
public Person getById(Long id){
    Person p = new Person();
    p.setId(id);
    p.setNama("abc");
    return p;
}
}

```

Ada satu cara yang bisa dilakukan jika kita tidak ingin mengimplementasikan semua method yang ada dalam interface, yaitu dengan menambahkan keyword `abstract` dalam deklarasi class serta mendeklarasikan method yang tidak ingin diimplementasikan menggunakan keyword `abstract`, seperti dalam kode `PersonDaoImpl` yang sedikit dimodifikasi di bawah ini :

```

public abstract class PersonDaoImpl implements PersonDao{
    public void save(Person p){
        System.out.println("menyimpan Person");
    }
    public void delete(Person p){
        System.out.println("menghapus person");
    }
    public abstract Person getById(Long id);
}

```

Pada dasarnya interface itu adalah sebuah class yang dideklarasikan dengan keyword `abstract` dan semua methodnya bersifat `public abstract`. Interface `PersonDao` statusnya setara dengan class `PersonDao` di bawah ini :

```

public abstract class PersonDao{
    public abstract void save(Person p);
    public abstract void delete(Person p);
    public abstract Person getById(Long id);
}

```

Tetapi tetap saja interface dan class adalah dua buah hal yang berbeda.

Interface, seperti halnya class, bisa mengextends interface lain. Interface yang mengextends interface lain akan memiliki semua method yang dipunyai oleh interface orang tuanya.

Collection dan Generics

Kemampuan menggunakan collection adalah wajib dimiliki oleh developer Java. Tanpa kemampuan menggunakan collection, membuat aplikasi yang mengolah data menjadi cukup sulit. Collection diajarkan dalam kurikulum ilmu komputer dalam kuliah struktur data, karena collection sendiri adalah sebuah struktur data. Kita bisa menyimpan data dengan dalam collection, struktur penyimpanan datanya bisa dipilih dari beberapa jenis collection yang sudah disiapkan oleh Java.

Generics banyak digunakan dalam library collections, dengan menggunakan generics kita bisa menentukan tipe dari isi collections. Jika akan membahas tentang List, Set dan Map, tetapi sebelum itu kita akan membahas bagaimana menentukan dua buah object sama atau berbeda dengan cara mengoverride method `equals` dan `hashCode`. Bagian terakhir kita akan membahas tentang bagaimana melakukan sorting terhadap collection agar isinya terurut.

toString

Setiap class selalu extends class `Object`, class `Object` memiliki tujuh buah method, setiap

method mempunyai tujuan tertentu. Method-method tersebut adalah :

- wait()
- notify()
- notifyAll()
- finalize()
- toString()
- equals()
- hashCode()

Method toString digunakan oleh Java untuk mendapatkan representasi String dari sebuah object atau class. Implementasi default dari Class object adalah mencetak nama class diikuti oleh "alamat memory dari object tersebut". Mari kita lihat bagaimana implementasi standard dari method toString ini :

```
public class ToStringTest {
    public static void main(String[] args) {
        ToStringTest test = new ToStringTest();
        System.out.println("implementasi toString dari class Object " +
            "menghasilkan : " + test);
    }
}
```

kalau kita compile dan jalankan kode di atas hasilnya :

```
$ javac ToStringTest.java
$ java ToStringTest
implementasi toString dari class Object menghasilkan : ToStringTest@54fc9944
$
```

terlihat bahwa string yang dikembalikan dari method toString tidak deskriptif sama sekali, dengan mengoverride method toString kita bisa mendapatkan string yang lebih deskriptif untuk menggambarkan object tersebut. Misalnya seperti di bawah ini :

```
public class ToStringTest {
    public String toString() {
        return "ini toString dari class ToStringTest";
    }
    public static void main(String[] args) {
        ToStringTest test = new ToStringTest();
        System.out.println("implementasi toString dari class Object " +
            "menghasilkan : " + test);
    }
}
```

Hasil eksekusi dari kode di atas akan menghasilkan output seperti di bawah ini :

```
$ javac ToStringTest.java
$ java ToStringTest
implementasi toString dari class Object menghasilkan : ini toString dari class
ToStringTest
$
```

method toString banyak digunakan dalam aplikasi Java, salah satunya adalah menampilkan text di dalam komponen JComboBox kalau item yang dimasukkan ke dalam JComboBox adalah class yang kita buat sendiri, bukan tipe data primitif atau class wrapper.

equals dan hashCode

Method equals dan hashCode berperan sangat penting dalam collection. Method equals digunakan untuk membandingkan antara dua buah object apakah sama atau tidak secara logis. Operator == bisa digunakan untuk membandingkan dua buah object, tetapi perbandingan ini

hanya akan menghasilkan true kalau dua buah object apakah berada dalam memory yang sama, atau bisa dikatakan dua buah object ini mempunyai reference yang sama persis. Kalau operator == mengembalikan nilai true berarti dua buah object ini adalah sama persis baik secara alamat memory dan otomatis sama secara logis.

Method equals akan mengembalikan true kalau kedua object sama secara logis walaupun kedua object mempunyai reference berbeda (tidak berada di memory yang sama). Kita ambil contoh sebuah String, dua buah object string akan mengembalikan false jika dioperasikan dengan operator == walaupun string yang ada di dalamnya sama. Tetapi method equals akan mengembalikan nilai true walaupun objectnya berada di memory berbeda asalkan nilai stringnya sama.

Mari kita lihat bagaimana penjelasan di atas kalau dilihat dalam kode. Perhatikan kode berikut ini :

```
public class EqualsTest {
    public static void main(String[] args) {
        String abc = new String("abc");
        String abc1 = new String("abc");
        boolean equalsOperator = (abc == abc1);
        System.out.println("abc == abc ? " + equalsOperator);
        boolean equalsMethod = abc.equals(abc1);
        System.out.println("abc.equals(abc) ? " + equalsMethod);
    }
}
```

apakah output dari kode di atas? Menggunakan operator == maka hasilnya adalah false karena kedua variabel String diinisialisasi dengan menggunakan new String. Sedangkan menggunakan method equals hasilnya adalah true karena kedua variabel mempunyai nilai string yang sama.

```
$ javac EqualsTest.java
$ java EqualsTest
abc == abc ? false
abc.equals(abc) ? true
$
```

Kalau ingin membandingkan dua buah object apakah sama secara logis, kita akan menggunakan method equals, tetapi masalahnya adalah implementasi method equals dari class Object sama persis dengan operator ==, sehingga harus dioverride agar walaupun dua buah object yang berbeda tetap dianggap sama asalkan suatu kriteria terpenuhi. Seperti yang terjadi dengan class String dalam contoh di atas, method equalsnya dioverride agar mengembalikan true kalau dua buah object dari class String mempunyai nilai string yang sama.

Misalnya untuk class Customer, dua buah object Customer dianggap sama asalkan idnya sama, oleh karena itu method equals akan dioverride seperti di bawah ini :

```
public class Customer {
    private Long id;
    public void setId(Long aId){
        this.id = aId;
    }
    public Long getId() {
        return this.id;
    }
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
    }
}
```

```

        final Customer other = (Customer) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }
}

```

Terlihat mengoverride method equals bukan pekerjaan yang gampang, cukup banyak kode yang harus diketik, nantinya menggunakan NetBeans proses pembuatan method equals bisa digenerate secara otomatis, sehingga meminimalisasi kode yang harus diketik manual.

Method hashCode akan mengembalikan nilai integer unik untuk setiap object yang berbeda. Aturannya adalah :

- Method hashCode dari sebuah object harus mengembalikan nilai yang sama walaupun dieksekusi berkali kali selama nilai property dalam object tidak berubah.
- Kalau dua buah object dioperasikan dengan method equals mengembalikan nilai true maka method hashCode dari kedua object harus mengembalikan nilai integer yang sama. Sebaliknya, kalau dua buah object mengembalikan nilai false maka hashCode untuk kedua object akan mengembalikan nilai integer yang berbeda.
- Kalau dua buah object dioperasikan dengan method equals mengembalikan nilai false maka method hashCode **tidak** harus mengembalikan nilai yang berbeda. Mengembalikan nilai yang sama pun tidak masalah.

Topik bagaimana menghitung nilai hashCode yang baik bisa menjadi topik PhD, tetapi tidak perlu bingung, menggunakan NetBeans kita bisa mengenerate method hashCode ini agar mematuhi aturan di atas. Class Customer menggunakan property id sebagai nilai unik, asal nilai idnya berbeda maka dianggap dua buah object Customer yang berbeda, oleh karena itu nilai hashCode akan dihitung dari id ini. Nah, kode lengkap class Customer setelah method hashCode dioverride adalah seperti di bawah ini :

```

public class Customer {
    private Long id;
    public void setId(Long aId){
        this.id = aId;
    }
    public Long getId() {
        return this.id;
    }
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Customer other = (Customer) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }
    public int hashCode() {
        int hash = 7;
        hash = 53 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}

```

hashCode sangat vital ketika kita menggunakan collection yang memanfaatkan nilai hashCode ini, seperti HashSet atau HashMap. Perhitungan hashCode yang salah akan membuat kedua jenis collection ini berantakan. Kita akan kembali lagi ke topik hashCode ketika membahas kedua collection tersebut di bab berikutnya.

Java Collection Framework

Membuat aplikasi yang cukup kompleks tanpa adanya collection yang baik akan sangat repot, banyak hal harus dilakukan hanya untuk membuat collection yang memenuhi kebutuhan kita. Untungnya dalam JDK terdapat banyak sekali collection yang sudah tersedia, kita hanya perlu mempelajari API tersebut dan memahami kapan saat terbaik untuk menggunakan collection API tersebut.

Dalam Java collection API ada beberapa interface kunci yang merupakan dasar dari collection yang ada dalam JDK. Interface-interface itu antara lain :

Collection	Set	SortedSet
List	Map	SortedMap
Queue	NavigableSet	NavigableMap

Dalam buku ini kita akan membahas beberapa interface saja yang sering dibutuhkan dalam aplikasi, antara lain: List, Set, Map dan Queue. Daftar class yang mengimplementasikan keempat interface di atas antara lain :

Map	Set	List	Queue
HashMap	HashSet	ArrayList	PriorityQueue
HashTable	LinkedHashSet	Vector	
TreeMap	TreeSet	LinkedList	
LinkedHashMap			

Setiap jenis collection di atas mempunyai empat varian : teratur (sorted), tidak teratur (unsorted), teratur (ordered) dan tidak teratur (unordered). Collection yang teratur (sorted) artinya isi dari collection tersebut teratur berdasarkan nilai tertentu dari objectnya, misalnya kalau yang dimasukkan dalam collection adalah object dari Customer, kita bisa mengurutkan berdasarkan id atau berdasarkan nilai property yang lain. Kita akan membahas lebih lanjut bagaimana caranya melakukan sorting atau menggunakan sorted collection di bab selanjutnya. Collection tidak teratur jelas kebalikan dari collection teratur.

Collection teratur (ordered) adalah collection yang diatur berdasarkan kapan item tersebut dimasukkan ke dalam collection, item yang dimasukkan pertama akan berada di awal, sedangkan item yang dimasukkan terakhir akan berada di akhir collection. Sedangkan collection tidak teratur (unordered) jelas kebalikan dari collection teratur.

Nah mari kita bahas satu per satu dari keempat collection di atas beserta variannya, dimulai dari List.

List

List adalah jenis collection yang teratur tetapi tidak teratur. List mempunyai index yang disusun berdasarkan urutan kapan item dimasukkan ke dalam List. Isi dari List bersifat tidak unik, alias dua buah item yang sama bisa dimasukkan berkali kali ke dalam List. Method penting dalam List antara lain :

- `get(int index)`; method ini digunakan untuk mengambil isi dari list berdasarkan index (urutan item dimasukkan ke dalam List)
- `indexOf(Object o)`; method ini digunakan untuk mengetahui berapa nomor index dari object yang ada dalam List.
- `add(Object o)`; digunakan untuk menambahkan object ke dalam List
- `add(int index, Object o)`; menambahkan object ke dalam List di index tertentu

Class ArrayList adalah class yang mengimplementasikan interface List, bayangkan class ArrayList ini adalah Array yang dapat bertambah ukurannya. Vector adalah class yang juga mengimplementasikan List, Vector adalah pendahulu dari ArrayList, kalau tidak ada yang menghalangi anda menggunakan ArrayList, sebaiknya hindari Vector karena performance ArrayList lebih bagus dan classnya lebih modern.

LinkedList adalah implementasi dari List yang menambahkan method baru untuk menambahkan atau menghapus isi dari List dari depan atau dari belakang. Class ini cocok digunakan untuk membuat tumpukan (stack) atau antrian (queue).

Mari kita lihat contoh penggunaan List dan ArrayList dalam kode di bawah ini :

```
public class ListTest {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();
        list.add("a");
        list.add("b");
        list.add("c");
        list.add("a");
        list.add("z");
        System.out.println("isi dari list : ");
        for(int i=0; i< list.size();i++) {
            System.out.println("index ke-" + i + ":" + list.get(i) );
        }
    }
}
```

Kode di atas diawali dengan deklarasi variabel List yang menggunakan generics : List<String>, kode ini artinya kita akan membuat variabel dengan tipe List dan isinya hanya String saja, tipe data lain tidak bisa dimasukkan ke dalam List ini.

Hasil eksekusi kode di atas adalah seperti di bawah ini :

```
$ javac ListTest.java
$ java ListTest
isi dari list :
index ke-0:a
index ke-1:b
index ke-2:c
index ke-3:a
index ke-4:z
$
```

secara visual isi dari list di atas bisa digambarkan dalam diagram seperti di bawah ini :

index item	0	1	2	3	4
	"a"	"b"	"c"	"a"	"z"

terlihat bahwa index adalah bagian terpenting dari List, sedangkan item dalam List sendiri bisa dobel, tidak unik. Selain itu item di dalam List juga tidak terurut, terlihat dari huruf "a" yang diletakkan setelah huruf "c".

Set

Set adalah collection yang bersifat unik. Set digunakan kalau anda memerlukan collection yang isinya harus unik. Definisi unik diimplementasikan dengan mengoverride method equals dan hashCode dari class yang akan dimasukkan ke dalam Set. Semua class wrapper seperti String telah mengoverride method equals dan hashCode sehingga dua buah object String dianggap sama kalau string yang ada di dalamnya sama. Untuk class yang kita buat sendiri, seperti class

Customer, maka method equals dan hashCode harus dioverride dahulu sebelum dimasukkan ke dalam Set. Kalau method equals dan hashCode tidak dioverride maka dua buah object dianggap sama kalau keduanya merefer ke object yang sama di dalam memory, tidak seperti yang kita harapkan.

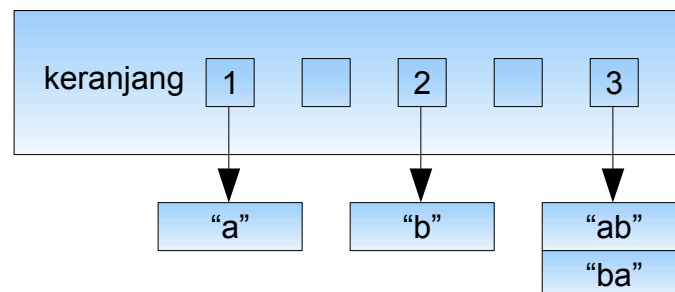
Class implementasi dari interface Set yang paling sering digunakan adalah HashSet. Nilai yang dikembalikan oleh hashCode sangat penting dalam class HashSet. Nilai hashCode digunakan untuk menentukan di mana object tersebut diletakkan dalam “keranjang” HashSet. Ketika method add di panggil dengan parameter object, HashSet akan memanggil method hashCode dari object tersebut untuk menentukan di keranjang mana object akan diletakkan. Setelah ketemu nomer keranjangnya, dicek apakah keranjang tersebut kosong. Kalau kosong maka object langsung diletakkan di keranjang tersebut. Kalau keranjang ada isinya, dilakukan pengtesan menggunakan method equals, kalau ada object di dalam keranjang dites menggunakan method equals menghasilkan true, maka object lama akan ditimpa dengan object baru yang dimasukkan ke dalam HashSet menggunakan method add tadi.

Kecepatan proses pencarian object di dalam HashSet tergantung dari bagaimana kita melakukan perhitungan hashCode. Kalau misalnya kita selalu return nilai 1 untuk semua object, maka semua object akan diletakkan dalam satu keranjang yang sama, proses pencarian menjadi linier ($O(n)$) karena object akan dibandingkan satu per satu menggunakan method equals. Kalau hashCode bersifat unik untuk setiap object, maka proses pencariannya sangat cepat yaitu $O(1)$, HashSet akan memanggil method hashCode untuk mendapatkan nilainya.

Kalau nilai hashCode tidak selalu unik, HashSet akan melihat di dalam keranjang tersebut adakah object di dalamnya, kalau ada satu buah maka object itu akan dikembalikan sebagai hasil pencarian, kalau kosong yang dikembalikan adalah null, kalau ada beberapa buah maka untuk setiap object di dalam keranjang dites menggunakan method equals, ketika method equals mengembalikan true maka object tersebut adalah hasil pencariannya.

Jika dua buah object dipanggil method equals menghasilkan nilai true tetapi hashCodenya berbeda (menyalahi aturan), maka oleh HashSet kedua object ini dianggap berbeda karena akan diletakkan di dalam keranjang yang berbeda, asal keranjangnya berbeda maka method equals akan diabaikan, tidak pernah dipanggil. Jadi harus ditaati aturan bahwa kalau dua buah object dioperasikan dengan method equals mengembalikan true, nilai hashCodenya harus sama.

Perhatikan visualisasi HashSet yang berisi String di bawah ini :



Algoritma sederhana untuk menghitung hashCode dari class String adalah menambahkan urutan dari hurufnya, misalnya string “a” nilai hashCodenya 1, “b” nilai hashCodenya 2, sedangkan string “ab” dan “ba” nilai hashCodenya sama yaitu 3. Bisa kita lihat juga string “ab” dan “ba” diletakkan dalam keranjang yang sama karena hashCodenya sama, tetapi tidak saling timpa, hal ini karena “ab”.equals(“ba”) bernilai false.

Tujuh paragraf menerangkan tentang Set kok rasanya masih kurang kalau kalau belum melihat kode bagaimana menggunakan Set ini. Kita akan menggunakan class Customer yang sudah dioverride method equals dan hashCodenya di bab sebelumnya, perhatikan juga bahwa equals dan hashCode dari class Customer di atas , kemudian buat class di bawah ini di folder yang sama dengan class Customer :

```
import java.util.Set;
import java.util.HashSet;
import java.util.Iterator;
```

```

public class CustomerSetTest{
    public static void main(String[] args) {
        Set<Customer> customers = new HashSet<Customer>();
        Customer id1 = new Customer();
        id1.setId(11);
        customers.add(id1);
        Customer id2 = new Customer();
        id2.setId(21);
        customers.add(id2);
        Customer c = new Customer();
        c.setId(11);
        customers.add(c); //mreplace id1 karena mempunyai id yang sama
        Iterator<Customer> i = customers.iterator();
        while(i.hasNext()){
            Customer current = i.next();
            System.out.println("keranjang no-" + current.hashCode()
                               + " idnya:" + current.getId());
        }
    }
}

```

Perhatikan juga kode di atas menggunakan Iterator untuk mengambil nilai dalam Set satu per satu, ada bentuk for each yang diperkenalkan sebagai bagian dari Java 5 Language enhancement, bentuk for each ini menyederhanakan kode untuk mengakses item di dalam Set. For each akan kita bahas di bab tentang java 5 language enhancement. Hasil eksekusi kode di atas adalah seperti di bawah ini :

```

$ javac Customer.java CustomerSetTest.java Customer.java
$ java CustomerSetTest
keranjang no-373 idnya:2
keranjang no-372 idnya:1
$

```

Anda mungkin juga terkejut, ternyata nomor keranjangnya pun tidak terurut dari kecil ke besar. HashSet adalah collection yang hanya menekankan pada sifat unik saja, sedangkan urutan dan keteraturan tidak dijamin sama sekali. Ada jenis collection lain yang bisa menjamin sifat unik sekaligus terurut, yaitu TreeSet, kita akan membahas tentang TreeSet nanti di bab setelah kita bahas konsep sorting.

Map

Map adalah bentuk struktur data yang berpasangan antara key-value. Key bersifat unik karena implementasinya menggunakan Set. Key dan value adalah object, bisa object apa saja, tipe data primitif tidak bisa dimasukkan ke dalam Map, juga tidak bisa dimasukkan ke semua jenis collection.

Implementasi Map memberikan kemampuan untuk mencari value berdasarkan key, setiap kali kita dihadapkan pada permasalahan bagaimana caranya melakukan pencarian berdasarkan value kita bisa menggunakan Map ini. Map bergantung pada method equals dan hashCode untuk menentukan apakah dua buah key sama atau tidak, jadi kalau kita ingin menggunakan object dari class yang kita buat sendiri sebagai key, maka kedua method itu harus dioverride untuk menentukan bagaimana dua buah object dianggap sama atau tidak.

HashMap adalah class yang mengimplementasikan interface Map, sifatnya adalah tidak terurut dan tidak teratur. Implementasi HashMap adalah yang paling efisien, implementasi Map lainnya yang membutuhkan keterurutan atau keteraturan akan menambahkan komputasi tambahan sehingga kurang efisien dibanding dengan HashMap. HashMap memperbolehkan satu buah key dengan nilai null dan banyak key bukan null yang valuenya adalah null.

Method penting yang dimiliki oleh Map antara lain :

- put(Object key, Object value); method yang digunakan untuk meletakkan pasangan key dan value

- `get(Object key)`; method digunakan untuk mendapatkan value dari key yang dimasukkan sebagai parameter
- `keySet()`; digunakan untuk mendapatkan Set yang berisi semua key, biasanya kita ingin mendapatkan Set dari key ini kalau ingin mengambil nilai dalam Map satu per satu

Mari kita lihat contoh kode penggunaan Map. Kita akan membuat sebuah map dimana key adalah kota dan value adalah kumpulan Customer yang tinggal di kota tersebut. Tipe data key adalah String dan tipe data valuenya adalah List of Customer:

```
import java.util.Map;
import java.util.HashMap;
import java.util.Set;
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
public class MapTest {
    public static void main(String[] args) {
        Map<String,List<Customer>> customerByCityMap =
            new HashMap<String,List<Customer>>();
        List<Customer> jakartaCust = new ArrayList<Customer>();
        Customer a = new Customer();
        a.setId(11);
        jakartaCust.add(a);
        Customer b = new Customer();
        b.setId(21);
        jakartaCust.add(b);
        customerByCityMap.put("jakarta",jakartaCust);
        List<Customer> surabayaCust = new ArrayList<Customer>();
        Customer c = new Customer();
        c.setId(31);
        surabayaCust.add(c);
        customerByCityMap.put("surabaya",surabayaCust);
        Set<String> keys = customerByCityMap.keySet();
        Iterator<String> iterator = keys.iterator();
        while(iterator.hasNext()) {
            String key = iterator.next();
            List<Customer> customers = customerByCityMap.get(key);
            for(int i = 0;i < customers.size(); i++) {
                Customer cust = customers.get(i);
                System.out.println("kota : " + key + ", Customer id : " +
                    cust.getId());
            }
        }
    }
}
```

hasil eksekusi kode di atas adalah :

```
$ javac MapTest.java Customer.java
$ java MapTest
kota : jakarta, Customer id : 1
kota : jakarta, Customer id : 2
kota : surabaya, Customer id : 3
$
```

Ada satu class lagi yang mengimplement Map interface sekaligus keynya diurutkan berdasarkan logic tertentu, yaitu class TreeSet. Kita akan membahas class TreeSet sekaligus HashSet setelah membahas bab Sorting di bawah ini.

Sorting

Sorting adalah cara untuk membuat sebuah collection terurut (sorted). Agar collection terurut kita perlu membuat item yang akan dimasukkan ke dalam Set mengimplementasikan interface Comparable. Interface ini digunakan untuk melakukan perbandingan antara dua buah objek, mana yang lebih besar atau lebih kecil atautkah sama persis.

Interface Comparable hanya mempunyai satu buah method, yaitu compareTo: method ini mempunyai sebuah parameter yang bertipe Object kalau tidak menggunakan generics, dan bertipe sama dengan class yang mengimplement interface Comparable kalau menggunakan generics. Method compareTo mengembalikan integer, nilai kembalinya positif berarti object yang dipanggil method comparenya lebih besar dari object yang dimasukkan ke parameter, nilai kembalinya negatif berarti sebaliknya dan kalau nilainya nol berarti kedua object sama besar.

Kita lihat contoh implementasi interface Comparable di class Customer. Pertama, kita akan mengimplementasikan interface Comparable tanpa menggunakan generics :

```
import java.lang.Comparable;
public class Customer implements Comparable{
    private Long id;
    public void setId(Long aId){
        this.id = aId;
    }
    public Long getId() {
        return this.id;
    }
    public int compareTo(Object o) {
        Customer c = (Customer) o;
        return getId().compareTo(c.getId());
    }
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Customer other = (Customer) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }
    public int hashCode() {
        int hash = 7;
        hash = 53 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}
```

Kalau menggunakan generics maka method compareTo tipe parameternya adalah Customer, bukan Object. Contohnya :

```
import java.lang.Comparable;
public class Customer implements Comparable<Customer>{
    private Long id;
    public void setId(Long aId){
        this.id = aId;
    }
    public Long getId() {
        return this.id;
    }
}
```

```

    }
    public int compareTo(Customer c) {
        return getId().compareTo(c.getId());
    }
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Customer other = (Customer) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }
    public int hashCode() {
        int hash = 7;
        hash = 53 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}

```

Perbandingan dalam method compareTo di atas akan menyebabkan Customer akan diurutkan berdasarkan idnya dari yang kecil ke besar. Kalau ingin diurutkan dari besar ke kecil, maka perbandingannya tinggal dibalik seperti di bawah ini :

```

public int compareTo(Customer c) {
    return c.getId().compareTo(getId());
}

```

Interface Comparable bisa digunakan kalau source code dari class Customer tersedia dan kita bisa melakukan modifikasi terhadap class Customer. Kalau misalnya class yang ingin diurutkan tidak tersedia source codenya, kita bisa membuat class baru kemudian mengextends class tersebut sekaligus mengimplementasikan interface Comparable, sehingga kita bisa membuat collection terurut dari class baru ini. Nah, bagaimana kalau classnya ditandai dengan final? Kita tidak mungkin mengextends class tersebut, ada cara kedua, yaitu dengan membuat class baru yang mengimplementasikan interface Comparator.

Interface Comparator mempunyai bentuk generics dan bentuk tanpa generics, seperti halnya interface Comparable, bentuk tanpa generics akan menyebabkan parameter bertipe Object sehingga perlu proses casting untuk mengubah menjadi Customer, sedangkan bentuk generics tipe data parameternya sudah bertipe Customer. mari kita lihat bagaimana cara membuat class yang mengimplementasikan interface Comparator dengan bentuk generics, bentuk non generics saya serahkan kepada anda untuk membuat sendiri:

```

import java.util.Comparator;
public class CustomerComparator implements Comparator<Customer>{
    public int compare(Customer c1, Customer c2) {
        return c1.getId().compareTo(c2.getId());
    }
}

```

Nah, setelah Customer mengimplementasikan interface Comparable, kita bisa menggunakan collection yang sifatnya terurut, misalnya TreeSet dan TreeMap. Kita ubah sedikit kode contoh penggunaan Set dengan mengganti HashSet menjadi TreeSet, seperti di bawah ini :

```

import java.util.Set;
import java.util.TreeSet;
import java.util.Iterator;
public class CustomerTreeSetTest{

```

```

public static void main(String[] args) {
    Set<Customer> customers = new TreeSet<Customer>();
    Customer id1 = new Customer();
    id1.setId(11);
    customers.add(id1);
    Customer id2 = new Customer();
    id2.setId(21);
    customers.add(id2);
    Customer c = new Customer();
    c.setId(11);
    customers.add(c); //mreplace id1 karena mempunyai id yang sama
    Iterator<Customer> i = customers.iterator();
    while(i.hasNext()){
        Customer current = i.next();
        System.out.println("keranjang no-" + current.hashCode()
            + " idnya:" + current.getId());
    }
}
}

```

kalau kita eksekusi kode di atas maka customer akan diurutkan berdasarkan idnya, seperti di bawah ini :

```

$ javac CustomerTreeSetTest.java Customer.java
$ java CustomerTreeSetTest
keranjang no-372 idnya:1
keranjang no-373 idnya:2
$

```

TreeSet dan TreeMap menyaratkan item yang dimasukkan mengimplementasikan Comparable, atau kalau itemnya tidak bisa mengimplementasikan Comparable, kita harus menyediakan class yang mengimplementasikan Comparator dan memasukkan object dari class Comparator tersebut di constructor dari TreeSet atau TreeMap, seperti cuplikan kode di bawah ini :

```

Set<Customer> customers = new TreeSet<Customer>(new CustomerComparator());
Map<Customer> customerMap = new TreeMap<Customer>(new CustomerComparator());

```

Kedua collection ini akan menyusun item dalam keadaan terurut menggunakan struktur data tree, kalau item yang dimasukkan lebih kecil dari root maka diletakkan di tree sebelah kiri, kalau lebih besar diletakkan di sebelah kanan. Algoritma seperti ini menyebabkan proses penambahan item baru ke dalam collection menjadi sedikit lebih lambat daripada HashSet. Tetapi ketika isi dari TreeSet diambil, keadaanya sudah dalam kondisi terurut, sehingga tidak diperlukan proses pengurutan lagi.

Jenis collection yang bersifat terurut hanya tersedia untuk Set dan Map, sedangkan untuk List tidak ada. Nah kalau kita ingin membuat sebuah List terurut kita bisa menggunakan class Collections yang di dalamnya terdapat method yang bisa mengurutkan List. Kita akan membahas tentang class Collections di bab selanjutnya.

Class Collections dan Class Arrays

Class Collections berisi method-method utility yang digunakan antara lain untuk: melakukan sorting, melakukan pencarian, mengacak isi List, membalik susunan List, mengcopy potongan List, mengisi list dengan pola tertentu dan seterusnya. Kita akan membahas satu per satu method di atas.

Method sort digunakan untuk mengurutkan List berdasarkan logika yang ada dalam method Comparable. Item di dalam List harus mengimplementasikan interface Comparable, kalau tidak maka gunakan method sort dengan dua parameter: List dan Object dari class yang mengimplementasikan interface Comparator, contoh potongan kodenya seperti di bawah ini :

```

import java.util.List;
import java.util.ArrayList;
import java.util.Collections;

```

```

public class SortListTest {
    public static void main(String[] args){
        List<Customer> customers = new ArrayList<Customer>();
        Customer cust1 = new Customer();
        cust1.setId(101);
        customers.add(cust1);
        Customer cust2 = new Customer();
        cust2.setId(21);
        customers.add(cust2);
        Customer cust3 = new Customer();
        cust3.setId(51);
        customers.add(cust3);
        System.out.println("isi dari list sebelum disorting: ");
        for(int i=0; i< customers.size();i++) {
            System.out.println("index ke-" + i + ":" + customers.get(i) );
        }
        Collections.sort(customers);
        System.out.println("isi dari list setelah disorting: ");
        for(int i=0; i< customers.size();i++) {
            System.out.println("index ke-" + i + ":" + customers.get(i) );
        }
    }
}

```

Kode di atas memperlihatkan bahwa kita membuat List of Customer dengan id yang tidak terurut, karena Customer mengimplementasikan interface Comparable maka isi dari List of Customer tersebut bisa disorting menggunakan method sort dari class Collections. Kalau kode di atas dcompile dan dieksekusi hasilnya adalah :

```

$ javac SortListTest.java Customer.java
$ java SortListTest
isi dari list sebelum disorting:
index ke-0:10
index ke-1:2
index ke-2:5
isi dari list setelah disorting:
index ke-0:2
index ke-1:5
index ke-2:10
$

```

Kalau misalnya Customer tidak mengimplementasikan interface Comparable, maka kita bisa membuat class CustomerComparator yang mengimplementasikan interface Comparator sebagai pembanding antara dua buah Customer mana yang lebih besar, kode di atas bisa diedit sebagian dengan memanggil method sort yang mempunyai dua buah parameter: List of Customer dan instance dari CustomerComparator.

```

Collections.sort(customers, new CustomerComparator());

```

Method binarySearch digunakan untuk melakukan pencarian terhadap isi List dengan lebih cepat dibanding dengan method indexOf dari interface List. Method binarySearch baru bisa dipanggil setelah List disorting dahulu. Method binarySearch memerlukan dua parameter kalau object dalam List mengimplementasikan interface Comparable: Listnya dan item yang ingin dicari. Contoh kode penggunaan method binarySearch adalah :

```

import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
public class BinarySearchTest {
    public static void main(String[] args){
        List<Customer> customers = new ArrayList<Customer>();

```

```

Customer cust1 = new Customer();
cust1.setId(101);
customers.add(cust1);
Customer cust2 = new Customer();
cust2.setId(21);
customers.add(cust2);
Customer cust3 = new Customer();
cust3.setId(51);
customers.add(cust3);
Collections.sort(customers);
int index = Collections.binarySearch(customers, cust3);
System.out.println("Customer dengan id:" + cust3.getId() +
    " ditemukan di index : " + index);
}
}

```

hasil eksekusi kode di atas adalah sebagai berikut :

```

$ javac BinarySearchTest.java Customer.java
$ java BinarySearchTest
Customer dengan id:5 ditemukan di index : 1
$

```

Kalau item tidak mengimplementasikan interface Comparable maka method binarySearch memerlukan tiga parameter: List, item yang dicari dan object dari class Comparator, seperti dalam kode berikut ini :

```

int index = Collections.binarySearch(customers, cust3,
    new CustomerComparator());

```

Untuk mengacak isi dari sebuah list, ada method shuffle. Method shuffle ada dua overload, yang pertama hanya mempunyai satu parameter berupa List, yang kedua mempunyai dua buah parameter: List dan object dari class Random. Object Random digunakan untuk menentukan jenis randomisasi yang ingin digunakan untuk mengacak isi dari List.

Method reverse digunakan untuk membalik isi dari List, dimana yang depan menjadi di belakang dan sebaliknya. Penggunaan method ini perlu hati-hati karena kecepatannya linier, semakin banyak isi dari List waktu eksekusinya membesar secara linier.

Method copy digunakan untuk mengcopy isi dari satu List ke List lain, method ini cukup praktis dibanding harus melakukan copy manual yang memerlukan proses iterasi, jadi hemat kira-kira empat sampai lima baris kode.

Method fill digunakan untuk mengganti isi dari sebuah list dengan sebuah object yang sama. Parameter method fill ada dua : List dan object item yang akan digunakan untuk menimpa semua item yang ada dalam List.

Method min dan max digunakan untuk mendapatkan nilai maximum dan minimum dari sebuah List. Method ini menyaratkan semua item di dalam List harus mengimplementasikan interface Comparable. Seperti biasa, kalau item di dalam List tidak mengimplementasikan interface Comparable maka kita harus menambahkan instance dari class yang mengimplementasikan interface Comparator ke dalam method min dan max.

Ada satu jenis method yang cukup berguna, yaitu unmodifiable, jenis method ini ada beberapa, antara lain: unmodifiableList, unmodifiableMap, unmodifiableSet dan unmodifiableCollection. Method jenis ini bagus sekali kalau digunakan untuk melindungi collection agar tidak bisa dimodifikasi, artinya collectionnya akan bersifat read only. Misalnya di dalam suatu class kita punya property yang bertipe collection, nah kita tidak ingin collection ini dimodifikasi di luar dari class, sehingga setiap kali method ini digunakan sebagai return type, yang kita return adalah versi read only dari collection tersebut. Sebagai contoh, kita modifikasi sedikit class Customer agar mempunyai property emails yang bertipe List of String, seperti di bawah ini :

```

import java.lang.Comparable;
import java.util.List;
import java.util.ArrayList;

```

```

import java.util.Collections;
public class Customer implements Comparable<Customer>{
    private Long id;
    private List<String> emails;
    public void setId(Long aId){
        this.id = aId;
    }
    public Long getId() {
        return this.id;
    }
    public void setEmails(List<String> aEmails) {
        this.emails = aEmails;
    }
    public List<String> getEmails() {
        return Collections.unmodifiableList(emails);
    }
    public void addEmail(String email){
        if(this.emails == null) {
            this.emails = new ArrayList<String>();
        }
        emails.add(email);
    }
    public int compareTo(Customer c) {
        return getId().compareTo(c.getId());
    }
    public boolean equals(Object obj) {
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Customer other = (Customer) obj;
        if (this.id != other.id && (this.id == null || !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }
    public int hashCode() {
        int hash = 7;
        hash = 53 * hash + (this.id != null ? this.id.hashCode() : 0);
        return hash;
    }
}

```

Dari kode di atas, terlihat bahwa pada saat method getEmails dipanggil, yang dikembalikan adalah versi read only dari emails. Dengan begitu, emails hanya bisa dimodifikasi dari dalam class Customer dengan memanggil method addEmail. Praktek seperti ini sangat disarankan terutama ketika kita membuat kode yang banyak digunakan orang lain dan tidak ingin collection yang dimanage oleh library tersebut diedit sembarangan. Pengaturan seperti ini biasanya cukup ditekankan kalau aplikasi dibuat oleh banyak sekali orang, sehingga bisa meminimalisasi kesalahan karena kurangnya informasi atau ketidaktahuan anggota tim lain. Dengan membuat collection yang ditandai dengan read only seperti di atas, kita menyampaikan informasi kepada orang lain yang mungkin menggunakan class Customer ini bahwa List emails tidak bisa diedit dari luar class Customer.

Class Arrays berisi method-method utility untuk bekerja dengan array, sama seperti class Collections yang berisi method-method utility untuk bekerja dengan collection. Method yang ada dalam class Arrays kurang lebih sama dengan method yang ada dalam class Collections,

seperti: proses sorting, pencarian, mengisi array dengan nilai tertentu, membuat copy dari array, memeriksa apakah dua buah array sama persis, mendapatkan nilai hashCode dari array, mendapatkan toString dari array dan merubah array menjadi List dengan cepat.

Method asList digunakan untuk mengubah array menjadi List dengan cepat. Tanpa bantuan method ini merubah array menjadi List memerlukan langkah-langkah : membuat instance dari List, kemudian mengkopy isi array ke dalam List dengan iterasi. Langkah-langkah di atas kira-kira memerlukan lima baris kode, tetapi dengan menggunakan method asList cukup satu baris kode saja.

Tanpa menggunakan method asList mengkopy array ke dalam List kodenya seperti di bawah ini :

```
import java.util.List;
import java.util.ArrayList;
public class CopyArrayManualTest {
    public static void main(String[] args) {
        String[] names = {"me","you","they","us"};
        List<String> nameList = new ArrayList<String>();
        for(int i = 0; i < names.length; i++) {
            nameList.add(names[i]);
            System.out.println("name:" + names[i]);
        }
    }
}
```

Dengan menggunakan class Arrays kode di atas menjadi sangat pendek, seperti di bawah ini :

```
import java.util.List;
import java.util.Arrays;
public class CopyArrayTest {
    public static void main(String[] args) {
        String[] names = {"me","you","they","us"};
        List<String> nameList = Arrays.asList(names);
    }
}
```

Method equals digunakan untuk menentukan apakah dua buah array isinya sama persis atau tidak, method ini sangat praktis digunakan daripada melakukan iterasi satu per satu isi dari array dan membandingkannya dengan isi array lain.

Method toString bisa digunakan untuk membuat sebuah string representasi dari array, string ini sudah diformat sekian rupa sehingga kalau diprint di dalam console akan terlihat bagus. Tanpa menggunakan method toString ini biasanya kita melakukan iterasi untuk mencetak satu persatu isi dari array, dengan adanya method ini tidak perlu lagi melakukan iterasi hanya untuk menampilkan array ke dalam console.

Method copyOf digunakan untuk mengkopy sebagian, keseluruhan atau malah membuat array baru dengan isi yang sama tetapi ukurannya lebih panjang dari sebuah array. Sedangkan method copyOfRange digunakan untuk mengkopy sebagian dari array dengan mendefinisikan awal dari index dan akhir dari index yang ingin dicopy. Perhatikan contoh berikut ini :

```
import java.util.Arrays;
public class CopyPartialArrayTest {
    public static void main(String[] args) {
        String[] names = {"me","you","they","us"};
        //membuat array baru dari sebagian isi array names
        String[] n = Arrays.copyOf(names, 2);
        System.out.println("setelah dipotong : " + Arrays.toString(n));
        //membuat array baru dari semua isi array names
        //sekaligus panjangnya bertambah
        n = Arrays.copyOf(names, 7);
        System.out.println("setelah ditambah panjang : " + Arrays.toString(n));
        //copy sebagian isi array names dari index 1 sampai index 3
    }
}
```



```

n = Arrays.copyOfRange(names, 1, 3);
System.out.println("setelah dipotong : " + Arrays.toString(n));
//copy sebagian isinya dan tambahkan default value untuk sisanya
n = Arrays.copyOfRange(names, 2, 10);
System.out.println("setelah dipotong dan bertambah panjang: "
    + Arrays.toString(n));
    }
}

```

Coba tebak hasil eksekusi kode di atas! Apakah tebakannya sama dengan hasil eksekusinya seperti di bawah ini ?

```

$ javac CopyPartialArrayTest.java
$ java CopyPartialArrayTest
setelah dipotong : [me, you]
setelah ditambah panjang : [me, you, they, us, null, null, null]
setelah dipotong : [you, they]
setelah dipotong dan bertambah panjang: [they, us, null, null, null, null, null,
null]
$

```

Sisanya, method-method seperti : sort dan binarySearch sama persis penggunaannya dengan method yang ada dalam class Collections.

Class Penting

JDK menyertakan banyak sekali class-class yang berguna untuk memudahkan developer membuat aplikasi. Beberapa class sangat penting fungsinya sehingga perlu diperhatikan bagaimana menggunakannya dengan benar.

String, StringBuilder, StringBuffer

String adalah class yang bersifat immutable, artinya sekali diset nilainya maka tidak bisa lagi diubah. String yang bersifat immutable juga berarti bahwa kalau kita melakukan modifikasi terhadap object String, pada dasarnya hasil modifikasi ini adalah object baru. Hal ini yang membuat String ini menjadi tidak efisien digunakan pada saat diperlukan manipulasi String dalam skala yang cukup besar. Mari kita lihat contoh kode yang menunjukkan class String immutable :

```

public class StringImmutableTest {
    public static void main(String[] args){
        String s = "ini string immutable ";
        System.out.println("sebelum operasi concat nilai s : " + s);
        //append tidak merubah variabel s, tetapi dibuat object baru
        //dan object baru ini direturn sedangkan variabel s nilainya tetap
        s.concat("concat");
        System.out.println("setelah operasi concat nilai s : " + s);
        String concat = s + s.concat("concat") + " object baru";
    }
}

```

Hasil eksekusi kode di atas adalah :

```

$ javac StringImmutableTest.java
$ java StringImmutableTest
sebelum operasi concat nilai s : ini string immutable
setelah operasi concat nilai s : ini string immutable
$

```

Sebelum dan sesudah operasi concat nilai variabel s tetap, hal ini terjadi karena proses concat dari class String akan membuat object baru kemudian dilakukan concat antara nilai yang sekarang dipunyai oleh variabel s dengan string yang ada dalam parameter, dan object baru ini

direturn. Proses concat tidak merubah nilai yang dipunyai oleh variabel s, karena class String immutable, sekali nilainya diset tidak bisa diubah lagi.

Perhatikan juga proses string concatenation dengan menggunakan operator + di atas, operasi ini jika dilakukan menggunakan String akan menghasilkan banyak sekali object baru, yang pertama adalah object hasil operasi append dari variabel s, kemudian object penggabungan variabel s dengan s.concat dan terakhir hasil penggabungan ketiga string tersebut.

Kalau proses concatenation atau pengolahan String secara umum dalam skala yang sangat besar, maka akan menyebabkan penggunaan memory yang tidak efisien. Kalau kita dihadapkan dengan masalah tersebut sebaiknya jangan menggunakan class String, selalu gunakan class StringBuilder atau StringBuffer.

Class StringBuffer dibuat untuk tujuan efisiensi, karena StringBuffer bukan class immutable seperti class String. StringBuffer mempunyai kemampuan untuk digunakan dalam lingkungan multi threading karena sifatnya yang thread safe, tetapi feature ini menyebabkan class StringBuffer tidak efisien dari sisi komputasi CPU, karena thread safe memerlukan pengecekan apakah yang melakukan operasi terhadap class StringBuffer ini adalah thread yang sama, kalau threadnya berbeda maka prosesnya disuruh mengantri dan hanya satu thread dalam satu waktu yang bisa melakukan manipulasi class StringBuffer ini.

Class StringBuilder mempunyai tujuan yang sama persis dengan class StringBuffer, tetapi StringBuilder tidak mempunyai fasilitas thread safe, sehingga performanya lebih cepat. Kalau kita tidak memerlukan fasilitas thread safe maka StringBuilder adalah class yang tepat digunakan untuk manipulasi string.

Mari kita rubah kode sebelumnya dengan mengganti class String menjadi StringBuilder dan kita lihat perbedaanya :

```
public class StringBuilderTest {
    public static void main(String[] args){
        StringBuilder s = new StringBuilder("ini StringBuilder tidak immutable ");
        System.out.println("sebelum operasi concat nilai s : " + s);
        //append merubah variabel s,
        //setelah proses append nilai variabel s akan berubah
        s.append("concat");
        System.out.println("setelah operasi concat nilai s : " + s);
        s.append(s.append("concat")).append(" object baru");
    }
}
```

Hasil eksekusi kode di atas adalah :

```
$ javac StringBuilderTest.java
$ java StringBuilderTest
sebelum operasi concat nilai s : ini StringBuilder tidak immutable
setelah operasi concat nilai s : ini StringBuilder tidak immutable concat
$
```

Mulai dari JDK 6, compiler java (javac) akan merubah operasi penggabungan String dengan menggunakan operasi append dari class StringBuilder secara otomatis. Misalnya kode di bawah ini :

```
String s = "ini string immutable ";
String concat = s + s.concat("concat") + " object baru";
```

kode di atas kalau di compile menggunakan JDK 6 akan diubah kodenya menjadi seperti di bawah ini :

```
String s = "ini string immutable ";
StringBuilder builder = new StringBuilder();
String concat = builder.append(s).append(s.concat("concat"))
    .append(" object baru").toString();
```

Jadi kalau sudah menggunakan JDK 6 tidak perlu khawatir melakukan penggabungan string menggunakan class String dan operator +, tetapi memang sebaiknya selalu gunakan

StringBuilder.append untuk menekankan disiplin penggabungan string yang baik.

Date, Calendar, DateFormat

Bekerja dengan tanggal bukan pekerjaan yang mudah, terkadang kita perlu melakukan perhitungan berdasarkan tanggal yang cukup rumit, misalnya mendapatkan hari terakhir dari bulan tertentu atau mendapatkan selisih hari antara dua tanggal, atau bahkan bekerja dengan calendar selain Masehi, misalnya penanggalan Budha. Java menyediakan class-class yang bisa digunakan untuk tujuan tersebut, sehingga tidak perlu membuat kode untuk melakukan hal tersebut secara manual.

Class Date adalah representasi dari waktu di Java, class ini terdapat dua jenis: java.util.Date dan java.sql.Date. Class java.util.Date adalah class yang sering digunakan dalam perhitungan tanggal, sedangkan java.sql.Date adalah representasi data tanggal di java yang setara dengan tipe data date di dalam database. Kalau tidak bekerja dengan database, sebaiknya selalu gunakan java.util.Date, sedangkan java.sql.Date hanya digunakan ketika kita bekerja dengan database. Istilah class Date di dalam bab ini merujuk ke java.util.Date.

Waktu dalam perhitungan komputer direpresentasikan sebagai bilangan Long yang merupakan durasi antara tanggal 1 januari 1970 00:00:000 hingga waktu sekarang dalam mili detik. Pencatatan tanggal ini disimpan dalam BIOS dari PC. Dari kode java kita bisa mendapatkan waktu sekarang menggunakan kode :

```
Long waktuSekarang = System.currentTimeMillis();
```

nilai waktuSekarang adalah jarak antara tanggal 1 januari 1970 00:00:000 hingga kode ini dieksekusi dalam mili detik. Kalau ingin mendapatkan representasi yang lebih baik, kita bisa menggunakan class Date untuk mendapatkan waktu sekarang, seperti di bawah ini :

```
Date d = new Date();
int tahun = d.getYear();
int bulan = d.getMonth();
int tanggal = d.getDate();
Long waktuSekarang = d.getTime();
```

variabel d adalah instance dari class Date yang merepresentasikan waktu sekarang dan mempunyai method-method untuk mendapatkan bagian dari waktu seperti tahun, bulan, tanggal dan seterusnya. Class Date juga menyediakan constructor dengan parameter Long, sehingga nilai dari System.currentTimeMillis() bisa dimasukkan sebagai parameternya, seperti di bawah ini :

```
Long waktuSekarang = System.currentTimeMillis();
Date d = new Date(waktuSekarang);
```

Semenjak java 1.1, di dalam JDK ditambahkan class Calendar untuk memanipulasi nilai dari class Date ini, oleh karena itu banyak sekali method di dalam class Date yang sifatnya deprecated, sudah tidak didukung dan sebaiknya tidak digunakan lagi. Misalnya method-method seperti getDate, getMonth, getYear dan seterusnya kalau digunakan dan kodenya dicompile maka pada waktu kompilasi akan muncul warning bahwa method-method tersebut sudah deprecated. Contohnya seperti ini :

```
import java.util.Date;
public class DateDeprecatedTest {
    public static void main(String[] args) {
        Date d = new Date();
        int tahun = d.getYear();
        int bulan = d.getMonth();
        int tanggal = d.getDate();
        Long waktuSekarang = d.getTime();
    }
}
```

Kalau kita compile kode di atas, warning menerangkan bahwa ada kode yang deprecated akan terlihat seperti di bawah ini:

```
$ javac DateDeprecatedTest.java
```

```
Note: DateDeprecatedTest.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
$
```

Class Date digunakan sebagai object untuk menyimpan waktu saja, hal ini terutama karena class Date mengimplementasikan interface Serializable, plus semua fungsi untuk perhitungan sudah ditandai deprecated. Sedangkan class Calendar digunakan layaknya kalkulator untuk proses perhitungan tanggal pada class Date, misalnya kita ingin menambahkan 1 hari, menambah 1 menit dan seterusnya. Calendar juga menyediakan sistem penanggalan Masehi (GregorianCalendar), penanggalan Budha dengan bahasa Thailand dan penanggalan Jepang.

Misalnya kita ingin membuat class Date dengan tanggal : 1 januari 2000, kodenya seperti di bawah ini :

```
Calendar calendar = Calendar.getInstance(); //secara default penanggalan yang
digunakan adalah Masehi (GregorianCalendar);
calendar.set(2000,Calendar.JANUARY,1,0,0,0);
Date d = calendar.getTime();
```

Kode di atas menggunakan Calendar.getInstance() untuk mendapatkan tanggal sekarang, class Calendar yang dikembalikan adalah GregorianCalendar yang merupakan implementasi penanggalan Masehi. Method set mempunyai beberapa bentuk (overloading) untuk mengeset nilai ke dalam calendar, bisa diset satu per satu atau diset hanya bulan-tanggal-tahun atau diset semuanya secara bersamaan seperti dalam contoh di atas.

Kita juga bisa langsung membuat instance dari GregorianCalendar menggunakan constructor seperti di bawah ini

```
Calendar calendar = new GregorianCalendar(2000,Calendar.JANUARY,1,0,0,0);
Date d = calendar.getTime();
```

Class Calendar mempunyai method setTime untuk memasukkan object class Date ke dalam calendar, kemudian bisa dilakukan perhitungan dan setelah selesai object date bisa dikeluarkan dari calendar dengan menggunakan method getTime.

Untuk mendapatkan nilai individual dari waktu, class Calendar menyediakan method get dengan sebuah parameter yang menerangkan bagian apa dari calendar yang ingin diambil. Contohnya seperti di bawah ini :

```
Calendar calendar = Calendar.getInstance();
int tahun = calendar.get(Calendar.YEAR);
int bulan = calendar.get(Calendar.MONTH);
int tanggal = calendar.get(Calendar.DATE);
int hari = calendar.get(Calendar.DAY_OF_WEEK);
```

Perlu diperhatikan bahwa nilai bulan yang diambil dari calendar dimulai dari 0 hingga 11!!, berbeda dengan tanggal yang tidak ada nilai 0, perilaku ini kerap menimbulkan bug di dalam aplikasi karena anggapan umum bulan dimulai dari 1 hingga 12.

Method add dari class Calendar digunakan untuk menambahkan satuan waktu ke dalam calendar. Parameternya ada dua: satuan waktu dan berapa banyak nilai yang akan ditambahkan. Perhatikan contoh di bawah ini :

```
Calendar calendar = Calendar.getInstance();
calendar.add(Calendar.DATE, 1); //menambah satu hari
calendar.add(Calendar.HOUR, 1); //menambah satu jam
calendar.add(Calendar.YEAR, -1); //mengurangi satu tahun
```

Proses penambahan atau pengurangan menggunakan add akan merubah satuan waktu lebih besar. Misalnya sekarang tanggal 31 januari dan ditambah satu hari, maka hasilnya adalah 1 Februari. Kalau kita tidak ingin satuan waktu lebih besar ikut berubah karena proses penambahan/pengurangan gunakan method roll, seperti contoh di bawah ini:

```
Calendar calendar = new GregorianCalendar(2000,Calendar.JANUARY,31,0,0,0);
calendar.roll(Calendar.DATE, 1); //hasilnya 1 januari 2000 bukan 1 februari 2000
```

Contoh di atas, kita menambah satu hari ke tanggal 31 Januari 2000, karena method yang

digunakan adalah roll dan 31 adalah tanggal terbesar untuk bulan januari maka tanggalnya akan dimulai lagi menjadi 1 tetapi bulannya tidak berubah, tetap januari.

Class Calendar juga mempunyai beberapa method lain yang cukup berguna, misalnya method `getActualMaximum` yang bisa digunakan untuk mengambil nilai maximum satuan waktu, kalau parameter dari `getActualMaximum` ini adalah `Calendar.DAY_OF_MONTH` maka akan direturn nilai tanggal terbesar dari bulan tersebut. Kalau Calendarnya mempunyai tanggal 1 Februari 2000 maka hasil dari `getActualMaximum(Calendar.DAY_OF_MONTH)` adalah 29. Contohnya seperti di bawah ini :

```
public class CalendarTest {
    public static void main(String[] args) {
        Calendar calendar = new GregorianCalendar(2000,Calendar.FEBRUARY,1);
        int maxFeb = calendar.getActualMaximum(Calendar.DAY_OF_MONTH);
        System.out.println(maxFeb);
    }
}
```

`DateFormat` adalah class yang digunakan untuk memformat tanggal menjadi string. Class turunan dari `DateFormat` yang sering digunakan adalah `SimpleDateFormat`. Class `SimpleDateFormat` bisa digunakan untuk mengubah string menjadi tanggal dan sebaliknya dari tanggal menjadi String. Class ini sebaiknya dibuat setiap kali akan digunakan, hal ini dikarenakan class `SimpleDateFormat` tidak thread-safe, artinya kalau diinstansiasi sekali dan digunakan dalam lingkungan multithreading maka hasilnya tidak konsisten.

Format tanggal di Java menggunakan `DateFormat` ada aturannya, ada beberapa huruf yang digunakan sebagai simbol satuan waktu. Daftar huruf yang digunakan sebagai simbol satuan waktu adalah :

Huru f	Satuan waktu	Tipe data	Contoh
G	Era	Teks	AD, BC
y	Tahun	Tahun	1996, 96
M	Bulan	Bulan	11,Jul,July
w	Minggu dalam satu tahun	Angka	27
W	Minggu dalam satu bulan	Angka	3
D	Hari dalam satu tahun	Angka	259
d	Hari dalam satu bulan	Angka	23
F	Minggu dalam satu bulan	Angka	4
E	Hari dalam satu minggu	Teks	Tuesday, Tue
a	Penanda AM/PM	Teks	AM, PM
H	Jam dalam satuan 24 jam (0-23)	Angka	10
k	Jam dalam satuan 24 jam (1-24)	Angka	24
K	Jam dalam satuan 12 jam (0-11)	Angka	11
h	Jam dalam satuan 12 jam (1-12)	Angka	12
m	Menit	Angka	30
s	Detik dalam satu menit	Angka	36
S	Milidetik	Angka	178
z	Daerah waktu (timezone)	Tanda daerah waktu	PST, GMT+7, Pacific Standard Time
Z	Daerah waktu (timezone)	Daerah waktu berdasar RFC-822	+0800, -0100

Berikut ini adalah contoh format tanggal dan hasilnya kalau dieksekusi :

Format tanggal dan waktu	Hasil
"dd/MM/yyyy"	21/03/2004
"dd/MM/yyyy HH:mm:ss"	21/03/2004 23:01:19
"dd.MM.yyyy HH:mm:ss z"	21.03.2004 23:01:19 SGT
"dd MMM yyyy"	01 Jan 09
"dd MMMM yyyy"	24 February 2010
"dd MMM yyyy hh:mm:ss a z"	23 Mar 2010 11:09:27 PM SGT

Contoh kodenya bisa kita lihat di bawah ini :

```
import java.util.Date;
import java.text.SimpleDateFormat;
import java.text.ParseException;
public class SimpleDateFormatTest {
    public static void main(String[] args) {
        Date now = new Date();
        System.out.println(new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(now));
        String tanggal = "21/12/1990 08:09:10";
        try {
            Date date = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss").parse(tanggal);
            System.out.println(new SimpleDateFormat("dd MMMM yyyy").format(date));
        } catch (ParseException ex) {
            ex.printStackTrace();
        }
    }
}
```

Hasil eksekusinya adalah seperti berikut :

```
$ javac SimpleDateFormatTest.java
$ java SimpleDateFormatTest
16/02/2011 01:32:26
21 December 1990
$
```

Manipulasi tanggal menggunakan Calendar dan memformat tanggal menggunakan SimpleDateFormat terkadang memerlukan kode yang cukup panjang dan tidak praktis. Masalah kurang handalnya library pengolahan tanggal yang ada dalam JDK sudah banyak diketahui, di JDK 7 nanti akan ada library tanggal yang baru di JSR-310. library ini diturunkan dari library tanggal yang sangat populer dan powerfull, yaitu Joda Time. Di bab berikutnya kita akan membandingkan bagaimana mengolah data menggunakan Calendar vs menggunakan Joda Time.

Joda Time

Joda Time adalah library pengolahan tanggal yang ditulis oleh Stephen Colebourne karena sudah cukup frustasi dengan library pengolahan tanggal di dalam JDK yang cukup merepotkan untuk perhitungan tanggal yang rumit. API untuk pengolahan tanggal bisa dibilang sebagai API yang paling buruk dalam JDK saat ini. API tanggal mulai ada semenjak JDK versi 1.0, James Gosling mencontoh implementasi tanggal dari C, dimana implementasinya masih sederhana dan memiliki banyak kelemahan secara design. Misalnya bulan di dalam class Date dimulai dari 0 hingga 11, jadi kalau kita panggil method getMonth dari class Date untuk bulan Desember maka nilainya adalah 11, bukan 12. Joda Time dibuat untuk memperbaiki hal ini.

Selain itu Joda Time juga mengenalkan beberapa konsep baru, seperti periode, interval dan implementasi daerah waktu (timezone) yang lebih baik. Feature chronology juga sangat membantu kalau kita ingin bekerja dengan sistem penanggalan yang tidak umum digunakan, seperti penanggalan islam, budha, julian dan sebagainya.

Joda Time bisa digunakan bersamaan dengan class Date dan Calendar dengan baik, kita bisa melakukan konversi dari class-class Joda Time menjadi kedua class tersebut dengan mudah. Jadi Joda Time digunakan sebagai library perhitungan tanggal yang bisa melakukan hal-hal rumit

dengan mudah, kemudian setelah selesai proses perhitungannya kita bisa mendapatkan hasilnya dalam bentuk class Date. Feature backward compatibility inilah salah satu daya tarik dari Joda Time.

Sebelum kita mulai belajar tentang konsep dalam Joda Time, kita perlu mendownload library Joda Time dari alamat di bawah ini :

<http://sourceforge.net/projects/joda-time/files/joda-time/>

pada saat buku ini ditulis versi yang terbaru adalah 1.6.2, kita hanya perlu satu buah jar saja yaitu joda-time.1.6.2.jar.

Joda Time memperkenalkan enam konsep waktu, yaitu :

- Instant : sebuah instant adalah nilai dalam mili detik dari 1 januari 1970 00:00:00 hingga sekarang.
- Parsial : representasi sebagian dari waktu, misalnya tanggal saja atau waktu saja.
- Interval : interval waktu dari awal hingga akhir. Dalam interval terdapat dua buah instant yaitu awal dan akhir.
- Durasi : durasi waktu dalam mili detik, misalnya 1000 mili detik.
- Periode : periode waktu yang direpresentasikan dalam satuan-satuan waktu, misalnya 1 hari 2 jam 3 menit dan 30 detik.
- Chronology : sistem penanggalan yang bisa digunakan sebagai basis perhitungan waktu.

Class-class implementasi dari setiap konsep di atas adalah sebagai berikut :

Konsep	Class Immutable	Class Mutable
Instant	Instant, DateTime, DateMidnight	MutableDateTime
Parsial	LocalDate, LocalTime, LocalDateTime, Partial	
Interval	Interval	MutableInterval
Periode	Period, Minutes, Hours, Days, Weeks, Months, Years	MutablePeriod
Durasi	Duration	
Chronology	GregorianCalendar, ISOChronology (default), BuddhistChronology, JulianChronology, EthiopicChronology, CopticChronology, GJChronology	

Mari kita bahas satu per satu konsep di atas dengan disertai contoh kode untuk mempermudah pemahaman terhadap konsep-konsep dalam Joda Time.

Instant adalah konsep paling sederhana, konsep ini sama dengan class Date, yaitu representasi jumlah mili detik dari tanggal 1 januari 1970 00:00:00 hingga sekarang. Joda Time mempunyai beberapa class yang digunakan untuk merepresentasikan instant ini, dua class yang sering digunakan adalah class Instant dan class DateTime. Class Instant berisi satu buah property dalam mili detik, perhatikan contoh berikut ini untuk memahami class Instant :

```
import org.joda.time.Instant;
public class InstantTest {
    public static void main(String[] args) {
        Instant instant = new Instant(1000); // 1 detik setelah 1970
        //ingat bahwa instant mutable sehingga perlu diset lagi setelah
        //diubah nilainya
        instant = instant.plus(100); //ditambah 100 milidetik
        instant = instant.plus(60000); //ditambah satu menit
        System.out.println(instant);
    }
}
```

```
}
```

Untuk mengcompile kode di atas, kita perlu menambahkan jar Joda Time ke dalam classpath. Letakkan joda-time-1.6.2.jar ke dalam folder yang sama dengan file java dari class di atas, kemudian jalankan command di bawah ini untuk mengcompile dan menjalankan kode di atas.

Jalankan perintah di bawah ini kalau anda bekerja dengan Linux, Unix dan Mac :

```
$ javac -cp joda-time-1.6.2.jar InstantTest.java
$ java -cp joda-time-1.6.2.jar:. InstantTest
1970-01-01T00:01:01.100Z
$
```

Sedangkan untuk Windows perintahnya sedikit berbeda, hal ini dikarenakan perbedaan pemisah classpath, dimana untuk Linux/Unix/Mac menggunakan : (titik dua) dan windows menggunakan ; (titik koma) :

```
$ javac -cp joda-time-1.6.2.jar InstantTest.java
$ java -cp joda-time-1.6.2.jar;. InstantTest
1970-01-01T00:01:01.100Z
$
```

Class `DateTime` juga merepresentasikan konsep `Instant`. Class ini fungsinya menggabungkan fungsi dalam class `Date` plus class `Calendar`, karena bisa digunakan sebagai representasi waktu sekaligus mempunyai method-method untuk melakukan manipulasi data waktu. Berikut ini adalah contoh penggunaan class `DateTime` :

```
import org.joda.time.DateTime;
import java.util.Date;
import java.util.Calendar;
public class DateTimeTest {
    public static void main(String[] args) {
        DateTime dateTime = new DateTime(); //waktu sekarang
        int date = dateTime.getDayOfMonth();
        int month = dateTime.getMonthOfYear(); //dimulai dari 1 hingga 12
        int year = dateTime.getYear();
        DateTime plusMonth = dateTime.plusMonths(2);
        DateTime plusMinutes = dateTime.plusMinutes(2);
        Date d = plusMonth.toDate(); //mengubah DateTime ke Date
        Calendar c = plusMinutes.toGregorianCalendar(); //mengubah DateTime ke Calendar
        System.out.println(dateTime);
    }
}
```

Partial digunakan untuk merepresentasikan waktu yang terpisah antara tanggal dan jam tanpa daerah waktu (timezone). Misalnya class `LocalDate` hanya mempunyai property tanggal, bulan dan tahun, sedangkan class `LocalTime` hanya mempunyai property jam, menit, detik dan mili detik. Class ini jarang digunakan dan lebih praktis menggunakan class `DateTime` saja untuk representasi waktu, class-class tersebut ada hanya untuk tujuan backward compatibility dengan versi Joda Time yang lebih lama. Contoh kodenya seperti di bawah ini :

```
import org.joda.time.LocalDate;
public class PartialTest {
    public static void main(String[] args) {
        LocalDate birthDay = new LocalDate(1965,1,23);
        long millis = birthDay.toDateTimeAtCurrentTime().getMillis();
        System.out.println(millis); //bernilai negatif karena sebelum 1970
        birthDay = birthDay.plusYears(27); //ultah ke 27
        int year = birthDay.getYear(); //tahun ultah ke 27
    }
}
```

Interval adalah representasi jarak antara satu instant dengan instant yang lainnya. Konsep interval mempunyai dua buah instant yaitu waktu mulai dan waktu selesai. Class `Interval` digunakan untuk

merepresentasikan konsep interval ini. Method `getStart` dari class `Interval` digunakan untuk mendapatkan waktu mulai, sedangkan method `getEnd` digunakan untuk mendapatkan waktu selesai. Kalau ingin mengubah waktu awal gunakan method `withStart` dan kalau ingin mengubah waktu akhir gunakan method `withEnd`.

Contoh penggunaan interval adalah sebagai berikut ini :

```
import org.joda.time.DateTime;
import org.joda.time.Interval;
public class IntervalTest {
    public static void main(String[] args) {
        DateTime now = new DateTime();
        DateTime oneMonthLater = now.plusMonths(1);
        Interval interval = new Interval(now, oneMonthLater);
        System.out.println("interval : " + interval);
        System.out.println("start : " + interval.getStart());
        System.out.println("end : " + interval.getEnd());
        System.out.println("duration : " + interval.toDuration());
        //merubah nilai end interval dengan menambahkan satu jam
        interval.withEnd(interval.getEnd().plusHours(1));
    }
}
```

Kalau kita eksekusi kode di atas, hasilnya adalah seperti di bawah ini :

```
$ javac -cp joda-time-1.6.2.jar IntervalTest.java
$ java -cp joda-time-1.6.2.jar:. IntervalTest
interval : 2011-02-16T03:34:37.877/2011-03-16T03:34:37.877
start : 2011-02-16T03:34:37.877+08:00
end : 2011-03-16T03:34:37.877+08:00
duration : PT2419200S
$
```

Kalau anda bekerja dengan windows, jangan lupa mengganti pemisah classpath dengan simbol ; (titik koma).

Konsep durasi dalam Joda Time merepresentasikan lama waktu dalam mili detik. Durasi bisa dihitung langsung dengan memasukkan nilai durasi dalam mili detik atau bisa juga dihitung dari sebuah interval. Contoh kode penggunaan durasi seperti di bawah ini :

```
import org.joda.time.Duration;
import org.joda.time.DateTime;
public class DurationTest {
    public static void main(String[] args) {
        Duration duration = new Duration(10000); // 10 detik
        System.out.println("duration : " + duration);
        DateTime now = new DateTime();
        DateTime oneMonthLater = now.plusMonths(1);
        duration = new Duration(now, oneMonthLater);
        System.out.println("duration of one month : " + duration);
        Duration oneHour = new Duration(1000 * 60 * 60);
        DateTime oneHourLater = now.plus(oneHour);
        System.out.println("one hour later : " + oneHourLater);
    }
}
```

Hasil eksekusi kode di atas adalah seperti berikut ini :

```
$ javac -cp joda-time-1.6.2.jar DurationTest.java
$ java -cp joda-time-1.6.2.jar:. DurationTest
duration : PT10S
duration of one month : PT2419200S
one hour later : 2011-02-16T04:47:30.445+08:00
```

```
$
```

Kalau anda bekerja dengan windows, jangan lupa mengganti pemisah classpath dengan simbol ; (titik koma).

Durasi adalah konsep komputer yang ketat, artinya durasi bisa diukur dalam mili detik dan tidak terikat dengan suatu waktu tertentu. Sedangkan periode adalah sebuah konsep durasi yang sedikit berbeda karena terikat dengan suatu waktu tertentu dan lebih condong sebagai interpretasi manusia terhadap waktu dibanding interpretasi mesin terhadap waktu. Contoh periode adalah : 1 tahun 2 bulan 30 hari, nah berapa total lama periode tersebut dalam mili detik tidaklah selalu sama, tergantung dari mana awal dari periode tersebut. Misalnya 1 tahun di tahun 2000 dengan 1 tahun di tahun 2001 akan berbeda dalam satuan mili detik karena tahun 2000 adalah tahun kabisat di penanggalan gregorian.

Di sisi lain, periode ini lebih praktis dibanding dengan durasi, terutama kalau nilainya lumayan tinggi dan didefinisikan dalam pecahan waktu. Berikut ini contoh penggunaan dari konsep periode:

```
import org.joda.time.Period;
import org.joda.time.DateTime;
import org.joda.time.Hours;
import org.joda.time.Minutes;
public class PeriodTest {
    public static void main(String[] args) {
        Period p = new Period(1000); //1 detik
        System.out.println("period : " + p);
        p = new Period(2,3,9,125); //2 jam 3 menit 9 detik dan 125 mili detik
        System.out.println("period : " + p);
        DateTime startTime = new DateTime(2000,1,1,9,0,0,0); //1 januari 2000 jam 9
        //menambahkan period ke instant untuk mendapatkan instant baru
        DateTime endTime = startTime.plus(p);
        System.out.println("end time : " + endTime);
        //Periode nilainya tidak menentu, satu hari belum tentu 24 jam,
        //tergantung apakah hari itu ada daylight savings atau tidak.
        //Begitupula satu tahun belum tentu 365 hari,
        //tergantung kabisat atau tidak.
        //Mengubah Period ke durasi harus ada waktu awal,
        //kemudian ditambah dengan period dapat waktu akhir dan dihitung durasinya
        Hours hours = Hours.hoursBetween(startTime,endTime);
        //mendapatkan durasi dalam jam dengan tipe data int
        int hoursBetween = hours.getHours();
        System.out.println("hours duration : " + hours);
        Minutes minutes = Minutes.minutesBetween(startTime, endTime);
        System.out.println("minutes duration : " + minutes);
    }
}
```

Hasil eksekusi kode di atas adalah sebagai berikut :

```
$ javac -cp joda-time-1.6.2.jar PeriodTest.java
$ java -cp joda-time-1.6.2.jar:. PeriodTest
period : PT1S
period : PT2H3M9.125S
end time : 2000-01-01T11:03:09.125+08:00
hours duration : PT2H
minutes duration : PT123M
$
```

Kalau anda bekerja dengan windows, jangan lupa mengganti pemisah classpath dengan simbol ; (titik koma).

Chronology adalah jantung dari Joda Time, semua perhitungan instant, periode dan durasi

didasarkan pada jenis penanggalan yang digunakan. Joda Time secara default menyediakan beberapa sistem penanggalan, antara lain :

- Gregorian : standard penanggalan masehi yang dimulai pada 15 september 1582, penanggalan ini mendefinisikan bahwa tahun kabisat hanya terjadi pada tahun yang habis dibagi 4, tidak habis dibagi 100 dan habis dibagi 400. Misalnya tahun 2008 adalah kabisat karena habis dibagi 4, tahun 2100 bukan kabisat karena walaupun habis dibagi 4 juga habis dibagi 100. Tahun 2000 adalah kabisat karena habis dibagi 4, walaupun habis dibagi 100 tetapi habis dibagi 400.
- Julian : standard penanggalan masehi yang sebelum tanggal 15 september 1582. Penanggalan ini mendefinisikan bahwa setiap tahun yang habis dibagi 4 adalah tahun kabisat.
- ISO8601 : standard penanggalan yang menggunakan sistem Gregorian plus standarisasi format penulisan tanggal. Joda Time secara default menggunakan ISO8601 sebagai default chronology
- GJ : penanggalan yang menggabungkan Julian dan Gregorian. Sebelum tanggal 15 September 1582 sistem penanggalan yang digunakan adalah Julian dan setelahnya adalah Gregorian.
- Penanggalan lain yang spesifik : Buddhist, Coptic, Ethiopic

Sampai di sini kita sudah belajar secara umum tentang konsep waktu dalam Joda Time beserta implementasinya, nah kita akan membahas lagi sedikit tentang bagaimana Joda Time menyederhanakan kode perhitungan tanggal yang dibuat dengan class dari JDK.

Contoh pertama : membuat instant, tambahkan 20 hari kemudian print hasilnya ke dalam console dengan format tanggal tertentu. Kode untuk melakukan perhitungan tersebut dengan menggunakan class-class dari JDK adalah seperti berikut ini :

```
import java.util.Calendar;
import java.text.SimpleDateFormat;
public class CalculateDateTest {
    public static void main(String[] args) {
        Calendar c = Calendar.getInstance();
        c.set(2000,Calendar.JANUARY,1,0,0,0);
        c.add(Calendar.DATE, 20);
        SimpleDateFormat format = new SimpleDateFormat("dd MMM yyyy HH:mm:ss");
        System.out.println("date : " + format.format(c.getTime()));
    }
}
```

Implementasinya menggunakan Joda Time memerlukan jumlah baris kode setengahnya dari kode di atas :

```
import org.joda.time.DateTime;
public class CalculateDateJodaTest {
    public static void main(String[] args) {
        DateTime d = new DateTime(2000,1,1,0,0,0,0);
        System.out.println("date : " +
            d.plusDays(20).toString("dd MMM yyyy HH:mm:ss"));
    }
}
```

Contoh kedua sedikit lebih rumit, misalnya kita ingin mengetahui hasil perhitungan ini : tanggal awal adalah 1 januari 2000, tambahkan 1 bulan 45 hari dari tanggal awal, kemudian cari tanggal berapa di hari minggu pada minggu tersebut. Perhitungan ini cukup rumit untuk dibuat implementasinya menggunakan library JDK, tetapi menggunakan Joda Time perhitungan ini sangat sederhana dan hanya memerlukan beberapa baris kode saja. Implementasinya :

```
import org.joda.time.DateTime;
public class CalculateComplicatedDateJodaTest {
```

```

public static void main(String[] args) {
    DateTime d = new DateTime(2000,1,1,0,0,0,0);
    System.out.println("date : " +
        d.plusMonths(1).plusDays(45).dayOfWeek()
        .withMaximumValue().toString("dd MMM yyyy HH:mm:ss"));
}
}

```

Library Joda Time adalah library yang wajib ada di setiap project yang kita buat. Perhitungan tanggal menjadi mudah menggunakan Joda Time.

BigDecimal dan Currency

Penanganan angka pecahan sangat penting untuk aplikasi bisnis, terutama untuk uang, hal ini karena kalau ada kesalahan pembulatan maka akan ada kesalahan perhitungan uangnya. Selisih sedikit saja akan membuat perhitungan accounting atau pajak menjadi tidak seimbang sehingga menyebabkan laporan keuangan tidak benar.

Bekerja dengan angka pecahan, kita pasti langsung teringat dengan tipe data float atau double, tergantung presisi dan besarnya data yang digunakan. Tetapi ketika berurusan dengan uang, kedua tipe data ini bisa menyebabkan masalah pelik, terutama kalau ada aturan berapa angka di belakang koma saja yang berpengaruh terhadap perhitungan. Misalnya hanya 2 buah angka di belakang koma saja yang digunakan, sisanya bisa diabaikan dengan menggunakan pembulatan. Tipe data float dan double tidak bisa diatur berapa angka di belakang koma, sehingga proses pembulatan juga tidak selalu berhasil dengan baik.

Mari kita lihat contoh perhitungan bilangan pecahan menggunakan double yang bisa menyebabkan kesalahan angka dibelakang koma. Misalnya kita menjual barang dengan harga Rp. 1.000.000,05 kemudian ada diskon 10%, baru setelah itu dihitung PPN 5%. Untuk melihat hasilnya digunakan class NumberFormat agar tampilan diformat menggunakan format uang Rupiah yang benar.

```

import java.text.NumberFormat;
import java.util.Locale;
public class SaleDecimalTest {
    public static void main(String args[]) {
        double amount = 1000000.05;
        double discount = amount * 0.10;
        double total = amount - discount;
        double tax = total * 0.05;
        double taxedTotal = tax + total;
        NumberFormat money = NumberFormat.getCurrencyInstance(
            new Locale("in","ID"));
        System.out.println("Subtotal : "+ money.format(amount));
        System.out.println("Discount : " + money.format(discount));
        System.out.println("Total : " + money.format(total));
        System.out.println("Tax : " + money.format(tax));
        System.out.println("Tax+Total: " + money.format(taxedTotal));
    }
}

```

Hasil eksekusi kode di atas sedikit mengejutkan, perhatikan baik-baik :

```

$ javac SaleDecimalTest.java
$ java SaleDecimalTest
Subtotal : Rp1.000.000,05
Discount : Rp100.000,00
Total : Rp900.000,04
Tax : Rp45.000,00
Tax+Total: Rp945.000,05
$

```

nilai total bukanya Rp 900.000,05 tetapi malah Rp 900.000,04 ternyata terjadi kesalahan

pembulatan uang pada NumberFormat. Kalau kita hilangkan proses memformat angkanya seperti di bawah ini :

```
import java.util.Locale;
public class SaleDecimalUnformattedTest {
    public static void main(String args[]) {
        double amount = 1000000.05;
        double discount = amount * 0.10;
        double total = amount - discount;
        double tax = total * 0.05;
        double taxedTotal = tax + total;
        System.out.println("Subtotal : "+ amount);
        System.out.println("Discount : " + discount);
        System.out.println("Total : " + total);
        System.out.println("Tax : " + tax);
        System.out.println("Tax+Total: " + taxedTotal);
    }
}
```

terlihat bahwa sebenarnya angka di belakang koma sedikit berantakan :

```
$ javac SaleDecimalUnformattedTest.java
$ java SaleDecimalUnformattedTest
Subtotal : 1000000.05
Discount : 100000.005
Total : 900000.045
Tax : 45000.002250000005
Tax+Total: 945000.0472500001
$
```

Dengan menggunakan BigDecimal kita bisa mengontrol bagaimana caranya melakukan pembulatan angka di belakang koma. Aturan pembulatan yang digunakan bisa dipilih salah satu dari nilai enum class RoundingMode yang diperkenalkan semenjak Java 6. Aturan pembulatan yang tersedia di dalam class RoundingMode antara lain :

- CEILING : pembulatan ke atas
- DOWN : pembulatan ke bawah mendekati nol, kalau nilainya negatif dibulatkan ke angka lebih besar yang lebih dekat dengan angka nol.
- FLOOR : pembulatan ke atas mendekati angka negatif tak terhingga. Kalau angkanya negatif dibulatkan ke angka lebih kecil lagi mendekati negatif tak terhingga.
- UP : pembulatan yang selalu menjauhi angka nol. Kalau nilai positif dibulatkan ke lebih besar, kalau angka negatif dibulatkan ke angka lebih kecil lagi.
- HALF_DOWN : dibulatkan ke angka terdekat, kecuali kalau sama jaraknya antara ke atas dan ke bawah, maka dibulatkan ke angka bawah mendekati nol.
- HALF_UP : dibulatkan ke angka terdekat, kecuali kalau sama jaraknya antara ke atas dan ke bawah, maka dibulatkan ke angka atas menjauhi nol. Pembulatan ini adalah sistem pembulatan yang digunakan dalam sistem keuangan.
- UNNECESSARY : tidak perlu dilakukan pembulatan

Selain masalah pembulatan, tipe data double juga mempunyai masalah serius dengan representasinya di dalam memory. Tidak semua angka dibelakang koma bisa direpresentasikan dalam binary, misalnya angka 0.035, kalau angka ini disimpan dalam tipe data double maka hasilnya adalah 0.034999999999999996. Masalah representasi data di belakang koma ini tidak muncul di BigDecimal, sehingga bisa dihindari masalah seperti ini.

Mari kita lihat contoh masalah ini dalam kode :

```
import java.math.BigDecimal;
import java.math.RoundingMode;
public class DoubleProblemTest {
```

```

public static void main(String[] args) {
    double dd = .35;
    BigDecimal d = new BigDecimal(dd);
    System.out.println(".35 = " + d); //hasilnya berantakan karena dari double
    d = d.setScale(2, RoundingMode.HALF_UP);
    System.out.println(".35 = " + d); //hasilnya bagus setelah pembulatan
    BigDecimal dec = new BigDecimal("0.35");
    System.out.println(".35 = " + dec); //hasilnya bagus
}
}

```

hasilnya seperti di bawah ini :

```

$ javac DoubleProblemTest.java
$ java DoubleProblemTest
.35 = 0.34999999999999997779553950749686919152736663818359375
.35 = 0.35
.35 = 0.35
$

```

BigDicimal yang dibuat dari double mempunyai nilai sama persis dengan double, makanya lebih baik kalau nilainya berasal dari String sehingga bisa sesuai yang kita harapkan.

Nah sekarang kita kembali lagi ke contoh pertama dalam bab ini. Kita akan mengimplementasikan perhitungan harga, diskon dan pajak menggunakan BigDecimal dan hasilnya harus cocok, tidak boleh ada selisih angka di belakang koma:

```

import java.math.BigDecimal;
import java.math.RoundingMode;
public class BigDecimalTest {
    public static void main(String args[]) {
        BigDecimal amount = new BigDecimal("1000000.05");
        BigDecimal discountPercent = new BigDecimal("0.10");
        BigDecimal discount = amount.multiply(discountPercent);
        discount = discount.setScale(2, RoundingMode.HALF_UP);
        BigDecimal total = amount.subtract(discount);
        total = total.setScale(2, RoundingMode.HALF_UP);
        BigDecimal taxPercent = new BigDecimal("0.05");
        BigDecimal tax = total.multiply(taxPercent);
        tax = tax.setScale(2, RoundingMode.HALF_UP);
        BigDecimal taxedTotal = total.add(tax);
        taxedTotal = taxedTotal.setScale(2, RoundingMode.HALF_UP);
        System.out.println("Subtotal : " + amount);
        System.out.println("Discount : " + discount);
        System.out.println("Total : " + total);
        System.out.println("Tax : " + tax);
        System.out.println("Tax+Total: " + taxedTotal);
    }
}

```

Hasil eksekusinya adalah seperti berikut ini :

```

$ javac BigDecimalTest.java
$ java BigDecimalTest
Subtotal : 1000000.05
Discount : 100000.01
Total : 900000.04
Tax : 45000.00
Tax+Total: 945000.04
$

```

terlihat bahwa tanpa menggunakan formatter hasil dari perhitungan BigDecimal sudah tepat dan

presisi, tidak ada kesalahan perhitungan sama sekali.

I/O

Input Output adalah topik yang sangat penting dalam aplikasi apapun, membaca atau menulis file merupakan rutin yang sering ada dalam aplikasi. I/O juga meliputi komunikasi dengan aplikasi lain yang ada dalam jaringan melalui komunikasi socket.

File

File adalah class yang merepresentasikan file dan folder yang ada di dalam file system, baik yang ada di hardisk lokal atau ada di dalam mesin lain yang diakses dari file sharing system. Dengan menggunakan Class ini kita bisa melakukan :

- mengecek sebuah path apakah berupa file ataukah folder
- membuat, merename atau menghapus file dan folder
- mengecek apakah folder atau file tersebut ada dalam filesystem
- mengambil daftar isi dari folder.

Mari kita lihat bagaimana class File ini beraksi :

```
import java.io.File;
import java.io.IOException;
public class FileTest {
    public static void main(String[] args) {
        File f = new File("newfile.txt");
        if(!f.exists()){
            try{
                f.createNewFile();
                System.out.println(f.getAbsolutePath());
            } catch(IOException e){
                e.printStackTrace();
            }
        }
        File folder = new File("newfolder");
        if(!f.exists()) {
            f.mkdir();
            //f.mkdirs(); //membuat folder di dalam folder
        }
        File currentFolder = new File(System.getProperty("user.dir"));
        File[] siblings = currentFolder.listFiles();
        for(int i=0;i<siblings.length;i++){
            File sibling = siblings[i];
            if(sibling.isFile()) {
                System.out.println("f " + sibling.getName());
            } else {
                System.out.println("d " + sibling.getName());
            }
        }
    }
}
```

Kode di atas akan membuat file, kemudian membuat folder dan akhirnya mengambil isi semua file dan folder dari user.dir (folder dimana aplikasi java dijalankan). Masih banyak method-method lain yang ada dalam class File yang belum kita coba, silahkan lihat javadoc untuk melihat semua method yang dimiliki class File ini.

Reader

Class-class yang mengextends class Reader biasa digunakan kalau ingin membaca karakter stream, artinya kalau yang dibaca adalah file maka file tersebut berisi text. Ada beberapa file yang menextends class Reader ini, tapi kita hanya akan membahas class BufferedReader untuk membaca file text.

BufferedReader akan menyimpan data yang sudah dibaca ke dalam memory, sehingga ketika method read atau readLine dipanggil dan data sudah ada dalam memory, BufferedReader tidak akan membaca lagi dari dalam file.

Mari kita lihat contoh kodenya :

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
public class ReaderTest {
    public static void main(String[] args) {
        try{
            BufferedReader reader = new BufferedReader(new
                FileReader("ReaderTest.java"));
            String line;
            while((line = reader.readLine())!=null) {
                System.out.println(line + "\n");
            }
            //jangan lupa untuk selalu memanggil close setelah reader tidak digunakan
            reader.close();
        } catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```

Writer

Writer digunakan untuk menulis String atau char[] ke dalam sebuah file. Kita akan menggunakan BufferedWriter sebagai contohnya.

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
public class WriterTest {
    public static void main(String[] args) {
        try{
            BufferedWriter writer = new BufferedWriter(new
                FileWriter("file.txt"));
            writer.write("HelloWorld ini file text saya\n");
            //menulis text ke dalam file
            writer.flush();
            writer.write("ini text di baris dua\n");
            //jangan lupa untuk selalu memanggil close
            //setelah reader tidak digunakan
            //close juga sekaligus memanggil flush dan menulis data ke dalam file
            writer.close();
        } catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```


InputStream

InputStream digunakan untuk membaca dari I/O yang berupa byte stream, semua data direpresentasikan dalam byte, jadi baik data berupa text maupun binary dapat dibaca menggunakan InputStream.

Class yang mengextends InputStream cukup banyak, karena InputStream ini adalah dasar dari IO di java. Semua sumber input output rata-rata bisa diakses menggunakan InputStream ini. Kita sayangnya tidak akan membahas panjang lebar mengenai InputStream ini, yang kita bahas hanya sedikit pengenalan bagaimana menggunakan InputStream.

Penggunaan InputStream yang pertama adalah membaca file binary, sering kali kita dihadapkan dengan upload file dari user ke server, nah proses upload file ini mau tidak mau akan melibatkan InputStream untuk membaca file yang dikirim ke server.

Mari kita lihat struktur kode untuk membaca file byte per byte :

```
InputStream is = new FileInputStream("file.txt");
int byteRead;
do {
    byteRead = is.read();
    //lakukan sesuatu dengan datanya
} while(byteRead!=-1);
```

Method read dari class InputStream digunakan untuk membaca byte per byte dari sumber data, variabel byteRead memegang nilai byte yang sedang dibaca, kalau nilai variabel byteRead adalah -1 artinya sudah tidak ada lagi data yang bisa dibaca, alias sudah sampai akhir dari data. Subclass dari InputStream mempunyai bermacam-macam method read, misalnya DataInputStream mempunyai method readBoolean, readObject, readDouble dan sebagainya.

Mari kita lihat contoh sebenarnya dari InputStream, kita akan membuat kode yang bisa memanggil URL dari internet kemudian membaca halaman dari url tersebut menggunakan InputStream :

```
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.net.MalformedURLException;
import java.net.URLConnection;
public class UrlInputStreamTest {
    public static void main(String[] args) {
        try{
            URL url = new URL("http://www.yahoo.com");
            URLConnection urlconn = url.openConnection();
            InputStream is = urlconn.getInputStream();
            int byteRead;
            do{
                byteRead = is.read();
                System.out.print((char)byteRead);
            } while(byteRead != -1);
        } catch(MalformedURLException ex) {
            ex.printStackTrace();
        } catch(IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

output dari kode di atas adalah html yang dibaca dari yahoo.com

OutputStream

OutputStream tentu saja adalah kebalikan dari InputStream, fungsi OutputStream adalah

menulis data ke sumber data, baik itu data dari filesystem ataupun dari koneksi socket.

Kita modifikasi kode di atas untuk menyimpan String ke dalam file menggunakan `FileOutputStream` :

```
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.FileOutputStream;
import java.net.URL;
import java.net.MalformedURLException;
import java.net.URLConnection;
public class UrlOutputStreamTest {
    public static void main(String[] args) {
        try{
            URL url = new URL("http://www.yahoo.com");
            URLConnection urlconn = url.openConnection();
            InputStream is = urlconn.getInputStream();
            OutputStream os = new FileOutputStream("yahoo.txt");
            int byteRead;
            do{
                byteRead = is.read();
                os.write(byteRead);
                System.out.print((char)byteRead);
            } while(byteRead != -1);
            os.flush();
        } catch(MalformedURLException ex) {
            ex.printStackTrace();
        } catch(IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Kode di atas akan membaca website yahoo.com kemudian mengambil isinya dan menyimpan text html ke dalam file yahoo.txt

ImageIO

ImageIO adalah class yang berisi utility untuk bekerja dengan gambar. Ada beberapa class di dalam java yang merepresentasikan gambar, antara lain `Icon`, `ImageIcon`, `Image` dan `BufferedImage`.

`Icon` adalah interface yang digunakan untuk menampilkan gambar dalam komponen Swing, misalnya membuat background atau membuat icon. `Image` adalah class untuk merepresentasikan sebuah gambar dalam memory java, kita bisa mengambil lebar dan tinggi serta ukuran dari gambar menggunakan class `Image` ini. `ImageIcon` adalah implementasi `Icon` yang diperoleh dari sebuah `Image`. Sedangkan `BufferedImage` adalah class yang digunakan untuk memanipulasi `Image`, misalnya melakukan resize atau menampilkan gambar yang bisa dicampur dengan bentuk-bentuk lain (seperti lingkaran atau persegi panjang) untuk kemudian disimpan menjadi gambar baru atau ditampilkan dalam canvas.

ImageIO memudahkan konversi dari satu class menjadi class lain, nah mari kita lihat bagaimana penggunaannya dalam kode. Contoh kode pertama adalah membaca file gambar dari internet, kemudian disimpan dalam file kemudian ditampilkan.

```
import java.io.*;
import java.net.*;
import javax.swing.*;
import javax.imageio.*;
public class ImageTest {
    public static void main(String[] args) {
```

```

try {
    URL url = new URL(
        "http://ifnubima.org/wp-content/uploads/2010/07/2010-07-25_0334.png");
    URLConnection urlconn = url.openConnection();
    InputStream is = urlconn.getInputStream();
    OutputStream os = new FileOutputStream("image.png");
    int byteRead;
    do {
        byteRead = is.read();
        os.write(byteRead);
    } while (byteRead != -1);
    os.flush();
    os.close();
    //menampilkan image
    Icon icon = new ImageIcon(
        ImageIO.read(new File("image.png")));
    JOptionPane.showMessageDialog(null, "ini gambar","menampilkan gambar",
        JOptionPane.INFORMATION_MESSAGE,
        icon);
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
}

```

Untuk mengetahui lebih banyak lagi tentang ImageIO utility, silahkan baca javadoc class ini.

Socket

Socket adalah satu ujung dari komunikasi dua arah antara dua buah aplikasi yang berjalan di lingkungan jaringan. Class Socket digunakan untuk merepresentasikan koneksi antara aplikasi client dan server. Package java.net menyediakan dua buah class : Socket dan ServerSocket yang mengimplementasikan client dan server.

Kita sudah sedikit melihat bagaimana socket beraksi, yaitu pada saat menggunakan URLConnection di bab sebelumnya untuk mengambil isi halaman yahoo.com, di dalam class URLConnection terdapat implementasi socket client untuk menghubungi yahoo.com dan mengambil halamanya lewat protokol HTTP.

Socket adalah jantung dari aplikasi network, setiap kali kita menghubungi aplikasi yang berada di server / workstation berbeda kita harus menggunakan socket ini, seperti koneksi ke database, koneksi ke server three tier dan sebagainya.

Komunikasi socket harus menggunakan protokol tertentu untuk mengatur bagaimana mekanisme komunikasi antara dua buah workstation. Protokol bisa dianalogikan dalam proses komunikasi dua orang manusia, kalau ingin berkomunikasi kita harus memanggil orang tersebut untuk mendapatkan perhatiannya, kemudian menunggu apakah orang ini bersedia berkomunikasi dengan kita atau tidak, setelah keduanya sepakat berkomunikasi maka wajah dan mata saling berhadapan untuk menandakan bahwa proses komunikasi bisa dilanjutkan, setelah itu percakapan terjadi hingga keduanya sepakat untuk mengakhiri percakapan dan saling memalingkan muka.

Protokol yang akan kita gunakan dalam contoh kita kali ini adalah TCP (Transmission Control Protocol) yang sangat terkenal. Protokol ini menyediakan mekanisme komunikasi point-to-point yang biasa digunakan dalam arsitektur client-server. Internet yang menggunakan protokol HTTP untuk menampilkan halaman web juga menggunakan protokol TCP sebagai mekanisme transfer data antara client (browser) dan server (webserver, website).

Contoh aplikasi yang akan kita buat sangat sederhana, yaitu aplikasi ping-pong. Server akan berjalan terlebih dahulu, membuka port tertentu dan menunggu client mengirim data ping, setelah ping diterima maka server akan mengirim balik data pong untuk diterima client.

Kode server sangat sederhana, langkah-langkah yang harus dilakukan oleh server adalah

sebagai berikut :

1. Membuka ServerSocket
2. Berusaha menerima client
3. membuat reader dan writer untuk menerima dan mengirim data ke client
4. dalam sebuah loop akan membaca data yang dikirim client
5. Kalau ada data yang dikirim client segera membalas dengan mengirim Pong ke client

Kode untuk Server :

```
import java.io.*;
import java.net.*;
public class PingPongServer {
    public static void main(String[] args) {
        try {
            //1. buka server socket
            ServerSocket server = new ServerSocket(4444);
            System.out.println("Server ready. Listening in port 4444");
            //2. berusaha menerima client
            Socket clientSocket = server.accept();
            System.out.println("Connection from client is excepted");
            //3. membuat reader dan writer untuk menerima dan mengirim data ke client
            BufferedWriter out = new BufferedWriter(
                new OutputStreamWriter(clientSocket.getOutputStream()));
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));
            String inputLine = null;
            //4. dalam sebuah loop akan membaca data yang dikirim client
            while((inputLine = in.readLine())!=null){
                System.out.println("Recieved " + inputLine + " from client");
                //5. Kalau ada data yang dikirim client segera membalas dengan mengirim
                // Pong ke client
                out.write("Pong\n");
                System.out.println("Send Pong to client");
                out.flush();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Kode untuk server ini harus dijalankan terlebih dahulu sebelum client dijalankan.

Client akan melakukan beberapa langkah berikut ini ketika berkomunikasi dengan server :

1. Membuat client socket sekaligus menghubungi server
2. Membuat reader dan writer untuk membaca dan menulis data ke server
3. Dalam sebuah loop, kirim data ke server
4. Dalam loop yang sama berusaha terima data dari server
5. Dalam loop tersebut lakukan 1 detik pause agar output bisa terlihat bagus

Kode implementasinya sebagai berikut :

```
import java.io.*;
import java.net.*;
public class PingPongClient {
```

```

public static void main(String[] args) {
    try {
        //1. Membuat client socket sekaligus menghubungi server
        Socket clientSocket = new Socket("localhost", 4444);
        System.out.println("client konek to server localhost:4444");
        //2. Membuat reader dan writer untuk membaca dan menulis data ke server
        BufferedWriter out = new BufferedWriter(
            new OutputStreamWriter(clientSocket.getOutputStream()));
        BufferedReader in = new BufferedReader(
            new InputStreamReader(clientSocket.getInputStream()));
        do {
            //3. Dalam sebuah loop, kirim data ke server
            out.write("Ping\n");
            System.out.println("Send ping to server");
            out.flush();
            //4. Dalam loop yang sama berusaha terima data dari server
            String inputLine = in.readLine();
            System.out.println("Recieved " + inputLine + " from server");
            //5. Dalam loop tersebut lakukan 1 detik pause agar output bisa bagus
            Thread.sleep(1000);
        } while (true); //lakukan terus menerus
    } catch (UnknownHostException ex) {
        ex.printStackTrace();
    } catch (IOException ex) {
        ex.printStackTrace();
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }
}
}

```

Selanjutnya compile dan jalankan server terlebih dahulu

```

$ javac PingPongServer.java
$ java PingPongServer
Server ready. Listening in port 4444
Connection from client is excepted
Recieved Ping from client
Send Pong to client
Recieved Ping from client
.
.
.

```

Output di baris pertama dan kedua akan segera muncul setelah server dijalankan, baris berikutnya akan muncul setelah client berjalan dan komunikasi client-server terjadi.

```

$ javac PingPongClient.java
$ java PingPongClient
client konek to server localhost:4444
Send ping to server
Recieved Pong from server
Send ping to server
Recieved Pong from server
.
.

```

Sampai di sini, dasar-dasar java fundamental sudah kita pelajari dengan baik. Kalau masih banyak hal-hal yang belum dipahami silahkan baca ulang bab ini atau rujuk ke buku tentang java fundamental yang lebih detail seperti Thinking in Java karya Bruce Eckel.

Java 5 Update

Feel of Java

Java pertama dibuat dan dikembangkan oleh Sun Microsystems untuk diimplementasikan dalam embedded sistem seperti perangkat elektronik sehari-hari. Pada saat proses perancangan, java didesain untuk menjadi "blue collar language" atau bahasa pemrograman untuk pekerjaan sehari-hari, bukannya bahasa canggih untuk riset Phd. Untuk mencapai tujuan tersebut, dibuatlah konsep "feel-of-java", konsep ini berlandaskan kesederhanaan dan kemudahan dibaca.

Konsep ini memudahkan programmer dari berbagai macam background merasa mudah belajar bahasa java. Sintaks java dibuat semirip mungkin dengan C dan C++, bahasa paling populer saat itu. Bahkan beberapa konsep OOP dari C++ yang dirasa menyulitkan dihilangkan dari feature java, misalnya operator overloading dan multiple inheritance.

Java juga mewarisi konsep static language dari C/C++. Static language mewajibkan setiap variabel harus didefinisikan terlebih dahulu sebelum diinisialisasi. Konsep ini memudahkan compiler untuk memeriksa statement java, sehingga error operasi variabel dapat ditemukan pada saat compile. Berbeda dengan dynamic language seperti ruby dan PHP, semua variabel tidak perlu didefinisikan dan kita bisa memasukkan tipe data apapun ke dalam variabel tersebut. Kalau ada error dalam operasi variabel, error tersebut akan terdeteksi ketika program dijalankan.

Feature Kesederhanaan dan kemudahan dibaca mempunyai implikasi bahasa Java sangat verbose, kode yang diketik relatif lebih banyak dari bahasa pemrograman lain. Hal inilah yang banyak dikeluhkan programmer java, sehingga Sun memutuskan untuk membuat perubahan terhadap bahasa java pada java versi 5. Perubahan tersebut antara lain:

- Enhanced for loop
- Autoboxing/Unboxing
- Static Import
- Varargs
- TypeSafe Enum
- Generics
- Annotation

Perubahan di atas semuanya dimaksudkan untuk mengurangi kode program yang harus ditulis programmer tanpa mengurangi semangat kesederhanaan java. Mari kita bahas satu per satu dari yang paling mudah.

Enhanced for Loop

For Loop Sebelum Java 5

Iterasi dalam sintaks bahasa java ada tiga bentuk: while, do-while dan for. Iterasi secara umum digunakan untuk melakukan satu tugas tertentu secara berulang-ulang. Salah satu kegunaan dari iterasi adalah mengambil satu-satu nilai dari sebuah kumpulan data.

Kumpulan data dalam java bisa dikategorikan tiga: array, Collection dan Map. Array adalah kumpulan data sejenis yang panjangnya tetap, setiap data diletakkan secara berurutan berdasarkan index. Cara paling lazim untuk mengambil satu per-satu data dari array adalah menggunakan suatu variabel index yang dimulai dari 0 kemudian diiterasi hingga n-1. Berikut ini adalah contoh kodenya :

```
String[] arr = {"a","b","c"};
for(int idx=0;idx<arr.length;idx++){
    System.out.println("current value" + arr[idx]);
}
```

bandingkan kode di atas dengan kode yang sama untuk PHP:

```
$arr = array("a","b","c");
```

```
foreach($val in $arr)
    echo $val;
}
```

Kode PHP mempunyai jumlah kode lebih sedikit dan lebih sederhana.

Collection dalam java mempunyai peran yang sangat penting dalam aplikasi, karena kita bisa mempunyai kumpulan data dengan behaviour berbeda hanya dengan memilih class Collection yang tepat, tidak perlu membuat algoritma sendiri hanya untuk melakukan sorting data. Namun untuk mengambil data dari collection kita harus menulis kode yang sedikit lebih rumit dibandingkan dengan array, berikut ini contohnya:

```
List lists = new ArrayList();
lists.add("a");
lists.add("b");
Iterator iterator = lists.iterator();
while(iterator.hasNext()){
    String current = (String) iterator.next();
    System.out.println("current value :" + current);
}
```

Terlihat kodenya cukup banyak, karena kita harus membuat variabel bertipe Iterator hanya untuk mengambil nilai dari Collection.

Map adalah bentuk kumpulan data yang mempunyai pasangan key-value. Key dalam Map berfungsi sebagai index yang mempunyai pasangan Value. Sebelum Java 5, cara mengakses setiap elemen dalam Map cukup rumit, berikut ini contohnya :

```
Map maps = new HashMap();
maps.put("Rp", "Rupiah");
maps.put("$", "US Dollar");
Iterator i = maps.keySet().iterator();
while(i.hasNext()){
    String key = (String) i.next();
    System.out.println("Key : " + key);
    System.out.println(" value :" + maps.get(key));
}
```

Kita harus mengambil dahulu daftar Key dari map, kemudian melakukan iterasi satu-satu terhadap Key dan dengan menggunakan key tersebut baru bisa diambil pasangan valuenya.

For Loop di Java 5

Bentuk penulisan for loop di atas cukup merepotkan dan dibandingkan dengan bahasa lain sudah ketinggalan jaman. Oleh karena itu diperlukan perubahan sintaks penulisan for loop agar proses pengambilan nilai dari kumpulan data menjadi lebih sederhana dan ringkas.

For loop untuk mengambil nilai dari array menjadi seperti berikut ini:

```
String[] arr = {"a","b","c"};
for(String current : arr){
    System.out.println("current value" + current);
}
```

Kode di atas menunjukkan bahwa kita tidak perlu lagi membuat variabel index untuk mengambil satu per satu isi dari array. Setiap kali for dilaksanakan, maka elemen yang sedang aktif akan disalin nilainya ke variabel current.

Bentuk for loop untuk Collection sama persis dengan Array di atas. Tidak lagi diperlukan Iterator untuk mengambil satu per satu elemen dari collection:

```
Collection<String> lists = new ArrayList<String>();
lists.add("a");
lists.add("b");
for(String current : lists){
```

```
    System.out.println("current value :" + current);  
}
```

Sintaks untuk mengakses setiap pasangan key-value juga menjadi lebih ringkas :

```
Map<String,String> maps = new HashMap<String,String>();  
maps.put("Rp", "Rupiah");  
maps.put("$", "US Dollar");  
for(String key : maps.keySet()){  
    System.out.println("Key : " + key);  
    System.out.println(" value :" + maps.get(key));  
}
```

Perbaikan for loop ini pada dasarnya tidak mengubah sintaks java secara drastis, yang terjadi adalah sebelum kode di atas dikompilasi menjadi file class, ada satu langkah untuk merubah kode di atas menjadi bentuk yang lama.

Autoboxing/Unboxing

Primitif dan Wrapper

Java dibuat pada pertengahan dekade 90-an, pada waktu itu memory RAM masih mahal. Arsitek Java memutuskan untuk tetap memasukkan tipe data primitif dengan alasan menghemat memory. Tipe data primitif hanya mempunyai nilai tunggal tanpa mempunyai method, sehingga membutuhkan kapasitas memory yang lebih kecil dibanding dengan object.

Tipe data Primitif mempunyai class padananya yang disebut dengan Wrapper class yang menyediakan method utility untuk melakukan convert dari satu tipe ke tipe yang lain, misalnya mengkonvert data dari tipe int menjadi double. Berikut ini adalah tabel tipe data primitif dan wrappernya:

int	Integer
double	Double
float	Float
short	Short
char	Character
byte	Byte
boolean	Boolean
long	Long

Dalam banyak kasus terkadang kita perlu melakukan convert dari nilai primitif ke nilai wrappernya atau sebaliknya, berikut ini contohnya :

```
int a = 10;  
Integer b = Integer.valueOf(a);  
Integer c = new Integer(a);  
int d = c.intValue();
```

kode di atas sangat merepotkan dan tidak ada gunanya.

Autoboxing/Unboxing memberikan kemudahan dengan menghilangkan kewajiban kita untuk menulis kode convert dari primitif ke wrapper atau sebaliknya, dengan inboxing/outoboxing kode di atas akan menjadi :

```
int a = 10;  
Integer b = a;  
Integer c = a;  
int d = c;
```

Seperti halnya enhanced for loop, inboxing/outoboxing dilaksanakan pada level compiler, sebelum kode dirubah menjadi .class, compiler akan merubah kode tersebut ke bentuk lamanya. Hal ini akan memberikan beberapa konsekuensi, perhatikan kode di bawah ini:


```
Integer a = null;
int b = a;
```

kode di baris kedua, ada proses perubahan dari wrapper ke primitif, sebenarnya kode tersebut akan diubah ke bentuk lama sebelum dicompile. Nilai dari variabel a adalah null, ketika akan dirubah ke bentuk primitif, a akan memanggil method intValue() dari class Integer, pemanggilan ini menyebabkan error NullPointerException ketika kode di atas dijalankan.

Kita harus hati-hati menggunakan fasilitas ini untuk menghindari error NullPointerException.

Static Import

Utility Class

Sering kita menjumpai jenis class yang disebut Utility Class, class ini hanya berisi static method, contohnya adalah class Math dan Assert. Setiap kali kita akan memanggil method dari class utility ini, kita harus menuliskan nama classnya berulang-ulang. Jika kita bekerja intensive dengan method dalam class Math, kode program kita terlihat tidak rapi, berikut ini contohnya:

```
Double result = Math.pow(Math.cos(180),3) /2;
```

Dengan menggunakan static import, kita tidak harus menuliskan nama classnya jika akan menggunakan method atau variabel static. Sintaks penulisan static import :

```
static import java.lang.Math.pow;
static import java.lang.Math.cos;
```

kita juga bisa menggunakan wildcard (*) untuk mengimport semua static member dalam class tersebut.

```
static import java.lang.Math.*;
```

Setelah static import didefinisikan, kode untuk memanggil method pow dan cos dari class Math menjadi lebih sederhana, seperti di bawah ini:

```
Double result = pow(cos(180),3) /2;
```

Varargs

Fungsi Dengan Jumlah Parameter Tidak Tetap

Acap kali kita membutuhkan jumlah parameter yang tidak tetap untuk sebuah method tertentu. Cara yang paling lazim dilakukan adalah dengan membuat parameter bertipe array atau collection. Cara ini tidak terlalu rumit, hanya saja kita terkadang harus membuat array atau collection padahal nilai parameteranya hanya satu. Contohnya seperti ini :

```
public void printAll(String[] array){
    for(String curr : array){
        System.out.println(curr);
    }
}
```

Kode untuk memanggil fungsi di atas :

```
String[] arr = new String[1];
arr[0] = "a";
printAll(arr);
```

Varargs mengijinkan method Java mempunyai jumlah argumen tidak tetap, artinya kita bisa memasukkan jumlah parameter semau kita pada kode pemanggilan method.

Mari kita implementasikan kembali method printAll di atas menggunakan fasilitas varargs:

```
public void printAll(String... array){
    for(String curr : array){
```

```
        System.out.println(curr);
    }
}
```

Sekarang kode untuk memanggil fungsi di atas menjadi lebih sederhana :

```
printAll("a");
printAll("a","b","c");
```

TypeSafe Enum

Tipe Data Enum

Java tidak mengenal tipe data enum seperti halnya pada C/C++, biasanya tipe data enum didefinisikan sebagai variabel konstant seperti di bawah ini :

```
public static final JENIS_KELAMIN_LAKI_LAKI = 0;
public static final JENIS_KELAMIN_PEREMPUAN = 1;
```

Pola ini mempunyai banyak kelemahan karena beberapa alasan, antara lain:

1. Tidak TypeSafe - karena tipenya hanya berupa int, kita bisa saja memasukkan nilai int apapun ketika kita memerlukan data jenis kelamin, atau menambahkan dua jenis kelamin, suatu operasi yang nyaris tidak logis.
2. Tidak mempunyai namespace - Kita harus secara manual memberikan awalan terhadap variabel enum agar tidak tertukar dengan variabel yang lainya.
3. Mudah Rusak - Misalnya suatu saat tanpa sengaja ada yang merubah nilai int dari variabel tersebut, maka integritas data dengan data yang ada di dalam database dengan sendirinya akan ikut berubah.
4. Nilai yang divetak tidak informatif - Setelah nilai Jenis kelamin dimasukkan ke dalam database, nilai yang dimasukkan hanyalah 0 atau 1. Informasi tentang Jenis kelaminnya tidak ada.

Java 5 mendukung TypeSafe enum langsung di sisi sintaksnya. Definisi enum di atas bisa dengan mudah diganti menjadi :

```
public enum JenisKelamin{LAKI_LAKI,PEREMPUAN};
```

Walaupun tampilan enum di atas mirip-mirip dengan enum di C/C++, enum di Java 5 mempunyai lebih banyak feature. Enum di atas adalah class java, yang bisa mempunyai method dan property. Enum juga merupakan implementasi dari class Object, yang Comparable dan Serializable.

Aplikasi berbasis database biasanya mengimplmentasikan enum Jenis kelamin dengan mempunyai dua jenis data, data pertama adalah simbol jenis kelamin, biasanya L dan P. Data kedua adalah nama lengkap dari jenis kelamin tersebut, biasanya digunakan sebagai pilihan dalam combo box. Kita bisa mendefinisikan ulang enum JenisKelamin menjadi seperti berikut ini :

```
public enum JenisKelamin {
    LAKI_LAKI("L","Laki-laki"),
    PEREMPUAN("P","Perempuan");

    private String symbol;
    private String name;
    JenisKelamin(String symbol, String name){
        this.symbol = symbol;
        this.name = name;
    }
    public String getName() {
        return name;
    }
    @Override
    public String toString() {
```

```
        return symbol;
    }
}
```

Enum juga bisa digunakan dalam clausa switch seperti di bawah ini :

```
switch(jk){
    case JenisKelamin.LAKI_LAKI:
        return "suaranya soprano";
    case JenisKelamin.PEREMPUAN:
        return "suaranya tenor";
}
```

Setiap Enum mempunyai nilai ordinal, nilai tersebut adalah nilai urut enum, urutan pertama mempunyai nilai 0 dan seterusnya.

Kita bisa mendapatkan semua nilai Enum secara programatik tanpa harus menyebutkan satu-satu, contohnya:

```
JenisKelamin[] jks = JenisKelamin.values();
for(JenisKelain jk : jks){
    System.out.println(jk);
}
```

Generics

Tipe Data di Dalam Collection

Ketika kita akan mengambil elemen dari dalam Collection, kita memerlukan proses casting ke tipe data sebenarnya. Proses casting ini diperlukan karena kita tidak bisa mendefinisikan tipe data apa yang ada di dalam collection, sehingga semua yang masuk ke dalam collection dianggap bertipe Object.

Masalah muncul kalau kita tidak tahu-menahu tentang apa isi dari collection tersebut, jika ternyata isi dalam collection tersebut beraneka ragam tipe data, kode kita akan mengalami error `ClassCastException`.

Compiler java tidak dapat mendeteksi potensi kesalahan ini pada saat compile. Berikut ini contohnya :

```
Collection lists = new ArraList();
lists.add("a");
lists.add(100);
Iterator iterator = lists.iterator();
for(;iterator.hasNext();){
    String current = (String) iterator.next();
}
```

Kode di atas akan menyebabkan error `ClassCastException`, karena dalam collection lists, kita memasukkan "a" yang bertipe String dan 100 yang bertipe Integer. Pada saat kita mengambil data dari lists dan melakukan cast ke tipe String terhadap nilai 100, error `ClassCastException` akan terjadi.

Generics dapat digunakan untuk mendefinisikan tipe data apa yang ada di dalam collection, kode di atas dapat dimodifikasi menjadi seperti di bawah ini:

```
Collection<String> lists = new ArraList<String>();
lists.add("a");
lists.add(100);
for(String s: lists){
    String current = iterator.next();
}
```

pada baris `lists.add(100);`, compiler akan memberikan keterangan error bahwa nilai yang dimasukkan ke dalam collection lists mempunyai tipe data selain String.

Pengecekan tipe data apa yang ada di dalam Collection sekarang terjadi pada saat kompilasi, bukan pada saat jalanya aplikasi, dengan begitu kesalahan bisa ditemukan lebih cepat.

Generics bisa digunakan untuk melakukan banyak hal, namun sebagai programmer kita tidak harus memikirkan secara detail tentang penggunaan generics, kita bertindak sebagai pemakai library, bukan library designer.

Annotations

Metadata

Banyak sekali API dalam java yang memerlukan konfigurasi yang terkadang diletakkan dalam file terpisah. Pola ini sangat memberatkan programmer karena ada tambahan file lagi yang harus dimantain. API lainnya mengharuskan kita meng-extend class tertentu atau membuat nama method dengan pola tertentu.

Annotation memberikan alternatif cara menambahkan informasi terhadap file java. Informasi tambahan ini disebut dengan metadata. Metadata digunakan oleh API atau IDE sebagai informasi tambahan yang digunakan sebagai patokan untuk memperlakukan file java tersebut dengan tindakan tertentu.

JUnit 3.x mengharuskan file yang akan dites memenuhi beberapa aturan, pertama harus meng-extend class TestCase, kemudian setiap method yang akan dites harus diawali dengan kata test, berikut ini contohnya:

```
public class PersonTest extends TestCase {
    public PersonTest(String testName) {
        super(testName);
    }
    public void testGetEmails() {
        fail("this test is not actually created");
    }
    public void testSetEmails() {
        fail("this test is not actually created");
    }
}
```

Dengan annotation kita bisa menyederhanakan struktur file java di atas, dan menghilangkan keharusan untuk melakukan extend terhadap file tertentu atau memberikan awalan terhadap method yang harus dites, berikut ini test yang sama untuk JUnit 4.x dengan annotation :

```
public class PersonTest {
    public PersonTest() {
    }
    @Test
    public void getEmails() {
        fail("this test is not actually created");
    }
    @Test
    public void setEmails() {
        fail("this test is not actually created");
    }
}
```

Terlihat bahwa kita menambahkan annotation @Test di atas method yang akan dites. Junit menggunakan informasi @Test ini sebagai penanda bahwa method yang akan dites.

Contoh annotation lain yang banyak digunakan adalah @Deprecated, jika annotation ini diletakkan di atas method, maka akan digunakan oleh java compiler dan IDE untuk memberitahukan kepada pemanggil dari method tersebut bahwa method yang dipanggil kemungkinan besar akan dihilangkan pada versi berikutnya. Hal ini membantu programmer

menghindari error di masa depan.

Menambahkan annotation `@Override` di atas method yang di-override juga merupakan praktek yang baik. Compiler akan memeriksa method yang ditandai dengan `@Override` apakah sudah dengan benar mengoverride method kepunyaan parentnya. Contohnya, misalkan kita akan berniat mengoverride method `equals` dari class `Object` seperti di bawah ini :

```
public boolean equals(){  
    return true;  
}
```

ternyata kode method `equals` di atas tidak memenuhi struktur method `equals` yang seharusnya mempunyai parameter `Object`. Java compiler tidak akan berkata apa-apa dan tidak akan memperingatkan kita bahwa ada kesalahan dalam proses override tersebut.

```
@Override  
public boolean equals(){  
    return true;  
}
```

Dengan menggunakan `@Override` kita memberitahukan java compiler bahwa kita berniat mengoverride method `equals`, dan jika terdapat kesalahan dalam proses overridenya, compiler akan memberikan peringatan "method does not override or implement a method from a supertype".

Masih banyak kasus lain dimana annotation akan memudahkan programmer mengimplentasikan aturan tertentu dari API yang digunakan. Misalnya hanya dengan menambahkan keterangan `@Stateless` terhadap class tertentu, maka kita sudah membuat `Stateless session bean`. `Hibernate` versi dahulu mewajibkan kita untuk membuat file `hbm.xml` untuk setiap class `Entity`, sekarang kita bisa meletakkan informasi mapping `hibernate` dalam file `java`-nya dengan menggunakan annotation.

Kesimpulan

Java 5 Language Enhancement ditujukan utamanya untuk menyederhanakan kode `java` tanpa menghilangkan kemudahannya untuk dibaca. Perubahan sintaks tersebut tidak serta merta merupakan perubahan signifikan terhadap sintaks `java` secara radikal, tetapi hanya perubahan kecil dalam compiler, dimana ada langkah tambahan untuk merubah sintaks bentuk baru ke dalam bentuk yang lama sebelum file `java` tersebut benar-benar dicompile menjadi file `class`.

Feature `generics` dan annotation ternyata membawa perubahan yang sangat signifikan terhadap banyak sekali API. Dengan menggunakan kedua feature ini API menjadi lebih mudah digunakan dan lebih sedikit file konfigurasi yang diperlukan untuk menjelaskan beberapa informasi dalam file `java`.

Pada akhirnya bahasa `java` menjadi lebih mudah digunakan dan lebih ringkas dibandingkan dengan versi sebelumnya.

BAGIAN 2

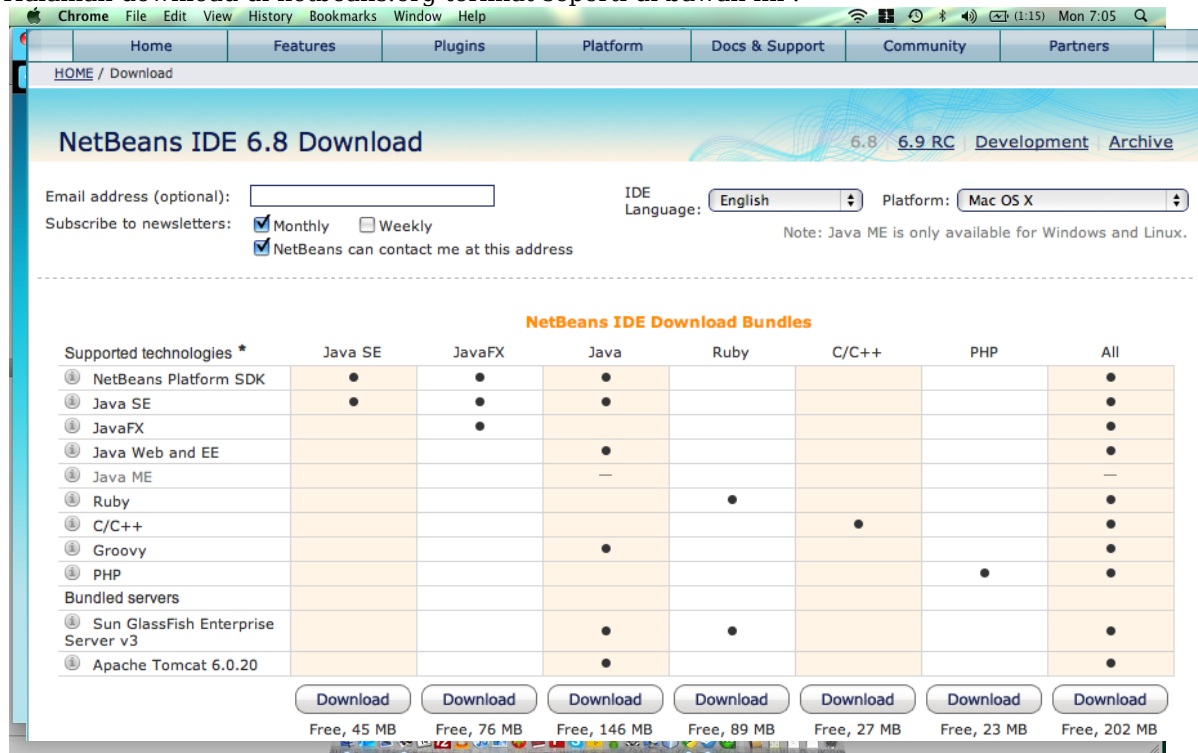
NETBEANS

Instalasi

Proses instalasi NetBeans sangat sederhana. Pertama download dahulu installer NetBeans dari website netbeans.org

<http://netbeans.org/downloads/index.html>

Halaman download di netbeans.org terlihat seperti di bawah ini :



Supported technologies *	Java SE	JavaFX	Java	Ruby	C/C++	PHP	All
NetBeans Platform SDK	•	•	•				•
Java SE	•	•	•				•
JavaFX		•					•
Java Web and EE			•				•
Java ME			—				—
Ruby				•			•
C/C++					•		•
Groovy			•				•
PHP						•	•
Bundled servers							
Sun GlassFish Enterprise Server v3			•	•			•
Apache Tomcat 6.0.20			•				•

Download buttons and sizes: Java SE (Free, 45 MB), JavaFX (Free, 76 MB), Java (Free, 146 MB), Ruby (Free, 89 MB), C/C++ (Free, 27 MB), PHP (Free, 23 MB), All (Free, 202 MB).

Pilih paket yang sesuai dengan kebutuhan anda, untuk tujuan buku ini cukup pilih paket java SE yang ukuran downloadnya hanya sekitar 45 MB.

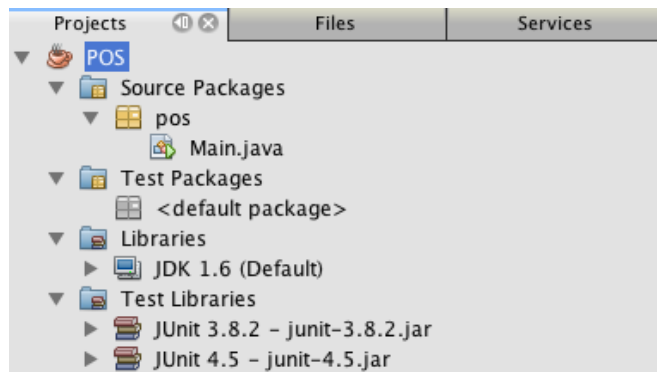
Setelah download selesai, double click file instalasi. Selanjutnya tinggal pilih next atau yes dan instalasi selesai.

Membuat Project

NetBeans akan menampilkan halaman utama ketika pertama kali dijalankan. Halaman utama NetBeans menampilkan link-link yang cukup bermanfaat, seperti blog dari NetBeans evangelist, tutorial dari netbeans.org hingga video demo feature-feature terbaru NetBeans.

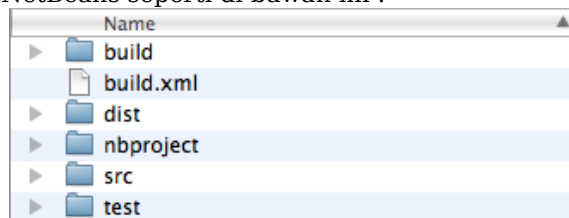
Untuk membuat project pilih menu File->New Project. Isikan nama project dan folder dimana project akan disimpan.

Setelah project selesai dibuat, tampilan di NetBeans seperti berikut ini :



Source Package menampilkan source code file java dan file-file konfigurasi yang nanti diperlukan. Test Package menampilkan semua file source code Junit test, Libraries menampilkan library-library yang diperlukan aplikasi, sedangkan Test Libraries menampilkan library yang diperlukan hanya untuk Junit test saja.

Kalau kita buka project NetBeans di atas di file explorer, akan terlihat struktur folder project NetBeans seperti di bawah ini :



Folder nbproject berisi file-file yang digenerate oleh NetBeans, sebaiknya file-file di dalam folder ini tidak diubah secara manual, biarkan saja seperti apa adanya. Folder build adalah dua folder yang dibuat NetBeans untuk menyimpan file hasil kompilasi java compiler, isinya berupa file .class. Sedangkan folder dist berisi file jar dari aplikasi yang kita buat beserta library yang diperlukan. Source code aplikasi akan diletakkan dalam folder src. Yang terakhir adalah folder test yang akan digunakan untuk menyimpan file test JUnit.

File build.xml adalah script ant yang digunakan untuk melakukan berbagai macam hal di NetBeans. Setiap kali kita melakukan aksi terhadap project, semisal menjalankan aplikasi, NetBeans akan menjalankan script dalam build.xml. Jika anda ingin belajar lebih lanjut tentang ant, skill yang amat sangat berguna, silahkan baca tutorial ant dari Endy Muhandin

<http://endy.artivisi.com/downloads/writings/Tutorial-Ant.pdf>

Kadang kala kita dihadapkan pada masalah dimana struktur project NetBeans yang rusak karena suatu hal. Jangan panik, cara pemecahannya sangatlah mudah. Pertama buat project NetBeans yang baru, kemudian cukup timpa folder src project baru dengan folder src project yang rusak, kemudian tambahkan library yang anda perlukan.

Menambahkan Library

Library pasti diperlukan dalam pembuatan aplikasi java. Semisal komponen Jcalendar untuk menampilkan tanggal yang tidak ada di dalam standard JRE, atau menambahkan library Spring dan Hibernate. Library Jcalendar adalah library UI yang bisa didrag and drop ke visual editor, sedangkan library Spring bukan UI library. Jcalendar dapat ditambahkan di palette NetBeans sehingga bisa didrag and drop ke visual editor.

Menambahkan library UI ke palette sangat mudah. Download component Jcalendar dari website berikut ini :

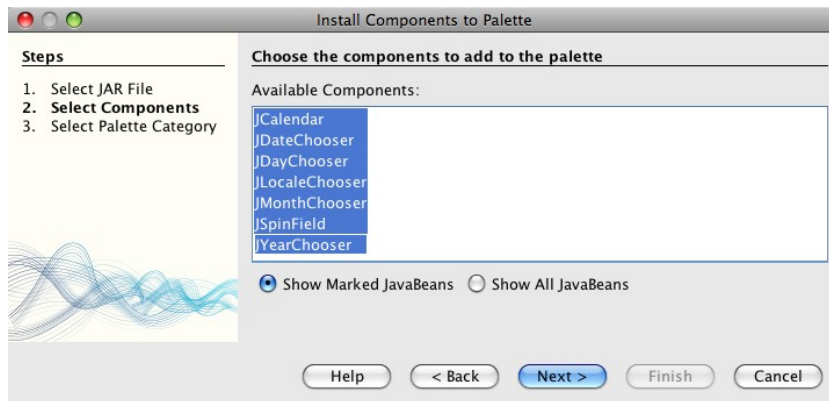
<http://www.toedter.com/download/jcalendar-1.3.3.zip>

Kemudian extract file zip, di dalam folder lib ada file jcalendar-1.3.3.jar. Kemudian lanjutkan langkah menambahkan Jcalendar ke palette dengan memilih menu berikut dari NetBeans menu bar :

Tools => Palette Manager => Swing/AWT Components

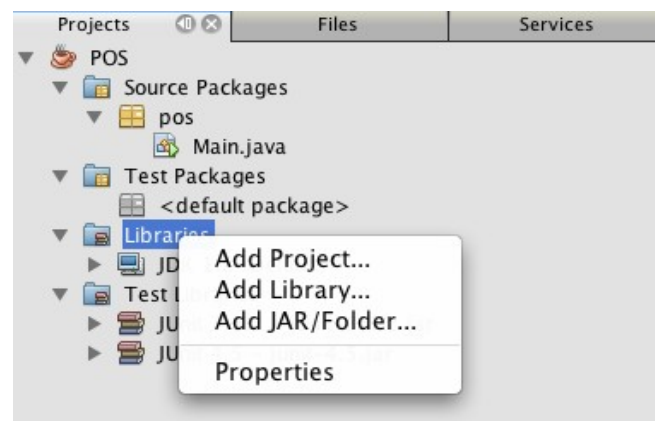
Menu ini digunakan untuk menampilkan Palette Manager. Sebelum memasukkan komponen JCalendar, sebaiknya buat dulu kategori baru untuk menampung komponen JCalendar. Tekan tombol "New Category" dan masukkan string "JCalendar" ke dalam input dialog.

Di jendela Palette Manager, tekan tombol "add from JAR", kemudian pilih file jcalendar-1.3.2.jar yang tadi telah disiapkan. Klik next. Setelah itu akan muncul dialog seperti di bawah ini.



Setelah proses penambahan Jcalendar selesai, anda bisa melihat komponen-komponen Jcalendar dalam palette, di bab-bab berikutnya akan dibahas bagaimana menggunakan library ini untuk mengambil input tanggal dari user.

Menambahkan library non UI ke dalam project langkahnya lebih mudah. Dari project tab, klik kanan node Libraries, kemudian akan tampil menu seperti di bawah ini :



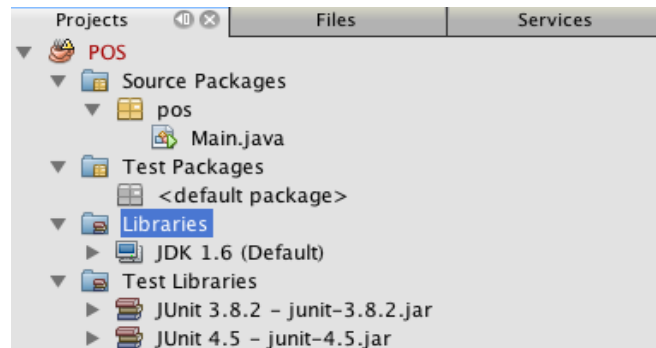
Untuk menambahkan library yang disediakan oleh NetBeans pilih Add Library, kemudian pilih library yang akan ditambahkan, misalnya Spring atau Hibernate.

Add Project digunakan untuk menambahkan project lain sebagai library. Add Jar/folder digunakan untuk menambahkan library jar yang tidak disediakan oleh NetBeans, misalkan library jcalendar. Properties digunakan untuk menampilkan semua library yang telah ditambahkan, bisa juga digunakan untuk menambahkan atau menghapus semua jenis library.

Ada kalanya library yang diperlukan oleh project NetBeans tidak dapat ditemukan sehingga membuat project menjadi tidak valid, seperti tampilan di gambar berikut ini. Hal ini terjadi kalau tidak sengaja menghapus file jar atau mendownload project NetBeans dari internet

sehingga jar yang dibutuhkan tidak ditemukan.

Cara mengatasinya tidak susah, buka menu properties di atas, kemudian remove jar yang hilang, ditandai dengan tanda seru, kemudian tambahkan jar yang sesuai.

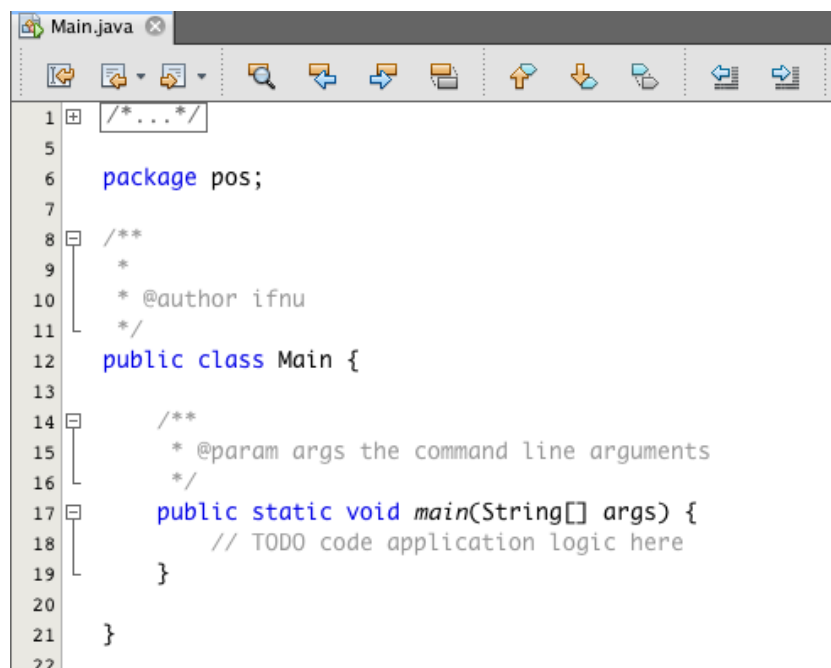


Praktek yang paling baik dalam mengelola library adalah dengan meletakkan semua jar yang dibutuhkan didalam folder project. Buat sebuah folder di dalam project NetBeans dan beri nama lib, kemudian letakkan semua jar external yang dibutuhak di dalam folder ini. NetBeans mampu mengenali folder di dalam project NetBeans dengan relative path, sehingga kalau project NetBeans dipindah-pindah ke folder berbeda atau dicopy dari satu komputer ke komputer lain atau dicommit ke code repository seperti subversion, struktur project tidak rusak dan tidak terjadi masalah library hilang seperti gambar di atas.

Menggunakan Editor

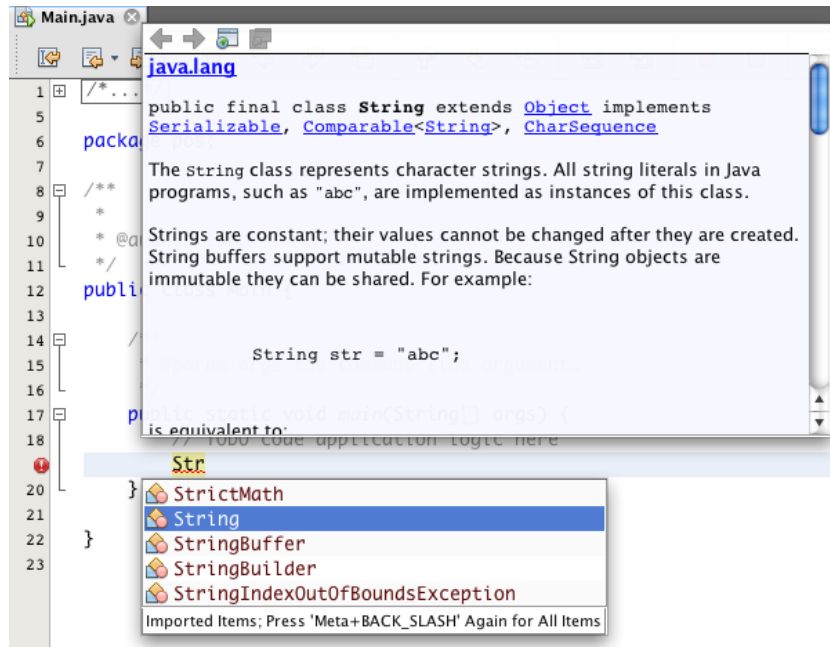
NetBeans mempunyai editor yang sangat bagus, hal ini penting karena sebagian besar waktu coding kita habiskan menggunakan editor. Editor NetBeans mempunyai feature yang sangat lengkap sehingga sangat membantu programmer meningkatkan produktifitasnya.

Tampilan editor NetBeans seperti di bawah ini :



Bagian atas adalah nama file, kemudian di bawahnya ada editor toolbar yang bisa digunakan untuk melakukan berbagai macam aksi terhadap jendela editor.

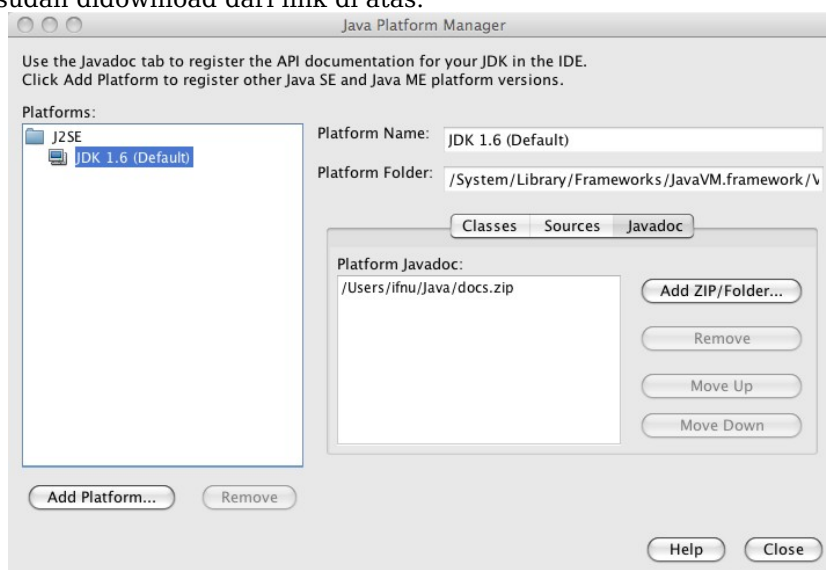
Feature pertama yang paling dicari user adalah code completion atau autocomplete. Feature ini dapat dipanggil dengan short cut ALT+SPACE. Ketik beberapa huruf pertama dari apapun yang anda ingin ketik, kemudian tekan ALT+SPACE, misalnya ketik Str kemudian tekan ALT+SPACE. Menu autocomplete akan menampilkan kemungkinan yang dimaksud dengan Str, tampilanya seperti di bawah ini :



Di bagian bawah terdapat daftar class yang diawali dengan Str, kemudian dibagian atas terdapat javadoc dari class yang sedang terpilih. Hal ini sangat bermanfaat jika sedang bekerja dengan library yang masih belum familiar. Jika NetBeans anda belum menampilkan javadoc, anda bisa mendownload javadoc dari link berikut ini :

<http://java.sun.com/javase/downloads/index.jsp#docs>

Kemudian menambahkan ke setting NetBeans secara manual. Buka menu Tools => Java Platform, akan tampil dialog seperti di bawah ini. Pindah ke tab Java Doc kemudian tambahkan file zip yang sudah didownload dari link di atas.

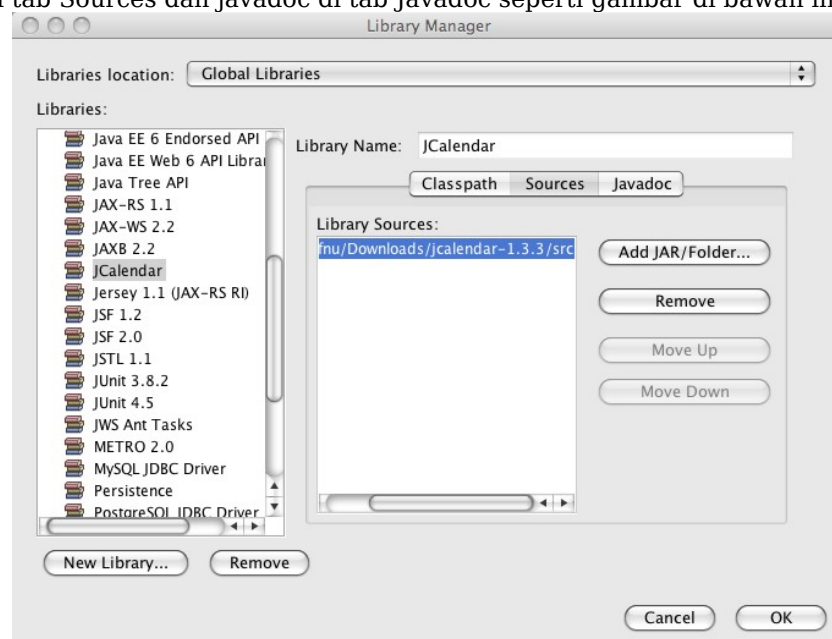


Feature berikutnya yang sangat berguna adalah fasilitas navigasi di dalam source code. Feature ini sangat berguna untuk menuju ke deklarasi dari class ataupun variabel. Arahkan pointer mouse ke atas deklarasi class atau variabel kemudian tekan CTRL + Click secara bersamaan. Jika source code dari sebuah class tersedia, maka class yang dimaksud akan ditampilkan. Misalnya anda ingin melihat source code class String, tetapi ternyata source code tersebut tidak tersedia di instalasi netbeans. Anda bisa menambahkan secara manual dengan mendownload source code JDK dari link berikut ini :

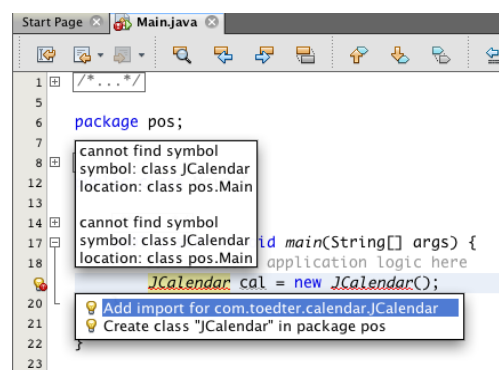
<http://download.java.net/openjdk/jdk6/>

Extract file tar.gz kemudian buka menu Tools => Java Platform yang akan membuka dialog seperti di atas. Buka tab source code dan tambahkan folder dimana anda mengekstrack file hasil download link di atas berada.

Jika ingin menambahkan source code ke jar selain JDK langkah-langkahnya sedikit lebih panjang. Langkah pertama adalah membuat library, pilih menu Tools -> Library kemudian tekan tombol New Library, beri nama librarynya, misalkan Jcalendar. Kemudian tambahkan jar di tab Classpath, source code di tab Sources dan javadoc di tab Javadoc seperti gambar di bawah ini

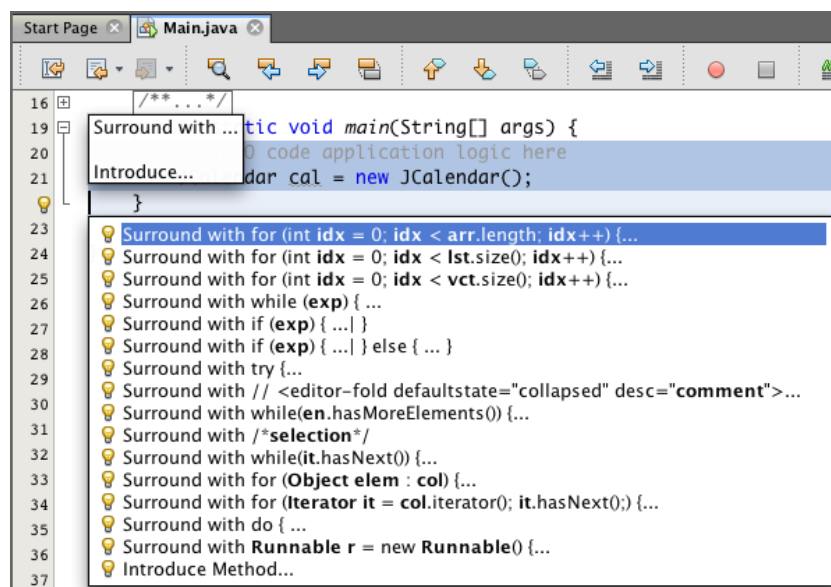


Feature berikutnya yang cukup membantu dalam menulis kode adalah autofix. Perhatikan gambar di bawah ini, setiap kali ada error dalam kode yang sedang ditulis di editor, akan ada icon berwarna merah dengan tanda seru di sebelah kiri nomor baris di mana error terjadi. Jika NetBeans tahu bagaimana memperbaiki kode yang error ini maka akan ada icon tambahan berupa lampu bolham warna kuning. Klik lampu bolham akan keluar menu untuk memilih alternatif memperbaiki error, misalnya menambahkan import untuk class atau membuat statement try/catch.

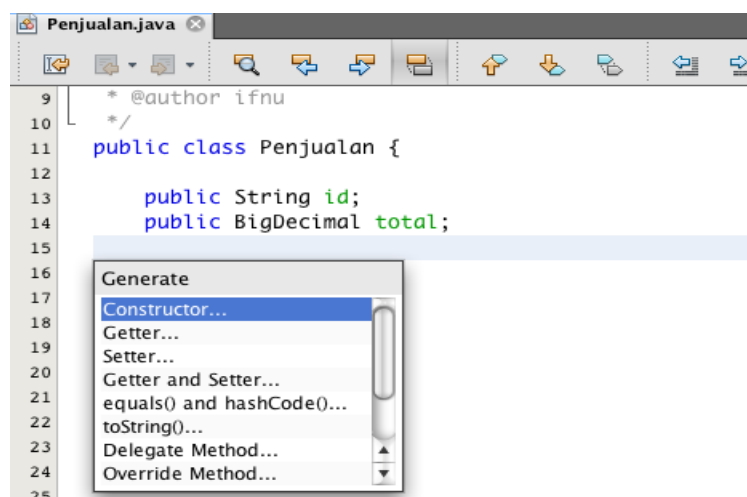


Sebagai contoh, lihat gambar di atas, ada statement untuk membuat instance dari JCalendar, ketika lampu bolham diklik akan ada menu untuk mengimport class Jcalendar, ketika menu tersebut diklik maka akan ada tambahan statement import class JCalendar. Shortcut untuk memanggil menu autofix adalah ALT + ENTER di baris yang sama dimana icon lampu bolham muncul.

Icon lampu bolham juga bisa dimunculkan walau tanpa error, caranya dengan mengeblok beberapa baris kode, lampu bolham kuning akan keluar di baris terakhir kode yang diblok. Klik atau tekan ALT + ENTER di baris munculnya bolham kuning, maka akan tampil berbagai macam alternatif kode yang akan melingkupi baris yang diblok. Misalnya pilih menu paling atas, maka statement for akan diletakkan mulai dari baris 20 hingga setelah baris 21. Silahkan coba-coba menggunakan pilihan yang ada untuk mengetahui kode apa yang akan ditambahkan ketika menu-menu tersebut dipilih.

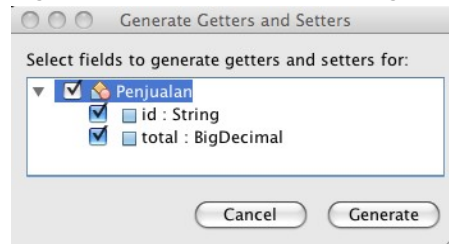


NetBeans Editor menyediakan feature untuk mengenerate kode-kode yang sering diperlukan, tekan ALT + INSERT atau klik kanan di Editor kemudian pilih menu Insert Code. Menu akan muncul dengan beberapa pilihan seperti di gambar di bawah ini.



Mari kita bahas satu persatu menu yang ada di atas. Getter and Setter digunakan untuk

mengenerate method yang akan mengencapsulasi property id dan total. Tekan menunya kemudian akan muncul dialog seperti gambar di bawah ini, pilih checkboxnya dan tekan generate. Method getId, setId, getTotal dan setTotal akan digenerate oleh netbeans.



Constructor digunakan untuk mengenerate constructor dari class Penjualan. Misalnya kita pilih 2 buah property id dan total sebagai parameter constructor, maka akan digenerate kode seperti di bawah ini :

```
public Penjualan(String id, BigDecimal total) {  
    this.id = id;  
    this.total = total;  
}
```

equals dan hashCode digunakan untuk mengenerate method yang akan mengoverride kedua method tersebut dari class Object. Kegunaan fungsi equals adalah membandingkan antara dua buah object apakah secara "logic" sama dengan object lainnya. Untuk menentukan dua buah object dari class Penjualan apakah sama atau tidak digunakan property id, asal dua buah object tersebut mempunyai id yang sama maka dianggap sama. Konsep equals dan hashCode sudah dijelaskan di bagian Java Fundamental, jadi tidak dibahas lagi secara mendetail di sini. Pembahasan hanya terbatas bagaimana caranya membuat method equals dan hashCode yang baik dan bug free. Menulis sendiri equals maupun hashCode tidak dilarang, tapi sebaiknya dihindari, gunakan fasilitas generate equals dan hashCode dari IDE. Hasil generate equals dan hashCode dari NetBeans sebagai berikut :

```
@Override  
public boolean equals(Object obj) {  
    if (obj == null) {  
        return false;  
    }  
    if (getClass() != obj.getClass()) {  
        return false;  
    }  
    final Penjualan other = (Penjualan) obj;  
    if ((this.id == null) ? (other.id != null) : !this.id.equals(other.id)) {  
        return false;  
    }  
    return true;  
}  
  
@Override  
public int hashCode() {  
    int hash = 3;  
    hash = 23 * hash + (this.id != null ? this.id.hashCode() : 0);  
    return hash;  
}
```

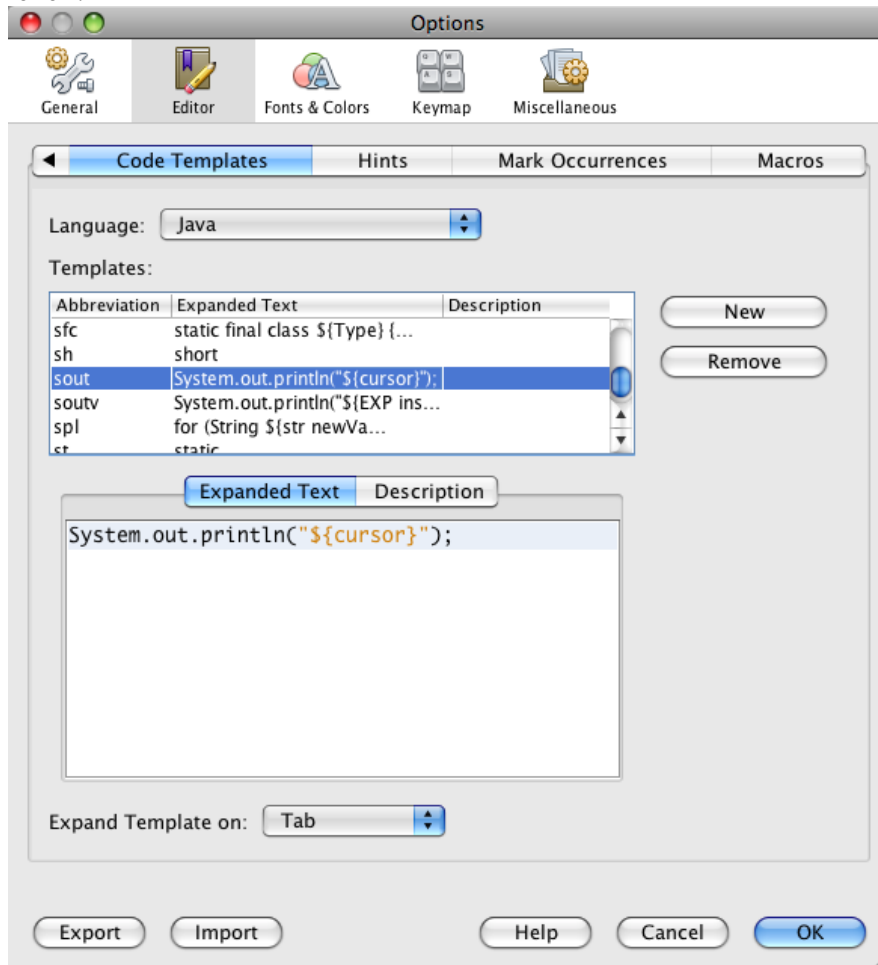
Kalau anda melihat di NetBeans yang anda gunakan tidak ada generate toString, tidak perlu panik, karena memang feature ini baru ada di NetBeans 6.9. Fungsinya untuk mengenerate method toString agar lebih mudah untuk mencetak nilai suatu object di console atau untuk tujuan logging. Kode hasil generate toString seperti berikut :

```
@Override  
public String toString() {
```

```
return "Penjualan{" + "id=" + id + "total=" + total + '}';
}
```

Feature lain yang cukup membantu mempercepat coding adalah code template. Lihat gambar di bawah ini, gambar tersebut adalah jendela yang menampilkan semua code template yang tersedia secara default dari NetBeans. Cara menampilkan jendela tersebut dengan memilih menu Tools -> Option. Code template sangat membantu mempercepat code dengan membuat shortcut dari kode yang ingin diketik. Misalnya ketik sout kemudian tekan tab, maka akan muncul kode System.out.println(), atau ketik psvm kemudian tab, akan muncul kode deklarasi method main.

Anda bisa menambahkan code template untuk generate kode yang anda inginkan atau sering digunakan.

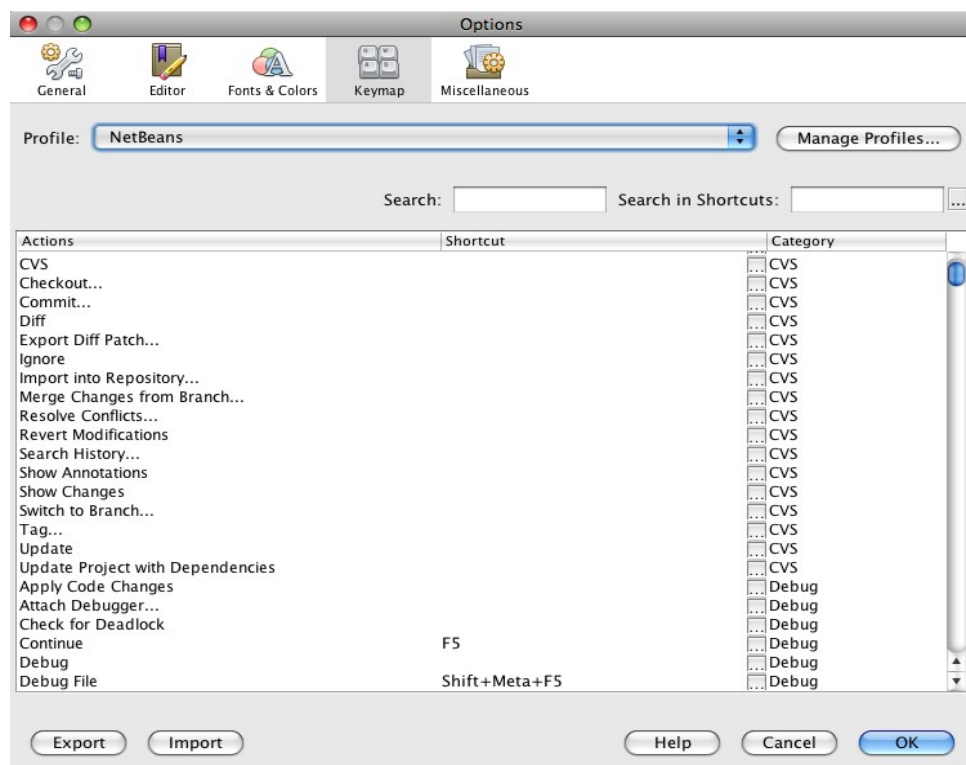


Berikut ini daftar feature dan shortcut lain di editor netbeans yang sering digunakan

- Menampilkan nomor baris View => Show Line Numbers
- CTRL + e menghapus baris yang sedang aktif atau semua baris yang sedang dipilih
- CTRL + k menebak kata yang pernah muncul berikutnya. Berbeda dengan code completion (CTRL + SPACE) karena menebak kata ini tidak menampilkan context menu tapi langsung menampilkan kata yang ditebak, sehingga lebih cepat. Kalau kata yang ditebak salah, tekan CTRL + k sekali lagi untuk menampilkan tebakan berikutnya.
- CTRL + / membuat baris yang sedang aktif atau baris yang sedang dipilih menjadi tidak aktif (komentar)
- CTRL + TAB berpindah dari satu file ke file yang lain

- CTRL + i memperbaiki import dengan mengimport class yang belum diimport dan menghapus import yang tidak digunakan
- CTRL + o mencari class yang ada dalam project
- CTRL + f mencari string dalam class / file yang sedang aktif di editor
- CTRL + SHIFT + f mencari string dalam semua file yang ada dalam project
- CTRL + b membuka deklarasi variabel atau class
- CTRL + SHIFT + Panah ke Bawah mengkopi satu baris atau satu blok baris ke bawah tanpa harus memencet CTRL + c trus CTRL + v
- ALT + SHIFT + Panah ke Bawah memindahkan satu baris atau satu blok baris ke bawah
- CTRL + o melakukan pencarian terhadap nama class dan membukanya di editor. Bisa menggunakan huruf besar dari setiap kata class untuk pencarian. Misalnya nama classnya FileInputStream cukup mengetikkan FIP di dialog pencarian.
- CTRL + SHIFT + o Mencari file di dalam project. Berbeda dengan CTRL + o yang hanya mencari class saja, perintah ini akan menampilkan semua file baik file java maupun file-file yang lain.
- CTRL + q menuju ke baris terakhir yang diedit. Tekan CTRL + q beberapa kali untuk menuju ke baris-baris yang diedit sebelumnya
- CTRL + 1 berpindah ke tab project, CTRL + 0 berpindah ke tab editor, CTRL + 4 berpindah ke tab output.

Masih banyak lagi feature Editor NetBeans yang belum dibahas di sini. Shortcut di atas hanya secuil feature yang sering digunakan. Kalau ingin melihat secara lengkap shortcut di dalam editor silahkan buka menu Tools => Option kemudian buka tab Keymap. Bahkan jika anda terbiasa dengan IDE lain semisal eclipse, anda bisa mengganti profile shortcut menjadi eclipse. Sekarang anda bisa menggunakan NetBeans dengan shortcut Eclipse.



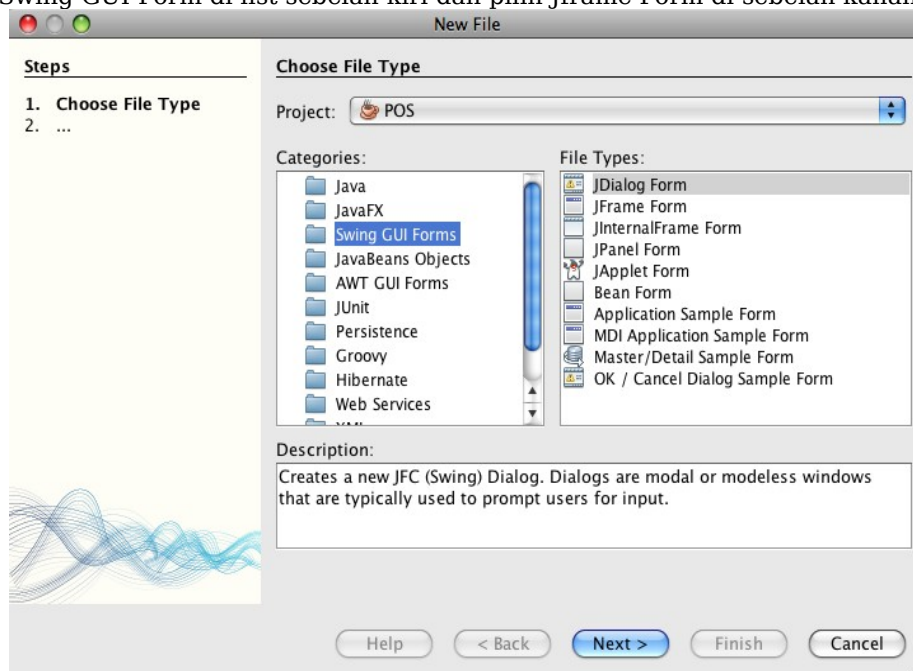
Menggunakan Visual Designer

NetBeans dilengkapi dengan Visual editor untuk AWT/Swing, anda bisa membuat UI aplikasi desktop dengan melakukan click and drag. Modul visual editor NetBeans disebut dengan Matisse. Modul ini dilengkapi sebuah sistem layout yang disebut Free Form Layout. Sistem layout ini khusus dibuat untuk ditulis menggunakan IDE, tidak dikode manual dengan tangan.

Beberapa netbeaners mengaku pada awalnya kesulitan menggunakan visual designer NetBeans karena terbiasa dengan Visual Basic atau Delphi. Perbedaan utamanya terletak di sistem layouting di NetBeans yang lebih fleksibel. Kalau sebuah panel diubah ukurannya maka anggota component di dalam panel akan ikut di-arrange sehingga tampilanya menyesuaikan dengan lebar / tinggi panel. Feature ini tidak ada di VB yang menggunakan null layout, berapapun ukuran dari panelnya component di dalamnya tidak ikut di-arrange, alias di situ situ saja letaknya. Karena sistem layout inilah visual editor NetBeans terasa sedikit lebih susah penggunaannya, karena componen akan di-snap agar bisa saling terkait dan saling menyesuaikan kalau ukuran panelnya dirubah. Kalau anda tidak suka dengan keadaan ini, anda bisa merubah layout panel ke null layout dan anda akan mendapatkan "rasa" yang sama dengan VB editor.

Membuat Visual Component

Langkah pertama menggunakan visual designer adalah membuat visual component yang merupakan child dari Container, misalnya JFrame. Klik kanan di project kemudian pilih new file dan pilih Swing GUI Form di list sebelah kiri dan pilih JFrame Form di sebelah kanan .



Klik tombol next dan beri nama class kemudian klik finish. Anda akan dibawa ke jendela visual designer NetBeans. Tampilanya seperti gambar di bawah ini. Sebelah kiri adalah tab project, sebelah tengah tab designer. Sebelah kanan atas adalah pallete dan sebelah kanan bawah adalah properties. Sebelah kiri bawah ada satu tab namanya inspector, kalau tab ini belum terlihat pilih menu windows => Navigator => Inspector untuk menampilkanya.

Berpindah dari visual editor dan editor bisa dilakukan dengan mengklik tombol source atau tombol design di bagian atas editor. Di bagian yang sama pula terdapat icon dengan lambang berwarna hijau yang digunakan untuk menampilkan preview. Sebelah kanan icon preview terdapat tombol-tombol untuk mengatur aligment component, apakah rata kiri, rata kanan, atas bawah atau di tengah-tengah.

Bekerja dengan jendela pallet

Jendela pallet menampilkan komponen-komponen yang bisa digunakan dalam aplikasi java desktop. Pallet dapat ditambah dan dihapus, di bagian sebelumnya sudah dibahas bagaimana menambahkan component baru ke dalam pallet.

Bekerja dengan jendela properties

Jendela Properties terdiri dari empat bagian, properties, binding, events dan code. Bagian properties berisi property dari component yang sedang dipilih di visual editor. Bagian binding memperlihatkan binding antara satu component dengan component lain, proses binding menggunakan framework BeansBinding. Bagian event memperlihatkan event apa saja yang diimplementasikan untuk component tersebut, biasanya bagian ini digunakan untuk menghapus event yang tidak digunakan, kenapa? Karena kode yang digenerate netbeans untuk handle event tidak dapat diedit/dihapus dari editor.

Bagian code digunakan untuk menyelipkan kode di area yang tidak bisa diedit dari tab editor. Seperti kode yang ingin diselipkan setelah komponen diinstansiasi, karena proses instansiasi component ada dalam method initComponents yang tidak dapat diedit dari editor, maka kode yang ingin diselipkan bisa dilalukan dari bagian code ini.

Bekerja dengan jendela Inspector

Jendela inspector termasuk dalam jendela yang gunanya untuk bernavigasi. Jendela inspector digunakan untuk bernavigasi dari satu component ke component yang lain, terkadang ada kesulitan untuk memilih component, misalnya karena ada di dalam tab component yang berlapis, sehingga untuk memilih component tersebut memerlukan sedikit waktu lebih banyak.

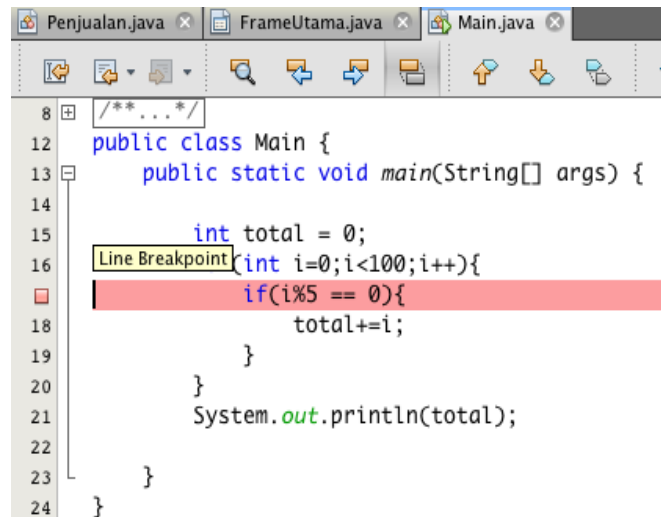
Debugging

Feature debugging di sebuah IDE sangat penting untuk membantu developer mencari masalah atau mengetahui keadaan internal dari aplikasi ketika sedang dijalankan. Dengan fasilitas debugging, aplikasi dapat di pause di tempat tertentu kemudian diperiksa keadaan internalnya, misalnya nilai dari suatu variabel, atau flow eksekusi.

Mari kita praktekkan proses debugging di NetBeans. Buat sebuah class, namakan Main.java, kemudian tambahkan kode berikut ini di dalam class Main

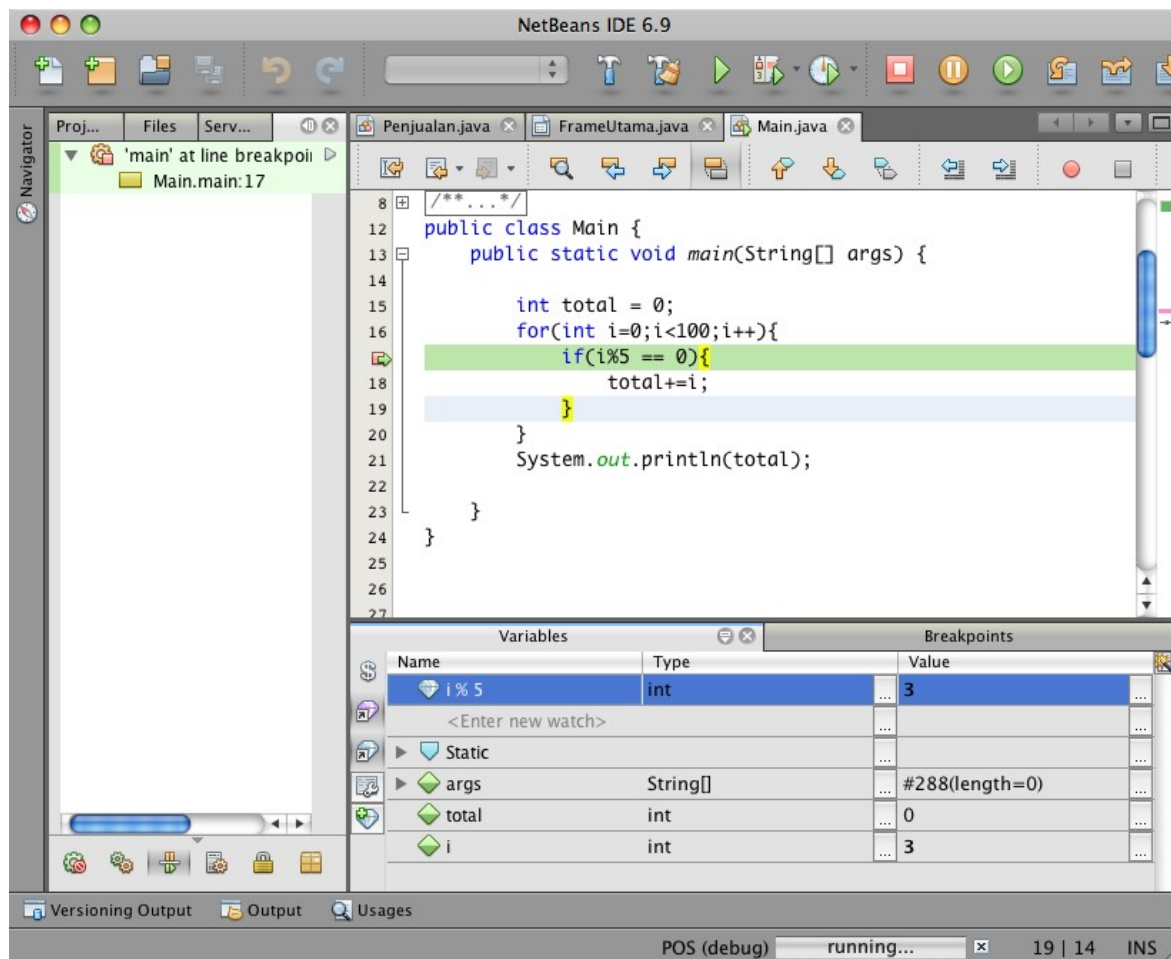
```
public class Main {
    public static void main(String[] args) {
        int total = 0;
        for(int i=0;i<100;i++){
            if(i%5 == 0){
                total+=i;
            }
        }
        System.out.println(total);
    }
}
```

Break point adalah sebuah tempat di dalam kode program dimana proses debugging akan berhenti. Cara membuat break point sangat gampang, klik bagian kiri editor dimana ada tampilan nomor baris, sehingga muncul kotak berwarna merah seperti gambar di bawah ini:



Setelah breakpoint berhasil dibuat, klik kanan di editor kemudian pilih menu Debug File. Eksekusi kode akan berhenti di tempat breakpoint berada dan ditandai dengan panah berwarna hijau seperti di gambar di bawah ini. Di bagian bawah NetBeans terdapat panel variables, di panel ini terdapat semua variables yang terlihat dari break point, ada variabel `i`, `total` dan `args`. Kolom di sebelahnya memperlihatkan nilai variabel pada waktu program berhenti di break point. Untuk melihat jalannya aplikasi baris per baris, tekan F8 berkali-kali sambil amati perubahan nilai variabel.

Watch adalah sebuah statement yang akan dievaluasi dalam proses debugging. Watch bisa mengevaluasi statement, misalnya dalam setiap eksekusi kode, kita ingin melihat apa hasil dari statement : `i % 5`. Caranya klik kanan di editor dan pilih new watch kemudian ketikkan statement `i % 5` di dialog yang muncul. Watch akan ditampilkan di panel yang sama dengan variabel. Watch ditandai dengan icon berlian dan variabel ditandai dengan icon belah ketupat berwarna hijau.



Di sebelah kiri terdapat panel debugging yang memperlihatkan stack trace, atau urutan pemanggilan fungsi dari Main class hingga breakpoint. Contoh di atas kita bisa lihat ada icon kotak warna kuning dengan tulisan `Main.main`, artinya method `main` pada class `Main` sedang dieksekusi pada saat eksekusi kode berhenti di breakpoint. Kalau aplikasinya cukup kompleks dan mempunyai banyak class, panel debugger akan memperlihatkan stack trace yang lengkap dari pertama kali kode dieksekusi hingga berhenti di breakpoint. Sangat berguna untuk urutan eksekusi kode dalam stack trace, dengan informasi ini kita bisa mengetahui bagaimana kode terangkai dan bagaimana kaitan antar class pada waktu eksekusi. Class apa memanggil class apa dan method apa yang dieksekusi.

F8 adalah shortcut untuk step over, artinya eksekusi satu baris ini dilakukan semua kemudian melompat ke baris berikutnya. Kadangkala dalam satu baris eksekusi terdapat beberapa method yang dieksekusi dalam satu rangkaian. Kalau ingin melihat eksekusi kode ke dalam method tersebut tekan F7 (step into), anda akan dibawa ke method yang sedang dieksekusi. Kalau mau keluar dari method yang sedang dieksekusi ke method pemanggilnya tekan CTRL + F7 (step out).

BAGIAN 3

AKSES DATA KE DATABASE

Akses Database dengan JDBC

Mengenal JDBC

Java Database Connectivity adalah API yang digunakan Java untuk melakukan koneksi dengan aplikasi lain atau dengan berbagai macam database. JDBC memungkinkan kita untuk membuat aplikasi Java yang melakukan tiga hal: konek ke sumber data, mengirimkan query dan statement ke database, menerima dan mengolah resultset yang diperoleh dari database.

JDBC mempunyai empat komponen :

1. JDBC API

JDBC API menyediakan metode akses yang sederhana ke sumber data relational (RDBMS) menggunakan pemrograman Java. dengan menggunakan JDBC API, kita bisa membuat program yang dapat mengeksekusi SQL, menerima hasil ResultSet, dan mengubah data dalam database. JDBC API juga mempunyai kemampuan untuk berinteraksi dengan lingkungan terdistribusi dari jenis sumber data yang berbeda-beda.

JDBC API adalah bagian dari Java Platform yang disertakan dalam library JDK maupun JRE. JDBC API sekarang ini sudah mencapai versi 4.0 yang disertakan dalam JDK 6.0. JDBC API 4.0 dibagi dalam dua package yaitu : `java.sql` dan `javax.sql`.

2. JDBC Driver Manager

Class `DriverManager` dari JDBC bertugas untuk mendefinisikan object-object yang dapat digunakan untuk melakukan koneksi ke sebuah sumber data. Secara tradisional `DriverManager` telah menjadi tulang punggung arsitektur JDBC.

3. JDBC Test Suite

JDBC Test Suite membantu kita untuk mencari driver mana yang cocok digunakan untuk melakukan sebuah koneksi ke sumber data tertentu. Tes yang dilakukan tidak memerlukan resource besar ataupun tes yang komprehensif, namun cukup tes-tes sederhana yang memastikan fitur-fitur penting JDBC dapat berjalan dengan lancar.

4. JDBC-ODBC Bridge

Bridge ini menyediakan fasilitas JDBC untuk melakukan koneksi ke sumber data menggunakan ODBC (Open DataBase Connectivity) driver. Sebagai catatan, anda perlu meload driver ODBC di setiap komputer client untuk dapat menggunakan bridge ini. Sebagai konsekuensinya, cara ini hanya cocok dilakukan di lingkungan intranet dimana isu instalasi tidak menjadi masalah.

Dengan keempat komponen yang dipunyainya, JDBC menjadi tools yang dapat diandalkan untuk melakukan koneksi, mengambil data dan merubah data dari berbagai macam sumber data. Modul ini hanya akan membahas dua komponen pertama dari keempat komponen yang dimiliki oleh JDBC, yaitu JDBC API dan `DriverManager`. Sumber data yang digunakan adalah Relational Database.

Database Driver

JDBC memerlukan database driver untuk melakukan koneksi ke suatu sumber data. Database driver ini bersifat spesifik untuk setiap jenis sumber data. Database driver biasanya dibuat oleh pihak pembuat sumber datanya, namun tidak jarang juga komunitas atau pihak ketiga menyediakan database driver untuk sebuah sumber data tertentu.

Perlu dipahami sekali lagi bahwa database driver bersifat spesifik untuk setiap jenis sumber data. Misalnya, Database Driver `MySQL` hanya bisa digunakan untuk melakukan koneksi ke database `MySQL` dan begitu juga database driver untuk `Postgre SQL` juga hanya bisa digunakan untuk melakukan koneksi ke database `Postgre SQL`.

Database driver untuk setiap DBMS pada umumnya dapat didownload dari website pembuat

DBMS tersebut. Beberapa vendor DBMS menyebut Database driver ini dengan sebutan Java Connector (J/Connector). Database driver biasanya dibungkus dalam file yang berekstensi jar. Setiap database driver harus mengimplement interface java.sql.Driver.

Membuat Koneksi

Melakukan koneksi ke database melibatkan dua langkah: Meload driver dan membuat koneksi itu sendiri. Cara meload driver sangat mudah, pertama letakkan file jar database driver ke dalam classpath. Kemudian load driver dengan menambahkan kode berikut ini:

```
Class.forName("com.mysql.jdbc.Driver");
```

Nama class database driver untuk setiap DBMS berbeda, anda bisa menemukan nama class tersebut dalam dokumentasi driver database yang anda gunakan. Dalam contoh ini, nama class database driver dari MySql adalah com.mysql.jdbc.Driver.

Memanggil method Class.forName secara otomatis membuat instance dari database driver, class DriverManager secara otomatis juga dipanggil untuk mengelola class database driver ini. Jadi anda tidak perlu menggunakan statement new untuk membuat instance dari class database driver tersebut.

Langkah berikutnya adalah membuat koneksi ke database menggunakan database driver yang sudah diload tadi. Class DriverManager bekerja sama dengan interface Driver untuk mengelola driver-driver yang diload oleh aplikasi, jadi dalam satu sesi anda bisa meload beberapa database driver yang berbeda.

Ketika kita benar-benar melakukan koneksi, JDBC Test Suite akan melakukan serangkaian tes untuk menentukan driver mana yang akan digunakan. Parameter yang digunakan untuk menentukan driver yang sesuai adalah URL. Aplikasi yang akan melakukan koneksi ke database menyediakan URL pengenalan dari server database tersebut. Sebagai contoh adalah URL yang digunakan untuk melakukan koneksi ke MySql :

```
jdbc:mysql://[host]:[port]/[schema]
```

contoh konkritnya :

```
jdbc:mysql://localhost:3306/latihan
```

Setiap vendor DBMS akan menyertakan cara untuk menentukan URL ini di dalam dokumentasi. Anda tinggal membaca dokumentasi tersebut tanpa harus khawatir tidak menemukan informasi yang anda perlukan.

Method DriverManager.getConnection bertugas untuk membuat koneksi:

```
Connection conn =  
    DriverManager.getConnection(  
        "jdbc:mysql://localhost:3306/latihan");
```

Dalam kebanyakan kasus anda juga harus memasukkan parameter username dan password untuk dapat melakukan koneksi ke dalam database. Method getConnection menerima Username sebagai parameter kedua dan password sebagai parameter ketiga, sehingga kode di atas dapat dirubah menjadi :

```
Connection conn =  
    DriverManager.getConnection(  
        "jdbc:mysql://localhost:3306/latihan",  
        "root", "");
```

Jika salah satu dari driver yang diload berhasil digunakan untuk melakukan koneksi dengan URL tersebut, maka koneksi ke database berhasil dilaksanakan. Class Connection akan memegang informasi koneksi ke database yang didefinisikan oleh URL tersebut.

Setelah sukses melakukan koneksi ke database, kita dapat mengambil data dari database menggunakan perintah query ataupun melakukan perubahan terhadap database. bagian berikut ini akan menerangkan bagaimana cara mengambil dan memanipulasi data dari database.

Menyiapkan Table

Dalam contoh berikutnya akan digunakan table T_PERSON, table ini terdiri dari tiga buah kolom : id, name dan password. Id adalah primary key dengan tipe integer dan nilainya increment berdasarkan urutan insert. Primary key dengan tipe integer sangat bagus untuk performance karena mudah diindex dan index bisa sangat efisien, selain itu id akan digunakan sebagai foreign key di table-table yang lain, sehingga sangat penting untuk menggunakan tipe data integer karena database bisa sangat efisien membandingkan integer dalam proses join. Kolom name harus unique tapi bukan primary key karena ukurannya yang cukup besar. Kalau name ini sering digunakan dalam where statement sebaiknya dibuatkan index untuk mempercepat proses query.

Dengan menggunakan mysql, berikut ini DDL dari table T_PERSON :

```
create table T_PERSON (
    id integer auto_increment primary key,
    `name` varchar(100) unique not null,
    password varchar(200) not null
) engine=InnoDB;
```

Perhatikan bagian akhir dari DDL di atas, ada attribute engine=InnoDB yang memerintahkan mysql menggunakan InnoDB database engine. InnoDB adalah database engine di mysql yang mendukung transaction (commit dan rollback), constraint foreign key dan constraint lainnya. Sedangkan MyISAM tidak mendukung transaction dan constraint. Dari sisi performance MyISAM masih lebih cepat dibanding InnoDB, jadi gunakan kedua engine ini pada kasus yang tepat. Jika diperlukan konsistensi data maka gunakan InnoDB, jika kecepatan adalah segalanya maka MyISAM lebih tepat digunakan.

Mengambil dan Memanipulasi Data dari Database

Proses pengambilan data dari database memerlukan suatu class untuk menampung data yang berhasil diambil, class tersebut harus mengimplement interface ResultSet.

Object yang bertipe ResultSet dapat mempunyai level fungsionalitas yang berbeda, hal ini tergantung dari tipe dari result set. Level fungsionalitas dari setiap tipe result set dibedakan berdasarkan dua area:

- Dengan cara bagaimana result set itu dapat dimanipulasi
- Bagaimana result set itu menangani perubahan data yang dilakukan oleh proses lain secara bersamaan (concurrent).

JDBC menyediakan tiga tipe result set untuk tujuan berbeda:

1. TYPE_FORWARD_ONLY : result set tersebut tidak bisa berjalan mundur, result set hanya bisa berjalan maju dari baris pertama hingga baris terakhir. result set hanya menggambarkan keadaan data ketika query dijalankan atau ketika data diterima oleh result set. Jika setelah itu ada perubahan data dalam database, result set tidak akan diupdate alias tidak ada perubahan dalam result set tipe ini.
2. TYPE_SCROLL_INSENSITIVE : result set dapat berjalan maju mundur. result set dapat berjalan maju dari row pertama hingga terakhir atau bergerak bebas berdasarkan posisi relatif atau absolute.
3. TYPE_SCROLL_SENSITIVE : result set dapat berjalan maju mundur. result set dapat berjalan maju dari row pertama hingga terakhir atau bergerak bebas berdasarkan posisi relatif atau absolute.

Instance dari object bertipe ResultSet diperlukan untuk menampung hasil kembalian data dari database. Sebelum kita bisa memperoleh instance dari ResultSet, kita harus membuat instance dari class Statement. Class Statement mempunyai method executeQuery yang digunakan untuk menjalankan perintah query dalam database kemudian mengembalikan data hasil eksekusi query ke dalam object ResultSet.

Berikut ini adalah contoh kode untuk membuat instance class Statement, kemudian menjalankan

query untuk mengambil data dari database yang hasilnya dipegang oleh ResultSet :

```
Statement statement =  
    conn.createStatement(  
        ResultSet.TYPE_SCROLL_SENSITIVE,  
        ResultSet.CONCUR_READ_ONLY);  
ResultSet rs = statement.executeQuery("select * from T_PERSON");
```

ResultSet akan meletakkan kursornya (posisi pembacaan baris) di sebuah posisi sebelum baris pertama. Untuk menggerakkan kursor maju, mundur, ke suatu posisi relatif atau ke suatu posisi absolute tertentu, gunakan method-method dari ResultSet:

- next() -- mengarahkan kursor maju satu baris.
- previous() -- mengarahkan kursor mundur satu baris.
- first() -- mengarahkan kursor ke baris pertama.
- last() -- mengarahkan kursor ke baris terakhir.
- beforeFirst() -- mengarahkan kursor ke sebelum baris pertama.
- afterLast() -- mengarahkan kursor ke setelah baris terakhir.
- relative(int rows) -- mengarahkan kursor relatif dari posisinya yang sekarang. Set nilai rows dengan nilai positif untuk maju, dan nilai negatif untuk mundur.
- absolute(int rowNumber) - mengarahkan kursor ke posisi tertentu sesuai dengan nilai rowNumber, dan tentu saja nilainya harus positif.

Interface ResultSet menyediakan method getter untuk mengakses nilai dari setiap kolom dalam baris yang sedang aktif. Parameter fungsi getter bisa menerima nilai index dari kolom ataupun nama kolomnya. Namun begitu, penggunaan nilai index lebih efisien dibanding menggunakan nama kolom.

Nilai index dimulai dengan satu hingga banyaknya kolom. Penggunaan nama kolom adalah case insensitive, artinya huruf kecil atau huruf besar tidak menjadi masalah.

getString digunakan untuk mengambil kolom dengan tipe data char, varchar atau tipe data string lainnya. getInt digunakan untuk mengambil kolom dengan tipe data integer.

Berikut ini adalah contoh program lengkap dari melakukan koneksi hingga mengambil data dari database.

```
Class.forName("com.mysql.jdbc.Driver");  
Connection conn = DriverManager.getConnection(  
    "jdbc:mysql://localhost:3306/latihan","root","");  
Statement statement = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.CONCUR_READ_ONLY);  
ResultSet rs = statement.executeQuery("select * from T_PERSON");  
while(rs.next()){  
    System.out.println(rs.getInt("id"));  
    System.out.println(rs.getString("name"));  
}
```

Method executeQuery hanya dapat menjalankan perintah SQL select, gunakan method executeUpdate untuk menjalankan perintah insert, update dan delete. Hasil dari eksekusi insert, update dan delete tidak mengembalikan result set, tetapi mengembalikan sebuah nilai integer yang merepresentasikan status hasil eksekusi method executeUpdate.

Berikut ini contoh insert, update dan delete :

```
result = statement.executeUpdate(  
    "update T_PERSON set name='roby' where name='andy'");  
result = statement.executeUpdate("delete T_PERSON where name='andy'");
```

Statement sangat fleksible namun eksekusi query-nya sangat lambat jika dibandingkan dengan PreparedStatement. Hal ini dikarenakan setiap kali melakukan eksekusi query menggunakan statement, database akan melalui berbagai macam langkah eksekusi query, antara lain :

1. melakukan parsing terhadap query
2. membuat execution plan
3. memasukkan parameter ke dalam query
4. mengeksekusi query tersebut.

PreparedStatement hanya akan melakukan langkah 1 dan 2 ketika pertama kali menyiapkan PreparedStatement, eksekusi berikutnya hanya menjalankan langkah 3 dan 4. Dengan menggunakan sedikit test kecil, kecepatan eksekusi PreparedStatement bisa mencapai puluhan kali lebih cepat dibanding Statement.

Selain lebih cepat, PreparedStatement juga aman dari serangan hacker yang disebut dengan sql injection. Sql Injection adalah serangan hacker dengan memanfaatkan proses pembentukan query yang melibatkan penggabungan string seperti yang terjadi dalam Statement. Perhatikan query berikut ini yang digunakan dalam proses login

```
String name = "Andy";
String password = "pwd123";
Statement statement = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
    ResultSet.CONCUR_READ_ONLY);
ResultSet rs = statement.executeQuery("select * from T_PERSON where name='" + name
    + "' and password='" + password + "'");
```

Username dan password diambil dari form login, bagaimana jika yang login adalah seorang hacker dan mencoba memasukkan string berikut ini ke dalam username

```
String name = "hacker' or 1=1 --";
String password = "password bypass";
```

Dengan memasukkan username berbentuk query statement seperti di atas, mari kita lihat query yang dihasilkan dari proses penggabungan string di atas :

```
select * from T_PERSON where name ='hacker' or 1=1 -- and password='password
bypass'
```

Dengan menambahkan query "or 1=1" maka statement where akan selalu bernilai true dan string "--" digunakan untuk membuat query di belakangnya hanya sekedar komentar dan tidak ikut dieksekusi. Artinya, hanya dengan mengetahui username tanpa mengetahui password, hacker dapat mengeksekusi query di atas dengan sukses.

Selain itu, statement juga rentan terkena error kalau ada karakter khusus dalam string yang digabung-gabung dalam query. Misalnya ada seorang user yang namanya "Martin O'neal", tanda ' di dalam string nama akan membuat statement gagal dieksekusi karena query yang tidak valid. Begitu juga dengan karakter-karakter buruk lainnya seperti '\ " ` ~.

Menggunakan statement sangat mudah dan fleksible, namun sangat tidak efisien, rentan serangan hacker dalam bentuk sql injection, dan rentan mengalami sql error karena ada bad character dalam string parameter. PreparedStatement menawarkan keunggulan dari sisi efisiensi, keamanan dan tahan terhadap input yang mengandung bad character.

Kunjungi blog ifnu.artivisi.com/?p=77 untuk mengetahui bagaimana cara menghilangkan bad character jika anda menggunakan statement dan tidak bisa berpindah ke PreparedStatement.

Menggunakan PreparedStatement

Memanggil method executeUpdate berulang-ulang, misalnya melakukan insert ratusan atau ribuan baris, sangat tidak efisien. Hal ini disebabkan karena DBMS harus memproses setiap query yang dikirimkan dalam beberapa langkah: memarsing query, mengcompile query dan kemudian baru mengeksekusi query tersebut.

PreparedStatement menawarkan solusi yang lebih baik dalam menangani keadaan tersebut. PreparedStatement menyaratkan query yang akan dieksekusi didefinisikan terlebih dahulu ketika PreparedStatement dibuat. Kemudian query tersebut dikirimkan ke dalam database untuk dicompile terlebih dahulu sebelum digunakan. Konsekuensinya, PreparedStatement bukan hanya mempunyai query, tetapi mempunyai query yang sudah dicompile. Ketika PreparedStatement

dijalankan, DBMS tidak perlu melakukan kompilasi ulang terhadap query yang dijalankan PreparedStatement. Hal inilah yang menyebabkan PreparedStatement jauh lebih efisien dibandingkan menggunakan method Statement.executeUpdate.

Berikut ini contoh pembuatan PreparedStatement menggunakan class Connection yang telah dibuat sebelumnya :

```
PreparedStatement ps = conn.prepareStatement(  
    "update T_PERSON set password = ? where name = ?");
```

Perhatikan tanda ? yang ada dalam query di atas, tanda ? disebut sebagai parameter. Kita bisa memberikan nilai yang berbeda ke dalam parameter dalam setiap pemanggilan PreparedStatement.

Method setString, setFloat, setInt dan beberapa method lain digunakan untuk memasukkan nilai dari setiap parameter. Method tersebut mempunyai dua parameter, parameter pertama adalah int yang digunakan untuk menentukan parameter PreparedStatement mana yang akan diberi nilai. Parameter kedua adalah nilai yang akan dimasukkan ke dalam PreparedStatement, tipe data dari parameter kedua tergantung dari method yang digunakan. Berdasarkan kode di atas, berikut ini contoh penggunaan method PreparedStatement.setString :

```
ps.setString(1,"pwdbaru");  
ps.setString(2,"rizal");
```

Kode di atas memberikan contoh bagaimana memasukkan nilai ke dalam parameter PreparedStatement. Baris pertama memasukkan String "andy" ke dalam parameter pertama dan baris kedua memasukkan String "rizal" ke parameter kedua. Sehingga pemanggilan query oleh PreparedStatement berdasarkan kode di atas sama dengan query statement di bawah ini :

```
"update T_PERSON set pwd = 'pwdbaru' where name = 'rizal'"
```

Berikut ini contoh lengkap penggunaan PreparedStatement untuk melakukan update dan insert data :

```
PreparedStatement pInsert = conn.prepareStatement(  
    "insert into T_PERSON(name,password) values(?,?)");  
pInsert.setString(1,"dian");  
pInsert.setString(2,"pwddian");  
pInsert.executeUpdate();  
PreparedStatement pUpdate = conn.prepareStatement(  
    "update T_PERSON set password=? where name=?");  
pUpdate.setString(1,"pwdandri");  
pUpdate.setString(2,"andri");  
pUpdate.executeUpdate();
```

Dalam contoh di atas, insert dan update data hanya dilaksanakan sekali saja, hal ini tidak memberikan gambaran yang tepat untuk melihat keunggulan PreparedStatement dibandingkan Statement.executeUpdate.

Sql Injection dapat dihindari dengan menggunakan PreparedStatement karena apapun yang dimasukkan hacker akan dianggap sebagai parameter dan tidak dianggap sebagai bagian dari query. Mari kita implementasikan lagi contoh query login sebelumnya yang menggunakan Statement menjadi menggunakan PreparedStatement.

```
String name = "Andy";  
String password = "pwd123";  
PreparedStatement loginStatement = conn.prepareStatement("select * from T_PERSON  
where name = ? and password = ?");  
loginStatement.setString(1,name);  
loginStatement.setString(2,password);
```

Walaupun hacker memasukkan username dan password seperti berikut ini :

```
String name = "hacker' or 1=1 --";  
String password = "password bypass";
```

Kedua text itu tidak dianggap sebagai bagian dari query tapi hanya sebagai parameter saja. PreparedStatement akan berusaha mencari username "hacker' or 1=1 --" di dalam database

dan pasti akan gagal, query untuk membandingkan password juga tetap dieksekusi walaupun ada string "--" di dalam username.

Pesan moralnya adalah gunakan selalu PreparedStatement dan hindari sebisa mungkin untuk menggunakan Statement.

Batch Execution

Misalnya kita ingin meng-insert seratus baris data dalam sebuah loop, kita bisa menggunakan fasilitas batch execution dari PreparedStatement. batch execution mengumpulkan semua eksekusi program yang akan dilaksanakan, setelah semuanya terkumpul batch execution kemudian mengirimkan kumpulan eksekusi program secara bersamaan ke DBMS dalam satu kesatuan. Metode ini sangat efisien karena mengurangi overhead yang diperlukan program untuk berkomunikasi dengan DBMS.

Dalam contoh di bawah ini kita akan menggunakan batch execution untuk melakukan insert data sebanyak seratus kali.

```
PreparedStatement pInsert = conn.prepareStatement(
    "insert into T_PERSON(name,password) values(?,?)");
for(int i=0;i<100;i++){
    pInsert.setString(1,"user ke " + i);
    pInsert.setString(2,"pwd ke " + i);
    pInsert.addBatch();
}
pInsert.executeBatch();
```

Setiap kali iterasi, method setString dipanggil untuk mengisi sebuah string ke dalam PreparedStatement, kemudian method addBatch dipanggil untuk mengumpulkan batch dalam satu wadah. Setelah iterasi selesai, method executeBatch dipanggil untuk melaksanakan semua keseratus instruksi insert secara berurut dengan sekali saja melaksanakan koneksi ke database.

Menangani Transaction

Dukungan transaction oleh JDBC tergantung dengan Databasenya, karena ada database yang mendukung transaction dan ada pula database yang tidak mendukung transaction. MySQL mendukung transaction jika kita menggunakan InnoDB sebagai sistem tablenya, kalau kita menggunakan MyISAM maka transaction tidak didukung.

Transaction merupakan konsep penting dari database. Transaction memastikan perubahan data dilaksanakan dengan kaidah ACID (Atomicity, Consistency, Isolation, Durability). Kaidah ini memastikan semua proses perubahan data berjalan secara benar, jika ada yang salah maka semua perubahan dalam satu kesatuan logika harus dibatalkan (rollback).

Mari kita evaluasi kode di atas agar menggunakan transaction, sehingga jika satu proses insert gagal, maka semua insert yang dilaksanakan sebelumnya akan dibatalkan :

```
try{
    connection.setAutoCommit(false);
    PreparedStatement pInsert = conn.prepareStatement(
        "insert into T_PERSON(name,password) values(?,?)");
    for(int i=0;i<100;i++){
        pInsert.setString(1,"user ke " + i);
        pInsert.setString(2,"pwd ke " + i);
        pInsert.addBatch();
    }
    pInsert.executeBatch();
    connection.commit();
    connection.setAutoCommit(true);
} catch (SQLException ex) {
    try{
```

```
        connection.rollback();
    }catch(SQLException e){
    }
}
```

Mendapatkan ID yang Digenerate Otomatis

MySQL mempunyai feature auto-increment untuk mengenerate primary key secara otomatis. Untuk mendapatkan ID yang baru saja digenerate ada teknik khusus, tidak bisa menggunakan cara dengan mengambil id yang paling besar dari table T_PERSON. Cara ini akan sangat berbahaya kalau aplikasi digunakan oleh banyak user, misalnya setelah user A menginsert satu baris kemudian select id terbesar, ada kemungkinan user B juga menginsert data sehingga id yang didapatkan user A dari select id terbesar bukan id yang baru saja digenerate user A, melainkan id yang digenerate user B.

Berikut ini contoh kode untuk mengambil id yang baru saja digenerate oleh database :

```
PreparedStatement pInsert = conn.prepareStatement(
    "insert into T_PERSON(name,password)", Statement.RETURN_GENERATED_KEYS);
pInsert.setString(1,"dian");
pInsert.setString(2,"pwddian");
pInsert.executeUpdate();

ResultSet rs = pInsert.getGeneratedKeys();
rs.next();
Long id = rs.getLong(1);
```

PreparedStatement harus digunakan untuk mengambil key yang digenerate MySQL secara otomatis. Para waktu membuat prepared statement, ada tambahan satu parameter yaitu Statement.RETURN_GENERATED_KEYS. Parameter tersebut digunakan untuk memberitahu PreparedStatement bahwa proses insert akan mengenerate primary id secara otomatis.

Setelah pInser.executeUpdate(); dilaksanakan, dengan menggunakan prepared statement yang sama eksekusi method getGeneratedKeys();, method ini akan mereturn ResultSet yang isinya cuma satu baris dan satu column. Di baris berikut nya dipanggil rs.next() untuk meloncat ke baris pertama dan rs.getLong(1) untuk mengambil id-nya.

JDBC-ODBC Bridge

ODBC atau Open Database Connectivity adalah standard pengaksesan relational database atau data source lain dengan memanfaatkan middle layer untuk menjembatani aplikasi dengan datasource tersebut. ODBC dikembangkan bersama oleh SQL Access group yang terdiri dari perusahaan-perusahaan besar seperti Microsoft dan IBM. Sebelum dikembangkan ODBC, setiap koneksi ke database berbeda-beda menggunakan cara yang berbeda-beda pula, hal ini menyulitkan pengembang untuk membuat kode yang portable antar database. Dengan menggunakan ODBC, aplikasi bisa mengakses database dengan cara yang seragam dan hanya perlu menyediakan driver yang sesuai dengan database tersebut. Konsep ini diambil oleh JDBC dimana API untuk mengakses database seragam, hanya perlu menyediakan driver sesuai untuk setiap database.

Sayangnya beberapa jenis database tidak menyediakan JDBC driver, seperti MS-Access. Untuk bisa connect ke MS-Access bisa memanfaatkan JDBC-ODBC driver bridge yang disediakan Sun. Setiap instalasi MS-Access 2003 akan disertakan database sample Northwind.mdb, letakkanya ada di [Office Installation]\Office11\SAMPLES, berikutnya siapkan beberapa langkah untuk membuat ODBC configuration.

Agar bisa menggunakan JDBC-ODBC bridge, anda perlu membuat DSN (Data Source Name) yang digunakan untuk mensetting driver ODBC dan databasenya. Berikut langkah-langkah membuat DSN :

- Buka Control Panel > Administrative Tool > Data Sources (ODBC)

- Klik tab System DSN. Kemudian klik tombol Add
- Dari daftar driver ODBC, pilih Microsoft Access Driver (*.mdb), ODBCJT32.DLL, dan tekan tombol finish.
- Sekarang bisa dilihat ada informasi DSN seperti berikut ini :

```
Name: Northwind
Description: DSN for the Northwind.mdb
Select Database: Northwind.mdb
```

- Tekan tombol OK. DSN "Northwind" sekarang sudah siap digunakan

Setelah DSN siap, langkah berikutnya adalah membuat connection ke DSN menggunakan JDBC-ODBC bridge.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn = DriverManager.getConnection("jdbc:odbc:Northwind");
```

JDBC-ODBC driver class `sun.jdbc.odbc.JdbcOdbcDriver` sudah tersedia di dalam JDK, terutama di windows. Sedangkan di OSX atau Linux masih harus diteliti lagi apakah class ini tersedia atau tidak.

Setelah koneksi berhasil dibuat, sisanya sama saja dengan menggunakan JDBC driver biasa. Proses pembuatan query dengan statement atau preparedstatement, transaction dan seterusnya sama.

Memanggil Function dan StoredProcedure

Stored procedure dan Function tidak bisa dipisahkan dari database, keduanya digunakan untuk mengeksekusi query di database. Menulis query di JDBC atau SP tergantung dari style pemrograman, bisa juga karena merek database tertentu mempunyai feature yang bisa digunakan dalam SP tetapi tidak tersedia di JDBC. SP juga sangat cepat, beberapa RDBMS bisa compile SP menjadi native code sehingga kecepatan eksekusinya meningkat. SP juga bisa mengurangi komunikasi network jika ada proses yang melibatkan query, loop, condition checking terhadap data yang cukup besar dan tidak melibatkan interaksi user.

MySQL semenjak versi 5 sudah mendukung StoredProcedure, informasi yang lengkap tentang stored procedure di mysql 5 bisa dibaca di sini:

```
http://dev.mysql.com/tech-resources/articles/mysql-storedprocedures.pdf
```

Sebelum bisa memanggil stored procedure dari JDBC, pertama kali jelas harus dibuat dulu stored procedurenya. Membuat stored procedure di mysql tidak terlalu susah, buka mysql client console dan jalankan perintah berikut ini:

```
mysql> create procedure get_persons () select * from T_PERSON;
```

Setelah procedure selesai dibuat, coba jalankan dengan menggunakan perintah ini :

```
mysql> call get_persons();
```

Kalau perintah di atas bisa dijalankan dengan baik, pertanda bahwa stored procedurenya berhasil digunakan.

Memanggil stored procedure dari JDBC tidak susah, prosesnya sama dengan menggunakan PreparedStatement. Berikut ini kode untuk memanggil stored procedure `get_users` di atas

```
Class.forName("com.mysql.jdbc.Driver");
Connection c = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/latihan", "root", "");

CallableStatement callableStatement = c.prepareCall("{call get_persons}");
ResultSet rs = callableStatement.executeQuery();
while(rs.next()){
    System.out.println(rs.getInt("id") + "," +
        rs.getString("name") + "," +
        rs.getString("password"));
}
```

```
}
```

Stored procedure juga bisa mempunyai parameter. Berikut ini cara membuat dan memanggil stored procedure di mysql 5 yang mempunyai parameter.

```
mysql> create procedure get_person_by_id (IN user_id int) select * from T_PERSON  
where id=user_id;
```

Memanggil stored procedure

```
mysql> call get_person_by_id(5000);
```

Memanggil stored procedure dengan parameter berupa variabel di mysql 5 client console

```
mysql> set @person_id = 5000;  
mysql> call get_person_by_id(@person_id);
```

Memanggil stored procedure dengan parameter dari JDBC

```
CallableStatement callableStatement =  
    c.prepareCall("{call get_person_by_id(?)}");  
callableStatement.setInt(1, 5000);  
ResultSet rs = callableStatement.executeQuery();  
while(rs.next()){  
    System.out.println(rs.getInt("id") + "," +  
        rs.getString("username") + "," + rs.getString("password"));  
}
```

Pemanggilan Stored procedure dapat dikombinasikan dengan Query biasa dalam satu skope transaction.

Artikel lengkap tentang JDBC bisa dilihat di website sun :

<http://java.sun.com/developer/onlineTraining/Database/JDBC20Intro/JDBC20.html>

Model, Dao dan Service Pattern

Akses terhadap database merupakan bagian yang sangat penting dari aplikasi database. Penggunaan pattern yang sesuai dapat memberikan manfaat sangat besar. Pattern yang sering digunakan dalam akses database adalah DAO (Data Access Object) dan Service/Facade pattern.

Kedua pattern ini digunakan untuk menerapkan “separation of concern” atau pemisahan kode program berdasarkan fungsi kode program. Semua kode untuk akses data harus dipisahkan dengan kode untuk pengaturan user interface. Hal ini memungkinkan kode akses data yang dibuat untuk aplikasi desktop, dengan mudah digunakan untuk aplikasi web.

Penerapan konsep separation of concern secara disiplin, dapat menghasilkan kode program yang dapat dites secara otomatis menggunakan JUnit atau DBUnit. Unit testing merupakan parameter utama dalam menentukan apakah kode program yang kita hasilkan mempunyai mutu yang tinggi atau tidak. Coverage unit testing yang tinggi mencerminkan kode program yang berkualitas tinggi pula.

Dao pattern berisi semua kode untuk mengakses data, seperti query. Semua kode yang spesifik terhadap implementasi akses data berhenti di sini, lapisan lebih atas tidak boleh tahu bagaimana akses data diterapkan, apakah menggunakan JDBC murni atau Hibernate atau JPA. Lapisan lain hanya perlu tahu fungsionalitas dari suatu method di dalam DAO class, tidak perlu tahu bagaimana method tersebut diimplementasikan. Class DAO akan mempunyai method seperti save, delete, getById atau getAll. Praktek yang lazim digunakan adalah satu buah Entity/Table akan mempunyai satu buah class DAO.

Bagian ini akan menerangkan bagaimana melakukan akses database menggunakan Hibernate serta menerangkan teori ORM dibalik Hibernate. Pembahasan dilanjutkan bagaimana menggunakan Spring untuk manage Hibernate dan mengimplementasikan Declarative Transaction. Bagian selanjutnya membahas feature Hibernate secara mendetail.

Entity Class / Model

Di dalam DAO layer ini, biasanya juga diterapkan konsep ORM. Setiap table dibuatkan Entity class yang merepresentasikan table. Hal ini memudahkan maintenance kode karena abstraksi dan logic aplikasi diterapkan di dalam kode. Hanya dengan melihat kode dan nama-nama class atau property, developer bisa melihat logic dibaliknya. Ditambah dengan javadoc dan komentar yang ekstensif, ORM bisa memudahkan maintenance code sangat signifikan.

Contoh-contoh di atas menggunakan table T_PERSON, dalam konsep ORM, table ini akan dimap dengan Entity class Person. Setiap property dalam class Person merepresentasikan kolom dalam table T_PERSON.

Buat class Person di dalam package com.googlecode.projecttemplate.pos.model. Agar kode yang kita buat terstruktur dengan rapi, class-class yang mempunyai fungsi yang sama akan dikelompokkan ke dalam package yang sama pula, misalnya class Person ini diletakkan dalam package model, kemudian class-class DAO diletakkan dalam package dao, class-class service diletakkan dalam package service dan class-class UI diletakkan dalam package ui.

```
public class Person {
    private Long id;
    private String name;
    private String password;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
```



```

        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}

```

Setelah data melewati DAO layer, tidak ada lagi class-class dari package javax.sql seperti ResultSet. Setiap query yang dieksekusi untuk mendapatkan data dari table T_PERSON di map ke dalam class Person, sehingga di dalam kode yang menggunakan DAO, tidak perlu lagi menghapus nama-nama kolom dari table, cukup menggunakan kelas Person.

DAO Pattern

DAO layer hanya melakukan query sederhana, sebaiknya tidak menempatkan kode bisnis di dalam layer DAO, karena satu DAO hanya ditujukan untuk melakukan manipulasi terhadap satu table saja. Proses transaksi juga tidak terjadi dalam layer DAO tetapi di layer lebih atasnya yaitu Service layer. Perhatikan juga semua DAO method men-throw SQLException, sehingga kalau ada error di dalam method dao, SQLException dilempar ke layer service dan layer service bisa merollback transaction.

Di bawah ini adalah contoh PersonDaoJdbc menggunakan JDBC untuk table T_PERSON :

```

public class PersonDaoJdbc {
    private Connection connection;
    private PreparedStatement insertStatement;
    private PreparedStatement updateStatement;
    private PreparedStatement deleteStatement;
    private PreparedStatement getByIdStatement;
    private PreparedStatement getAllStatement;
    private final String insertQuery = "insert into T_PERSON(name,password) " +
        " values(?,?)";
    private final String updateQuery = "update T_PERSON set name=?, " +
        " password=? where id=?";
    private final String deleteQuery = "delete from T_PERSON where id=?";
    private final String getByIdQuery = "select * from T_PERSON where id =?";
    private final String getAllQuery = "select * from T_PERSON";

    public void setConnection(Connection connection) throws SQLException{
        this.connection = connection;
        insertStatement = this.connection.prepareStatement(insertQuery,
            Statement.RETURN_GENERATED_KEYS);
        updateStatement = this.connection.prepareStatement(updateQuery);
        deleteStatement = this.connection.prepareStatement(deleteQuery);
        getByIdStatement = this.connection.prepareStatement(getByIdQuery);
        getAllStatement = this.connection.prepareStatement(getAllQuery);
    }

    public Person save(Person person) throws SQLException{
        if (person.getId() == null) {
            insertStatement.setString(1, person.getName());
            insertStatement.setString(2, person.getPassword());
        }
    }
}

```

```

        int id = insertStatement.executeUpdate();
        person.setId(id);
    } else {
        updateStatement.setString(1, person.getName());
        updateStatement.setString(2, person.getPassword());
        updateStatement.setInt(3, person.getId());
        updateStatement.executeUpdate();
    }
    return person;
}

public Person delete(Person person) throws SQLException{
    deleteStatement.setInt(1, person.getId());
    deleteStatement.executeUpdate();
    return person;
}

public Person getById(Long id) throws SQLException{
    getByIdStatement.setLong(1, id);
    ResultSet rs = getByIdStatement.executeQuery();
    //proses mapping dari relational ke object
    if (rs.next()) {
        Person person = new Person();
        person.setId(rs.getLong("id"));
        person.setName(rs.getString("name"));
        person.setPassword(rs.getString("password"));
        return person;
    }
    return null;
}

public List<Person> getAll() throws SQLException{
    List<Person> persons =
        new ArrayList<Person>();
    ResultSet rs = getAllStatement.executeQuery();
    while(rs.next()){
        Person person = new Person();
        person.setId(rs.getLong("id"));
        person.setName(rs.getString("name"));
        person.setPassword(rs.getString("password"));
        persons.add(person);
    }
    return persons;
}
}

```

Service Pattern

Service pattern digunakan utamanya untuk menyederhanakan class-class DAO yang ada, misalnya kita mempunyai 50 buah table maka lazimnya akan ada 50 buah class DAO. Class DAO tersebut perlu disederhanakan, caranya adalah dengan mengelompokkan class-class DAO dalam satu modul aplikasi ke class Service. Misalnya DAO yang berhubungan dengan user management ke dalam class UserService.

Transaction diatur dalam class Service, praktek yang lazim digunakan adalah satu method dalam class service adalah satu scope transaction. Jadi ketika method dalam service mulai dieksekusi transaction akan dimulai (begin), ketika method akan berakhir, transaction akan dicommit, jika terjadi exception pada saat method dilaksanakan dilakukan rollback data untuk mengembalikan

keadaan seperti sebelumnya.

Berikut ini adalah contoh class Service :

```
public class ServiceJdbc {
    private PersonDaoJdbc personDao;
    private Connection connection;

    public void setDataSource(DataSource dataSource){
        try {
            connection = dataSource.getConnection();
            personDao = new PersonDaoJdbc();
            personDao.setConnection(connection);
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }

    public Person save(Person person){
        try {
            connection.setAutoCommit(false);
            personDao.save(person);
            connection.commit();
            connection.setAutoCommit(true);
        } catch (SQLException ex) {
            try{
                connection.rollback();
            }catch(SQLException e){
                e.printStackTrace();
            }
        }
        return person;
    }

    public Person delete(Person person){
        try {
            connection.setAutoCommit(false);
            personDao.save(person);
            connection.commit();
            connection.setAutoCommit(true);
        } catch (SQLException ex) {
            try{
                connection.rollback();
            }catch(SQLException e){
                e.printStackTrace();
            }
        }
        return person;
    }

    public Person getPerson(Long id){
        try {
            return personDao.getById(id);
        } catch (SQLException ex) {
            e.printStackTrace();
        }
        return null;
    }
}
```

```

    public List<Person> getPersons(){
        try{
            return personDao.getAll();
        } catch (SQLException ex) {
            e.printStackTrace();
        }
        return new ArrayList<Person>();
    }
}

```

Setelah class DAO dan service berhasil dibuat, mari kita lihat bagaimana cara menggunakannya :

```

public class MainJdbc {

    public static void main(String[] args) {

        MysqlDataSource dataSource = new MysqlDataSource();
        dataSource.setUser("root");
        dataSource.setPassword("");
        dataSource.setDatabaseName("latihan");
        dataSource.setServerName("localhost");
        dataSource.setPortNumber(3306);

        ServiceJdbc service = new ServiceJdbc();
        service.setDataSource(dataSource);

        Person person = new Person();
        person.setName("administrator");
        person.setPassword("pwd");
        service.save(person);

        System.out.println("id : " + person.getId());
        System.out.println("name: " + person.getName());

        try {
            dataSource.getConnection().close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}

```

Seperti terlihat dalam kode di atas, abstraksi OOP jadi terlihat dengan jelas, kode menjadi rapi dan mudah dibaca dengan hilangnya try-catch untuk menhandle SQLException.

Pemisahan layer seperti ini juga sangat strategis dalam pembagian tugas antar developer. Misalnya dalam satu team terdapat developer senior dan junior, keduanya bisa mengerjakan bagian yang berbeda tanpa harus menunggu bagian lain selesai. Developer senior mengerjakan bagian backend ke database, kemudian membuat kerangka kode DAO dan Service ini, di tahap awal project, kode DAO dan Service masih kosong menunggu requirement di approve oleh client. Dengan menggunakan kerangka kode ini, Developer junior yang bertugas untuk membuat UI sudah bisa langsung menggunakan kode DAO dan Service. Setelah proses requirement selesai, Developer senior bisa mulai memfinalisasi kode DAO dan Service, developer junior tidak perlu merubah kodenya sama sekali karena perubahan di sisi DAO dan Service tidak berpengaruh sama sekali di kode UI. Semakin besar ukuran tim dalam project, semakin penting pemisahan layer kode ini untuk mendukung kerja tim agar tidak saling menunggu anggota tim lain menyelesaikan tugasnya, selama kerangka kodenya sudah dibuat, anggota tim lain sudah bisa mulai mengerjakan tugasnya.

Setelah mengenal JDBC, kita akan membahas cara mengakses database yang lebih baik dengan

menggunakan Hibernate. Dengan menggunakan Hibernate kode program yang akan dibuat akan lebih ringkas dan terlihat konsep OOP dibanding dengan JDBC murni. Hibernate adalah framework ORM untuk memetakan tabel dalam database dan class dalam konsep OOP. Dengan menggunakan Hibernate kode program kita akan lebih rapi dan terlihat OOP.

ORM, Hibernate dan Spring

Akses database menggunakan kode JDBC murni memerlukan kode yang cukup panjang, bisa dilihat dari class DAO dan class Service di bagian sebelumnya. Selain itu SQLException juga ada di mana-mana, penggunaan transaction juga masih manual dengan memanggil kode begin, rollback dan commit.

Bagian ini akan membahas konsep akses database menggunakan OOP menggunakan konsep ORM dan framework hibernate. Selain itu akan dibahas juga penggunaan Spring untuk menangani transaction dengan menerapkan konsep declarative transaction menggantikan programatic transaction yang digunakan di bagian sebelumnya.

Kalau masih awam dengan semua istilah di atas, jangan khawatir. Akan kita bahas satu per satu konsep ORM, Hibernate, declarative transaction vs programatic transaction dan spring di bagian-bagian selanjutnya.

Object Relational Mapping

Object Relational Mapping (ORM) adalah sebuah framework yang dapat menjembatani perbedaan sistem basis data yang bersifat relational dengan paradigma pengembangan aplikasi yang berorientasi objek. Setiap objek yang akan memetakan menjadi tabel-tabel pada basis data relational dibungkus oleh suatu interface dengan menerapkan konsep design pattern. Hal tersebut bertujuan untuk memudahkan lapisan aplikasi (controller) mengakses data tersebut.

Object Relational Mapping merupakan teknik otomatisasi dan transparansi dari *object persistence* ke dalam tabel pada basis data, menggunakan metadata yang mendeskripsikan pemetaan antara objek dan basis data. ORM berperan dalam lapisan model dalam konsep MVC. Model adalah sebuah lapisan yang paling dekat dengan sumber data, baik itu berasal dari basis data, *webservice*, maupun *file system*.

Object Relational Mapping (ORM) juga mengatasi perbedaan sistem basis data yang bersifat relational dengan paradigma pengembangan aplikasi yang berorientasi objek. Selain itu, ORM juga menjembatani dialek SQL yang digunakan, sehingga apapun produk RDBMS yang digunakan tidak berpengaruh terhadap kode program. ORM merupakan solusi yang mengatasi perbedaan aspek-aspek ketidaksesuaian antara konsep pemrograman berorientasi objek dengan konsep basis data relational.

Hibernate

Project hibernate dimulai pada tahun 2001 oleh Gavin King, project ini mulai mendapat tanggapan serius setelah Gavin King dipekerjakan oleh Jboss dan Jboss mulai mengerahkan pasukan lebih banyak lagi untuk mengelola Hibernate secara serius. Keberhasilan Hibernate sangat fenomenal, bahkan di satu titik Hibernate justru lebih terkenal dibanding konsep ORM itu sendiri.

Hibernate + Spring benar-benar menohok EJB 2.1 secara telak, sehingga dalam rilis versi berikutnya, EJB 3.0, diperkenalkan konsep JPA yang mengambil ilham dari Hibernate. Sekarang ini Hibernate, bersama iBatis, sudah menjadi pemimpin pasar di backend framework untuk mengabstraksi JDBC.

Semenjak versi 3.5 hibernate dilengkapi dengan Hibernate Annotation dan juga Hibernate secara resmi menjadi salah satu implementasi JPA. Sedangkan JPA sendiri pada dasarnya hanya API yang wujud nyatanya adalah sekumpulan interface seperti halnya JDBC. Implementasi JPA yang dikenal selain Hibernate adalah Toplink.

Annotation dengan Hibernate bisa diimplementasikan dengan tiga cara: menggunakan JPA annotation, menggunakan murni Hibernate Annotation atau mencampur antara keduanya. Bagaimana membedakan hibernate annotation dan JPA annotation? Cukup dilihat import dari annotationnya, jika diimport dari `javax.persistence` berarti JPA annotation, kalau diimport dari `org.hibernate.mapping` berarti Hibernate annotation.

Feature dari Hibernate annotation lebih lengkap dibanding JPA annotation. Pendekatan yang digunakan dalam buku ini adalah secara default menggunakan JPA Annotation, jika ada feature mapping yang tidak ada di dalam JPA annotation, maka baru digunakan Hibernate annotation. Anda tidak perlu bingung harus menggunakan yang mana, karena pada dasarnya implementasi di belakang annotation ini ya Hibernate itu sendiri, jadi menggunakan cara yang manapun hasilnya sama saja. Secara teknis tidak ada bedanya, hanya masalah pilihan saja.

Project Hibernate mempunyai pengaruh yang sangat besar, selain mempengaruhi EJB dengan dilahirkannya JPA, Hibernate project juga menghasilkan Hibernate Validation yang merupakan cikal bakal Beans Validation yang akhirnya masuk JSR. Selain itu ada project Hibernate Search yang bertujuan untuk membuat full text indexing dari table-table yang dikelola Hibernate, sehingga bisa dilakukan full text search terhadap table tersebut, sangat powerfull. Project berikutnya adalah Hibernate Shards, project ini bertujuan untuk memberikan kemampuan pemisahan table dalam beberapa instance database, alias distributed database.

Object persistence

Dalam pengembangan sistem, pengertian *persistence* adalah objek yang dihasilkan dari suatu sistem yang dapat disimpan dalam waktu lama bahkan bersifat permanen. *Object Persistence* bisa disimpan dalam bentuk basis data relasional, *file system* dalam *harddisk*, *file XML*, *Web Service*, *ODBMS (Object Data Base Management System)*, dan *LDAP*. Namun *Object Persistence* yang paling banyak digunakan dalam pengembangan perangkat lunak adalah basis data relasional. Basis data relasional mudah dibuat dan diakses.

Dengan *object persistence*, data yang terkandung pada objek tersebut dapat dengan mudah disimpan dan diakses. *Object persistence* bersifat abstrak, dapat menyembunyikan rincian mekanisme bagaimana suatu data disimpan dan diakses, sehingga tidak terlihat oleh objek lainnya.

Object persistence yang dimaksud dalam buku ini dibatasi hanya ke RDBMS.

Object-relational mismatch

Object-relational mismatch adalah ketidaksesuaian antara basis data yang menggunakan konsep relasional dengan pengembangan aplikasi yang menggunakan konsep berorientasi objek. Ketidaksesuaian tersebut meliputi aspek:

Granularity

Pemecahan *entity* pada atribut-atribut yang lebih kompleks. Misalnya *class Person* mempunyai atribut *address* dengan tipe data *Address*. Sedangkan pada basis data relasional tidak memungkinkan pada tabel *PERSON* ada kolom *address* dengan tipe data *address*, yang mungkin dilakukan adalah memecah *address* menjadi *street_address*, *city_address*, *zipCode_address*.

Subtypes

Pembeda antara *superclass* dan *subclass*. Pada pemrograman berbasis objek dikenal istilah *inheritance* (pewarisan) dari *class parent* kepada *class child*. Sedangkan pada basis data relasional tidak dikenal proses pewarisan antar tabel.

Identity

Terdapat perbedaan fungsi antara lambang sama dengan (=) dan *method equals()*, pada objek tetapi merujuk nilai yang sama pada *primary key* basis data relasional.

Association

Hubungan dua entitas pada objek dikenal dengan *references* sedangkan pada relasional dikenal dengan *foreign key*. Asosiasi antar *entity* terdiri dari: *one-to-one*, *one-to-many*, dan *many-to-many*.

Navigasi data

Proses pencarian pada basis data menggunakan *join table* sedangkan pada objek memanggil suatu *method getter*. Navigasi dibagi dua macam berdasarkan arahnya, antara lain: *directional* dan *bidirectional*.

Implementasi ORM untuk Mengatasi Masalah Ketidaksesuaian

Entity class merupakan *class* yang merepresentasikan setiap tabel pada basis data. Entity berfungsi sebagai jembatan penghubung antar lapisan dalam aplikasi. Dengan pola ini proses perpindahan data menjadi sederhana dan terintegrasi. Entity dibuat berdasarkan UML *class diagram* yang telah dirancang. Entity berisi *class JavaBean* yang setiap propertinya akan merepresentasikan atribut-atribut pada tabel.

Proses pemetaan *Entity* menjadi tabel terjadi ketidaksesuaian antara tabel yang berasal dari basis data relasional dan *class javaBeans* yang berorientasi objek. Ketidaksesuaian tersebut dapat di atasi dengan menggunakan konsep ORM sehingga setiap tabel bisa merepresentasikan *class* DTO dan diakses sebagai objek. Uraian berikut menjelaskan implementasi konsep ORM untuk mengatasi ketidaksesuaian yang meliputi tiga aspek yaitu aspek identitas, asosiasi, dan navigasi.

Identitas

Terdapat perbedaan pengenalan identitas antara objek dan tabel. Pada objek identitas dibedakan menjadi nilai dan alamat memorinya. Sehingga terdapat dua notasi yang melambangkan kesamaan suatu identitas pada objek Java, yaitu lambang samadengan (==) dan *method equals()*. Contoh : `a == b`, berarti variabel a dan b memegang alamat *reference* yang sama pada memori. `(a.equals(b))`, secara logik mempunyai nilai yang sama.

Pada basis data relasional, identitas dari suatu tabel disebut dengan *primary key*. Dengan demikian, sering kali terjadi objek yang secara logik sama (`a.equals(b)`) dan mewakili satu baris dalam tabel basis data, tetapi objek tersebut tidak berada pada satu lokasi alamat memori yang sama.

ORM mengatasi ketidaksesuaian tersebut dengan menambah properti *identity* pada setiap entity. Dengan demikian pengujian apakah *class* a sama dengan *class* b bisa ditulis seperti berikut: `a.getId().equals(b.getId())`.

Asosiasi

Kebanyakan entitas pada basis data mempunyai keterhubungan dengan entitas yang lainnya. Elemen yang menghubungkan kedua tabel tersebut ditandai dengan *foreign key*. Pada objek, elemen penghubung dua objek ditandai dengan sebuah *reference object*. Namun permasalahannya adalah pada *reference object* yang menghubungkannya bergantung pada arah asosiasinya (*direction of relationship*). Apabila asosiasi antara dua objek terjadi pada dua arah, maka harus didefinisikan dua kali pada setiap *classnya*.

Akan tetapi *foreign key* pada tabel relasional, tidak mengenal arah dari asosiasinya, karena asosiasi antara dua tabel dihubungkan dengan *table join* atau *projection*. Asosiasi antara dua entitas terdiri dari *one-to-one*, *one-to-many*, dan *many-to-many*.

Navigasi data

Masalah navigasi data adalah problem bagaimana mengakses suatu objek dari objek lain. Menurut arahnya, navigasi terbagi menjadi dua macam, yaitu: *unidirectional* dan *bidirectional*. Pada pemrograman berbasis objek, proses akses properti objek dari objek lain bisa langsung menggunakan *method getter*. Apabila arah navigasinya *unidirectional*, maka tidak terdapat properti objek tersebut di object lawanya. Sehingga dari satu arah relasi bisa terlihat, tapi di sisi lain relasi tidak terlihat.

Sedangkan pada basis data relasional, konsep arah navigasi tidak mempengaruhi proses akses data. Misalnya sebuah tabel mempunyai relasi *one-to-many* kemudian *foreign key* berada pada tabel lawanya, akan tetapi arah navigasi dari asosiasi yang menghubungkannya tidak

berpengaruh. Selama ada *foreign key* yang menghubungkan kedua tabel, proses query yang melibatkan kedua tabel bisa dijalankan.

Teorinya cukup dua halaman setengah saja, bahasan berikutnya lebih banyak praktek dan membuat kode.

Mapping Sederhana

Hibernate mapping dapat dilakukan dengan dua cara : menggunakan xml atau menggunakan annotation. Karena sekarang Java 5 dan Java 6 sudah lazim di gunakan maka hanya akan dibahas mapping hibernate menggunakan annotation saja. Kalau si bos masih memaksa menggunakan xml silahkan mulai organisasi teman-teman satu project untuk mengadakan pemogokan kerja :D. Pemaksaan menggunakan XML adalah melanggar hak asasi programmer untuk menggunakan teknologi terbaru dan tools terbaik yang tersedia.

Persiapan yang harus dilakukan sebelum memulai membuat mapping sederhana adalah menambahkan library Hibernate JPA ke dalam project Java. Di dalam library Hibernate JPA terdapat

Di bab sebelumnya yang membahas JDBC sudah ada class Person yang dimapping dengan table T_PERSON, nah class tersebut akan diubah dan didekorasi dengan hibernate annotation sehingga proses mapping bisa terjadi secara otomatis. Setelah ditambahkan hibernate annotation, class Person akan terlihat seperti ini :

```
@Entity
@Table(name="T_PERSON")
public class Person implements Serializable {

    @Id @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="ID")
    private Long id;
    @Column(name="name",unique=true,length=100)
    private String name;
    @Column(name="PASSWORD",unique=true,length=200)
    private String password;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
}
```

Mari kita bahas satu-satu a keong (@) yang ada di dalam class Person ini.

@Entity digunakan untuk mendeklarasikan bahwa class Person adalah Entity class. @Entity hanya bisa dipasangkan dengan deklarasi class saja.

@Table digunakan untuk meletakkan definisi dan konfigurasi dari table yang akan dimapping dengan class Entity. Di dalam @Table terdapat attribute name yang digunakan untuk mendefinisikan nama table yang akan dimapping dengan class Person yaitu T_PERSON.

@Id digunakan untuk menandakan sebuah property sebagai primary key. Konsekuensinya adalah nilai property harus unique dan tidak boleh null. @Id harus ada di salah satu property Entity class, kalau tidak ada hibernate akan teriak-teriak bahwa syarat entity tidak dipenuhi. Bagaimana jika ingin punya table yang tidak mempunyai primary key tapi ingin dijadikan Entity? Tidak bisa. Setiap Entity akan dimapping ke table dan tablenya harus mempunyai primary key, kalau tidak ada primary key table tersebut tidak bisa dimapping ke Entity class, biasanya dimapping ke @Component, apa itu @Component? Akan dibahas di bagian, bagian berikutnya, sabar ya.

@GeneratedValue digunakan berpasangan dengan @Id untuk menandakan primary digenerate secara otomatis oleh database. Di MySQL @GeneratedValue akan diterjemahkan dengan menandai kolom ID sebagai auto_increment. @GeneratedValue mempunyai dua attribute. Attribute pertama adalah strategy, yang digunakan untuk menentukan bagaimana primary key akan digenerate. Attribute kedua adalah generator, nilai attribute ini berupa nama generator untuk id-nya, bisa juga diisi dengan nama Sequence kalau menggunakan RDBMS yang mendukung sequence, seperti Oracle. Topik ini akan dibahas panjang lebar di bab Hibernate Mapping.

@Column digunakan untuk mengatur struktur kolom. Berikut ini attribute apa saja yang mungkin digunakan dalam @Column :

- name : nama kolom, nama kolom default sama dengan nama property
- unique : menambahkan unique constraint agar tidak ada 2 baris yang sama
- nullable : mengijinkan nilai column bisa diisi null atau tidak
- insertable : apakah kolom ini diikuti dalam query insert apa tidak
- updatable : apakah kolom ini diikuti dalam query update apa tidak
- columnDefinition : override sql DDL untuk kolom ini, tidak disarankan untuk digunakan karena tidak portable untuk semua database. Setiap query DDL untuk database berbeda akan mempunyai sql yang berbeda pula.
- Table : mendefinisikan table target, default adalah table yang ada di @Entity
- length : panjang data dari kolom ini
- precision : menyimpan berapa panjang angka pecahan di belakang koma dari angka desimal
- scale : menyimpan panjang digit angka desimal

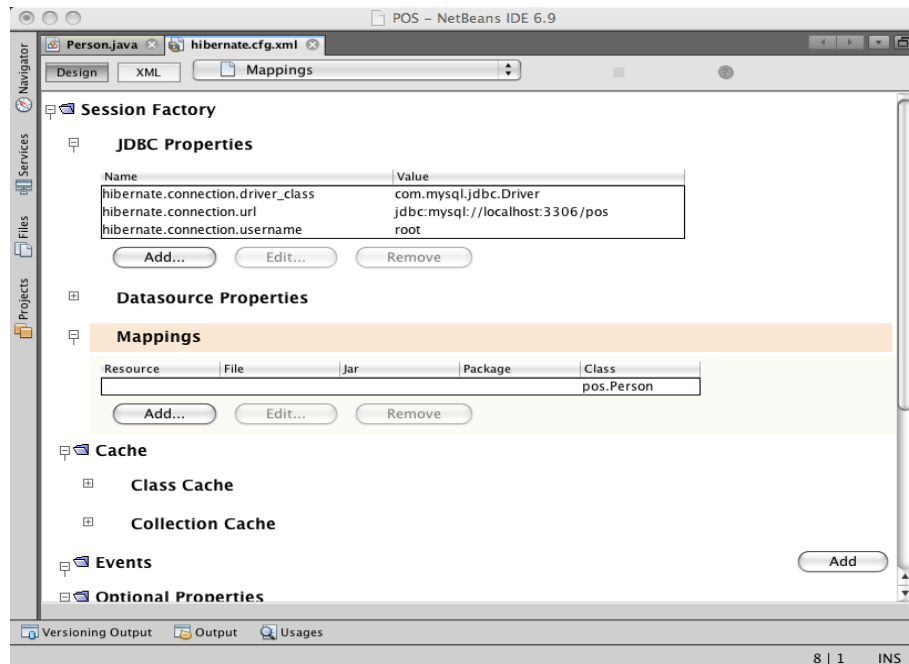
Proses mapping di atas sangat sederhana dan belum mencakup semua mapping feature di hibernate secara lengkap. Untuk sementara proses mapping dibatasi hanya untuk mapping sederhana, kemudian dilanjutkan ke konfigurasi hibernate hingga setting spring ORM. Setelah mengetahui feel dari hibernate, baru akan dibahas mapping yang lebih advance lagi.

Konfigurasi Hibernate

Setelah selesai membuat Entity class dan annotation mapping, langkah berikutnya adalah membuat konfigurasi hibernate : hibernate.cfg.xml

Membuat hibernate.cfg.xml dengan NetBeans sangat mudah, pilih menu File -> New, kemudian di dalam jendela pilih Hibernate di sebelah kiri dan Hibernate Configuration Wizard di sebelah kanan.

Jendela berikutnya meminta untuk memilih connection ke database yang akan digunakan. Jika belum ada silahkan pilih New Database Connection untuk membuat koneksi ke database yang baru. Hibernate.cfg.xml harus berada dalam classpath ketika aplikasi di jalankan, cara paling gampang yaitu dengan menyimpan hibernate.cfg.xml di dalam src folder. Klik OK, berikutnya jendela visual editor untuk hibernate.cfg.xml akan muncul seperti gambar di bawah ini :



Yang visual-visual sudah kuno dan gak macho, jadi kita buka jendela XML kemudian edit hibernate.cfg.xml menjadi seperti berikut ini :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
    <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/pos</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.hbm2ddl.auto">create</property>
    <property name="hibernate.show_sql">true</property>
    <property name="hibernate.format_sql">true</property>
    <mapping class="com.googlecode.projecttemplate.pos.model.Person"/>
  </session-factory>
</hibernate-configuration>
```

Sekarang kita bahas satu persatu konfigurasi di atas.

Hibernate.cfg.xml dimulai dengan tag <hibernate-configuration> kemudian diikuti dengan tag <session-factory> di dalam tag session-factory terdapat tag-tag property dan tag-tag mapping. Tag property mendefinisikan hal-hal umum dalam konfigurasi Hibernate, sedangkan tag mapping digunakan untuk mendaftarkan semua class Entity.

- hibernate.dialect digunakan untuk mendefinisikan database apa beserta dialect khusus dari datababase tersebut. Misalnya MySQLInnoDBDialect atau OracleDialect. Dengan menggunakan hibernate, kalau mau ganti database tinggal mengganti dialect ini maka aplikasi akan jalan dengan sangat lancar, tidak perlu melakukan perubahan signifikan.
- hibernate.connection.driver_class mendefinisikan driver yang dipakai untuk database.
- hibernate.connection.url berisi JDBC url connection ke database server.
- hibernate.connection.username dan hibernate.connection.password berisi informasi username dan password ke database server.
- hibernate.hbm2ddl.auto digunakan untuk memerintahkan hibernate mengenerate table-

table yang akan dimapping secara otomatis. Isi dari konfigurasi ada beberapa, antara lain:

- create semua table yang sudah ada akan didrop dan dibuat table-table baru.
- update table yang lama tidak didrop hanya dirubah seperlunya saja dengan menggunakan alter table.
- validate table yang ada tidak diotak-atik, hanya kalau antara mapping dan table yang sudah ada tidak sinkron, maka hibernate akan mengeluarkan error.
- create-drop setiap kali hibernate ditutup semua table akan didrop dan dibikin ulang ketika hibernate dibuka kembali.
- hibernate.show_sql digunakan untuk memerintahkan hibernate menampilkan query yang digenerate oleh Hibernate dalam jendela Output ketika aplikasi berjalan.
- hibernate.format_sql sql yang ditunjukkan dalam Output diformat agar tampilanya bagus dan mudah dibaca

Sekarang hanya ada satu tag mapping yang isinya class Person yang sudah kita buat sebelumnya, jika ada lebih dari satu class Entity dalam project, buat lagi tag mapping untuk mendaftarkan semua class Entity.

Menjalankan Hibernate

Setelah class Entity dan hibernate.cfg.xml sudah selesai dibuat, saatnya untuk menjalankan Hibernate dan mencoba menyimpan atau menampilkan data dari database.

Buat class baru, namakan saja MainHibernate, kemudian ketik kode di bawah ini :

```
public class MainHibernate {
    public static void main(String[] args) {
        SessionFactory sessionFactory = Configuration().buildSessionFactory();

        Person p = new Person();
        p.setName("dian");
        p.setPassword("pwddian");

        Session session = sessionFactory.openSession();
        try {
            session.beginTransaction();
            session.save(p);
            session.getTransaction().commit();
        } catch (HibernateException hibernateException) {
            session.getTransaction().rollback();
        }
        session.close();

        session = sessionFactory.openSession();
        Query query = session.createQuery("from Person");
        List<Person> persons = query.list();
        for (Person person : persons) {
            System.out.println("id : " + person.getId());
            System.out.println("name : " + person.getName());
            System.out.println("password : " + person.getPassword());
        }

        session.close();
        sessionFactory.close();
    }
}
```

```
}  
}
```

Sebelum bisa menjalankan kode di atas, anda harus menambahkan library Hibernate JPA ke project, kemudian klik kanan file, pilih Run. Setelah proses run selesai, table T_PERSON akan dibuat di database dan satu baris data akan diinsert. Silahkan cek database untuk mengetahui apakah kode sukses dieksekusi atau tidak.

Nah terlihat bahwa kode sangat rapi, untuk proses insert tidak perlu membuat query insert, table digenerate secara otomatis, query menjadi pendek dan abstraksi OOP dari aplikasinya memudahkan kode dimengerti / dibaca orang lain.

Class-class Penting dalam Hibernate

SessionFactory adalah jantung dari Hibernate. SessionFactory hanya dibuat satu kali dan cuma satu object untuk setiap aplikasi. Pada waktu aplikasi start, SessionFactory akan dibuat dan akan ditutup tepat pada waktu aplikasi diakhiri. Dalam JDBC posisi SessionFactory ini setara dengan Connection.

Session adalah class yang dibuat sebelum proses eksekusi perintah-perintah yang berurusan dengan database. Session mempunyai daur hidup yang sangat pendek, seperti terlihat dalam kode di atas, setelah proses insert dilaksanakan maka session segera ditutup, proses query di bawahnya menggunakan object session yang berbeda. Session harus diperlakukan seperti ini, jangan sampai dalam satu aplikasi hanya membuka satu session kemudian digunakan di semua bagian aplikasi. Kenapa tidak boleh seperti itu? Karena kalau proses ke database yang dilakukan oleh session gagal, maka object session ini menjadi tidak synchronized dengan database, untuk menghindari hal seperti ini sebaiknya object session segera ditutup ketika satu atomic activity selesai dijalankan.

Session mempunyai reference ke class Transaction yang mengatur proses transaction. Session mempunyai kemampuan untuk memulai, mengcommit dan merolback transaction. Kemungkinan besar kode aplikasi kita hanya akan berisi class Session saja, sedangkan SessionFactory hanya ada dalam konfigurasi saja.

Query adalah class yang mengimplementasikan query spesifik hibernate yaitu Hibernate Query Language. HQL ini nantinya akan diterjemahkan ke query yang spesifik ke database sesuai dengan settingan hibernate.dialect.

Menggunakan Hibernate dengan Spring ORM

Spring adalah Dependency Injection (DI) atau Inversion of Control (IoC). Tugas utama Spring adalah manage object-object yang dibutuhkan dalam aplikasi. Misalnya SessionFactory dan Session adalah object-object yang diperlukan untuk bekerja dengan Hibernate, Spring dapat membantu manage object-object ini sehingga kode infrastruktur (boiler code) dapat dikurangi secara signifikan karena dihandle oleh Spring.

Kode infrastruktur (boiler plate code) contohnya adalah membuka dan menutup Session, menangani transaction seperti transaction begin, commit dan rollback. Spring akan mengambil alih tanggung jawab untuk manage kode infrastruktur ini dari tangan developer, hal ini sangat strategis karena developer seringkali lupa untuk menulis kode infrastruktur yang dapat mengakibatkan memory leak atau merusak konsistensi data.

Peranan utama Spring dalam aplikasi desktop cukup besar, salah satu tugas utama Spring adalah manage Hibernate agar developer tidak perlu secara manual harus membuat kode untuk membuka menutup session (transparent Session management) dan menangani transaction (declarative transaction). Transaction yang ditangani secara manual dengan kode disebut gaya pemrograman Programatic Transaction, sedangkan gaya pemrograman yang menyerahkan penanganan transaction ke framework disebut Declarative Transaction. Bagaimana perbandingan kedua gaya ini akan dibahas lebih lanjut di bagian berikutnya.

Spring mempunyai feature Aspect Oriented Programming (AOP) yang menjadi tulang punggung dibalik implementasi declarative transaction dan transparent Session management.

AOP memungkinkan Spring menyisipkan kode untuk memulai transaction kemudian membuka session pada saat sebuah method dalam service akan dieksekusi, kemudian disisipkan pula kode untuk merollback dan menutup session ketika method dalam service muncul exception, dan setelah method selesai dilaksanakan Spring menyisipkan kode untuk mengcommit transaction plus menutup session. Semua proses penyisipan kode ini dilakukan secara runtime dan transparan terhadap developer. Sehingga developer cukup berkonsentrasi untuk mengimplementasikan bussiness logic tanpa harus khawatir lupa mengcommit transaction atau menutup Session.

Hibernate DAO menggunakan Spring

Langkah berikutnya setelah mapping Entity Person dan hibernate configuration selesai dibuat adalah membuat DAO dan Service. Kedua pattern ini mengikuti pattern yang sudah diterangkan dalam bagian JDBC, tidak perlu diterangkan secara panjang lebar tentang teori dan kegunaanya.

Hibernate DAO dengan bantuan Spring menjadi sangat rapih karena tidak memerlukan kode untuk manage Session, proses pembukaan dan penutupan session dilakukan oleh Spring secara transparan.

Sebelum proses pembuatan DAO dimulai, persiapkan dulu project dengan menambahkan library Spring Framework versi 2.5.6. Kalau anda menggunakan NetBeans versi 6.8 dan ke bawah, silahkan download dahulu Spring Framework dari <http://www.springsource.org/download>, Spring library yang dibawa NetBeans 6.9 mempunyai versi 2.5.6 dan versi 3.0, versi Spring ini tidak mempunyai beberapa class yang diperlukan dalam buku ini.

Buat class PersonDao di package com.googlecode.projecttemplate.pos.dao kemudian ketik kode berikut ini :

```
@Component
public class PersonDao {
    @Autowired private SessionFactory sessionFactory;
    public Person save(Person person){
        sessionFactory.getCurrentSession().saveOrUpdate(person);
    }
    public Person delete(Person person){
        sessionFactory.getCurrentSession().delete(person);
    }
    public Long count(){
        return (Long) sessionFactory.getCurrentSession()
            .createQuery("select count(*) from Person p")
            .uniqueResult();
    }
    public Person getById(Long id){
        return (Person) sessionFactory.getCurrentSession()
            .createQuery("from Person p where p.id=:id")
            .setParameter("id", id)
            .uniqueResult();
    }
    public List<Person> getAll(){
        return sessionFactory.getCurrentSession()
            .createQuery("from Person p")
            .list();
    }
    public List<Person> getAll(int start, int num){
        return sessionFactory.getCurrentSession()
            .createQuery("from Person p")
            .setFirstResult(start)
            .setFetchSize(num)
            .list();
    }
}
```

Kode di atas memperlihatkan annotation dari Spring, yaitu @Component, annotation ini digunakan untuk menandai suatu class akan dimanage oleh Spring, istilahnya adalah Spring Bean. Artinya class DAO ini proses pembuatan object-nya akan dilakukan oleh Spring. Object dari DAO akan dibuat satu dan hanya satu saja, pattern ini disebut Singleton. Pattern Singleton menyatakan bahwa sebuah class hanya mempunyai satu instance / object. Semua class yang masuk dalam category Spring Bean dapat diinject ke Spring Bean lain dengan menggunakan annotation @Autowired.

Annotation kedua dari Spring adalah @Autowired, annotation ini digunakan untuk menandai sebuah property harus diinject oleh Spring dan diambil dari kumpulan Spring Bean. Seperti dalam contoh DAO ini, Spring akan berusaha mencari instance dari sessionFactory di dalam koleksi Spring Bean. Kalau ketemu, instance akan diletakkan dalam property tersebut, kalau tidak ketemu Spring akan mengeluarkan exception.

Di chapter sebelumnya, pada bagian JDBC kode untuk membuat DAO, Service dan connection kita harus rangkai sendiri. DAO diinstantiasi, Service juga diinstantiasi, kemudian proses pembuatan koneksi juga dilakukan di dalam kode Java. Nah proses merangkai kode ini bisa dilakukan dengan elegan menggunakan feature Dependency Injection dari Spring. Setiap class yang ditandai dengan @Component akan diinstantiasi oleh spring, kemudian semua dependency dari class @Component yang ditandai dengan @Autowired diambil (dirangkai) dari kumpulan Spring Bean. Nanti akan kita lihat lebih banyak lagi penggunaan Spring Bean di topik yang lebih advance.

Generic DAO

Bentuk DAO cukup repetitif, terutama fungsi-fungsi dasar seperti save, delete, getById dan getAll. Proses pembuatan DAO bisa dipersingkat dengan membuat generic DAO, kemudian semua DAO akan extends generic DAO ini dan menambahkan generic classnya. Tidak perlu banyak dijelaskan lagi, langsung saja kita lihat bentuknya

```
public class BaseDaoHibernate<T> {

    @SuppressWarnings("unchecked")
    protected Class domainClass;

    @Autowired
    protected sessionFactory sessionFactory;
    @SuppressWarnings("unchecked")
    public BaseDaoHibernate() {
        this.domainClass = (Class) ((ParameterizedType)
            getClass().getGenericSuperclass())
            .getActualTypeArguments()[0];
    }
    public T save(T domain) {
        sessionFactory.getCurrentSession().saveOrUpdate(domain);
    }
    @SuppressWarnings("unchecked")
    public T getById(Long id) {
        return (T) sessionFactory.getCurrentSession().get(domainClass, id);
    }
    public T delete(T domain) {
        sessionFactory.getCurrentSession().delete(domain);
    }
    @SuppressWarnings("unchecked")
    public Long count() {
        List list = sessionFactory.getCurrentSession().createQuery(
            "select count(*) from " + domainClass.getName() + " x").list();
        Long count = (Long) list.get(0);
        return count;
    }
    @SuppressWarnings("unchecked")
```

```

public List<T> getAll() {
    return sessionFactory.getCurrentSession().createQuery("from " +
        domainClass.getName())
        .list();
}
@SuppressWarnings("unchecked")
public List<T> getAll(int start, int num) {
    return sessionFactory.getCurrentSession().createQuery("from " +
        domainClass.getName())
        .setFirstResult(start).setMaxResults(num)
        .list();
}
}

```

Nah setelah generic DAO di atas udah jadi, PersonDao yang sebelumnya harus menulis satu-satu kodenya tinggal extends BaseDaoHibernate di atas dan mengganti T dengan class Person. Seperti di bawah ini :

```

@Component
public class PersonDao extends BaseDaoHibernate<Person>{

}

```

Ringkas sekali bukan? Dari sekian puluh baris hanya menjadi empat baris saja. Fungsi-fungsi DAO yang ada di dalam BaseDaoHibernate di atas sangat terbatas untuk fungsi-fungsi dasar dari operasi data, besar kemungkinan akan dibutuhkan fungsi-fungsi yang jauh lebih kompleks, fungsi-fungsi yang kompleks dan spesifik ini harus dibuat manual.

Spring Service

Setelah menyelesaikan DAO, langkah berikutnya adalah membuat Spring Service. Berbeda dengan DAO yang merupakan class, Spring Service sebaiknya dibuatkan interface, kemudian diimplementasikan oleh class. Misalnya untuk contoh Entity Person, buat PersonService yang merupakan interface dan PersonServiceImpl yang merupakan class implementasi dari PersonService. Di bagian selanjutnya akan diterangkan lebih lanjut tentang magic yang ada dalam Spring Service ini dan kenapa sebaiknya Spring Service dibuatkan interface dan implementasinya.

Sama halnya dengan Service yang sudah dibuat di bab JDBC, Spring Service ini tugasnya untuk menangani transaksi dan tempat untuk menulis bussiness process logic. Di contoh PersonService logic bussiness processnya masih sangat sederhana, sehingga terlihat cuma sebagai kepanjangan tangan DAO. Di kasus yang sebenarnya, dalam satu method di dalam Spring Service biasanya terdapat banyak sekali kode untuk melakukan banyak hal, mengupdate beberapa table sekaligus. Service dapat memanfaatkan kode yang sudah ada di dalam DAO, dan sebaiknya di dalam Service tidak menulis kode yang langsung mengakses SessionFactory, semua kode yang mengakses SessionFactory sebaiknya diletakkan di dalam DAO. Hal ini dimaksudkan untuk mengelola kode dengan rapi dan disiplin.

Transaction management dengan menggunakan Spring Service berbeda dengan service yang kita buat di bab JDBC. Spring Service tidak perlu manage transaksi secara manual, dengan memulai transaksi, commit dan rollback. Transaction Management di Spring Service dilaksanakan secara transparan, artinya kita sebagai programmer Spring tidak perlu tahu detail bagaimana Spring Service menangani transaction, kita hanya perlu tahu bahwa method-method dalam Spring Service yang ditandai @Transactional akan memulai transaction ketika masuk ke method, mencommit sesaat setelah keluar dari method dan merollback transaction kalau ada exception ketika method tersebut dieksekusi.

Teknik penanganan transaction secara transparan ini disebut sebagai Declarative Transaction, sedangkan penanganan transaction secara manual seperti di JDBC Service, disebut sebagai Programatic Transaction. Di ekosistem Java, hanya EJB dan Spring saja yang mempunyai feature Declarative Transaction. Spring Service mempunyai karakteristik yang mirip dengan Stateless Session Bean dari EJB, yaitu method-methodnya transactional.

Berikut ini contoh PersonService di package com.googlecode.projecttemplate.pos.service dan class PersonServiceImpl di package com.googlecode.projecttemplate.pos.service.impl

```
public interface PersonService {  
  
    void save(Person person);  
    void delete(Person person);  
    Long count();  
    Person getPerson(Long id);  
    List<Person> getPersons();  
    List<Person> getPersons(int start, int num);  
  
}
```

Setiap method dari PersonService di atas diimplementasikan oleh class PersonServiceImpl

```
@Service("personService")  
@Transactional(readOnly=true)  
public class PersonServiceImpl implements PersonService{  
  
    @Autowired private PersonDao personDao;  
  
    @Transactional(readOnly=false)  
    public void save(Person person) {  
        personDao.save(person);  
    }  
    @Transactional(readOnly=false)  
    public void delete(Person person) {  
        personDao.delete(person);  
    }  
    public Long count() {  
        return personDao.count();  
    }  
    public Person getPerson(Long id) {  
        return personDao.getById(id);  
    }  
    public List<Person> getPersons() {  
        return personDao.getAll();  
    }  
    public List<Person> getPersons(int start, int num) {  
        return personDao.getAll(start, num);  
    }  
}
```

Dalam class PersonServiceImpl di atas, terdapat beberapa annotation dari Spring. Annotation pertama adalah @Service yang digunakan untuk menandai bahwa class ini adalah Spring Service, terdapat satu parameter yaitu "personService" yang merupakan nama dari Spring Service ini. @Service mempunyai fungsi yang sama dengan @Component dalam class DAO, yaitu menandai bahwa class ini nantinya akan diinstansiasi oleh Spring dan instansiasi-nya disebut sebagai Spring Bean.

@Transactional digunakan untuk menandai method-method dalam Spring Service merupakan satu area transaksi. Parameter readOnly menandai apakah method tersebut dapat melakukan operasi insert/update/delete dalam database. @Transactional dapat digunakan untuk mendandai class sehingga semua method dalam class tersebut secara default menggunakan konfigurasi tersebut, kemudian setiap method dapat juga ditandai @Transactional yang mengoveride @Transactional yang diletakkan di class.

Semua method yang melakukan proses manipulasi data harus ditandai dengan @Transactional(readOnly=false) atau cukup @Transactional sebagai bentuk pendeknya. Kalau

tidak ditandai dengan `@Transactional` maka Spring akan mengeluarkan error bahwa transaksi dalam method tersebut bersifat `readonly` dan tidak diperkenankan untuk melakukan manipulasi data dalam database.

Declarative Transaction vs Programatic Transaction

Setelah melihat implementasi Spring Service dan JDBC Service di bab-bab sebelumnya, kita jadi tahu ternyata ada dua jenis variasi Transaction Management. Cara yang pertama adalah Programatic Transaction, dimana proses penanganan transaksi dilakukan secara manual, programmer harus menulis kode untuk memulai transaksi, kemudian `commit` transaksi dan kalau ada `exception` melakukan `rollback`. Contoh lengkapnya bisa dilihat pada bab yang membahas JDBC di bagian Service Pattern.

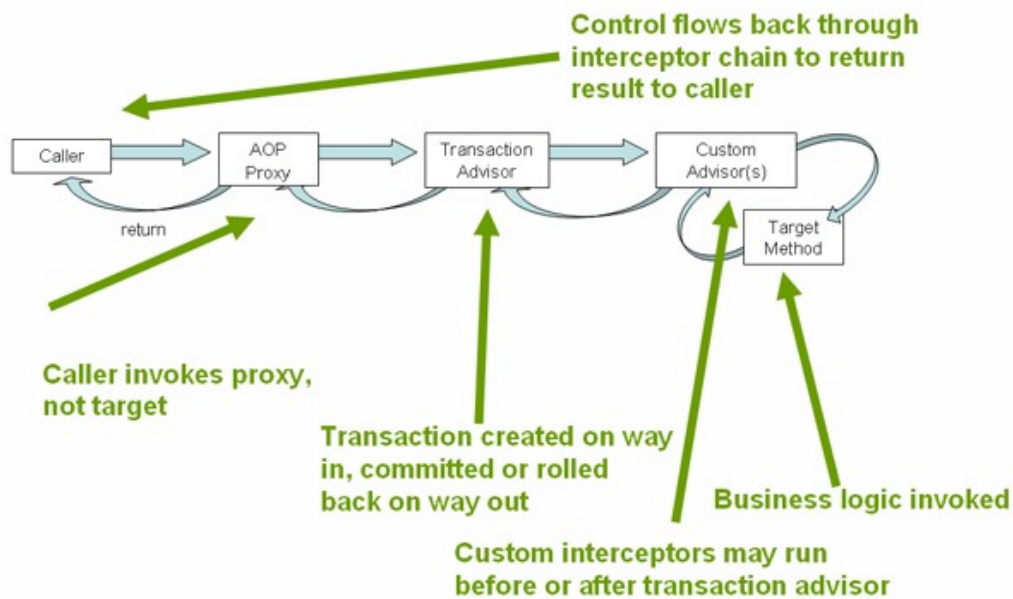
Programatic Transaction sangat mudah dipahami karena konsepnya ada di setiap bahasa pemrograman. Kode yang ditulis untuk melakukan Penanganan transaksi secara manual ini disebut dengan boiler plate code atau kode infrastruktur, kode jenis ini tidak berkontribusi sama sekali terhadap bussiness proses aplikasi. Dengan kata lain, kodenya harus ditulis tapi nilai tambahnya kosong. Semenjak awal, teknologi di ekosistem Java berusaha membuat teknologi agar proses penanganan transaksi tidak lagi dilakukan secara manual, sehingga munculah jenis kedua proses penanganan transaksi, yaitu Declarative Transaction.

Framework yang mempunyai feature Declarative Transaction mengambil alih proses penanganan transaksi dari kode yang ditulis manual oleh programmer ke framework itu sendiri. Programmer cukup mendeklarasikan bahwa method ini adalah transactional, dengan kata lain setiap method yang dideklarasikan sebagai transactional tersebut akan dieksekusi, framework akan memulai transaction, ketika eksekusi methodnya akan berakhir framework akan melakukan `commit`, dan jika terdapat `exception` di tengah-tengah method maka framework akan `rollback` transaksi. Dengan begitu, programmer dapat berkonsentrasi di proses bisnis aplikasi tanpa harus direpotkan dengan boiler plate code untuk menangani transaksi.

EJB dan Spring adalah framework yang mempunyai feature Declarative Transaction. EJB lewat feature Container Managed Transaction (CMT) membuat semua method dalam Stateless Session Bean (SLSB), Statefull Session Bean (SLSB) dan Message Driven Bean (MDB) bersifat transactional. Dalam Spring, method dalam Spring Service yang ditandai dengan `@Transactional` bersifat transactional.

Spring menggunakan AOP untuk mengimplementasikan Declarative Transaction. Lihat gambar di bawah ini, di kotak paling kiri adalah kode program yang memanggil service. Spring Service yang dipanggil sebenarnya adalah AOP Proxy, bukan `PersonServiceImpl`. AOP Proxy adalah class yang dibuat `on-the-fly` pada waktu aplikasi berjalan, didalam class ini disisipkan Transaction Advisor untuk menangani transaction begin, `commit` dan `rollback`. Jadi sebenarnya kode untuk transaction begin, `commit` dan `rollback` ada dalam AOP Proxy ini tanpa perlu ditulis manual, alias dibuat oleh Spring. Custom Advisor bisa ditambahkan dalam Spring Service, biasanya digunakan untuk proses logging agar kode log tidak bercampur dengan kode bussiness process.

Pada akhirnya kode dalam `PersonServiceImpl` akan dieksekusi, kemudian `return` dari eksekusi kode tersebut dikembalikan ke pemanggilnya. Di bagian berikutnya kita akan membahas bagaimana cara membuat aplikasi

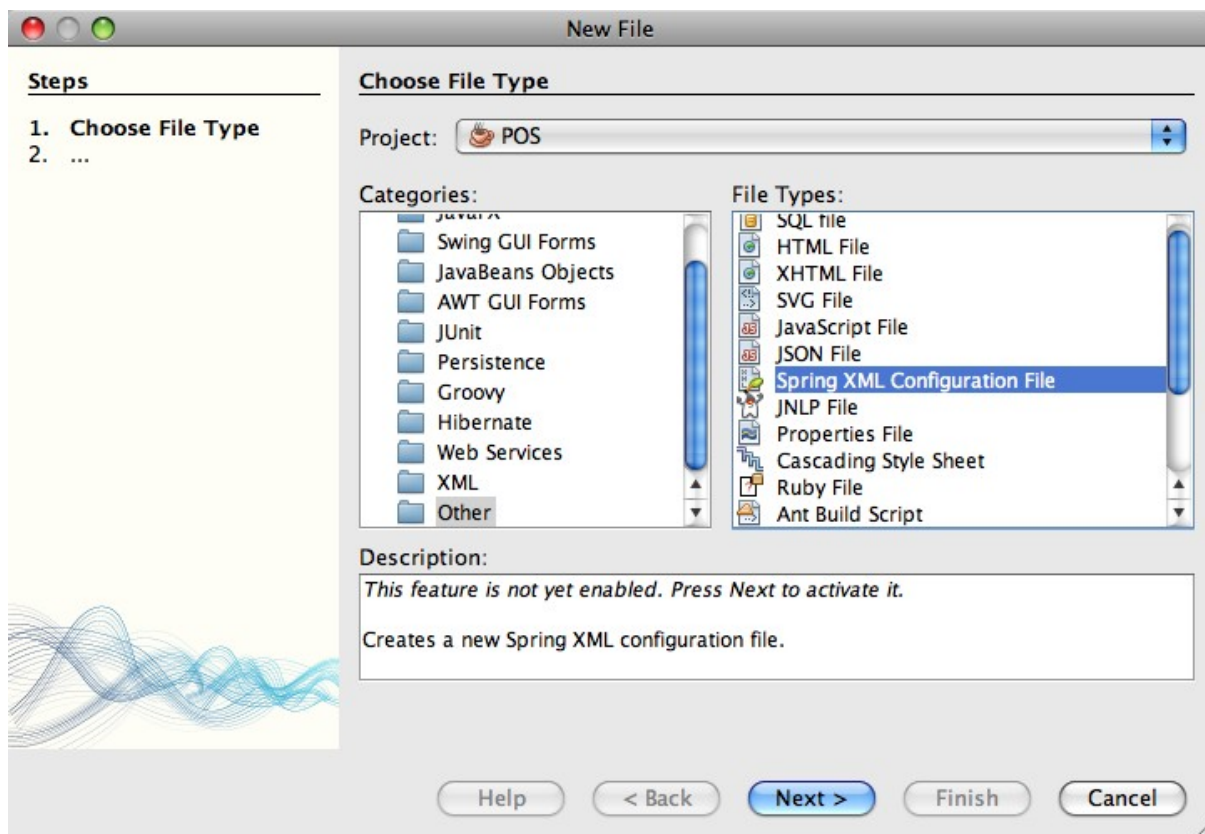


Spring Transaction Management
<http://static.springsource.org/spring/docs/3.0.3.RELEASE/spring-framework-reference/html/transaction.html>

Spring Configuration dan Spring Application Context

Setelah Entity, hibernate configuration, DAO dan Spring Service selesai dibuat, langkah berikutnya adalah membuat Spring configuration, atau biasa disebut dengan Spring Application Context Configuration. Bentuknya sama dengan hibernate configuration yaitu file XML. Di dalam Spring Application Context ini akan didefinisikan class-class yang dimanage oleh spring, sering disebut dengan Spring Bean. Apa sih sebenarnya Spring Bean itu? Spring Bean pada dasarnya hanya object biasa, tetapi punya keistimewaan karena berada dalam Spring Application Context, sehingga object ini bisa digunakan dalam proses Deppendency Injection. Contoh kongkritnya bisa dilihat dari class DAO, class ini membutuhkan SessionFactory agar bisa digunakan, kata lain class DAO dipenden (tergantung) terhadap SessionFactory, nah proses memasukkan (inject) SessionFactory ke dalam DAO dilakukan menggunakan annotation @Autowired. Mekanisme ini bisa dilakukan jika SessionFactory didefinisikan dalam Spring Application Context Configuration, DAO juga ditandai dengan @Component dan dikonfigurasi agar @Component ini bisa dikenali oleh Spring Application Context.

Konfigurasi Spring Application Context dapat di buat dengan mudah menggunakan NetBeans, pilih menu File -> New File kemudian pilih node Others dan akhirnya pilih Spring XML Configuration File seperti di dialog berikut ini :



Klik Next dan lanjutkan ke halaman berikutnya, di halaman ini terdapat dialog untuk memilih Spring namespace apa saja yang akan disertakan dalam konfigurasi, pilih 3 buah namespace: context, tx dan p. Lanjutkan ke halaman berikutnya dengan menekan next, di halaman ini anda akan diminta untuk memberikan nama file konfigurasi, beri nama file-nya applicationContext.xml setelah itu tekan Finish.

Setelah applicationContext.xml selesai dibuat, modifikasi filenya menjadi seperti ini :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:p="http://www.springframework.org/schema/p"

  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

  <context:component-scan base-package="com.googlecode.projecttemplate.pos"/>
  <context:annotation-config/>
  <tx:annotation-driven/>

  <context:property-placeholder location="classpath:jdbc.properties"/>
  <bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
```

```

    p:driverClassName="${jdbc.driver}"
    p:url="${jdbc.url}"
    p:username="${jdbc.username}"
    p:password="${jdbc.password}"/>

    <bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"
    p:dataSource-ref="dataSource"
    p:configLocation="classpath:hibernate.cfg.xml"/>

    <bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager"
    p:sessionFactory-ref="sessionFactory"/>
</beans>

```

Kalau anda menggunakan NetBeans versi 6.9, kemungkinan besar file di atas akan menggunakan Spring 3.0, perbedaan dalam konfigurasi hanya ada di xmlns saja. Buku ini menggunakan Spring 2.5.6 sehingga anda perlu mengganti xmlns yang berakhiran 3.0.xsd menjadi 2.5.xsd seperti dalam konfigurasi di atas.

Mari kita bahas satu persatu konfigurasi di atas.

- **context:component-scan** digunakan untuk mendefinisikan base-package dimana class-class yang dianotasi @Component, @Repository dan @Service berada, di contoh-contoh sebelumnya class yang dimaksud adalah DAO dan Service. Kalau hanya ingin memasukkan package yang ada class DAO dan servicenya bisa menggunakan tanda koma (,) untuk memisahkan package-nya, seperti contoh di bawah ini

```

<context:component-scan base-package =
"com.googlecode.projecttemplate.pos.dao,com.googlecode.projecttemplate.pos.service.impl"/>

```

- **context:annotation-config** digunakan untuk memberitahu Spring bahwa dalam project ini akan digunakan annotation untuk mengkonfigurasi.
- **tx:annotation-driven** digunakan untuk memberitahu Spring bahwa konfigurasi Declarative Transaction akan menggunakan annotation @Transactional
- **context:property-placeholder** digunakan untuk mendaftarkan file-file property yang akan digunakan untuk menyimpan konfigurasi, misalnya dalam hal ini adalah konfigurasi koneksi ke database. Semua property yang ada bisa diakses dalam application context ini menggunakan sintaks \${nama_property}. Misalnya dalam Application Context ini terdapat \${jdbc.driver}, konfigurasi ini menghendaki ada property jdbc.driver di dalam file jdbc.properties. Awalan classpath: menerangkan bahwa file jdbc.properties akan berada dalam classpath. Jika ada file properties lain yang digunakan, pisahkan daftar file-nya dengan tanda koma (,).

Agar konfigurasi ini berjalan, maka perlu dibuat file jdbc.properties di dalam folder src atau di dalam <default-package>. Langkahnya: pilih menu File -> New File, kemudian pilih Other dan akhirnya pilih Properties File, beri nama jdbc.properties. Lalu isikan kode berikut ini :

```

jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/pos
jdbc.username=root
jdbc.password=

```

nilai jdbc.password sengaja dikosongkan karena user root tidak mempunyai password.

- **bean** digunakan untuk membuat Spring Bean, alias meminta spring untuk menginstansiasi class yang ada dalam attribute class. Contoh dalam konfigurasi di atas, akan dibuat Spring Bean dengan id datasource yang diinstansiasi dari class org.springframework.jdbc.datasource.DriverManagerDataSource. Kemudian di property

driverClassName dari class tersebut diisikan nilai dari jdbc.driver, dan seterusnya. Kalau konfigurasi di atas ditulis dalam kode Java, maka padananya adalah sebagai berikut

```
DriverManagerDataSource datasource = new DriverManagerDataSource();
datasource.setDriverClassName("com.mysql.jdbc.driver");
datasource.setUrl("jdbc:mysql://localhost:3306/pos");
datasource.setUsername("root");
datasource.setPassword("");
```

Setiap kali ada tag <bean> di Spring Application Context, anda harus membayangkan bahwa sebenarnya yang dilakukan oleh Spring adalah menginstansiasi sebuah class, kemudian menset property yang dibutuhkan oleh object yang baru saja diinstansiasi. Nah proses instansiasi plus menset property yang dibutuhkan oleh object inilah inti dari Spring, sangat sederhana kan?

Walaupun konsepnya sangat sederhana, namun efeknya sangat powerfull. Sekarang kalau misalnya ingin mengganti class dari datasource ke Apache Commons DBCP, maka kita tidak perlu mengganti kode Java, cukup mengganti konfigurasi Spring maka kita sudah mempunyai Connection Pooling. Dengan cara inilah Spring mewujudkan konsep modular, komponen dalam aplikasi bisa diganti, disubstitusi atau dibongkar pasang hanya dengan mengganti konfigurasi di dalam Spring Application Context ini, luar biasa.

- **bean** berikutnya yang dibuat adalah sessionFactory. Property yang diset untuk sessionFactory ini ada datasource yang diambil dari bean datasource, kemudian property configLocation digunakan untuk mendefinisikan file hibernate.cfg.xml. Prefix classpath: digunakan untuk memberitahu Spring bahwa file hibernate.cfg.xml akan berada di dalam classpath.
- **bean** terakhir adalah transactionManager. Bean ini digunakan oleh Spring untuk memenage transaction dari Hibernate. Parameter yang diperlukan adalah bean sessionFactory.

Setelah selesai membuat Spring Application Context, kode sudah siap dijalankan. Buat sebuah class, namakan MainSpring kemudian ketik kode berikut ini untuk mencoba menggunakan kode yang sudah dibuat:

```
public class MainSpring {
    public static void main(String[] args) {

        ApplicationContext appContext =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        PersonService personService =
            (PersonService) appContext.getBean("personService");

        Person person = new Person();
        person.setName("ifnu");
        person.setPassword("pwdifnu");
        personService.save(person);

        List<Person> persons = personService.getPersons();
        for (Person p : persons) {
            System.out.println("name:" + p.getName() + ", password:"+p.getPassword());
        }
    }
}
```

Sebelum menjalankan class di atas, tambahkan library Spring Framework 2.5.6.SEC01 ke dalam project, klik kanan file kemudian pilih run. Class ini akan menginstansiasi ApplicationContext yang diambil dari file konfigurasi applicationContext.xml, kemudian dari ApplicationContext tersebut kita ingin mendapatkan service bean, caranya dengan memanggil method getBean dan menggunakan string personService sebagai parameternya. Setelah service diambil dari ApplicationContext, kita bisa menggunakannya untuk menyimpan object Person yang baru saja dibuat menggunakan method save atau mengambil semua Person dari table T_PERSON menggunakan method getPersons.

Utility class untuk mengenerate table dan initial data

Lihat lagi file konfigurasi hibernate.cfg.xml, di dalam file tersebut ada property sessionFactory.hibernate.hbm2ddl.auto yang nilainya create-drop, property ini memberitahu sessionFactory untuk mendrop table yang sudah ada kemudian membuat table yang baru. Terkadang kita ingin mengenerate table dari awal karena ada perubahan struktur table, terkadang kita juga tidak ingin mendrop table-table yang sudah dibuat, nah berulang-ulang kali kita harus mengganti nilai sessionFactory.hibernate.hbm2ddl.auto dari create-drop menjadi none agar tablenya tidak digenerate ulang. Kadang kala cukup menyebalkan kalau sudah menyiapkan data untuk mengetest aplikasi tapi lupa mengganti nilai sessionFactory.hibernate.hbm2ddl.auto menjadi none, sehingga pada waktu aplikasi dijalankan semua table dicreate ulang.

Untuk menghindari kesalahan di atas, kita bisa secara permanen mengeset nilai sessionFactory.hibernate.hbm2ddl.auto menjadi none, kemudian membuat satu class utility GenerateTables untuk mengenerate table. Setiap kali ingin mengenerate table, cukup jalankan class ini. Di dalam class GenerateTables bisa juga diletakkan kode-kode untuk membuat initial data, misalnya data user atau data master.

Buat class GenerateTables di dalam package com.googlecode.projecttemplate.pos.util kemudian ketikkan kode seperti berikut ini.

```
public class GenerateTables {
    public static void main(String[] args) throws SQLException {
        AbstractApplicationContext appContext =
            new ClassPathXmlApplicationContext("classpath:applicationContext.xml");

        DataSource dataSource = (DataSource) appContext.getBean("dataSource");

        Configuration cfg =
            new AnnotationConfiguration().configure("hibernate.cfg.xml");
        Connection conn = dataSource.getConnection();
        new SchemaExport(cfg, conn).create(true, true);

        PersonService personService =
            (PersonService) appContext.getBean("personService");

        Person person = new Person();
        person.setName("ifnu");
        person.setPassword("pwdifnu");
        personService.save(person);

        System.exit(0);
    }
}
```

Sampai di sini, struktur utama dari project sudah tersedia, ada class Entity, DAO dan Service kemudian ada file konfigurasi hibernate.cfg.xml, applicationContext.xml dan jdbc.properties. Kode untuk menjalankan project juga sudah dibuat. Bagian selanjutnya akan membahas Hibernate secara lebih mendalam, diawali dengan pembahasan tentang Hibernate Mapping, kemudian dilanjutkan dengan HQL.

Hibernate Mapping

Hibernate, seperti halnya semua framework O/R mapping, membutuhkan metadata yang digunakan sebagai informasi pemetaan dari class ke table database. Sebelum Java 5 metadata yang digunakan adalah file hbm.xml, masalah utama dengan hbm.xml adalah kesalahan mapping akan terlihat kalau aplikasi dijalankan, sedangkan IDE yang ada saat itu belum cukup canggih untuk bisa melihat kesalahan mapping yang terjadi di dalam hbm.xml, konfigurasi yang terpisah dalam file berbeda seperti ini juga menimbulkan masalah penurunan produktifitas, karena programmer harus bolak-balik melihat kode Java dan kode xml. Setelah Java 5 memperkenalkan annotation, Hibernate mengeluarkan Hibernate Annotation sebagai

alternatif hbm.xml. Annotation juga memudahkan IDE seperti NetBeans untuk membuat feature autocomplete dan pemeriksaan kesalahan.

Melihat kesuksesan Hibernate yang luar biasa, JCP akhirnya memutuskan untuk membuat teknologi standard ORM, yang kemudian diberi nama JPA (Java Persistence API). Seperti halnya semua standard di dalam Java, JPA ini pada dasarnya hanya kumpulan interface dan annotation, setiap vendor bisa membuat implementasi dari JPA ini. Reference implementation (RI) dari JPA adalah TopLink Essentials, semua vendor lain yang ingin membuat implementasi JPA bisa meneliti behaviour Toplink Essentials sebagai patokan. Hibernate juga mempunyai implementasi JPA yang disebut Hibernate EntityManager.

Dalam buku ini kita akan menggunakan Hibernate plus JPA annotation untuk basic mapping, serta beberapa annotation dari Hibernate Annotation. JPA annotation ditandai dengan package import javax.persistence sedangkan Hibernate Annotation ditandai dengan package org.hibernate.annotations. Kenapa masih harus menggunakan Hibernate Annotation? Karena pada dasarnya JPA itu hanya mempunyai feature mapping yang umum dipakai ORM, ada feature tertentu yang hanya ada di Hibernate Annotation, sehingga kita masih harus mencampur antara JPA dan Hibernate Annotation.

Selain annotation yang berbeda, JPA dan Hibernate juga mempunyai class-class yang berbeda. Jika di Hibernate ada SessionFactory maka di JPA ada EntityManagerFactory, di Hibernate ada Session di JPA ada EntityManager, dan seterusnya. Tidak perlu merasa bingung atau bimbang, JPA dan Hibernate mempunyai tujuan yang sama, tidak ada perbedaan essensial antara keduanya, anda bisa memilih salah satu tanpa merasa bahwa yang lain jauh lebih baik atau sebaliknya.

Entity dan Basic mapping

Bab sebelumnya sudah memperlihatkan basic mapping, kita akan bahas sekali lagi tentang basic mapping dalam bab ini, kalau anda merasa sudah cukup mengerti tentang basic mapping, silahkan lanjut ke bagian berikutnya.

Basic mapping adalah kumpulan annotation untuk memapping sebuah class / Entity ke table dalam database. Sebuah Entity minimum harus mempunyai dua buah annotation, yaitu @Entity dan @Id, kedua annotation ini wajib ada di dalam Entity. JPA annotation menggunakan konsep Configuration by Exception dimana ada aturan bahwa annotation lain yang tidak disebutkan akan diperlakukan secara default. Bagaimana konsep ini diterapkan dalam proses mapping? Mari kita lihat contoh berikut ini :

```
@Entity
public class Person implements Serializable {
    @Id
    private Long id;
    private String name;
    private String password;
    //getter setter
}
```

Mapping class Person di atas berbeda dengan mapping class Person yang ada di bab sebelumnya, semua annotation lain selain @Entity dan @Id dihilangkan, sehingga Hibernate akan menggunakan nilai default untuk mappingnya ke dalam table. Nilai defaultnya antara lain :

- nama tablenya akan sama dengan nama class, yaitu Person
- nama kolom akan sama dengan nama property
- panjang varchar untuk tipe String adalah 255
- primary key yang ditandai dengan @Id tidak digenerate, dengan kata lain kita harus set nilainya sebelum menyimpan dalam database.

@Table digunakan untuk mendefinisikan table yang akan dimapping dengan Entity, selain nama dari table yang didefinisikan dalam attribute name, @Table bisa menerima catalog dan schema. Attribute yang tidak kalah pentingnya adalah uniqueConstraints yang digunakan untuk mendefinisikan unique constraints dari table tersebut. Misalnya di table T_PERSON kolom nama harus unique, maka @Table di Entity person bisa diganti seperti di bawah ini :

```
@Table(name="T_PERSON",uniqueConstraints={@UniqueConstraint(columnNames="NAME")})
```


kalau kombinasi kolom NAME dan PASSWORD mempunyai nilai unique, @Table diganti seperti di bawah ini

```
@Table(name="T_PERSON",uniqueConstraints={@UniqueConstraint(columnNames={"NAME",
"PASSWORD"})})
```

Tipe data Date memerlukan informasi apakah data yang disimpan hanya tanggal, hanya waktu atau tanggal dan waktu, @Temporal digunakan untuk kebutuhan ini, @Temporal bisa berisi TemporalType.DATE, TemporalType.TIMESTAMP atau TemporalType.TIME. . Misalnya di class Person kita tambahkan satu kolom lagi untuk menampung informasi tanggal lahir, karena tanggal lahir hanya perlu informasi tanggal saja, maka mappingnya seperti ini :

```
@Temporal(TemporalType.DATE)
@Column(name="BIRTH_DATE")
private Date birthDate;
```

Terkadang kita perlu tipe data enum untuk menampung informasi berupa nilai pilihan yang terbatas, misalnya jenis kelamin, atau status perkawinan. Hibernate mendukung tipe data enum untuk digunakan dalam entity, annotation yang digunakan adalah @Enumerated. Kita bisa memilih untuk menyimpan string dari enum atau menyimpan urutannya, jika ingin menyimpan string dari enum gunakan EnumType.STRING, sedangkan EnumType.ORDINAL digunakan kalau ingin menyimpan urutan enumnya. Jika anda masih awam tentang enum, silahkan membaca lagi bab sebelumnya yang membahas tentang tipe data enum.

Sebagai contoh, buat enum MaritalStatus di package model seperti berikut ini

```
public enum MaritalStatus {
    SINGLE,MARRIED,DIVORCED;
}
```

Kemudian mapping class Person ditambah kode berikut ini :

```
@Enumerated(EnumType.STRING)
@Column(name="STATUS",length=20)
private MaritalStatus status;
```

Hibernate juga bisa menampung data binary seperti foto profile user ke dalam database, tipe data yang digunakan adalah byte array (byte[]), kemudian annotation @Lob (Large object) digunakan untuk menandai peroperty ini. Contohnya seperti berikut ini

```
@Lob
@Column(name="PICTURE")
private byte[] picture;
```

@Lob juga digunakan untuk menandai kolom dengan tipe data String yang mempunyai ukuran sangat panjang. MySQL varchar hanya sanggup menampung 255 huruf, jadi kalau punya kolom yang ingin mempunyai panjang lebih dari 255 harus menggunakan tipe data Text. Hibernate akan membuat kolom bertipe text kalau ada property String ditandai dengan @Lob

```
@Lob
@Column(name="REMARK")
private String remark;
```

Semua annotation di atas sudah cukup mengcover sebagian besar basic mapping di Hibernate. Bagian berikutnya kita akan membahas annotation @GeneratedValue yang digunakan bersamaan dengan @Id.

Id Generation

@GeneratedValue adalah annotation yang digunakan bersamaan dengan @Id, annotation @GeneratedValue menandai bahwa primary key akan digenerate oleh database. Nilai attribute strategy menentukan bagaimana caranya primary key akan digenerate oleh database, nilai attribute strategy antara lain:

- GenerationType.AUTO proses generate id tergantung database yang digunakan, berdasarkan databasanya hibernate akan memilih proses generate menggunakan TABLE, SEQUENCE atau IDENTITY. Pilihan AUTO ini yang paling aman dan portable ke semua database.

- GenerationType.TABLE proses generate id menggunakan satu buah table yang menyimpan nilai terakhir yang digunakan. Sebelum insert data, ambil satu baris dalam table sequence kemudian baca nilai terakhir dan tambahkan satu, setelah proses insert berhasil update nilai terbesarnya dengan nilai terakhir yang baru saja digunakan untuk insert. Konfigurasi lengkapnya seperti di bawah ini :

```
@GeneratedValue(strategy=GenerationType.TABLE, generator="PERSON_ID")
@TableGenerator(name="PERSON_ID", table="T_RUNNING_NUMBER",
pkColumnName="name", valueColumnName="COUNT",
pkColumnValue="PERSON_ID")
```

Konfigurasi di atas menggunakan sebuah table dengan nama T_RUNNING_NUMBER yang mempunyai kolom NAME dan COUNT seperti contoh di bawah ini. Setiap kali insert ke table T_PERSON nilai count dari PERSON_ID akan bertambah satu secara otomatis untuk mencatat id terakhir yang digunakan.

NAME	COUNT
PERSON_ID	123
PENJUALAN_ID	550

- GenerationType.SEQUENCE proses generate id berdasarkan sequence, oracle tidak mempunyai feature auto_increment, untuk mengenerate id oracle menggunakan sequence. Sebuah sequence dapat digunakan oleh beberapa table yang berbeda. Misalnya untuk beberapa entity digunakan sequence yang sama, maka bisa dideklarasikan dengan menyebutkan nama sequence-nya secara eksplisit seperti di bawah ini :

```
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="SEQ_STORE")
```

Sequence SEQ_STORE bisa digunakan di entity Person atau di entity yang lain. SEQUENCE mendukung preallocation untuk mengalokasikan sekian buah angka setiap kali increment. SEQUENCE adalah metode yang paling tidak portable dan hanya oracle, db2 dan postgres yang mendukung SEQUENCE. Performance SEQUENCE yang paling bagus karena dapat mengatasi masalah pemanggilan SEQUENCE secara bersamaan. Contoh SEQUENCE generation yang menggunakan preallocation seperti berikut :

```
@GeneratedValue(strategy=GenerationType.SEQUENCE, generator="EMP_SEQ")
@SequenceGenerator(name="EMP_SEQ", sequenceName="EMP_SEQ",
allocationSize=100)
```

- GenerationType.IDENTITY menggunakan identity sebagai proses generate id, nilai id akan terus bertambah, unique dan nilainya hanya bisa digunakan dalam table tersebut. MySQL mendukung pilihan ini dengan feature auto_increment.

Proses generate id tidak hanya tergantung dengan database, hibernate memungkinkan programmer menentukan generate id dari aplikasi Java, tidak dari database. Misalnya aplikasi yang akan dibuat memerlukan sinkronisasi data dari cabang ke pusat, jika setiap cabang tablenya menggunakan ID bertipe IDENTITY, kemungkinan table T_PERSON antara cabang satu dengan cabang yang lain akan mempunyai ID yang sama, sehingga diperlukan proses generate id yang unique untuk semua cabang. Java mempunyai class UUID yang akan mengenerate string yang dipastikan unique setiap kali digenerate. Berikut ini konfigurasi @GeneratedValue yang menggunakan UUID string :

```
@GeneratedValue(generator="system-uuid")
@GenericGenerator(name="system-uuid", strategy="uuid")
```

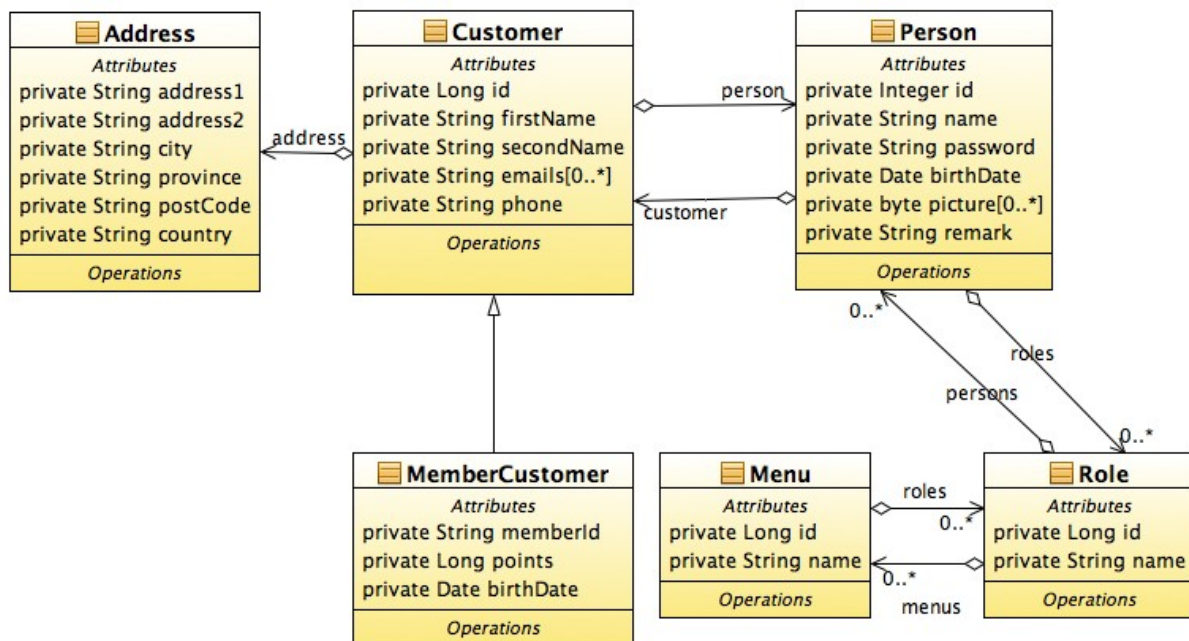
Pembahasan mengenai @Id masih panjang, salah satu topik advance adalah menggunakan composite primary key, dimana primary key dari sebuah table terdiri dari dua kolom atau lebih. Penggunaan composite primary key tidak disarankan, karena akan membuat proses mapping relationship dengan entity lain menjadi lebih rumit. Topik ini tidak akan dicover dalam buku ini, anda bisa melihat ke hibernate reference untuk mengetahui bagaimana cara menggunakan composite primary key.

Class Diagram

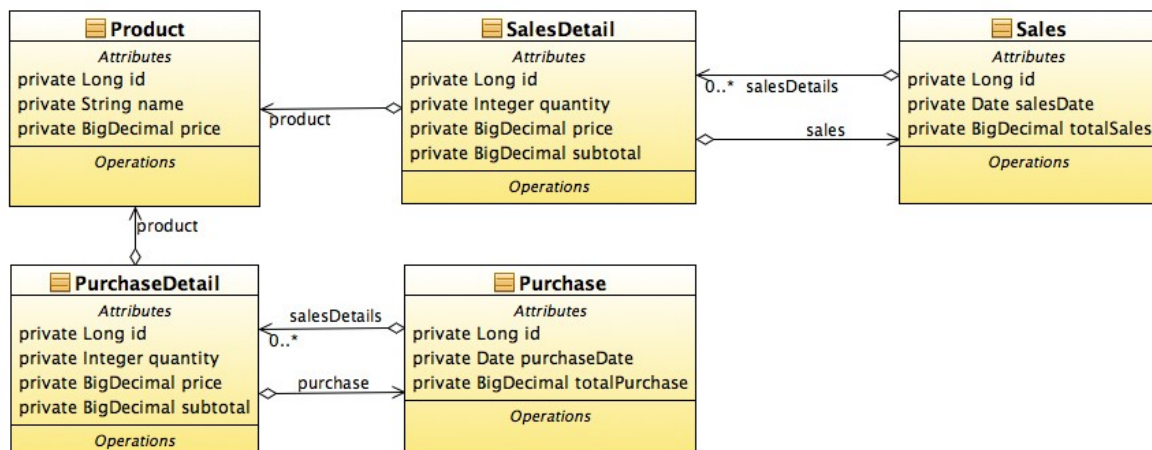
Bagian berikutnya akan membahas lebih banyak tentang entity relationship mapping, sebelum mulai membahas topik relationship mapping, kita bahas dulu class Diagram contoh yang akan digunakan di bagian selanjutnya.

UML class diagram pertama memperlihatkan 6 class yang berhubungan dengan user management. Salah satu class-nya adalah class Person yang sudah digunakan dalam contoh-contoh di bagian sebelumnya. Class Person mempunyai beberapa relationship, salah satunya adalah One-to-One relationship ke class Customer. Class Person juga mempunyai relasi Many-to-Many dengan Role, class Role mempunyai Many-to-Many relationship dengan class Menu.

Class Customer mempunyai inheritance MemberCustomer, jadi dari struktur ini akan digunakan sebagai contoh untuk membahas inheritance mapping. Class Customer juga mempunyai property dengan tipe Address, dimana Address ini bukan merupakan entity, hanya class yang bisa di-embed di class lain. Class Customer juga mempunyai hubungan CollectionOfElement dengan tipe List<String> untuk menampung data email yang dimiliki customer, dimana setiap customer bisa mempunyai banyak email. CollectionOfElement akan dibuatkan satu table tersendiri untuk menampung email-email setiap customer.



Class Diagram kedua merepresentasikan data yang digunakan dalam Transaksi, yaitu produk, pembelian dan penjualan. Produk mempunyai relasi dengan SalesDetail dan CustomerDetail berupa relasi One-to-Many. Sales dan SalesDetail mempunyai hubungan Many-to-One yang disebut sebagai Master-Detail, dimana tanpa ada data Master-nya (Sales) data Detail (SalesDetail) menjadi tidak berguna, sehingga setiap kali data Master-nya dihapus, data Detail-nya juga harus ikut dihapus. Dengan kata lain, daur hidup data Detail tergantung daur hidup Masternya.



Setelah memahami studi kasus yang akan digunakan dalam bagian-bagian berikutnya, mari kita bahas satu per satu relationship yang ada dalam Hibernate.

One-To-One

Relationship ini memapping dua buah entity sepasang, artinya satu entity dimapping tepat ke satu entity yang lain. Contohnya adalah relationship antara Person dan Customer. Setiap Customer dibuatkan data personnya, relasi ini memungkinkan customer untuk bisa login ke system, misalnya untuk mengecek berapa banyak poin yang dimiliki oleh Customer yang tipenya MemberCustomer.

Relationship One-to-One menggunakan annotation `@OneToOne`, kemudian diikuti dengan `@JoinColumn` untuk mendefinisikan Foreign Key. Di sisi lawanya juga menggunakan Annotation `@OneToOne` tanpa diikuti `@JoinColumn` tetapi cukup didefinisikan saja attribute `mappedBy` yang isinya nama property.

Mari kita lihat mapping untuk class Person dan class Customer :

```

@Entity
@Table(name="T_CUSTOMER")
public class Customer {

    @Id @GeneratedValue
    @Column(name="ID")
    private Long id;

    @Column(name="FIRST_NAME",nullable=false,length=100)
    private String firstName;

    @Column(name="SECOND_NAME",length=150)
    private String secondName;

    private Address address;

    @CollectionOfElements(targetElement=String.class)
    @IndexColumn(name="emails_index")
    private List<String> emails;

    @Column(name="PHONE",nullable=false,length=15)
    private String phone;

    @OneToOne
    @JoinColumn(name="PERSON_ID")
    private Person person;
}

```

```
//getter dan setter di sini  
}
```

Proses mapping One-to-One dimulai de class Customer ini dengan menggunakan @OneToOne dan @JoinColumn. Nama dari @JoinColumn akan diterjemahkan menjadi foreign key ketika Hibernate mengenerate table dari mapping ini.

```
@Entity  
@Table(name="T_PERSON",uniqueConstraints={@UniqueConstraint(columnNames={"NAME"})})  
public class Person implements Serializable {  
  
    @OneToOne(mappedBy = "person")  
    private Customer customer;  
  
    @Id @GeneratedValue(strategy=GenerationType.AUTO)  
    @Column(name="ID")  
    private Integer id;  
  
    @Column(name="NAME",unique=true,length=100)  
    private String name;  
  
    @Column(name="PASSWORD",unique=true,length=200)  
    private String password;  
  
    @Temporal(TemporalType.DATE)  
    @Column(name="BIRTH_DATE")  
    private Date birthDate;  
  
    @Enumerated(EnumType.STRING)  
    @Column(name="STATUS",length=20)  
    private MaritalStatus status;  
  
    @Lob  
    @Column(name="PICTURE")  
    private byte[] picture;  
  
    @Lob  
    @Column(name="REMARK")  
    private String remark;  
  
    @ManyToMany(mappedBy="persons")  
    private Set<Role> roles;  
  
    //getter setter di sini  
}
```

Sedangkan di sisi Person mapping dilakukan cukup dengan menambahkan @OneToOne dan mensetting attribute mappedBy dengan nama property person yang ada dalam class Customer.

One-To-Many dan Master-Detail

Relasi One-to-Many menggunakan annotation @OneToMany, @ManyToOne dan @JoinColumn, Relasi antara Sales dan SalesDetail adalah OneToMany plus Master-Detail.

Mari kita lihat kode untuk class Sales dan SalesDetail

```
@Entity  
@Table(name="T_SALES")  
public class Sales {
```

```

@Id @GeneratedValue
@Column(name="ID")
private Long id;

@Temporal(TemporalType.TIMESTAMP)
@Column(name="SALES_DATE",nullable=false)
private Date salesDate;

@OneToMany(mappedBy="sales",cascade=CascadeType.ALL)
@Cascade(org.hibernate.annotations.CascadeType.DELETE_ORPHAN)
private List<SalesDetail> salesDetails;

@Column(name="TOTAL_SALES",precision=18,scale=0,nullable=false)
private BigDecimal totalSales;

//getter setter di sini
}

```

Annotation yang ada dalam class Sales adalah @OneToMany dan @Cascade. Seperti yang diterangkan dalam bagian sebelumnya, daur hidup class Detail tergantung dari Master-nya, sehingga dalam mapping @OneToMany ditambahkan attribute cascade yang nilainya adalah CascadeType.ALL. Konfigurasi ini menyebabkan SalesDetail akan disimpan kalau Sales disimpan, tidak perlu kode untuk menginsert SalesDetail satu per satu, cukup simpan Sales-nya maka semua SalesDetail yang ada akan ikut disimpan (insert). Begitu juga kalau class Sales dihapus, maka semua SalesDetail-nya juga akan dihapus secara otomatis tanpa harus menghapus satu per satu.

Annotation @Cascade diperlukan untuk proses edit class Sales. Misalnya pembeli pada waktu di kasir membawa empat item barang, kemudian kasir sudah menginput keempat sales tersebut dan menyimpan transaksi, maka satu baris data Sales dan empat baris data SalesDetail disimpan dalam database. Kemudian customer tersebut membatalkan salah satu itemnya karena ada kerusakan fisik, kemudian menambahkan satu item lain untuk menggantikannya. Kasir akan menghapus satu item dari SalesDetail, menambahkan satu item ke dalam SalesDetail kemudian menyimpan transaksi yang baru saja diedit tersebut. Dengan attribute cascade all di dalam @OneToMany dan annotation @Cascade di atas, hibernate akan memeriksa satu per satu SalesDetail apa yang ada dalam List<SalesDetail> dan SalesDetail apa yang ada dalam database. Kemungkinan hasil perbandingan antara SalesDetail dalam List dan SalesDetail yang ada dalam database ada tiga jenis:

1. Data SalesDetail yang ada di dalam List dan di dalam database, maka nilainya di database diupdate.
2. Data SalesDetail yang ada dalam List tetapi belum ada dalam Database, maka data yang di dalam List diinsert.
3. Data SalesDetail yang tidak ada dalam List tetapi ada dalam database, maka data yang ada dalam database dihapus.

Poin pertama dan kedua sudah bisa dilaksanakan dengan menggunakan mapping @OneToMany dan attribute cascade all. Poin ketiga dapat dilakukan dengan menggunakan annotation @Cascade.

Mapping di sisi class SalesDetail menggunakan annotation @ManyToOne dan @JoinColumn untuk mendefinisikan nama kolom Foreign Key. Kode class SalesDetail seperti di bawah ini :

```

@Entity
@Table(name="T_SALES_DETAIL")
public class SalesDetail {

    @Id @GeneratedValue
    @Column(name="ID")
    private Long id;

```

```

@ManyToOne
@JoinColumn(name="PRODUCT_ID",nullable=false)
private Product product;

@Column(name="QUANTITY",nullable=false)
private Integer quantity;

@Column(name="PRICE",nullable=false,precision=18,scale=0)
private BigDecimal price;

@Column(name="SUBTOTAL",nullable=false,precision=18,scale=0)
private BigDecimal subtotal;

@ManyToOne
@JoinColumn(name="SALES_ID",nullable=false)
private Sales sales;

//getter setter di sini
}

```

Many-to-Many

Relationship Many-to-Many terjadi antara class Role dengan class Menu, dimana setiap Role bisa mempunyai banyak Menu dan sebaliknya. Setiap mapping Many-to-Many akan menghasilkan sebuah table baru. Annotation yang digunakan dalam relationship Many-to-Many adalah @ManyToMany di kedua class. Salah satu class akan mempunyai mapping tambahan yaitu @JoinTable. Class yang mempunyai @JoinTable ini disebut dengan class pengendali relationship.

Mari kita lihat kode class Role di bawah ini :

```

@Entity
@Table(name="T_ROLE")
public class Role {

    @Id @GeneratedValue
    @Column(name="ID")
    private Long id;

    @Column(name="NAME",length=50,unique=true)
    private String name;

    @ManyToMany
    @JoinTable(name="T_ROLE_PERSON",
        joinColumns={@JoinColumn(name="ROLE_ID")},
        inverseJoinColumns={@JoinColumn(name="PERSON_ID")})
    private Set<Person> persons;

    @ManyToMany
    @JoinTable(name="T_ROLE_MENU",
        joinColumns={@JoinColumn(name="ROLE_ID")},
        inverseJoinColumns={@JoinColumn(name="MENU_ID")})
    private Set<Menu> menus;

    //getter setter di sini
}

```

Perhatikan class Role di atas, dalam relationship Many-to-Many dengan class Menu, class Role bertindak sebagai class Pengendali karena mempunyai mapping @JoinTable. Proses perangkaian relasi ini akan terjadi ketika class Role ini disimpan, dengan syarat class Menu

disimpan terlebih dahulu atau sudah berada dalam database. Kalau urutan ini dibalik, class Role disimpan terlebih dahulu sebelum class Menu, maka akan muncul error yang memberitahu bahwa class Menu belum disimpan dalam database.

Berikut ini kode dari class Menu :

```
@Entity
@Table(name="T_MENU")
public class Menu {

    @Id @GeneratedValue
    @Column(name="MENU_ID")
    private Long id;

    private String name;

    @ManyToMany(mappedBy="menus")
    private Set<Role> roles;

    //getter setter di sini
}
```

Component

Mapping component ini terjadi kalau sebuah class yang bukan entity melainkan embeddable class berada dalam class lain yang tipenya Entity. Class component ditandai dengan @Embeddable, class ini tidak akan dibuatkan table, tetapi semua property yang ada dalam class ini akan dibuatkan table di dalam class Entity-nya.

Konsep component ini mirip dengan konsep Abstract Data type, dimana sebuah class Entity bisa mempunyai property dengan tipe class selain class fundamental Java (String, Date, BigDecimal, Integer dst). Tujuan utamanya adalah untuk menyederhanakan kode kalau class component ini digunakan dalam Entity secara berulang-ulang. Misalnya kita buat class Address yang menjadi property class Customer, ada kemungkinan class Address akan digunakan dalam Entity lain sehingga dibuatkan class Address agar tidak perlu mengetik semua property dalam class Address.

Berikut ini kode untuk class Address :

```
@Embeddable
public class Address {

    @Column(name="ADDRESS1",length=180,nullable=false)
    private String address1;

    @Column(name="ADDRESS2",length=180)
    private String address2;

    @Column(name="CITY",length=50,nullable=false)
    private String city;

    @Column(name="PROVINCE",length=50,nullable=false)
    private String province;

    @Column(name="POST_CODE",length=50)
    private String postCode;

    @Column(name="COUNTRY",length=50,nullable=false)
    private String country;

    //getter setter di sini
}
```


Perhatikan bahwa dalam class Customer tidak diperlukan annotation apapun untuk memapping class Address dalam class Customer. NetBeans biasanya memberi tanda garis bawah merah di deklarasi variabel address yang menerangkan bahwa basic attribute harus menggunakan tipe primitive, wrapper atau array. Tidak perlu khawatir mengenai tanda error ini, karena ketika dicompile atau dijalankan tidak ada masalah sama sekali.

CollectionOfElement

CollectionOfElement digunakan untuk menyimpan nilai property yang jumlahnya lebih dari satu. Dalam contoh kali ini, misalnya setiap customer bisa mempunyai lebih dari satu buah email. Email-email ini akan disimpan ke dalam table berbeda. Tipe data dari collectionOfElement bisa berupa tipe data basic seperti String atau class, jika tipe datanya class maka class tersebut tidak bisa digunakan dalam hibernate query selayaknya Entity.

CollectionOfElement bisa digabungkan dengan @IndexColumn untuk membuat index urutan, misalnya email-email tersebut diurutkan berdasarkan prioritasnya. Contoh mapping-nya seperti di bawah ini diambil dari class Customer:

```
@CollectionOfElements(targetElement=String.class)
@IndexColumn(name="emails_index")
private List<String> emails;
```

Inheritance

Dalam OOP inheritance adalah suatu konsep yang sangat penting, tetapi konsep inheritance ini tidak ada dalam Relational Database, masalah perbedaan konsep ini sering disebut dengan Object-Relational Impedance. ORM, dalam hal ini Hibernate, mengatasi Object-Relational Impedance dengan tiga strategi :

1. One Table Per Class, strategi ini akan membuat table sebanyak class-nya. Misalnya dalam contoh ada class Customer yang merupakan parent class MemberCustomer, dengan menggunakan One Table Per Class, maka akan dibuat masing-masing satu table untuk kedua class ini. Contoh mappingnya :

```
@Entity
@Table(name="T_CUSTOMER")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Customer implements Serializable{
}
```

dan di sisi MemberCustomer adalah :

```
@Entity
@Table(name="T_MEMBER_CUSTOMER")
public class MemberCustomer extends Customer implements Serializable{
}
```

Strategi ini punya banyak kelemahan, misalnya jika ingin mendapatkan semua customer dengan query "from Customer c" maka data dari table T_CUSTOMER dan table T_MEMBER_CUSTOMER harus disatukan, karena sebenarnya MemberCustomer adalah Customer juga. Hibernate mengimplementasikan masalah penggabungan table ini dengan mengenerate union all query antara table T_CUSTOMER dan T_MEMBER_CUSTOMER. Strategi ini tidak mendukung auto generate id dengan tipe IDENTITY atau AUTO karena id harus disharing antara dua table tersebut, sehingga hanya menyisakan SEQUENCE dan TABLE untuk strategi generate id. Hal ini menyebabkan RDBMS yang tidak mendukung SEQUENCE seperti MySQL harus menggunakan strategi TABLE agar bisa menggunakan TABLE_PER_CLASS.

2. Single table, Strategi ini akan menyimpan seluruh class dari parentnya hingga subclass ke dalam satu table saja. Antar class dibedakan menggunakan satu kolom yang disebut dengan Discriminator. Setiap class mempunyai nilai berbeda dalam kolom discriminator ini. Annotation @DiscriminatorColumn digunakan untuk mendefinisikan nama kolom diskriminator dan @DiscriminatorValue digunakan untuk mendefinisikan nilainya untuk setiap class. Jika kedua annotation ini tidak diset, maka digunakan kolom DTYPE dan nilainya diambil dari mana class Entity

Contoh mappingnya adalah sebagai berikut :

```
@Entity
@Table(name="T_CUSTOMER")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
    name="CUSTOMER_TYPE",
    discriminatorType=DiscriminatorType.STRING,
    length=6)
@DiscriminatorValue("CUST")
public class Customer implements Serializable{
}
```

Mapping di class turunanya adalah sebagai berikut :

```
@Entity
@Table(name="T_MEMBER_CUSTOMER")
@DiscriminatorValue("MCUST")
public class MemberCustomer extends Customer implements Serializable{
}
```

Mapping ini cocok digunakan jika satu class mempunyai banyak turunan tetapi kolom di turunanya cuma berbeda sedikit. Kalau kolom di turunanya mempunyai banyak perbedaan maka nantinya akan ada banyak kolom kosong bernilai null. Selain itu kolom yang dipunyai oleh turunan harus bisa diisi dengan nilai null, hal ini dikarenakan untuk entity dengan tipe Customer nilai di kolom yang hanya dipunyai oleh MemberCustomer akan bernilai null.

3. Joined subclasss, strategi ini akan menyimpan kolom yang sama di satu table dan membuat satu table lagi untuk menyimpan property yang hanya dipunyai subclass. Strategi ini dianggap paling bagus, merupakan gabungan antara table per class dan single table. Dalam contoh kita, semua property dalam class Customer akan disimpan di dalam table T_CUSTOMER, baik itu dari entity Customer maupun MemberCustomer. Table T_MEMBER_CUSTOMER sekarang hanya mempunyai property-property yang dipunyai MemberCustomer tetapi tidak dipunyai Customer.

Setiap kali data Customer diambil dari database, Hibernate akan menggenerate query untuk mengambil dari table T_CUSTOMER. Jika yang diambil adalah MemberCustomer maka query yang digenerate hibernate akan mengambil data dari T_CUSTOMER yang disjoint dengan T_MEMBER_CUSTOMER.

Mappingnya adalah sebagai berikut ini :

```
@Entity
@Table(name="T_CUSTOMER")
@Inheritance(strategy=InheritanceType.JOINED)
public class Customer implements Serializable{
}
```

Mappingnya di subclassnya :

```
@Entity
@Table(name="T_MEMBER_CUSTOMER")
@PrimaryKeyJoinColumn(name="CUSTOMER_ID")
public class MemberCustomer extends Customer implements Serializable{
}
```

Mapping inheritance class bukan pekerjaan gampang, perlu pemahaman cukup jelas mengenai bagaimana pengaruh inheritance di Mappingnya maupun di HQLnya. Secara umum mapping inheritance sering dihindari selama masih ada alternatif solusi yang sama baiknya, tetapi kalau sudah tidak ada pilihan lain ya tidak ada masalah menggunakan inheritance mapping.

Database Index

Database index adalah feature yang sangat penting dalam persistence database. Setiap kolom yang digunakan dalam klausa where sebaiknya dibuatkan indexnya, bahkan kalau perlu dibuat kombinasi index yang sama persis dengan klausa where-nya. Misalnya di dalam DAO class Person ada query untuk mengambil data berdasarkan nama-usernya, maka kolom name dalam table

T_PERSON perlu diindex agar query tersebut berjalan dengan cepat.

Proses indexing biasanya dilakukan untuk tujuan optimisasi, indexing adalah solusi pertama yang mengatasi sebagian besar masalah performa query yang lambat. Setiap RDBMS administration tools mempunyai kemampuan untuk memprofile query yang sedang berjalan. Kalau ada query yang lambat ditunjukkan oleh hasil pemeriksaan, cara terbaik mengatasinya adalah melihat klausa where-nya, kemudian pastikan ada index yang dengan kombinasi sama dengan klausa wher tersebut. Kemudian kalau ada join di dalam slow query tersebut, pastikan definisi kolom join-nya sama persis, baik tipe data maupun panjang kolom-nya. Kemudian index juga kolom yang digunakan sebagai join tersebut.

Indexing bisa dilakukan kasus per kasus seperti dalam penjelasan di atas, dengan mencari slow query. Indexing bisa juga dilakukan di depan ketika aplikasi akan dibuat kalau sudah tahu bahwa ukuran datanya akan sangat besar dan potensial mengakibatkan slow query.

Mapping hibernate untuk membuat database index adalah sebagai berikut :

```
@Entity
@Table(name="T_PERSON",uniqueConstraints={@UniqueConstraint(columnNames={"NAME"})})
@org.hibernate.annotations.Table(
    appliesTo="T_PERSON",
    indexes={
        @Index(columnNames={"NAME"})
    }
)
public class Person implements Serializable {
}
```

Annotation yang digunakan adalah @org.hibernate.annotations.Table, annotation adalah komplement dari @javax.persistence.Table yang sudah ada di atasnya. Di dalamnya ada attribute appliesTo yang isinya adalah nama table yang akan diidex. Indexes digunakan untuk mendaftarkan semua index yang akan dibuat dalam table T_PERSON, dalam hal ini kita hanya akan membuat satu buah index saya yang isinya cuma satu kolom, yaitu kolom NAME.

HQL

Query adalah bagian terpenting dari aplikasi yang mengakses database, inti dari aplikasi berada di kode yang mengandung query ini. Hibernate mempunyai query yang disebut HQL, di dalam HQL semua object yang digunakan berasal dari class mapping, tidak ada lagi query ke table langsung. HQL pada akhirnya akan diubah menjadi query yang sesuai dengan RDMS yang digunakan, teknik ini memungkinkan Hibernate mempunyai feature portability antar RDMS.

Hibernate yang sudah matang secara teknologi menjamin semua query yang dibuat merupakan query yang optimal untuk setiap RDBMS, tuning preformance biasanya tidak dilakukan terhadap query yang digenerate hibernate melainkan dilakukan dengan memastikan bahwa struktur table yang dibuat sudah optimum.

HQL bersifat case-insensitive kecuali untuk nama Entity dan propertynya, semua keyword dalam HQL bisa ditulis dalam huruf kecil ataupun huruf besar. Mari kita mulai dengan beberapa contoh HQL untuk mendapatkan gambaran konkrit bagaimana bentuk HQL. Contoh paling sederhana adalah melakukan select terhadap Entity Person, seperti yang ada dalam contoh DAO di bagian sebelumnya :

```
from Person p
```

Seperti terlihat di atas, tidak diperlukan statement select. Query di atas akan mengembalikan List<Person>, kita bisa juga melakukan select satu per satu terhadap property class Person seperti di bawah ini :

```
select p.id, p.name, p.password from Person p
```

Query di atas akan mengembalikan tiga property yaitu id, name dan password dari class Person, hasil querynya adalah List<Object[]> bukannya List<Person> seperti di contoh

sebelumnya. Hal ini dikarenakan query di atas memilih satu per satu property class Person. Object[] akan mempunyai panjang tiga, index nol berisi id dengan tipe Long, index satu berisi name dengan tipe String dan index dua berisi password dengan tipe String.

Dalam contoh query di atas, dapat dilihat juga adanya feature alias untuk mempermudah merefer entity apa yang sedang digunakan. Misalnya dalam contoh di atas p digunakan sebagai alias dari Entity Person.

Projection

Projection digunakan untuk mengambil nilai-nilai tertentu dari sebuah entity. Projection menggunakan klausa select untuk mendaftarkan property apa saja yang akan dikembalikan oleh HQL tersebut. Contoh kedua di atas menunjukkan bagaimana menggunakan feature projection dalam HQL.

Condition

HQL mempunyai feature condition layaknya SQL Query, operator kondisi yang digunakan pun tidak berbeda dengan SQL Query. Condition dalam HQL menggunakan klausa where seperti halnya dalam SQL Query, nama-nama yang ada dalam klausa where adalah property dari Entity. Berikut ini contoh-contoh penggunaan klausa where :

```
from Person where name='ifnu'  
from Person where picture is not null  
from Person where id = 2
```

Dalam klausa where bisa digunakan property chaining untuk mengakses property dari entity, misalnya seperti contoh di bawah ini

```
from Person p where p.customer.firstName = 'ifnu'
```

Query di atas akan mencari Person yang juga Customer dengan nama depan 'ifnu', query di atas kalau diimplementasikan dengan SQL Query memerlukan join antara table T_PERSON dan table T_CUSTOMER. Property chaining adalah salah satu feature HQL yang sangat powerfull karena mengurangi baris kode query secara signifikan tanpa harus menulis klausa join, cukup menggunakan tanda titik untuk mengakses property dari sebuah entity.

Operator, fungsi dan symbol yang boleh ada dalam klausa where adalah sebagai berikut :

- Operator matematika +, -, *, /
- Operator perbandingan =, >=, <=, >, <, <>, !=, dan like
- Operator logika menggunakan and, or, dan not
- Tanda (), digunakan untuk menandai pengelompokan kondisi
- in, not on, between, is null, is not null, is empty, is not empty, member of, dan not member of, berikut ini contoh-contoh penggunaanya :

```
from Person p where p.name in ('ifnu','dian')  
from Person p where p.name between 'A' and 'J'  
from Person p where p.customer is null  
from Customer c where c.emails is empty  
from Customer c where 'ifnubima@gmail.com' member of c.emails
```

Serta bentuk negasinya

```
from Person p where p.name not in ('ifnu','dian')  
from Person p where p.name not between 'A' and 'J'  
from Person p where p.customer is not null  
from Customer c where c.emails is not empty  
from Customer c where 'ifnubima@gmail.com' not member of c.emails
```

- Bentuk case sederhana: case ... when ... then ... else ... end
- Bentuk case pencarian : case when ... then ... else ... end

- Penggabungan String ... || ... atau concat(...,...)

```
from Customer c where c.firstName || c.lastName = 'ifnu bima'
from Customer c where concat(c.firstName, c.lastName) = 'ifnu bima'
```

- current_date(), current_time(), current_timestamp()

```
from Person p where p.birthDate = current_date()
```

- second(...), minute(...), hour(...), day(...), month(...) dan year(...)

```
from Person p where year(p.birtDate) = 1986
```

- fungsi-fungsi untuk manipulasi string dan perhitungan matematis : substring(), trim(), lower(), upper(), length(), locate(), abs(), sqrt(), bit_length(), mod()

```
from Person p where substring(p.name,0,1)='i'
```

- coalesce() dan nullif()
- str() untuk menconvert nilai numerik atau temporal menjadi menjadi nilai string
- cast(... as ...) dimana argumen kedua adalah nama tipe data Hibernate, dan extract(... from ...) jika kedua fungsi cast dan extract didukung oleh RDBMS yang digunakan
- fungsi index() yang dapat dioperasikan terhadap collection yang mempunyai index hasil dari operasi join
- fungsi-fungsi dalam HQL yang digunakan untuk operasi collection, seperti: size(), minelement(), maxelement(), minindex(), maxindex(), plus fungsi elements() dan indices() yang dapat dikuantifikasi menggunakan keyword : some, all, exists, any dan in.

```
from Customer c where c.emails.size = 1
from Customer c where size(c.emails) = 1
from Customer c where maxelement(c.emails) > 1
from Customer c where minelement(c.emails) = 1
from Customer c where exist elements(c.emails)
select m from Menu m, Role r where m in elements(r.menus)
select m from Menu m, Role r where all elements(r.menus) > 10
```

- fungsi skalar SQL seperti sign(), trunc(), rtrim() dan sin()
- query parameter seperti dalam PreparedStatement JDBC : ?
- named parameter, misalnya : :id, :name, :birthdate. Kedua style parameter di atas akan dibahas di bab berikutnya
- SQL literal seperti : 'ifnu', 17, 6.66E+2, '1986-03-01 10:00:00.0'
- Constant Java yang ditandai dengan public static final, misalnya Color.BLUE

Parameter Binding

Di bagian sebelumnya telah disinggung dua gaya untuk memasukkan parameter ke dalam HQL, cara pertama menggunakan symbol ? Sebagai penanda letak parameter seperti di dalam PreparedStatement, contohnya seperti berikut ini:

```
sessionFactory.getCurrentSession()
    .createQuery("from Person p where p.name = ? or p.customer.firstName=?")
    .setParameter(1,"ifnu")
    .setParameter(2,"ifnu");
```

Cara kedua adalah menggunakan named parameter, dimana parameternya tidak menggunakan index posisi, tapi menggunakan sebuah nama diawali dengan tanda :, cara ini jauh lebih baik dan gampang dibaca dibanding cara pertama. Mari kita lihat query di atas jika diimplementasikan menggunakan named parameter :

```
sessionFactory.getCurrentSession()
    .createQuery("from Person p where p.name = :name or "
        + " p.customer.firstName=:fname")
    .setParameter("name","ifnu")
```

```
.setParameter("fname","ifnu");
```

Order By

Layaknya SQL, HQL juga mendukung sorting hasil query menggunakan klausa order by, cara penggunaan order by di HQL sama dengan di SQL. Contohnya seperti berikut ini :

```
from Person p order by p.name asc
```

Agregat

HQL juga mendukung fungsi-fungsi agregat, klausa group by dan klausa having seperti dalam SQL Query. Berikut ini contoh penggunaannya :

```
select pd.product.name, sum(pd.subtotal) from PurchaseDetail pd
group by pd.product.name
having sum(pd.subtotal) > 1000000
```

Subquery

HQL mendukung subquery di dalam query, feature ini hanya tersedia untuk RDBMS yang juga mendukung subquery. Cara penggunaannya sama dengan subquery dalam SQL Query, yaitu dengan meletakkan subquery di dalam tanda kurung buka-tutup () . Berikut ini contoh penggunaan subquery dalam HQL

```
from Person p where p.name in (select c.firstName from Customer c)
```

Join

Join dalam HQL bisa digunakan secara implisit maupun eksplisit. Cara implisit terjadi kalau kita menulis HQL dengan mengakses property dari entity dimana property tersebut tipenya juga merupakan entity atau element lain yang dimapping ke table berbeda. Misalnya kita ingin mengambil property nama pertama (firstName) customer dari Entity Person, contohnya :

```
select p.customer.firstName from Person p
select p from Person p where p.customer.firstName like 'ifnu%'
```

Join cara kedua dilakukan secara eksplisit dengan menggunakan keyword join, seperti contoh di bawah ini :

```
select p from Purchase p join p.purchaseDetail pd where pd.subtotal > 1000000
```

Masalah LazyInitializationException, N+1 Select, Lazy fetch dan Eager fetch

Masalah paling sering dihadapi programmer ketika menggunakan Spring-Hibernate adalah LazyInitializationException (LIE), error ini terjadi ketika kode berusaha mengakses property yang belum diinisialisasi. Secara default, Hibernate Annotation menggunakan JPA annotation seperti yang dibahas dalam buku ini akan menginisialisasi single property seperti customer dalam class Person atau product dalam class PurchaseDetail. Tetapi property dengan tipe collection seperti purchaseDetail dalam Class Purchase tidak diinisialisasi.

LIE terjadi karena setiap kali method dalam Service selesai dieksekusi maka Session yang digunakan dalam method service tersebut akan ditutup, sehingga ketika kode kita berusaha mengakses property yang belum diinisialisasi maka error LIE akan muncul.

Ada beberapa cara untuk mengatasi LIE, cara pertama adalah dengan mengeset mapping dari Purchase ke purchaseDetail dengan FetchType.EAGER, seperti contoh di bawah ini :

```
@OneToMany(mappedBy="purchase", cascade=CascadeType.ALL, fetch=FetchType.EAGER)
@Cascade(org.hibernate.annotations.CascadeType.DELETE_ORPHAN)
private List<PurchaseDetail> purchaseDetail;
```

Mapping di atas akan secara otomatis menginisialisasi purchaseDetails setiap kali entity Purchase di-select. Solusi ini malah akan menimbulkan masalah yang disebut dengan *N+1 Select*, hal ini terjadi karena untuk setiap baris dalam table T PURCHASE, Hibernate akan melakukan satu query untuk mendapatkan semua baris dalam T PURCHASE DETAIL, sehingga satu query :

```
from Purchase p
```

yang menghasilkan N buah baris T_PURCHASE akan dilakukan query ke RDBMS sebanyak N+1

kali. Kesimpulannya adalah solusi eager fetch ini tidak boleh digunakan secara sembarangan, bahkan disarankan untuk tidak menggunakan eager fetch sama sekali karena akan menyebabkan pemborosan resource.

Solusi kedua adalah dengan menggunakan klausa left join fetch ketika ingin mendapatkan nilai Purchase plus purchaseDetail seperti dalam query berikut ini :

```
from Purchase p left join fetch p.purchaseDetail pd
```

HQL di atas akan diterjemahkan oleh Hibernate dengan melakukan join antara T_PURCHASE dan T_PURCHASE_DETAIL sehingga masalah N+1 select dan masalah LIE terpecahkan dengan baik. Best Practice untuk mengatasi masalah ini adalah dengan selalu menggunakan left join fetch untuk semua property yang merupakan entity agar tidak terkena masalah LIE dan N + 1 Select.

Transformation

Terkadang kita dihadapkan pada suatu kebutuhan untuk mengambil nilai-nilai tertentu saja dari suatu table, tetapi tidak ingin mendapatkan nilai List<Object[]> dari hasil query-nya, karena abstraksi kode menjadi susah dibaca kalau menggunakan tipe data List<Object[]>, kebutuhan seperti ini biasanya sering ditemui di dalam report. Hibernate mempunyai feature yang disebut Transformation yang dapat digunakan untuk mentransformasi hasil select query ke dalam class tertentu.

Mari kita lihat contoh penggunaan feature Transformation ini untuk membuat laporan penjualan harian berdasarkan produk. Pertama, buat sebuah class dengan struktur seperti di bawah ini :

```
public class DailySalesReport {
    private String productName;
    private Long quantity;
    private BigDecimal subTotal;
    public String getProductName() {
        return productName;
    }
    public void setProductName(String productName) {
        this.productName = productName;
    }
    public Long getQuantity() {
        return quantity;
    }
    public void setQuantity(Long quantity) {
        this.quantity = quantity;
    }
    public BigDecimal getSubTotal() {
        return subTotal;
    }
    public void setSubTotal(BigDecimal subTotal) {
        this.subTotal = subTotal;
    }
}
```

Kemudian kode dalam DAO untuk mengambil nilai DailySalesReport dari query dengan feature Transformation adalah sebagai berikut ini :

```
sessionFactory.getCurrentSession()
    .createQuery("select p.name as productName, sum(pd.quantity) as quantity, "
        + " sum(pd.subtotal) as subTotal from PurchaseDetail pd "
        + " group by pd.product.name "
        + " where pd.purchase.purchaseDate = current_date")
    .setResultTransformer(Transformers.aliasToBean(DailySalesReport.class))
    .list();
```

Kode di atas menggunakan alias to bean transformer yang menyalurkan kode HQL

menggunakan nama alias yang sama dengan property yang ada dalam class DiallySalesReport, misalnya p.name diberi alias productName yang sama persis dengan property DiallySalesReport.productName dan seterusnya.

Hibernate Cache

Aplikasi yang mempunyai load tinggi sering kali berhadapan dengan performa yang buruk karena akses ke database yang sangat tinggi. Hibernate mempunyai feature cache yang dapat mempertahankan performa aplikasi dikala akses ke database sangat tinggi. Cache dibuat untuk tujuan ini, meningkatkan performa aplikasi dalam keadaan load yang sangat tinggi. Performa cache sendiri bisa ratusan hingga ribuan kali lebih cepat dibanding database akses karena datanya yang terletak di dalam memory, sedangkan RDBMS biasanya meletakkan data di dalam Hard disk.

Hibernate mempunyai feature cache yang terdiri dari dua level: First level cache dan second level cache. Cache di hibernate berfungsi untuk mengurangi hit ke database sehingga resource bisa lebih efisien. Cara kerja cache adalah dengan menyimpan data dalam memory, dalam hal ini sebagai object Java. Selama object masih terus disimpan oleh cache, Hibernate tidak akan mengambil nilai data itu dari database tetapi dari cache ini.

Cache pada teorinya adalah sebuah Map / Collection untuk menyimpan object-object berupa data. Hibernate pertama kali akan berusaha mencari data di cache, jika tidak diperoleh maka hibernate memutuskan untuk mengquery data dari RDBMS.

First level cache

First level cache digunakan untuk menyimpan resource yang digunakan selama satu scope transaksi atau bisa juga diartikan dalam satu Thread. Cache ini built in di dalam Hibernate dan tidak perlu konfigurasi tambahan untuk menghidupkannya. Dalam buku ini, satu scope transaksi adalah ketika method dalam Spring service mulai dieksekusi hingga method tersebut selesai dieksekusi.

First Level cache berada dalam Hibernate Session, selama Session masih belum ditutup data akan disimpan dalam Session, sehingga query ke data yang sama selama session masih belum ditutup tidak akan diambil dari dalam Database, tetapi diambil dari First level cache di dalam Session ini. Selain itu Session tidak boleh digunakan atau dishare untuk thread berbeda, setiap kali transaksi selesai atau setiap kali thread selesai maka session harus diclose, hal ini mengakibatkan first level cache dalam Session tersebut juga dibuang.

Perhatikan kode yang berada dalam suatu method class DAO berikut ini :

```
for(int=0;i<100;i++){
    sessionFactory.getCurrentSession()
        .createQuery("from Person")
        .list();
}
```

Terlihat bahwa query "from Person" dieksekusi seratus kali dalam kode Java, tetapi kalau dilihat outputnya, Hibernate hanya akan mengeksekusi query sekali saja pada saat pertama kali. Hibernate tidak akan mengeksekusi SQL untuk query berikutnya, tetapi menggunakan nilai yang sama yang diambil dari First level cache.

Second level cache

Sifat First level cache yang hanya hidup dalam satu scope transaksi yang sangat pendek, terkadang tidak sesuai dengan kebutuhan optimisasi data yang sering dibaca oleh aplikasi dalam proses berbeda. Second level cache memenuhi kebutuhan ini dengan melakukan cache dalam satu scope aplikasi, dimana cache akan berada dalam SessionFactory, sehingga cache akan bisa terus dipertahankan selama sessionFactory tersebut tidak diclose, yang artinya aplikasi dimatikan. SessionFactory akan dibuat ketika distart aplikasinya dan diclose ketika aplikasi ditutup.

Perlu diwaspadai dalam penggunaan second level cache ini, karena cache tidak akan diupdate kalau proses update data dilakukan dari proses lain tanpa menggunakan SessionFactory yang sama, misalnya ada satu proses di background yang akan melakukan perubahan data atau

perubahan data dilakukan manual langsung ke database dari RDBMS admin tool. Walaupun bisa saja dibuat sebuah konfigurasi yang secara periodik akan mengupdate cache.

Second level cache mempunyai beberapa setrategi, antara lain : read-only, non strict read-write, read-write, dan transactional. Mari kita bahas satu per satu mode di atas.

Read-Only digunakan kalau data yang dicache tidak pernah dirubah sama sekali, misalnya entity Menu, kalau ada perubahan di Entity menu maka apalikasi harus direstart agar cachenya direfresh. Strategi ini mempunyai kinerja paling bagus dan aman digunakan dalam lingkungan cluster.

Read-Write digunakan kalau data yang dicache akan diupdate. Strategi ini tidak boleh digunakan jika transaction isolation level yang digunakan adalah Serializable. Jika digunakan dalam lingkungan cluster, harus dipastikan bahwa cache provider mendukung locking terhadap cache data.

Nonstrict Read-Write digunakan jika data hanya sesekali diupdate dan dipastikan bahwa tidak akan pernah ada proses mengupdate data ini secara simultan dari dua buah mesin berbeda dalam satu cluster.

Transactional menyediakan dukungan penuh terhadap transactional cache dalam lingkungan cluster. Cache dengan feature ini harus digunakan bersama denga JTA provider yang mendukung clustering.

Sebelum memulai menggunakan Second level cache, yang pertama harus dilakukan adalah memilih cache provider yang sesuai, karena tidak ada satupun cache provider yang mendukung semua strategi, pilih cache provider yang sesuai dengan kebutuhan aplikasi. Ada beberapa cache provoder yang didukung oleh Hibernate, setiap cache provider mempunyai feature dan kemampuan yang berbeda-beda, silahkan pilih cache provider yang sesuai dengan kebutuhan aplikasi.

Berikut ini cache provider yang didukung oleh Hibernate

Nama	Read-Only	Non strict read write	Read Write	Transactional
EhCache	Yes	Yes	Yes	No
OSCache	Yes	Yes	Yes	No
SwarmCache	Yes	Yes	No	No
Jboss TreeCache	Yes	No	No	Yes

Sebagai contoh, kita akan menggunakan EhCache sebagai cache provider. Berikut ini konfigurasi yang harus ditambahkan dalam hibernate.cfg.xml :

```
<property name="cache.provider_class">
    org.hibernate.cache.EhCacheProvider
</property>
<property name="cache.use_second_level_cache">true</property>
```

Selanjutnya buat sebuah xml konfigurasi dengan isi sebagai berikut :

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache>
    <diskStore path="java.io.tmpdir"/>
    <defaultCache
        maxElementsInMemory="10000"
        eternal="false"
        timeToIdleSeconds="120"
        timeToLiveSeconds="120"
        overflowToDisk="true"
    />
    <cache name="com.googlecode.projecttemplate.model.Menu"
        eternal="true"
        overflowToDisk="false"
    />
```

```
<cache name="com.googlecode.projecttemplate.model.Person"
    maxElementsInMemory="1000"
    eternal="false"
    timeToIdleSeconds="900"
    timeToLiveSeconds="1800"
    overflowToDisk="true"
/>
</ehcache>
```

Konfigurasi di atas memperlihatkan bahwa entity yang dicache adalah Menu dan Person, kemudian ada beberapa konfigurasi lain yang diset, mari kita bahas satu per satu :

- diskStore konfigurasi ini digunakan untuk meletakkan cache di disk kalau memory tidak mencukupi. Disk Store berguna kalau database dan aplikasi diletakkan dalam server berbeda, sehingga akses data dari local hardisk lebih cepat daripada akses database yang berada di hardisk server lain
- defaultCache digunakan untuk mendefinisikan attribute default dari ehcache. Attribute yang ada dalam defaultCache akan dioverride oleh attribute yang sama dalam tag cache
- cache tag digunakan untuk mendefinisikan Entity apa saja yang akan dicache dalam Hibernate, attribute name berisi nama class dari Entitynya.
- maxElementsInMemory digunakan untuk mendefinisikan berapa maximum jumlah object entity yang akan disimpan dalam cache.
- eternal true digunakan untuk menandai bahwa cache ini akan hidup selamanya, tidak diganti atau diupdate selama aplikasi berjalan. Perhatikan bahwa untuk menu nilai eternal adalah true, karena selama aplikasi berjalan menu tidak kan berubah sama sekali, sedangkan untuk Person nilai eternal adalah false, artinya ada kemungkinan nilai Persin ini bisa berubah dan cache harus mengupdate nilainya jika ada perubahan.
- overflowToDisk true menandakan bahwa cache untuk Person bisa disimpan dalam hardisk jika memory sudah tidak mencukupi. Sedangkan cache untuk Menu mempunyai nilai false, artinya semua cache untuk menu akan disimpan dalam memory dan tidak ada yang disimpan dalam hardisk.
- timeToLiveSeconds digunakan untuk mendefinisikan berapa lama cache hidup sebelum berusaha diupdate dari database. Setelah time to live dilewati maka cache akan diremove dan request berikutnya yang memerlukan cache tersebut akan diambil lagi dari database.
- timeToIdleSeconds digunakan untuk menandai berapa lama cache tidak digunakan sebelum deremove. Kalau data tersebut sudah mencapai waktu di timeTiidleSeconds tanpa sekalipun diakses, maka data akan diremove dari cache. Hal ini dimaksudkan untuk menghindari cache diisi oleh data yang jarang digunakan, sehingga ukuranya akan semakin membengkak.

Tidak perlu ada konfigurasi untuk memberitahu hibernate tentang adanya file ehcache.xml ini, library ehcache akan secara otomatis mencari file ini, ehcache.xml harus berada dalam classpath agar dapat ditemukan.

Perubahan lain yang perlu dilakukan adalah disisi mappingnya, di dalam class Menu dan Person perlu ditambahkan annotation @Cache dan mendeklarasikan strategi apa yang digunakan.

```
@Entity
@Table(name="T_MENU")
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY)
public class Menu implements Serializable{
}
```

Seperti yang kita bahas sebelumnya, entity Menu akan menggunakan strategi read-only karena sifatnya yang tidak akan berubah selama aplikasi berjalan, jika menu akan berubah maka aplikasi harus direstart.

```
@Entity
```

```
@Table(name="T_PERSON",uniqueConstraints={@UniqueConstraint(columnNames={"NAME"})})
@Cache(usage=CacheConcurrencyStrategy.READ_WRITE)
public class Person implements Serializable {
}
```

Entity Person menggunakan strategi read-write karena ada kemungkinan nilai-nilai entity ini berubah dengan adanya penambahan atau edit data person.

Second level cache paling tepat digunakan untuk data yang sering dibaca tetapi jarang ditulis seperti table-table catalog atau table master. Seperti dalam contoh di atas adalah Entity Menu dan Entity Person. Perlu diketahui juga bahwa cache ini hidupnya di dalam SessionFactory, sehingga akan memberikan efek jika digunakan dalam arsitektur Three Tier, dimana hanya ada satu SessionFactory yang berada di tier application server. Jika aplikasinya menggunakan arsitektur client server dimana setiap client akan langsung terhubung dengan database dan setiap client mempunyai SessionFactory sendiri-sendiri, maka second level cache menjadi tidak relevan dan sebaiknya tidak digunakan dalam arsitektur client-server.

Java Foundation Class

Java Foundation Class (JFC) merupakan sekumpulan class-class Java yang digunakan untuk mengembangkan perangkat lunak berbasis GUI (Graphical User Interface). Selain itu, JFC juga mempunyai class-class yang digunakan untuk menambahkan fungsi dan kemampuan interaksi yang variatif dari pemrograman Java. Dari definisi ini, JFC tidak hanya berisi class-class GUI saja tetapi juga class-class lain yang dapat meningkatkan kemampuan pemrograman Java baik dari segi fungsionalitasnya maupun dari segi kemampuan interaksi pemrograman Java yang sangat kaya.

Feature JFC

Fitur-fitur yang dipunyai oleh JFC	
Fitur	Deskripsi
Komponen Swing	Memuat semua class-class yang dibutuhkan untuk membuat aplikasi berbasis GUI, dari tombol, table, tab, menu, toolbar dan sebagainya
Look and Feel (LaF)	Memberikan kemampuan kepada program Java yang dikembangkan menggunakan library Swing untuk memilih tema tampilan. Misalnya sebuah program yang sama dapat mempunyai tampilan windows LaF atau Java LaF, atau LaF lain yang dikembangkan oleh komunitas seperti JGoodies.
Accessibility API	Fasilitas untuk mengembangkan aplikasi bagi penyandang cacat, misalnya dukungan untuk membuat huruf braile, kemampuan mengambil input dari layar sentuh dan sebagainya.
Java 2D API	Berisi kumpulan class-class yang dapat digunakan untuk memanipulasi object-object 2 dimensi, seperti garis, kotak, lingkaran, kurva dan lain sebagainya. Selain itu Java 2D API juga memberikan kemampuan program yang ditulis menggunakan Java untuk mencetak output ke alat pencetak seperti printer.
Drag-and-drop	Menyediakan kemampuan drag-and-drop antara program Java dan program lain yang ditulis spesifik untuk suatu platform sistem operasi tertentu.
Internationalization (i18n)	Membantu pengembang perangkat lunak untuk membangun aplikasi yang dapat mendukung semua bahasa dan huruf yang ada di dunia.

Modul ini akan berkonsentrasi untuk membahas komponen Swing. Pemilihan komponen dan library Swing yang tepat dapat mempengaruhi kualitas program yang kita buat secara signifikan. Hal ini dikarenakan, dalam dunia Java Standard Edition, lebih spesifik lagi aplikasi Java yang dibangun menggunakan Swing, belum ada framework yang benar-benar komprehensif membimbing pengembang untuk membuat aplikasi yang berkualitas.

Pada umumnya aplikasi yang dikembangkan dengan Swing mempunyai kode yang sangat 'kotor', dimana kode yang berisi pengendalian terhadap event komponen Swing bercampur aduk dengan kode yang berisi aturan bisnis dan kode yang berisi manipulasi terhadap data.

Swing Package

Swing API sangat bagus dan lengkap, Java 6.0 menyertakan setidaknya tujuh belas (17) buah package yang berisi class-class Swing yang siap digunakan.

javax.accessibility	javax.swing.plaf	javax.swing.text
javax.swing	javax.swing.plaf.basic	javax.swing.text.html
javax.swing.border	javax.swing.plaf.metal	javax.swing.text.rtf
javax.swing.colorchooser	javax.swing.plaf.multi	javax.swing.table
javax.swing.event	javax.swing.plaf.synth	javax.swing.tree
javax.swing.filechooser		javax.swing.undo

Utungnya kita tidak akan menggunakan semua class-class dalam package Swing, hanya sebagian kecil saja dari class-class tersebut yang nantinya akan benar-benar kita gunakan. Sehingga kita bisa berkonsentrasi untuk memahami beberapa komponen penting saja. Dalam modul ini nanti kita hanya akan menggunakan beberapa class komponen Swing yang penting saja. Beberapa kelas ini sudah cukup sebagai bahan pemembuat perangkat lunak berkualitas.

Komunitas Java juga menyediakan banyak sekali library Swing, antara lain dari Swingx dan JGoodies yang mengembangkan library standard Swing dengan menambahkan berbagai macam feature menarik. Sedangkan komunitas dari javadesktop.org mengembangkan banyak sekali library Swing untuk keperluan khusus. Nyaris semua komponen yang kita perlukan baik komponen umum hingga komponen untuk tujuan khusus banyak tersedia di internet, kita tinggal mencari dan menggunakan.

Praktek yang baik dalam memilih komponen apa yang tepat adalah dengan mencari dahulu informasi di internet. Hal ini sangat bermanfaat untuk mengurangi waktu kita mengembangkan komponen, sehingga kita bisa lebih banyak berkonsentrasi untuk mengembangkan sisi bisnis dan usability dari software yang kita kembangkan. Sebaik apapun software yang kita buat tapi tidak memberikan nilai tambah terhadap masalah yang dihadapi adalah kesia-siaan belaka. Banyak sekali software yang dianggap gagal memberikan nilai tambah terhadap masalah yang dihadapi hanya karena tampilan GUI-nya sangat susah dipahami dan tidak intuitif.

Swing HelloWorld

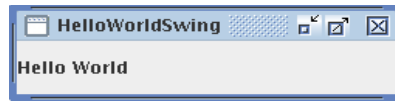
Menggunakan contoh langsung adalah cara yang tepat untuk memulai proses belajar. Cara ini memberikan gambaran kongkrit tentang subject yang akan dipelajari, sehingga proses belajar lebih cepat diserap. Untuk tujuan ini, kita akan membuat sebuah program kecil yang menampilkan kata "HelloWorld" menggunakan komponen Swing. Berikut ini adalah langkah-langkah yang harus anda lakukan untuk membuat program "HelloWorld" berbasis komponen Swing:

1. Install Java Development Kit (JDK)
2. Membuat program HelloWorld itu sendiri
3. Melakukan kompilasi program HelloWorld
4. Menjalankan program HelloWorld

Install Java Development Kit

Yang perlu kita lakukan dalam langkah ini hanyalah mendownload JDK dari website java.sun.com. kemudian jalankan program instalasinya dan ikuti perintah-perintah dalam langkah-langkah instalasi tersebut. Setelah proses instalasi selesai, kita siap untuk membuat program Java.

Membuat program HelloWorld



Program Java dengan tampilan seperti di atas dapat dibuat dengan dua cara. Cara yang pertama adalah dengan menggunakan text editor dan mengetik kode program. Cara yang kedua adalah dengan menggunakan Netbeans Matisse GUI Builder.

Lakukan langkah-langkah berikut ini untuk membuat program di atas menggunakan text editor:

1. Buka text editor kesayangan anda.
2. Ketikkan kode program di bawah ini dan simpan dengan nama file HelloWorld.java :

```
public class HelloWorld {  
  
    public void display(){  
        JFrame.setDefaultLookAndFeelDecorated(true);  
        JLabel label = new JLabel("HelloWorld");  
        JFrame frame = new JFrame();  
        frame.getContentPane().add(label);  
        frame.setVisible(true);  
        frame.pack();  
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
    public static void main(String[] str){  
        HelloWorld hello = new HelloWorld();  
        hello.display();  
    }  
}
```

Melakukan kompilasi program HelloWorld

Kompilasi program tersebut dengan cara menjalankan program javac (Java compiler). Jika anda bekerja di lingkungan windows buka command prompt, kemudian ketik program berikut ini :

```
c:\latihan> javac HelloWorld.java
```

Jika anda bekerja di lingkungan GNU/Linux, buka console dan ketikkan perintah berikut ini :

```
shell$ javac HelloWorld.java
```

Menjalankan program HelloWorld

Proses kompilasi akan menghasilkan file yang berekstensi .class, file inilah yang akan kita eksekusi. Jika anda bekerja di lingkungan windows lakukan perintah berikut ini:

```
c:\latihan> java HelloWorld
```

Jika anda bekerja di lingkungan GNU/Linux jalankan perintah berikut ini:

```
shell$ java HelloWorld
```

Membuat Swing HelloWorld dengan NetBeans

Netbeans 6.1 dilengkapi dengan GUI builder yang dikenal dengan Matisse. Dalam modul ini selanjutnya, Matisse akan digunakan untuk menyebut Netbeans GUI Builder. Tools ini sangat powerful dan produktif dalam membuat komponen GUI. Langkah-langkah yang harus anda lakukan untuk membuat Swing HelloWorld dengan Matisse adalah sebagai berikut:

1. Buat project baru dalam Netbeans, caranya pilih menu :

```
File > New Project
```

- Langkah berikutnya anda harus menentukan kategori project yang akan anda buat, caranya pilih :

General > Java Application

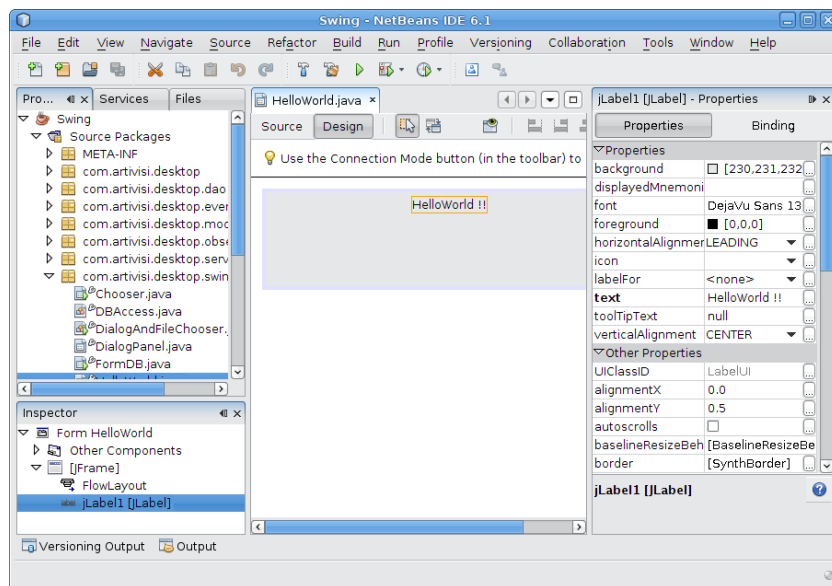
Anda akan dibawa ke dialog untuk menentukan nama project dan folder dimana anda meletakkan project tersebut, pilih folder sesuai keinginan anda.

- Klik kanan di project yang baru anda buat, popup menu akan muncul, kemudian pilihlah menu :

New > JFrame Form...

Kemudian masukkan nama class JFrame yang akan dibuat, misalnya HelloWorld.java, klik finish.

- Tampilan Netbeans akan berganti dengan tampilan GUI builder, dimana di sisi kanan akan terlihat Swing Pallet. Klik item Label di Swing Pallet kemudian klik di atas JFrame, sebuah JLabel akan dibuat.



Jendela Design dan Pallette Netbeans Matisse

- Untuk memenuhi tujuan kita membuat Swing HelloWorld, kita akan memasukkan string "HelloWorld" ke dalam JLabel yang baru saja kita buat. Caranya, double klik di atas JLabel tersebut, kursor muncul bersama text field dan ketikkan "Helloworld".
- Klik kanan di file HelloWorld.java pada jendela explorer di sebelah kiri, pilih menu **Run File...** untuk mengcompile dan menjalankan class HelloWorld.java atau tekan tombol **SHIFT + F6**.

Matisse mempunyai sistem Layouting yang sangat fleksible, sistem layout yang digunakan oleh Matisse adalah GroupLayout. Dalam chapter berikutnya kita akan belajar bagaimana menggunakan GroupLayout ini dengan benar dan memanfaatkan keunggulannya dalam menata component GUI dengan sangat rapi.

Swing helloworld ini hanya sebagian kecil saja dari pekerjaan yang harus dilakukan dalam membangun aplikasi desktop berbasis Java. Selanjutnya kita akan membahas penggunaan JLabel, JButton, JCheckBox, JTextField dan JRadioButton untuk membuat aplikasi GUI sederhana dengan menggunakan Matisse.

Komponen Swing

Swing toolkit menyediakan banyak sekali komponen untuk membangun aplikasi GUI desktop. Swing toolkit juga menyediakan class-class untuk menangani interaksi antara aplikasi dan user menggunakan standard input seperti keyboard dan mouse. Komponen-komponen yang disediakan Swing mencakup semua GUI toolkit yang lazim digunakan dalam aplikasi desktop, seperti : JLabel, JList, JTree, JButton, JLabel dan masih banyak komponen-komponen lainnya yang sudah teruji dan siap pakai.

Selain komponen GUI, Swing juga menyediakan fasilitas untuk proses undo, komponen untuk mengolah text, internationalization, Komponen GUI yang mendukung penyandang cacat (accessibility support) dan fasilitas drag-and-drop.

Look and Feel merupakan fasilitas yang unik dalam Swing. Dengan fasilitas Look and Feel ini kita bisa dengan mudah merubah tampilan dari program kita sesuai dengan keinginan dan tujuan kita. Misalnya, agar program terlihat fancy atau agar program terlihat konsisten dalam segala keadaan.

Swing juga menyediakan library Java 2D untuk pengolahan data secara visual, seperti mengolah gambar, object 2D, bahkan animasi. SwingLabs.org menyediakan library Swing Painter yang merupakan pengembangan dari Java 2D, Swing Painter ini memungkinkan aplikasi Swing mempunyai tampilan yang indah dan terlihat profesional.

Java 6.0 menambahkan banyak sekali fitur-fitur baru ke dalam package Swing, termasuk dukungan untuk library OpenGL menggunakan JOGL, Tray Icon dan Web Service. Dengan adanya dukungan ini Swing menjadi lebih powerful dan mempunyai masa depan yang cerah.

Struktur Komponen Swing

Secara arsitektur, Swing dibangun di atas arsitektur AWT (Abstract Windows Toolkit). AWT adalah GUI toolkit yang dikembangkan oleh Sun engineer sebelum Swing muncul. Kelemahan utama AWT adalah fleksibilitas tampilan GUI, seperti painting method yang masih sangat primitif.

Swing dimaksudkan untuk memperbaiki kekurangan dari AWT tanpa harus membuang teknologi yang sudah dibuat dan membuat GUI toolkit baru dari nol.

Komponen AWT diletakkan dalam satu package yaitu java.awt, didalamnya terdapat komponen-komponen GUI dasar, salah satunya adalah Component. Class Component adalah moyang dari sebagian besar komponen AWT maupun Swing. CheckBox, Label, Button dan beberapa komponen AWT lainnya adalah turunan langsung dari class Component. Namun dalam kenyataannya arsitektur demikian tidak memberikan fleksibilitas yang cukup memadai untuk membuat berbagai macam komponen baru yang dibutuhkan dalam desktop application.

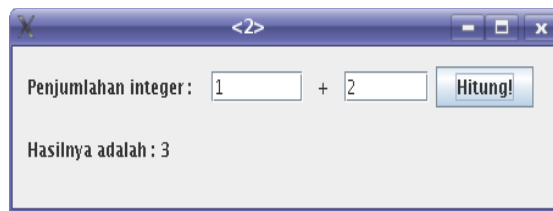
Swing muncul dengan membawa teknologi AWT yang telah ditambahkan dengan banyak kemampuan. Nyaris semua komponen GUI dari Swing merupakan turunan class Container dan class Container adalah turunan dari class Component.

Bekerja dengan JLabel, JTextField dan JButton

Bekerja dengan komponen Swing menggunakan Matisse sangat menyenangkan dan mudah. GroupLayout yang sangat fleksibel memungkinkan kita untuk membuat aplikasi dengan tampilan seperti yang kita harapkan.

Label, textfield dan tombol adalah komponen-komponen dasar yang selalu ada dalam setiap aplikasi berbasis desktop. Ketiga komponen ini mempunyai fungsi yang sangat sederhana, textfield menyimpan data berbentuk text (string) yang relatif pendek, label banyak digunakan untuk memberikan keterangan penjelas terhadap komponen lain dan tombol digunakan user untuk menjalankan satu instruksi tertentu.

Berikut ini adalah contoh aplikasi sederhana yang melakukan penjumlahan dua buah bilangan.



Contoh program menggunakan JLabel, JTextField dan JButton

Untuk membuat aplikasi ini menggunakan Matisse, lakukan langkah-langkah berikut ini:

1. Buat project baru di Netbeans (kalau sudah membuat project, tidak perlu membuat lagi) dengan cara memilih menu :

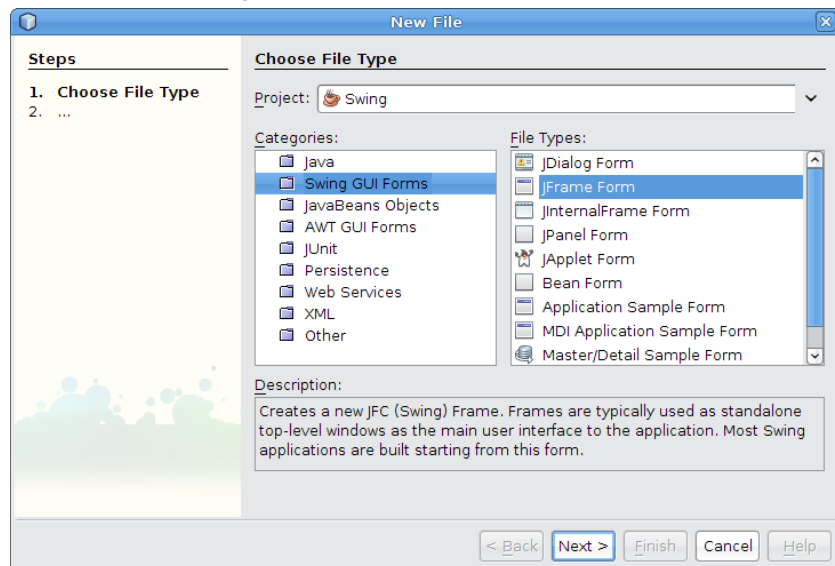
File > New Project

Kemudian ikuti petunjuk yang diberikan dialog.

2. Buat class JFrame baru, caranya dengan memilih menu :

File > New File

Kemudian akan muncul dialog seperti di bawah ini :



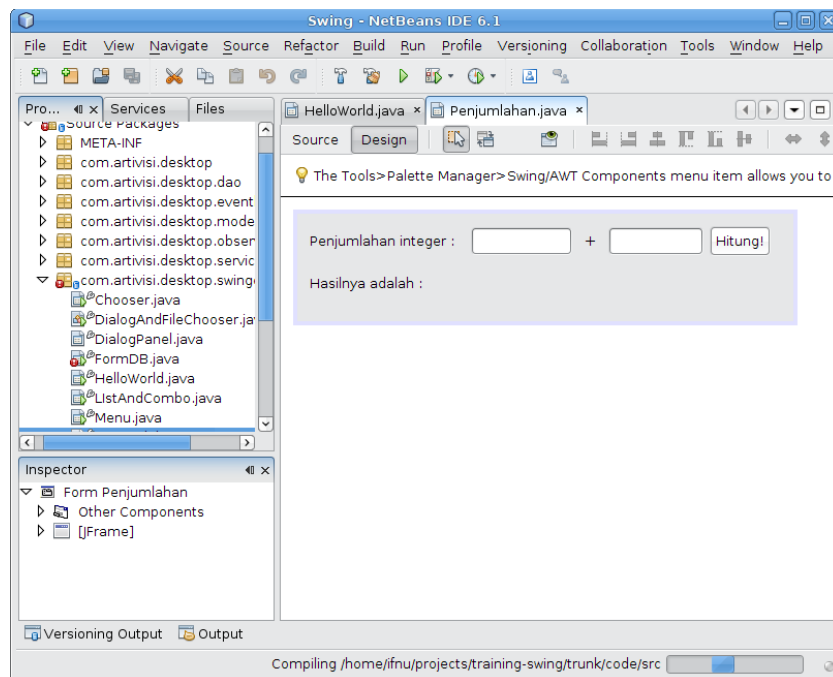
Jendela dialog new file

3. Pilih kategori :

Java GUI Forms > JFrame Form

Seperti terlihat di dialog New File dialog di atas, kemudian beri nama Penjumlahan.java

4. Buat tampilan form seperti gambar bawah ini, caranya dengan klik Jendela Pallette di sebelah kanan untuk memilih komponen apa yang akan dibuat, kemudian klik di jendela Design untuk menempatkan komponen yang sudah dipilih tadi ke dalam form. Hasilnya terlihat seperti pada gambar di bawah ini:



Jendela design Netbens Matisse

5. Ganti nama setiap komponen agar mudah dikenali. Klik kanan di atas setiap komponen yang ada dalam Jendela Design di atas, kemudian pilih menu :

Klik kanan > Change Variable Name ...

Ganti nama komponen-komponen tersebut (sesuai urutan dari kiri ke kanan, atas ke bawah) menjadi : lblKeterangan, txtA, lblPlus, txtB, btnHitung, lblHasil.

6. Menambahkan variabel untuk menampung nilai yang akan dijumlahkan. Klik tombol Source untuk membuka jendela yang menampilkan kode sumber dari program di atas kemudian tambahkan kode di bawah ini tepat di bawah definisi dari class Penjumlahan:

```
private String str = "Hasilnya adalah : ";
private int a, b;
```

7. Menangani penekanan tombol btnHitung. Klik kanan di atas komponen btnHitung kemudian pilih menu :

Events > Action > actionPerformed

Anda akan dibawa ke jendela Source, dan akan menemukan kode program seperti di bawah ini :

```
private void btnHitungActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

Ubah kode program di atas menjadi :

```
private void btnHitungActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    a = Integer.parseInt(txtA.getText());
    b = Integer.parseInt(txtB.getText());
    int hasil = a + b;
    lblHasil.setText(str + hasil);
}
```

8. Compile dan jalankan program. Tekan tombol SHIFT + F6, atau klik kanan file Penjumlahan.java kemudian pilih menu Run File.

Catatan :

- Method Integer.parseInt digunakan untuk merubah String menjadi Integer.
- Method btnHitungActionPerformed akan dipanggil setiap kali kita memencet tombol btnHitung.

Sekarang anda bisa melihat bahwa bekerja dengan JLabel, JTextField dan JButton sangat sederhana. Untuk latihan, silahkan rubah fungsi yang digunakan dalam program di atas, misalnya perkalian dua bilangan atau pengurangan dua bilangan.

Bekerja dengan JCheckBox dan JRadioButton

JCheckBox dan JRadioButton hanya bisa mempunyai dua buah kemungkinan nilai, benar atau salah. Kedua komponen ini digunakan untuk merepresentasikan data yang berupa pilihan. JCheckBox digunakan jika pilihanya berupa multiple selection, sedangkan JRadioButton digunakan jika pilihanya berupa single selection.

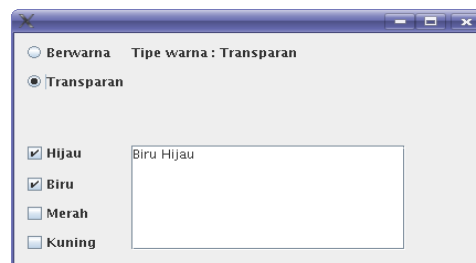
JRadioButton digunakan misalnya untuk merepresentasikan pilihan jenis kelamin. JCheckBox digunakan misalnya untuk merepresentasikan pilihan hobby.

ButtonGroup diperlukan untuk mengumpulkan JRadioButton yang mempunyai grup pilihan yang sama. Misalnya grup pilihan jenis kelamin digunakan untuk mengumpulkan JRadioButton yang merepresentasikan pilihan laki-laki dan JRadioButton yang merepresentasikan pilihan perempuan dalam satu group. Jika JRadioButton tidak diletakkan dalam satu group, maka pilihan laki-laki dan pilihan perempuan bisa dipilih bersamaan.

Status dari JRadioButton dan JCheckBox dapat diketahui dengan melihat nilai kembalian dari method isSelected, jika dipilih maka nilai kembalian method isSelected adalah benar, dan false jika sebaliknya.

Setiap JRadioButton dan JCheckBox mempunyai text yang menerangkan pilihan yang diwakilinya. Method getText dan setText digunakan untuk memanipulasi text.

Di bawah ini adalah contoh program yang menggunakan JCheckBox dan JRadioButton.



Contoh aplikasi menggunakan JCheckBox dan JRadioButton

Di bagian atas aplikasi ini, terdapat dua JRadioButton untuk merepresentasikan pilihan tipe warna, transparan atau berwarna. Di bawahnya terdapat pilihan warna yang dapat dipilih lebih dari satu buah menggunakan JCheckBox.

Untuk membuat program di atas ikuti langkah-langkah berikut ini:

1. Buat class baru bertipe JFrame Form, kemudian beri nama Pilihan.java
2. Buat tampilan di atas menggunakan Matisse. komponen yang harus dibuat adalah :
 - Dua object JRadioButton : radioBerwarna dan radioTransparan.
 - Satu object ButtonGroup : groupTipeWarna.
 - Empat object JCheckBox : chkHijau, chkBiru, chkMerah, chkKuning.
 - Satu object JTextArea : txtWarna.
 - Satu object JScrollPane : scrollWarna

Untuk melihat semua komponen yang ada dalam Jendela Design, gunakan Jendela Inspector di sisi kiri bawah.

3. Masukkan object radioBerwarna dan radioTransparan ke dalam object groupTipeWarna. Caranya dengan :

- a) Memilih komponen radioBerwarna di Jendela Design
- b) Klik tab code di Jendela Properties
- c) Pilih properti : Post-Creation Code
- d) Masukkan kode berikut ini kedalam dialog yang muncul :

```
groupTipeWarna.add(radioBerwarna);
```

Lakukan langkah yang sama terhadap object radioTransparan.

4. Menangani event ketika JRadioButton diklik. Caranya dengan :

- a) Memilih komponen radioBerwarna di Jendela Design
- b) Klik kanan komponen radioBerwarna, kemudian pilih menu:

```
Event > Action > actionPerformed
```

- c) Anda akan dibawa ke dalam Jendela Code, dan menemukan kode berikut ini :

```
private void radioBerwarnaActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}
```

Ubahlah kode di atas menjadi :

```
private void radioBerwarnaActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    if(radioBerwarna.isSelected()){  
        lblTipeWarna.setText("Tipe warna : " +  
            radioBerwarna.getText());  
    }  
}
```

Lakukan langkah yang sama terhadap radioTransparan.

5. Buat sebuah private method untuk menangani event pemilihan terhadap JCheckBox. Method tampilkanWarna ini nantinya akan dipanggil setiap kali salah satu dari JCheckBox dipilih. yang dilakukan oleh metod tampilkanWarna adalah mengecek status setiap JCheckBox, apakah sedang dipilih atau tidak. Jika sedang dipilih maka text dari JCheckBox tersebut akan ditampilkan dalam txtWarna.

Class StringBuffer digunakan untuk menampung nilai text dari JCheckBox yang statusnya terpilih.

```
private void tampilkanWarna(){  
    StringBuffer warna = new StringBuffer();  
    if(chkBiru.isSelected()){  
        warna.append(chkBiru.getText() + " ");  
    }  
    if(chkHijau.isSelected()){  
        warna.append(chkHijau.getText() + " ");  
    }  
    if(chkKuning.isSelected()){  
        warna.append(chkKuning.getText() + " ");  
    }  
    if(chkMerah.isSelected()){  
        warna.append(chkMerah.getText() + " ");  
    }  
    txtWarna.setText(warna.toString());  
}
```

6. Menangani event pemilihan JCheckBox. Caranya sebagai berikut :

- a) Pilih komponen chkHijau di Jendela Design.
- b) Klik kanan komponen chkHijau untuk memunculkan context (popup) menu.
- c) Pilih menu :

Event > Action > actionPerformed

- d) Anda akan dibawa ke Jendela Code, kemudian dalam method chkHijauActionPerformed tersebut panggil method tampilkanWarna. seperti di bawah ini :

```
private void chkHijauActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    tampilkanWarna();  
}
```

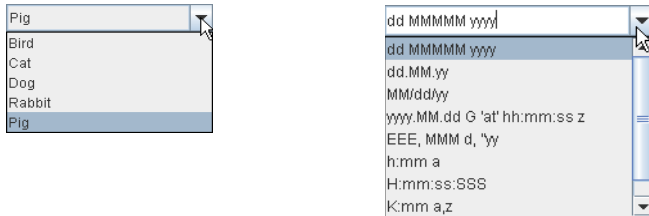
Lakukan hal ini untuk semua JCheckBox.

7. Compile dan jalankan program dengan menekan tombol SHIFT + F6.

Cara lain dalam menampilkan pilihan adalah dengan menggunakan JList dan JComboBox. Kedua komponen ini mempunyai fleksibilitas yang lebih tinggi dan lebih mudah digunakan jika object yang dimasukkan dalam pilihan lebih kompleks. JList dan JComboBox bisa mempunyai ComponentEditor agar pilihan yang ditampilkan tidak hanya berupa text, bisa berupa warna atau icon. Bagian berikutnya akan membahas bagaimana bekerja menggunakan JList dan JComboBox.

Bekerja dengan JList dan JComboBox

JComboBox memerlukan tempat yang minimalis dibandingkan dengan JRadioButton, selain itu JComboBox mempunyai bentuk ComboBox yang dapat diedit, sehingga memungkinkan user untuk memilih pilihan yang tidak ada dalam item JComboBox.

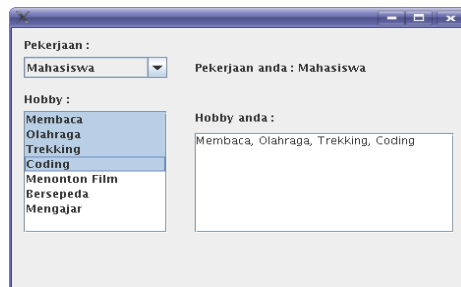


Contoh JComboBox

JList memungkinkan multiple selection dengan menekan tombol : SHIFT + Left Click atau CTRL + Left Click. Kemampuan ini membantu user jika harus melakukan multiple selection.

JComboBox dan JList sangat fleksibel, kita dapat menambah dan menghapus item di dalamnya dengan sangat mudah. Sehingga cocok digunakan untuk merepresentasikan pilihan yang item pilihannya bersifat dinamis.

Aplikasi di bawah ini adalah contoh penggunaan JComboBox dan JList.



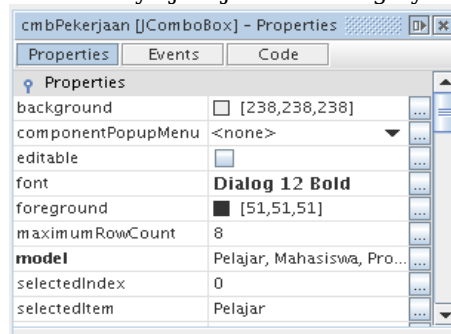
Contoh program menggunakan JComboBox dan JList

Bagian pertama program ini terdapat sebuah JComboBox dan JLabel, setiap kali item di dalam JComboBox dipilih, JLabel di sebelahnya akan menampilkan item yang dipilih tersebut.

Bagian kedua program ini terdapat sebuah JList dan JTextArea. Setiap kali item-item di dalam JList dipilih, JTextArea akan menampilkan item-item yang dipilih tersebut dipisahkan dengan koma (,).

Ikuti langkah-langkah berikut ini untuk membuat program di atas:

1. Buatlah class JFrame Form baru dan beri nama ListAndCombo.java.
2. Buat tampilan program di atas menggunakan Matisse, kemudian tambahkan komponen-komponen:
 - a) Empat buah JLabel : lblPekerjaan, lblPilihanPekerjaan, lblHobby, lblPilihanHobby.
 - b) Satu buah JComboBox : cmbPekerjaan
 - c) Satu buah JList : lstHobby
 - d) Satu buah JTextArea : txtPilihanHobby
3. Merubah isi JComboBox. Untuk merubah isi dari JComboBox dan JList kita akan menggunakan Jendela Properties, Jendela ini letaknya di sebelah kanan bawah, di bawah Jendela Pallette dan akan muncul hanya jika jendela Design yang dipilih.



Jendela Properties

Pilih komponen JComboBox di Jendela Design, Jendela Properties akan menampilkan properties dari JComboBox.

Pada bagian model di dalam Jendela Properties masukkan item Pelajar, Mahasiswa, Programmer, Technical Writer dan Tester. Setiap item dipisahkan dengan koma (,).

4. Merubah isi JList. Pilih JList di Jendela Design maka Jendela Properties untuk JList akan muncul. Di bagian model isikan item : Membaca, Olahraga, Trekking, Coding, Menonton Film, Bersepeda dan Mengajar. Setiap item dipisahkan dengan koma (,).
5. Menangani pemilihan JComboBox. Klik kanan JComboBox di Jendela Design, kemudian pilih menu :

Events > Action > actionPerformed

Jendela Code akan terbuka, tambahkan code seperti di bawah ini :

```
private void cmbPekerjaanActionPerformed(java.awt.event.ActionEvent evt) {  
    lblPilihanPekerjaan.setText("Pekerjaan anda : " +  
        cmbPekerjaan.getSelectedItem());  
}
```

method getItem() dari JComboBox digunakan untuk memperoleh item yang sedang di pilih dalam JComboBox.

6. Menangani event pemilihan dari JList. Event yang digunakan untuk menangani pemilihan item dari JList berbeda dengan JComboBox. JList akan mengaktifkan ListSelection event ketika user memilih item dalam JList. Untuk menangani event ini, lakukan langkah-langkah berikut :

a) Klik kanan pada JList di dalam Jendela Design, kemudian pilih menu :

Events > ListSelection > valueChanged

b) Dalam jendela kode yang ketik kode seperti berikut ini :

```
private void lstHobbyValueChanged(javax.swing.event.ListSelectionEvent evt) {
    Object[] selectedItems = lstHobby.getSelectedValues();
    if(selectedItems == null || selectedItems.length == 0){
        txtPilihanHobby.setText("");
    }else{
        StringBuffer strValues = new StringBuffer();
        for(Object item : selectedItems){
            strValues.append(item.toString() + ", ");
        }
        txtPilihanHobby.setText(
            strValues.substring(0, strValues.length() - 2));
    }
}
```

Catatan : Method `getSelectedValues` dari JList mengembalikan item-item yang terpilih.

Bekerja dengan Menu, Popup Menu dan Toolbar

Menu, Popup menu dan Toolbar digunakan untuk melakukan navigasi dalam aplikasi. dengan ketiga komponen itu navigasi dalam aplikasi menjadi lebih fleksibel dan mudah digunakan oleh user. Menu dan Toolbar pada umumnya diletakkan di bagian atas dari aplikasi agar mudah ditemukan oleh user. Sedangkan Popup Menu bisa muncul di mana saja sesuai dengan konteks aplikasi.

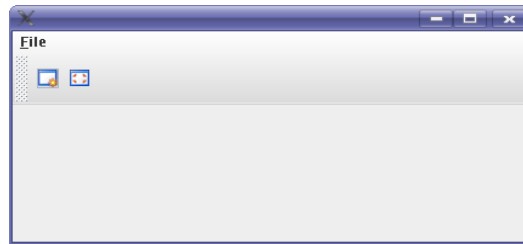
JMenuBar adalah class yang digunakan untuk menampung JMenu. JMenu dapat menampung satu atau lebih JMenuItem. JMenuItem merupakan bagian terluar dari struktur menu yang tidak bisa mempunyai child. JSeparator digunakan untuk memisahkan antara satu menu item dan menu item yang lain. Jika didalam menu terdapat sub menu, gunakan JMenu untuk menampung sub menu tersebut. Selain JMenuItem, JMenu juga dapat menerima class JCheckBoxMenuItem dan JRadioButtonMenuItem.

JPopupMenu mempunyai banyak kesamaan dibandingkan dengan JMenuBar. Perbedaan utamanya adalah : JMenuBar hanya bisa berada di atas sebuah jendela JFrame. Sedangkan JPopupMenu bisa muncul di mana saja sesuai dengan konteks dari aplikasi.

Perbedaan lainnya terletak di dalam penggunaan umum keduanya. JMenuBar berisikan menu/instruksi yang bersifat umum dan berlaku untuk semua keadaan dalam aplikasi. Sedangkan JPopupMenu akan mempunyai menu/instruksi yang berbeda-beda berdasarkan dari konteks aplikasi. Oleh karena itu JPopupMenu terkadang disebut juga sebagai konteks menu.

Toolbar memberikan cara yang lebih praktis dibandingkan menu, bahkan bisa dikatakan bahwa toolbar adalah cara cepat untuk mengakses menu. Oleh karena itu, setiap item dalam toolbar biasanya juga tersedia dalam menu. Pada umumnya toolbar diwakili hanya dengan gambar/icon yang melambangkan perintah dari toolbarnya. Di internet banyak tersedia toolbar icon gratis yang dapat kita gunakan.

Berbeda dengan JMenuBar dan JPopupMenu yang hanya bisa menerima menu item, JToolBar dapat menampung JButton atau control lainnya. Seperti contohnya : JCheckBox, JRadioButton, JToggleButton dan lainnya. Normalnya, JToolBar akan diisi dengan JButton yang dihilangkan text-nya dan diganti dengan icon. Kita juga perlu merubah dekorasi JButton agar tampilannya terlihat cantik dan menarik.



Contoh program dengan Menu, Popup Menu dan Toolbar

Untuk membuat program seperti di atas, ada beberapa tahap yang perlu dilakukan. Tahap pertama adalah membuat Menu, yang kedua adalah membuat Popup Menu dan yang ketiga adalah membuat Toolbar.

Membuat Menu

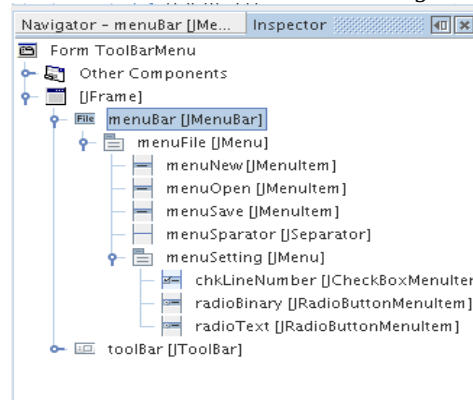
Bekerja dengan Menu dalam Java melibatkan enam komponen Swing, antara lain :

1. **JMenuBar** : Class yang menampung semua menu, hanya bisa menampung JMenu sebagai child.
2. **JMenu** : Class yang mempunyai child menu item. Biasanya JMenu ini yang jadi child langsung dengan JMenuBar
3. **JMenuItem** : Ujung dari menu, di sinilah object Action diletakkan, sehingga ketika kita memilih JMenuItem ada action tertentu yang dijalankan aplikasi.
4. **JCheckBoxMenuItem** : Ujung dari menu, namun bentuknya lebih mirip JCheckBox.
5. **JRadioButtonMenuItem** : Ujung dari menu, namun bentuknya lebih mirip JButton.
6. **JSeparator** : pemisah antar JMenuItem atau antar JMenu

Setiap komponen menu mempunyai fungsi dan kegunaan masing-masing. Jika kita perhatikan, menu dalam aplikasi umumnya mempunyai shortcut untuk mengakses menu tanpa menggunakan bantuan mouse. Misalnya menu File biasanya dapat diakses menggunakan tombol ALT + F, menu Format dapat diakses dengan ALT + O. Fasilitas shortcut menu ini disebut sebagai Keyboard Mnemonic. File mempunyai mnemonic F, Format mempunyai mnemonic o, dan seterusnya. Pada umumnya tampilan mnemonic dari sebuah menu diwakili dengan huruf yang bergaris bawah.

Matisse mempunyai fasilitas yang sangat OK untuk bekerja dengan Menu, fasilitas drag-and-dropnya membantu banyak pekerjaan membangun aplikasi berbasis GUI secara signifikan.

Dalam program di atas kita akan membuat struktur menu sebagai berikut:



Struktur menu dari aplikasi

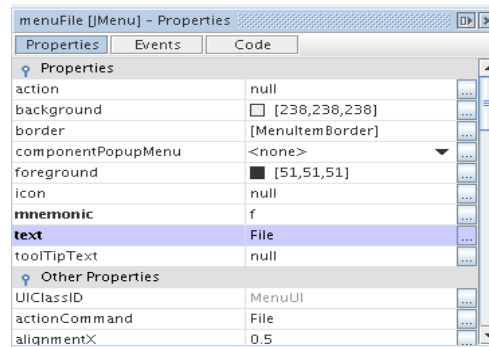
Ikuti langkah-langkah berikut ini untuk membuat struktur menu seperti di atas:

1. Buat sebuah class JFrame dan beri nama ToolbarMenu.java

- Menambahkan JMenuBar ke dalam JFrame. Pilih komponen Menu Bar dari Jendela Palette kemudian klik JFrame di Jendela Design. Sebuah class JMenuBar akan ditambahkan di dalam JFrame. Ganti namanya menjadi menuBar.
- Menambahkan JMenu ke dalam JMenuBar. Klik kanan JMenuBar yang baru saja kita buat di Jendela Inspector, kemudian pilih menu :

Add > JMenu

Ganti nama JMenu tersebut menjadi menuFile. Kemudian alihkan perhatian anda ke Jendela Properties



Jendela Properties dari class JMenu

Isi properti text dengan string "File". Kemudian set isi properti mnemonic dengan string "f", hal ini akan menyebabkan tampilanya menuFile menjadi File dan user dapat menekan tombol ALT + F untuk mengaktifkan menu menuFile.

- Menambahkan JMenuItem. Langkah berikutnya adalah menambahkan JMenuItem ke dalam JMenu menuFile yang telah dibuat di langkah sebelumnya. caranya, klik kanan di JMenu menuFile di Jendela Inspector, kemudian pilih menu :

Add > JMenuItem

Tambahkan berturut-turut menuNew, menuOpen dan menuSave. Pilih JMenuItem dari Jendela Inspector, kemudian untuk masing-masing JMenuItem set text dan mnemonic yang sesuai dari Jendela Properties.

- Menambahkan JSeparator. Dalam struktur menu yang bagus, menu yang mempunyai fungsi serupa diletakkan dalam urutan berderdekatan dan dipisahkan dengan separator (pemisah). Langkah menambahkan JSeparator tidak berbeda dengan langkah menambahkan JMenuItem, klik kanan di JMenu menuFile kemudian pilih menu:

Add > JSeparator

- Menambahkan JMenu. Berikutnya kita akan menambahkan JMenu baru ke dalam JMenu menuFile. JMenu yang baru ini akan bertindak sebagai sub menu. Caranya juga sama : klik kanan di JMenu menuFile kemudian pilih menu :

Add > JMenu

Beri nama menuSetting, set text dan mnemonic yang sesuai pada Jendela Properties.

- Menambahkan JCheckBoxMenuItem. Perilaku JCheckBoxMenuItem tidak berbeda jauh dengan JCheckBox biasa, bedanya hanyalah JCheckBoxMenuItem berada dalam struktur menu. Cara menambahkan JCheckBoxMenuItem sama dengan komponen lain : klik kanan JMenu menuSetting kemudian pilih menu :

Add > JCheckBoxMenuItem

Beri nama chkLineNumber, set text dan mnemonic yang sesuai pada Jendela Properties.

JCheckBoxMenuItem sedikit spesial dibandingkan dengan JMenuItem, karena JCheckBoxMenuItem memiliki properties selected. Properties selected ini digunakan untuk menentukan apakah JCheckBoxMenuItem dalam keadaan terpilih atau tidak.

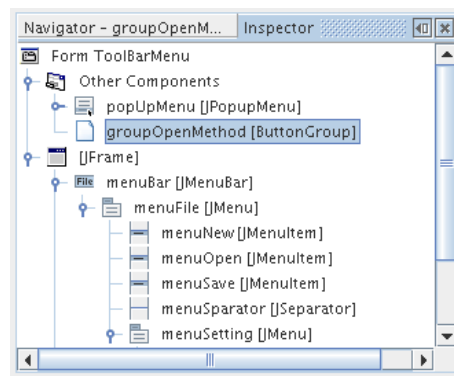
8. Menambahkan `JRadioButtonMenuItem`. Dalam contoh ini kita akan mempunyai dua buah `JRadioButtonMenuItem`, `radioBinary` dan `radioText`. Keduanya dibuat dengan langkah yang sama dengan komponen lain, klik kanan di `JMenu menuSetting`, kemudian pilih menu :

Add > `JRadioButtonMenuItem`

Set text dan mnemonic yang sesuai dari `Jendela Properties`.

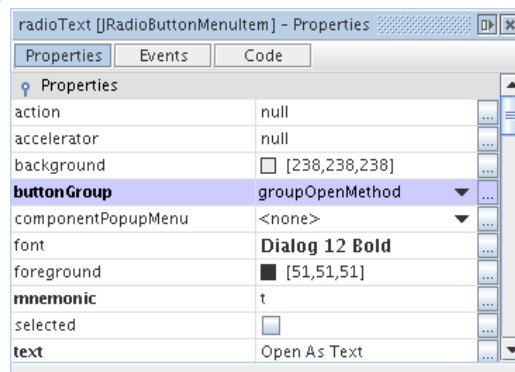
9. Menambahkan `ButtonGroup`. Seperti halnya `JRadioButton`, `JRadioButtonMenuItem` juga memerlukan `ButtonGroup` agar hanya satu buah `JRadioButtonMenuItem` yang bisa dipilih. Cara menambahkan `ButtonGroup` sangat mudah, klik item `ButtonGroup` dari `Jendela Palette` kemudian klik `Jendela Design`, maka otomatis `ButtonGroup` akan ditambahkan. Ganti namanya menjadi `groupOpenMethod`.

Dalam `Jendela Inspector`, `ButtonGroup` yang baru dibuat tadi akan berada dalam kategori `Other Components`, seperti terlihat dalam gambar di bawah ini :



`ButtonGroup` berada dalam kategori `Other Components`

10. Menambahkan `JRadioButtonMenuItem` ke dalam `ButtonGroup`. Pilih masing-masing `JRadioButtonMenuItem` dari `Jendela Inspector`, kemudian perhatikan `Jendela Properties` dari `JRadioButtonMenuItem` tersebut, pada bagian `groupButton` pilih item `groupOpenMethod`, seperti terlihat dalam gambar di bawah ini :



Properties dari `JRadioButtonMenuItem`

11. Compile dan jalankan class `ToolBarMenu.java`. Klik kanan class `ToolBarMenu` dari `Jendela Design` kemudian pilih menu `Run File` atau tekan tombol `SHIFT + F6`.

Bekerja dengan Menu menggunakan `Matisse` sangatlah menyenangkan dan produktif. Hal ini berbeda sekali jika harus mengetik satu demi satu kode untuk menyusun struktur menu seperti contoh program di atas.

Membuat `Popup Menu` menggunakan `Matisse` juga sama mudahnya. Hanya saja kita harus menentukan dimana dan dengan cara apa `popup menu` itu muncul, apakah dengan penekanan tombol tertentu dari keyboard atau ketika tombol mouse ditekan.

Membuat Popup Menu

Popup menu pada dasarnya tidak jauh berbeda dibandingkan dengan menu biasa, hanya saja popup menu dapat muncul di mana saja, tidak hanya di bagian atas JFrame seperti halnya JMenuBar. Selain itu kita harus menentukan kapan popup muncul, pada umumnya popup akan muncul ketika user melakukan klik kanan terhadap suatu komponen Swing. Misalnya, ketika suatu table di klik kanan terdapat popup yang muncul, dan sebagainya.

Popup menu terutama digunakan sebagai “context sensitive menu”, dimana menu yang ditampilkan oleh popup menu tergantung konteks dari aplikasi, semisal : komponen apa yang dikenai aksi klik kanan, bagaimana keadaan data dalam komponen tersebut dan sebagainya.

Aplikasi yang memerlukan interaksi yang sangat intens dengan user sebaiknya menggunakan popup menu untuk memudahkan user mengakses action tertentu. Hal ini jauh lebih praktis dibanding user harus mengakses menu dalam JMenuBar di bagian atas JFrame.

Popup menu dalam contoh program di atas akan muncul ketika user melakukan klik kanan terhadap JFrame. menu yang ditampilkannya pun hanya ada tiga buah: cut, copy dan paste.

Ikuti langkah-langkah berikut ini untuk membuat Popup menu :

1. Buka class ToolbarMenu.java, yang telah dibuat dalam langkah sebelumnya dalam Jendela Design.
2. Klik Jendela Palette dan pilih JPopupMenu, kemudian klik Jendela Design. Secara otomatis JPopupMenu akan ditambahkan dalam class ToolbarMenu.java. Ganti nama variabel JPopupMenu yang baru saja dibuat menjadi popUpMenu. JPopupMenu tidak terlihat dalam Jendela Design, namun anda bisa mengksesnya melalui Jendela Inspector.
3. Menambahkan JMenuItem. Seperti halnya JMenuBar, JPopupMenu dapat memiliki child berupa JMenuItem, JMenuItem, JCheckBoxMenuItem, JRadioButtonMenuItem dan JSeparator. Menambahkan JMenuItem ke dalam JPopupMenu sangat sederhana, caranya : klik kanan pada JPopupMenu di Jendela Design, kemudian pilih menu :

Add > JMenuItem

Ganti nama objectnya menjadi menuCut, beralihlah ke Jendela Properties kemudian set text dan mnemonic yang sesuai.

Lakukan langkah ini untuk JMenuItem yang lain, menuCopy dan menuPaste.

4. Memunculkan JPopupMenu. Ketika tombol kanan mouse di klik di atas JFrame, JPopupMenu akan tampil. Agar behavior tersebut berjalan, kita perlu menangani event mouseClicked terhadap JFrame. Caranya :

- a) Klik kanan JFrame di Jendela Design, kemudian pilih menu :

Events > Mouse > mouseClicked

- b) Di dalam jendela source yang terbuka masukkan kode berikut ini :

```
private void formMouseClicked(java.awt.event.MouseEvent evt) {  
    if(evt.getButton() == MouseEvent.BUTTON3){  
        popUpMenu.show((Component)evt.getSource(),  
            evt.getX(),evt.getY());  
    }  
}
```

Kondisi if di atas digunakan apakah tombol yang diklik mouse adalah tombol sebelah kanan, jika nilai kembalian method getButton sama dengan nilai BUTTON3 maka benar tombol kanan yang ditekan.

Method show digunakan untuk memunculkan popup menu, parameter pertama diisi dengan Component dimana nantinya popup menu akan ditampilkan, sedangkan parameter kedua dan ketiga diisi dengan letak koordinat popup menu akan ditampilkan.

5. Simpan file ToolbarMenu.java, compile dan jalankan. Kemudian coba munculkan popup menu dengan mengklik kanan JFrame.

Popup menu sangat berguna jika aplikasi yang kita kembangkan membutuhkan interaksi yang intensif dengan user. Popup menu menyediakan cara mudah untuk mengakses menu/action yang sesuai dengan konteks dari aplikasi.

Membuat Toolbar

Toolbar memberikan dimensi lain dalam mengakses menu dibandingkan menu ataupun popup menu. Pada umumnya Toolbar merupakan cara singkat untuk mengakses menu. Menu yang diwakili toolbar adalah menu yang bersifat umum dan tidak terikat pada konteks tertentu.

Kegunaan lain dari toolbar adalah mempercantik tampilan aplikasi, karena toolbar biasanya adalah tombol yang didekorasi dengan icon yang menarik. Selain itu toolbar juga memberikan kesempatan kepada user untuk mengkustomisasi tampilan dari aplikasi. Karena layout toolbar sangat fleksibel, user bisa memindah-mindahkan letak toolbar di dalam aplikasi, di atas, di bawah atau di samping, atau bahkan mengambang (floating) di atas jendela yang sedang aktif.

Dalam contoh program di atas kita akan membuat sebuah JToolBar dengan dua buah JButton yang telah didekorasi dengan icon cantik. Icon yang digunakan banyak tersedia di internet, format file yang dipilih adalah .png, karena format file ini paling bagus dalam menangani transparansi komponen.

Sebelum mulai membuat JToolBar, kita perlu mempersiapkan terlebih dahulu icon yang akan digunakan sebagai dekorasi JButton. Ikuti langkah-langkah berikut ini :

1. Buatlah sebuah Java package baru untuk menampung semua icon yang akan digunakan. caranya klik kanan di jendela Projects bagian nama project, pilih menu :

New > Java Package

Beri nama images untuk Java package yang baru saja kita buka.

2. Memasukkan Icon ke dalam package. Untuk memasukkan image ke dalam package kita perlu tahu dimana project disimpan, misalkan project disimpan dalam folder :

c:\javaswing

Buka file explorer, kemudian navigasi ke folder

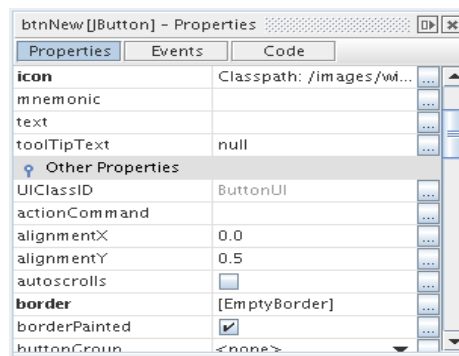
c:\javaswing\src\images

Copy semua icon yang diperlukan ke dalam folder di atas.

3. Build project. Langkah ini diperlukan untuk mengcompile semua file .java menjadi file .class. Selain itu proses ini juga akan mengkopi file selain file .java (termasuk file icon) ke dalam folder build\classes. Jika proses ini tidak dilaksanakan, maka ketika program dijalankan, file icon tidak akan ditemukan dan program menjadi error .

Setelah proses persiapan selesai, lakukan langkah-langkah berikut ini untuk membuat Toolbar :

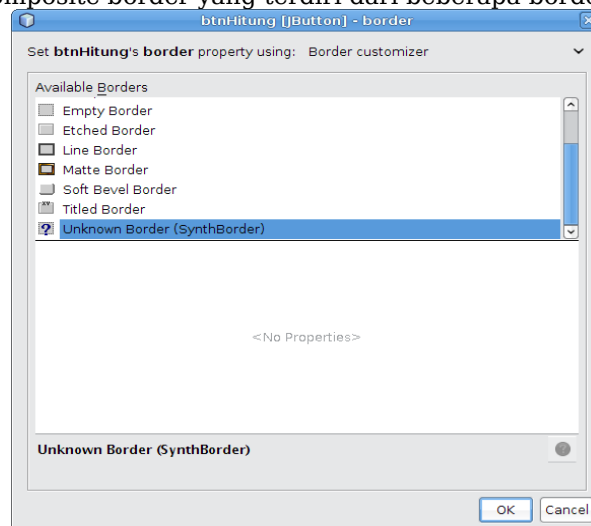
1. Buka class ToolbarMenu.java yang sudah dibuat di langkah sebelumnya.
2. Buat sebuah object JToolBar, caranya : klik item JToolBar dari Jendela Palette, kemudian klik JFrame di Jendela Design. Secara otomatis sebuah object JToolBar akan dimasukkan ke dalam JFrame. Ganti namanya menjadi toolbar.
3. Menambahkan JButton dalam JToolBar. Klik item JButton dalam Jendela Palette kemudian klik komponen JToolBar yang baru saja kita buat tadi. JButton baru akan diletakkan di atas JToolBar, ganti nama JButton tersebut menjadi btnNew. Letakkan lagi satu buah JButton di atas JToolBar dan beri nama btnMaximize.
4. Mendekorasi Tampilan JButton. Agar tampilan JButton terlihat cantik, kita perlu mengeset beberapa nilai dari properti JButton, seperti terlihat pada gambar di bawah ini :



Jendela Properties JButton

- Text, hapus nilai textnya.
- Border, pilih bordernya menjadi empty border dan set nilai bordernya menjadi [5,5,5,5]. Tujuan pemberian empty border ini agar tombol berukuran lebih besar dibandingkan dengan icon yang akan digunakan nanti, dan setiap mouse melewati JButton, ada efek transisi yang cantik.

Untuk mengedit border dari JButton, Matisse menyediakan Jendela Border untuk memilih border yang kita inginkan untuk JButton. Border yang dipilih bisa single border, atau composite border yang terdiri dari beberapa border.



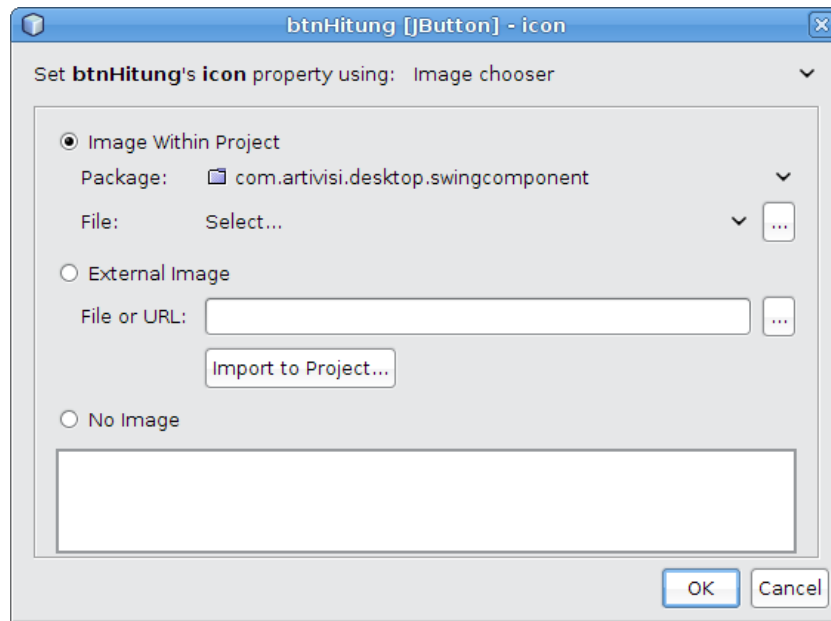
Jendela Border Editor dari JButton

- Opaque, uncheck nilai opaque. Bertujuan agar tombolnya berwarna transparan, sehingga mempunyai warna background yang sama dengan background JToolBar.
- Icon, ganti iconnya dengan icon yang telah disiapkan. Untuk memasukkan icon ke dalam JButton, tekan tombol di samping pilihan Icon di dalam Jendela Properties, kemudian akan muncul Dialog Icon Editor seperti di bawah ini :

Jendela icon editor

Pilih radio button Classpath, kemudian tekan tombol Select File dan pilih salah satu icon yang telah disiapkan. Tekan OK. Lakukan langkah-langkah yang sama terhadap JButton yang lain.

5. Compile dan jalankan class ToolbarMenu untuk melihat hasilnya.



Membuat Dialog dan JFileChooser

Dialog memerankan peran yang penting dalam aplikasi berbasis desktop. Interaksi antara user dengan aplikasi terkadang tidak berjalan dengan baik karena user memberikan aksi yang tidak valid kepada aplikasi. Ketika hal tersebut terjadi, aplikasi harus memberitahukan kepada user apa yang telah terjadi dan bagaimana seharusnya user memperbaikinya. Model interaksi seperti ini tepat dilaksanakan menggunakan dialog.

Skenario lain adalah ketika aplikasi memerlukan input dari user agar aplikasi bisa terus melaksanakan tugasnya, misalnya meminta konfirmasi apakah user yakin akan melaksanakan sebuah aksi penting terhadap aplikasi seperti delete, update atau add data.

Dialog juga memberikan pembatasan kepada user, sebelum dialog selesai diproses, user tidak akan bisa berinteraksi dengan bagian aplikasi lainnya. Dialog mencegah hal ini terjadi dengan memastikan bahwa jendela yang bisa diaktifkan hanyalah jendela dialog, sedangkan jendela aplikasi yang lain tidak dapat diaktifkan selama jendela dialog masih aktif.

Aplikasi sangat sering menggunakan dialog untuk berinteraksi dengan user, tetapi jenis interaksinya selalu seragam dan berulang-ulang. Swing menyediakan dialog yang didesign untuk keperluan yang sering muncul dalam aplikasi, seperti JOptionPane dan JFileChooser. Swing juga menyediakan class JDialog jika kita ingin membuat dialog custom sesuai keinginan kita.

Membuat pre-defined dialog dengan JOptionPane




JOptionPane menyediakan beberapa dialog yang siap pakai dan sering digunakan dalam aplikasi. JOptionPane sangat memudahkan kita dalam meminta user suatu input tertentu atau memberitahu user apa yang terjadi dalam aplikasi.

JOptionPane mempunyai banyak static method untuk menampilkan popup dialog dengan mudah. Terdapat empat method utama yang dapat kita gunakan sebagai landasan membuat dialog. Keempat method tersebut secara rinci digambarkan dalam table berikut ini:

Method	Deskripsi
showConfirmDialog	Meminta konfirmasi dari user, seperti yes/no/cancel
showInputDialog	Meminta input dari user, baik berupa input text menggunakan JTextField maupun pilihan menggunakan JComboBox
showMessageDialog	Memberitahukan user tentang apa yang baru saja terjadi
showOptionDialog	Gabungan dari ketiga jenis dialog di atas

Swing juga menyediakan method `showInternalXXX` yang digunakan jika kita bekerja dengan `JInternalFrame`.

Parameter dari keempat method tersebut mengikuti pola yang konsisten. Terurut dari kiri ke kanan, berikut ini parameter-parameter yang bisa diterima oleh method-method dalam class `JOptionPane`:

1. `parentComponent` : Mendefinisikan komponen yang akan menjadi parent dari dialog box ini. Frame dari parent component tersebut akan menjadi frame dari dialog dan dialog akan ditampilkan di tengah-tengah parent component. Jika nilai dari `parentComponent` diset null, maka dialog akan menggunakan frame default dan dialog akan diletakkan ditengah-tengah layar monitor (tergantung L&F).
2. `message` : Pesan yang deskriptif menerangkan perihal dialog yang muncul. Pada umumnya `message` berupa pesan String yang akan diletakkan dalam dialog, namun jenis object lain juga diijinkan digunakan sebagai `message`. Object-object yang diijinkan akan diperlakukan berbeda, object-object tersebut antara lain
 - a) `Object[]` : Setiap object akan ditampilkan dalam dialog berurut dari atas ke bawah. Aturan ini berlaku rekursif untuk semua object didalam array.
 - b) `Component` : Jika object yang dimasukkan sebagai `message` bertipe `Component`, maka `Component` tersebut akan ditampilkan ditengah-tengah dialog.
 - c) `Icon` : Icon akan dimasukkan ke dalam sebuah `JLabel` kemudian ditampilkan di sebelah kiri dari dialog.
 - d) `others` : Object lainya akan ditampilkan dalam dialog dengan mengambil nilai kembalian dari method `toString` dari setiap object.
3. `messageType` : Mendefinisikan jenis dari pesan. Pada umumnya memberikan custom icon untuk setiap jenis pesan. Setiap L&F manager akan memperlakukan setiap jenis pesan dengan berbeda, namun perbedaanya tidak akan terlalu mencolok. Pilihan yang mungkin dan icon yang mewakilinya adalah:
 - a) `ERROR_MESSAGE` 
 - b) `INFORMATION_MESSAGE` 
 - c) `WARNING_MESSAGE` 
 - d) `PLAIN_MESSAGE` (tanpa icon)
4. `optionType` : Mendefinisikan tombol-tombol yang akan ditampilkan di bagian bawah dari dialog.
 - a) `DEFAULT_OPTION`
 - b) `YES_NO_OPTION`
 - c) `YES_NO_CANCEL_OPTION`
 - d) `OK_CANCEL_OPTION`

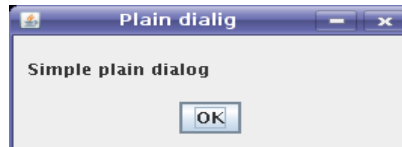
Namun kita tidak dibatasi untuk hanya menggunakan empat jenis set tombol di atas, kita dapat mendefinisikan tombol-tombol yang akan muncul sesuai kebutuhan.

5. `Options` : Deskripsi yang lebih detail dari set tombol yang digunakan dialog. Nilai yang lazim adalah sebuah array String berisi text yang akan ditampilkan di setiap tombol. Namun Object lain juga dapat diterima, antara lain:
 - a) `Component`
Component akan diletakkan dalam baris tombol secara langsung.
 - b) `Icon`
Sebuah `JButton` akan dibuat dan didekorasi dengan icon ini.
 - c) `other`
Object dengan tipe selainnya akan dirubah ke dalam bentuk String dengan mengambil nilai kembalian dari method `toString` dari object tersebut.

6. Icon : Icon yang digunakan untuk mendekorasi dialog. Jika icon ini didefinisikan maka akan menempa icon default yang didefinisikan oleh messageType.
7. Title : Judul dari dialog yang diletakkan di bagian paling atas dari dialog.
8. InitialValue : Nilai default dari pilihan yang mungkin ada dalam dialog.

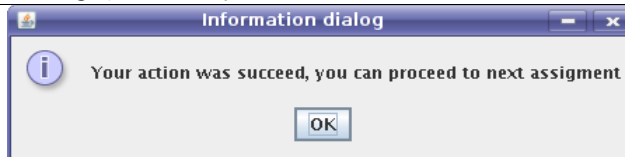
Untuk lebih jelasnya, berikut ini beberapa contoh kode penggunaan JOptionPane beserta hasil tampilanya :

```
JOptionPane.showMessageDialog(null,
    "Simple plain dialog","Plain dialog", JOptionPane.PLAIN_MESSAGE);
```



Tampilan dialog sederhana

```
JOptionPane.showMessageDialog(null,
    "Your action was succeed, " +
    "you can proceed to next assigment",
    "Information dialog",    JOptionPane.INFORMATION_MESSAGE);
```



Tampilan dialog dengan tipe dialog Information

```
JOptionPane.showMessageDialog(null,
    "You neet to be sure to do this action!",
    "Dialog Peringatan", JOptionPane.WARNING_MESSAGE);
```



Dialog dengan tipe Warning

```
JOptionPane.showMessageDialog(null,
    "Something goes wrong and generate error message",
    "Error Dialog", JOptionPane.ERROR_MESSAGE);
```

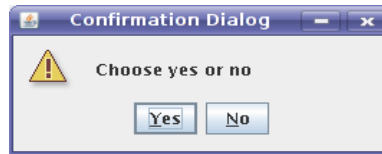


Dialog dengan tipe Error

```
JOptionPane.showConfirmDialog(null,
    "Choose yes or no","Confirmation Dialog",
    JOptionPane.YES_NO_OPTION,
```

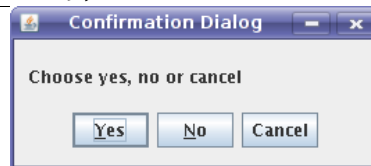


```
JOptionPane.WARNING_MESSAGE);
```



Option dialog dengan tipe Information dan pilihan YES_NO

```
JOptionPane.showConfirmDialog(null,  
    "Choose yes, no or cancel","Confirmation Dialog",  
    JOptionPane.YES_NO_CANCEL_OPTION,  
    JOptionPane.PLAIN_MESSAGE);
```



OptionDialog dengan tipe Plain dan pilihan YES NO CANCEL

```
JOptionPane.showInputDialog(null,  
    "Input your name here","Input Dialog",  
    JOptionPane.INFORMATION_MESSAGE);
```



InputDialog dengan tipe message Information

```
String[] options = {"Apple","Mango","Grape","Guava"};  
JOptionPane.showInputDialog(null,  
    "Choose this one Option","Input dialog",  
    JOptionPane.WARNING_MESSAGE,null,options,"Apple");
```



InputDialog dialog dengan tipe Warning, Options berupa array of String dan initialValue = 'Apple'

Membuat JFileChooser

JFileChooser digunakan untuk bernavigasi dalam file system, kemudian memilih satu atau lebih file atau folder dari list file dan folder. JFileChooser pada dasarnya adalah pengembangan dari dialog yang dapat digunakan untuk memilih file. JFileChooser dapat digunakan sebagai dialog untuk menyimpan file atau untuk membuka file.

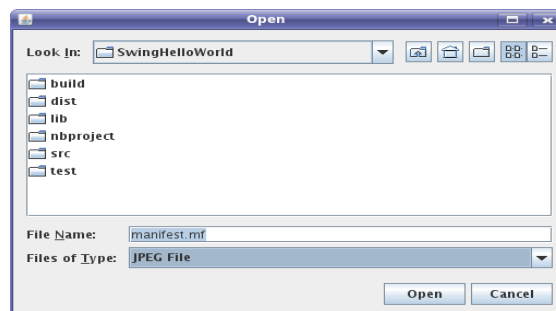
JFileChooser hanya memberikan fasilitas untuk memilih file atau folder, sedangkan mekanisme untuk menyimpan atau membuka file dilakukan sendiri menggunakan library I/O.

Aplikasi berikut ini adalah contoh penggunaan JFileChooser untuk membuka dan menyimpan file.



Contoh program menggunakan JFileChooser

Tampilan JFileChooser ketika tombol open ditekan adalah seperti di bawah ini :



Tampilan JFileChooser

Untuk membuat aplikasi di atas lakukan langkah-langkah berikut ini :

1. Buat class JFrame Form baru, beri nama Chooser.java
2. Masukkan dua buah JTextField : txtOpen dan txtSave, dua buah JButton : btnOpen dan btn save, sebuah JLabel : lblStatus. Sesuaikan penataan komponen sesuai dengan gambar di atas.
3. Tambahkan sebuah object JFileChooser sebagai field dari class Chooser, beri nama chooser.

```
public class Chooser{
    JFileChooser chooser = new JfileChooser();
    //kode lain di sini
}
```

4. FileNameExtentionFilter digunakan sebagai file filter dalam JFileChooser. Metode filteringnya adalah mencocokkan ekstensi file dalam file system dengan ekstensi yang ada dalam FileNameExtentionFilter. Contoh kode di bawah ini akan menyebabkan JFileChooser mempunyai pilihan "JPEG File", dan jika pilihan tersebut dipilih, maka file dengan ekstensi "jpg", "jpeg", "JPG" atau "JPEG" saja yang akan ditampilkan oleh JFileChooser.

```
FileNameExtensionFilter JPEGFilter = new FileNameExtensionFilter(
    "JPEG File","jpg","jpeg","JPG","JPEG");
chooser.addChoosableFileFilter(JPEGFilter);
```

5. Set direktori yang akan dituju oleh JFileChooser. Untuk mengetahui dimana direktori aktif aplikasi, kita bisa menggunakan system property "user.dir". Kode berikut ini akan menyebabkan JFileChooser dibuka pada direktori aktif aplikasi :

```
String dir = System.getProperty("user.dir");
chooser.setCurrentDirectory(new File(dir));
```

6. Menghandle event penekanan tombol btnSave. Ketika tombol btnSave ditekan, chooser akan menampilkan dialog save file, kemudian mengambil nama file yang dipilih dan menampilkannya dalam txtSave, serta menampilkannya dalam lblStatus. Berikut ini kodenya :

```
private void btnSaveActionPerformed(ActionEvent evt) {
    int ret = chooser.showSaveDialog(this);
    if(ret == JFileChooser.APPROVE_OPTION){
        File f = chooser.getSelectedFile();
        lblStatus.setText("Status : saving file" + f.getAbsolutePath());
    }
}
```

```
        txtSave.setText(f.getAbsolutePath());  
    }  
}
```

7. Menghandle penekanan tombol btnOpen. Kode untuk menangani penekanan tombol btnOpen mirip dengan kode untuk menangani penekanan tombol btnSave, perbedaanya adalah btnOpen akan menampilkan dialog open file, berikut ini kodenya :

```
private void btnBrowseActionPerformed(ActionEvent evt){  
    int ret = chooser.showOpenDialog(this);  
    if(ret == JFileChooser.APPROVE_OPTION){  
        File f = chooser.getSelectedFile();  
        lblStatus.setText("Status : opening file" + f.getAbsolutePath());  
        txtOpen.setText(f.getAbsolutePath());  
    }  
}
```

8. Compile dan jalankan aplikasinya dengan menekan tombol SHIFT + F6

Bekerja dengan JOptionPane dan dengan JFileChooser sangat sederhana. Keduanya menggunakan modal dialog untuk mengambil input dari user. Modal dialog akan mencegah user mengakses bagian aplikasi lain sebelum dialog ditutup, atau dalam hal ini memutuskan pilihan apa yang diambil oleh user.

Masih banyak lagi komponen Swing yang disediakan oleh JDK, anda tinggal melanjutkan membaca dari referensi yang diberikan modul ini pada bagian akhir untuk melanjutkan pembelajaran anda tentang Java desktop.

Konsep MVC

MVC adalah arsitektur aplikasi yang memisahkan kode-kode aplikasi dalam tiga lapisan, Model, View dan Control. MVC termasuk dalam arsitektural design pattern yang menghendaki organisasi kode yang terstruktur dan tidak bercampur aduk. Ketika aplikasi sudah sangat besar dan menangani struktur data yang kompleks, harus ada pemisahan yang jelas antara domain model, komponen view dan kontroler yang mengatur penampilan model dalam view.

Arsitektur MVC ini memungkinkan adanya perubahan dalam domain model tanpa harus mengubah code untuk menampilkan domain model tersebut. Hal ini sangat bermanfaat ketika aplikasi mempunyai domain model dan view komponen sangat besar dan kompleks.

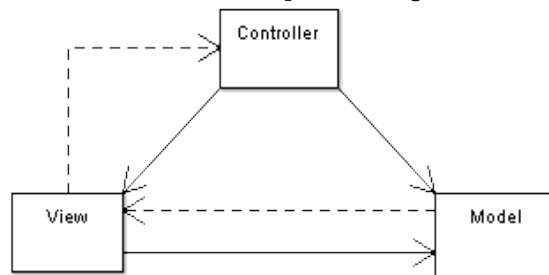


Diagram interaksi antar komponen dalam arsitektur MVC (Wikipedia.org)

Model adalah representasi dari object yang sedang diolah oleh aplikasi, dalam Java, model ini biasanya direpresentasikan sebagai Java Bean. Java Bean adalah class Java biasa atau POJO (Plain Old Java Object). Syarat sebuah POJO dianggap sebagai Java Bean adalah :

1. Mempunyai constructor default, constructor yang tidak mempunyai parameter.
2. Semua field-field yang bisa diakses dilengkapi dengan getter dan setter method.

Lebih jelasnya lihat kode dari class Person di bawah ini :

```
public class Person {
    private Long id;
    private String name;
    private String email;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String aEmail) {
        this.email = aEmail;
    }
}
```

Kode di atas adalah representasi Model dalam Java untuk Entity Person. Beberapa orang terkadang salah mengartikan model ini sebagai data akses domain. Dimana data dari sumber

data, misalnya database, diambil dan diolah. Pada hakekatnya Model adalah representasi data dari object sebenarnya, bukan kumpulan kode untuk mengakses data dari database.

Pendekatan terbaik adalah memisahkan kode untuk melakukan akses sumber data ke dalam lapisan tersendiri, lapisan ini biasanya disebut sebagai service. Service diimplementasikan dalam bentuk class-class yang disebut sebagai manager, misalnya SQLManager, PrintManager, ReportManager, XMLManager, WebServiceManager dan seterusnya. Dengan begitu kode akan menjadi lebih rapi dan terstruktur. Manfaat paling terasa adalah kemudahan pencarian kesalahan dan penambahan modul-modul baru tidak harus merombak seluruh struktur aplikasi.

View adalah komponen untuk merepresentasikan Model dalam bentuk visual. Semisal komponen Swing, seperti : JTable, JList, JComboBox dan sebagainya. View juga bertanggung jawab untuk menangkap interaksi user terhadap sistem, semisal : klik mouse, penekanan tombol keyboard, barcode scanning dan sebagainya.

Controller sebenarnya hanya sekumpulan kode-kode untuk mensinkronisasi keadaan Model dan View. Jika ada perubahan data dari Model, Controller harus mengupdate tampilan View. Dan sebaliknya jika user memberikan event terhadap View, Controller harus mengupdate Model sesuai dengan hasil interaksi user terhadap View.

Model dalam Komponen Swing

Sebagian besar komponen Swing mempunyai model. JButton mempunyai model yaitu ButtonModel yang memegang 'state' dari JButton - apa keyboard mnemonicnya, apakah JButton tersebut sedang dipilih atau tidak dan seterusnya. Ada pula komponen Swing yang mempunyai lebih dari satu model. JList mempunyai ListModel yang memegang isi dari JList dan ListSelectionModel untuk mencatat item JList yang sedang dipilih.

Pada banyak kasus normal kita tidak perlu pusing memikirkan model ini. Semisal kita tidak perlu memikirkan model dari JButton karena pada kasus umum kita tidak perlu memodifikasi model dari JButton.

Lalu, kenapa model komponen Swing dibuat? Alasan utamanya adalah fleksibilitas untuk menentukan bagaimana data disimpan dan diambil dari komponen Swing. Misalnya kita mempunyai aplikasi spreadsheet yang menggunakan komponen JTable, karakteristik utama spreadsheet adalah banyak cell yang kosong, dengan begitu kita bisa memilih model data yang sesuai dengan karakteristik tersebut.

Contoh lainnya adalah JTable yang digunakan untuk menampilkan data dari database dengan jumlah baris luar biasa banyak. Kita bisa mengatur agar tampilan JTable dibuat halaman-per-halaman dalam menampilkan baris data, tidak semua data ditampilkan dalam satu halaman, hal ini ditujukan untuk efisiensi dan mempertahankan agar aplikasi tetap responsif walau bekerja dengan data yang besar.

Model dalam komponen Swing juga mempunyai keuntungan lain, yaitu tidak perlu ada dua data terpisah, untuk struktur data aplikasi dan untuk komponen Swing.

Kegunaan Model yang cukup penting juga adalah adanya konsep event-listener, dimana jika terjadi event perubahan data dalam model, semua listener yang terdaftar dalam model tersebut akan diberitahu dan tindakan yang tepat dapat diambil untuk menangani event yang muncul. Sebagai contoh, untuk menambahkan item dalam JList kita bisa memanggil method addItem dari JList. Penambahan item dalam JList ini akan mengakibatkan ListModel memicu event dalam JList dan listener lainnya. Komponen Swing—dalam hal ini JList—akan diupdate tampilanya untuk merefleksikan perubahan item dalam ListModel.

Walaupun terkadang banyak yang menyebut arsitektur komponen Swing sebagai MVC, tetapi pada dasarnya arsitektur komponen Swing tidak sepenuhnya MVC. Komponen Swing secara umum dibuat agar View dan Controller diletakkan dalam satu tempat (class) yaitu class UI yang disediakan oleh Look-and-Feel. Arsitektur komponen Swing lebih tepat disebut sebagai "Arsitektur dengan Model yang terpisah".

Selanjutnya kita akan membahas beberapa model yang seringkali harus kita kustomisasi sesuai dengan kebutuhan. Sedangkan model yang nyaris tidak pernah kita rubah—ButtonModel—tidak dibahas dalam bagian ini.

TableModel

TableModel adalah class model yang paling sering dikustomisasi. Karakteristik data dari JTable yang berbentuk koleksi data dua dimensi membutuhkan perhatian khusus agar efisien digunakan dalam aplikasi. Jika kita tidak hati-hati, maka aplikasi kita bisa menjadi sangat lambat dan tidak efisien.

TableModel adalah interface yang digunakan oleh JTable untuk mendefinisikan ciri-ciri dari data tabular yang akan ditampilkan oleh JTable. Misalnya : jumlah kolom, nama kolom, class dari object dalam kolom, jumlah baris dan nilai setiap cell. Dengan adanya data-data ini JTable dapat secara efisien menentukan bagaimana menampilkan data tersebut.

Berikut ini adalah kode untuk menampilkan koleksi object Person. Class ArrayList<Person> adalah implementasi dari generics, konsep dalam Java yang digunakan untuk mendefinisikan isi dari koleksi. ArrayList<Person> artinya adalah membuat sebuah object koleksi ArrayList yang harus diisi oleh object Person dan tidak bisa diisi oleh object lainya, misalnya String.

```
public class PersonTableModel extends AbstractTableModel{
    private List<Person> persons;
    public PersonTableModel(List<Person> persons) {
        this.persons = persons;
    }
    public int getRowCount() {
        return persons.size();
    }
    public int getColumnCount() {
        return 3;
    }
    public Object getValueAt(int rowIndex, int columnIndex) {
        Person p = persons.get(rowIndex);
        switch(columnIndex){
            case 0 : return p.getId();
            case 1 : return p.getName();
            case 2 : return p.getEmail();
            default : return "";
        }
    }
    @Override
    public String getColumnName(int column) {
        switch(column){
            case 0 : return "ID";
            case 1 : return "NAME";
            case 2 : return "EMAIL";
            default : return "";
        }
    }
}
```

Yang perlu diperhatikan bahwa dalam AbstractTableModel, method isCellEditable selalu mengembalikan nilai false, artinya semua cell tidak dapat diedit. Kemudian method setValueAt adalah method kosong belaka, artinya jika kita memanggil method ini tidak akan terjadi apa-apa.

Class kedua adalah DefaultTableModel yang telah mengimplementasi semua method abstract dari interface TableModel. Representasi data DefaultTableModel menggunakan dua jenis data tabular, yaitu array dua dimensi, Object[][], dan Vector dari Vector, Vector<Vector<Object>>. Jika kita mempunyai struktur data selain kedua jenis tersebut kita harus melakukan konversi data ke dalam salah satu bentuk struktur data tersebut. Cara yang lebih cerdas adalah mendefinisikan sendiri class yang mengimplement interface TableModel seperti class CustomerTableModel di atas.

Setelah TableModel selesai didefinisikan kita tinggal memanggil method setTableModel dari

object `JTable`, atau membuat object `JTable` baru menggunakan constructor yang menerima argumen `TableModel`. Contohnya seperti potongan kode di bawah ini :

```
JTable table = new JTable(new DefaultTableModel());
JTable table1 = new JTable();
table1.setModel(new DefaultTableModel());
```

ListModel

`JList` adalah komponen Swing yang mempunyai dua model sekaligus, `ListModel` dan `ListSelectionModel`. `ListModel` digunakan untuk mendefinisikan item/element yang dikandung oleh `JList`. Sedangkan `ListSelectionModel` digunakan untuk mendefinisikan bagaimana representasi data jika terjadi proses pemilihan di `JList`.

Seperti halnya `TableModel`, `ListModel` mempunyai dua class yang mengimplement `ListModel`, `AbstractListModel` dan `DefaultListModel`. Kita bisa menggunakan salah satu dari tiga tipe tersebut untuk membuat object `ListModel`. Cara pertama dengan membuat class baru yang mengimplement `ListModel`. Cara kedua dengan membuat class baru yang `extends` `AbstractListModel` dan cara ketiga dengan langsung menggunakan `DefaultListModel`.

Struktur data `JList` tidak terlalu rumit seperti `JTable`, dan pada umumnya, cukup hanya dengan menggunakan `DefaultListModel` sudah memenuhi sebagian besar kebutuhan penggunaan `JList`.

Berikut ini contoh bagaimana membuat `ListModel` untuk data customer, contoh ini menggunakan cara kedua untuk membuat object `ListModel`, yaitu dengan cara membuat class baru yang `extends` `AbstractListModel` :

```
public class CustomerListModel extends AbstractListModel{
    private ArrayList<Person> customer =
        new ArrayList<Person>();
    public CustomerListModel(List<Person> cust){
        customers.addAll(cust);
    }
    public Object getValueAt(int index) {
        return customers.get(index);
    }
    public int getSize() {
        return customers.size();
    }
}
```

Implementasi `ListModel` sangat mudah dan tidak serumit `TableModel`, namun implementasi dari `ListSelectionModel` sangat rumit, karena kita harus mengimplementasi dua puluh buah method. Lebih baik menggunakan implementasi standard dari `ListSelectionModel` yaitu `DefaultListSelectionModel`.

Renderer

`Renderer` dan `Editor` adalah View dari pattern MVC. `Renderer` digunakan untuk menampilkan data di komponen Swing. Beberapa komponen Swing memungkinkan programmer untuk mendefinisikan `Renderer` yang akan digunakan. Jika `renderernya` menggunakan default `renderer` yang disediakan oleh Swing, maka String yang ditampilkan dalam komponen adalah hasil return method `toString` object tersebut. Misalnya entity `Role` akan ditampilkan dalam sebuah combobox, maka string yang tampil akan berasal dari method `toString` class `Role`, kalau method `toString`-nya masih bawaan dari class `Object` dan tidak `dioverride` oleh class `Role`, return dari String adalah nama class-nya diappend oleh alamat memory dimana object `role` itu berada. Misalnya `com.googlecode.projecttemplate.pos.model.Role@54fc9944`, hal seperti ini bisa dihindari dengan `mengoverride` method `toString` dari `role`, misalnya :

```
@Override
public String toString() {
    return "Role{" + "name=" + name + '}';
}
```

```
}
```

Misalnya kita perlu menampilkan class Role dalam beberapa combobox yang berbeda tetapi string yang ditampilkan berbeda pula, misalnya di satu combo box akan ditampilkan id-nya di combo box yang lain ingin ditampilkan name-nya. Masalah ini tidak bisa diselesaikan dengan hanya mengoverride method toString, kita harus membuat renderer baru untuk menampilkan id role atau menampilkan nama role.

Renderer untuk combo box dibuat dengan mengimplementasikan ListCellRenderer, tetapi membuat renderer langsung dengan mengimplementasikan ListCellRenderer tidaklah mudah. Cara terbaik adalah mengextends renderer yang sudah ada yaitu DefaultListCellRenderer kemudian mengoverride method untuk menampilkan String di dalam JComboBox, contohnya seperti berikut ini :

```
public class RoleIdComboBoxRenderer extends DefaultListCellRenderer{
    @Override
    public Component getListCellRendererComponent(JList list, Object value, int
        index, boolean isSelected, boolean cellHasFocus) {
        JLabel renderer = (JLabel) super.getListCellRendererComponent(
            list, value, index, isSelected, cellHasFocus);
        if(value instanceof Role){
            Role r = (Role) value;
            renderer.setText(String.valueOf(r.getId()));
        }
        return renderer;
    }
}
```

Class RoleIdComboBoxRenderer menextends DefaultListCellRenderer kemudian hanya mengoverride satu buah method yaitu getListCellRendererComponent. Method tersebut sudah ada implementasinya, kita hanya ingin mengubah cara renderer ini menampilkan String di dalam combo box saja, oleh karena itu method yang sudah diimplementasikan oleh DefaultListCellRenderer dieksekusi dengan menggunakan keyword super. Kemudian parameter value akan berisi object yang dimasukkan ke dalam combobox, value ditest menggunakan keyword instanceof untuk memastikan bahwa value tersebut adalah Role. Setelah itu set text dari JLabel menggunakan nilai id dari Role.

Setelah Renderer selesai dibuat, mari kita buat kode untuk memasang renderer tersebut ke dalam JComboBox

```
JComboBox combo = new JComboBox();
combo.setRenderer(new RoleIdComboBoxRenderer());
```

Editor

Editor digunakan oleh beberapa komponent untuk menentukan komponen apa yang digunakan untuk mengedit data dalam component. Sebagian besar component menggunakan component yang sama untuk Renderer dan Editor sehingga component tersebut akan terlihat sama ketika berada dalam mode normal atau mode edit. Salah satu component yang menggunakan Editor dan Renderer berbeda adalah Jtable. Perpindahan mode dari standard ke edit dalam Jtable terjadi jika user melakukan double click terhadap cell-cell dalam Jtable. Dalam mode normal Jtable menggunakan JLabel untuk menampilkan datanya, tetapi akan menggunakan JTextField untuk data String dan Number ketika berada dalam mode Edit dan menggunakan JcheckBox untuk data dengan tipe Boolean. Untuk data lain Jtable tetap akan menggunakan JTextField.

Jika kita ingin misalnya menambahkan Jcalendar untuk tipe data Date, maka kita harus membuat custom Editor dan memasangkanya ke Jtable untuk tipe data Date, setelah itu harus dipastikan bahwa method getColumnClass mengembalikan class Date.

Sebagai contoh, kita akan membuat Jtable untuk Person, di property birthDate akan digunakan Editor yang khusus untuk tipe Date menggunakan component dari Jcalendar. Langkah pertama adalah membuat class JdateChooserCellEditor seperti di bawah ini :


```

public class JDateChooserCellEditor extends AbstractCellEditor implements
    TableCellEditor {
    private static final long serialVersionUID = 917881575221755609L;
    private JDateChooser dateChooser = new JDateChooser();
    public Component getTableCellEditorComponent(JTable table, Object value,
        boolean isSelected, int row, int column) {
        Date date = null;
        if (value instanceof Date) {
            date = (Date) value;
        }
        dateChooser.setDate(date);
        return dateChooser;
    }
    public Object getCellEditorValue() {
        return dateChooser.getDate();
    }
}

```

Setelah itu buat TableModel yang mengoverride getColumnClass agar mengembalikan class Date untuk column birthDate

```

public class PersonTableModel extends AbstractTableModel{
    private static final long serialVersionUID = 1L;
    private List<Person> persons;
    public PersonTableModel(List<Person> persons) {
        this.persons = persons;
    }
    public int getRowCount() {return persons.size();}
    public int getColumnCount() {return 3;}
    public Object getValueAt(int rowIndex, int columnIndex) {
        Person p = persons.get(rowIndex);
        switch(columnIndex){
            case 0: return p.getId();
            case 1: return p.getName();
            case 2: return p.getBirthDate();
            default: return "";
        }
    }
    @Override
    public Class<?> getColumnClass(int columnIndex) {
        switch(columnIndex){
            case 0: return Long.class;
            case 1: return String.class;
            case 2: return Date.class;
            default: return Object.class;
        }
    }
    @Override
    public boolean isCellEditable(int rowIndex, int columnIndex) {
        if(columnIndex == 1 || columnIndex == 2){
            return true;
        }
        return false;
    }
}

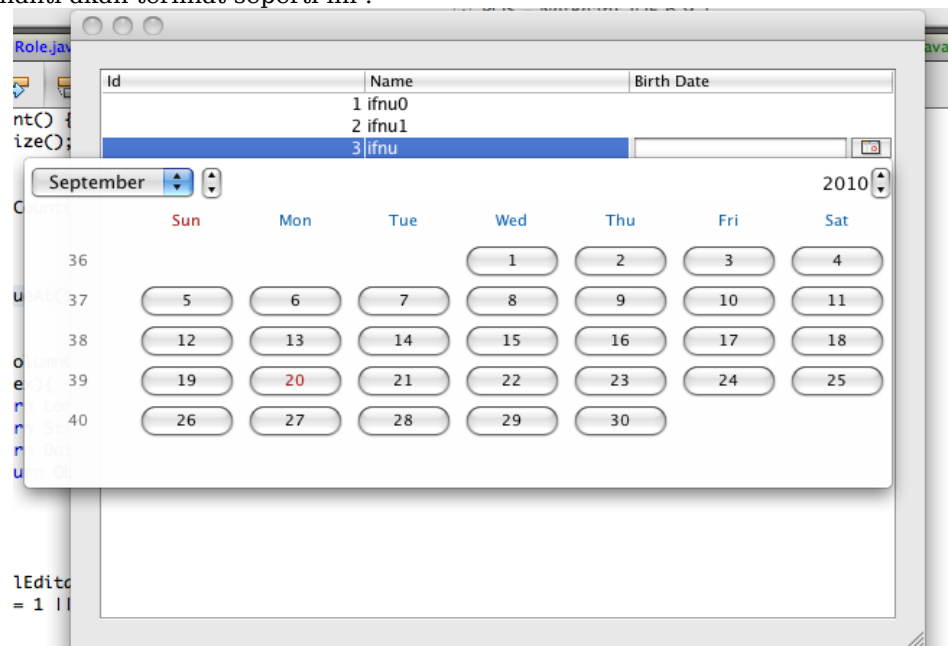
```

Langkah terakhir adalah mendaftarkan Editor ke dalam Jtable

```
Jtable table = new Jtable();
```

```
table.setDefaultEditor(Date.class, new JdateChooserCellEditor());
```

Hasilnya nanti akan terlihat seperti ini :



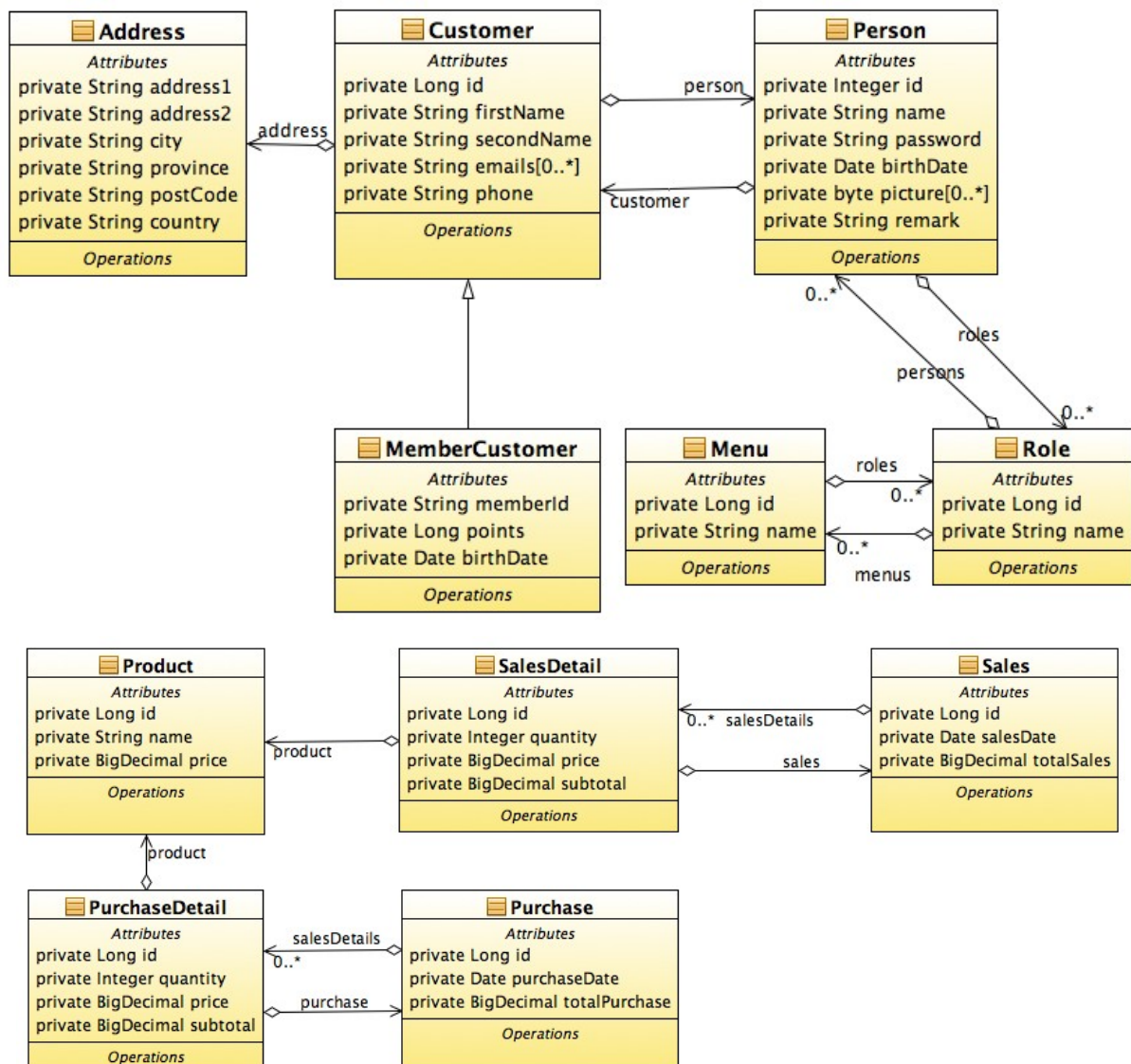
BAGIAN 4

APLIKASI POS

Aplikasi POS

Aplikasi POS digunakan oleh toko-toko atau retail kecil untuk mencatat transaksi. Jenis aplikasi ini cocok diimplementasikan dengan menggunakan Java Desktop karena karakteristiknya yang harus berinteraksi dengan hardware seperti printer, scanner dan terkadang cash drawer. Feature yang akan diimplementasikan dalam contoh buku ini meliputi Administrasi Pengguna hingga ke Access Control List (ACL), Produk, Pembelian Barang dan Penjualan. Feature advance seperti accounting dan stock tidak diimplementasikan untuk mempertahankan contoh aplikasi yang sederhana.

Struktur data yang digunakan dalam aplikasi POS ini sama persis dengan UML yang dibuat untuk contoh Mapping di bab Hibernate, untuk mengingat kembali struktur data tersebut, mari kita lihat class diagram di bawah ini :

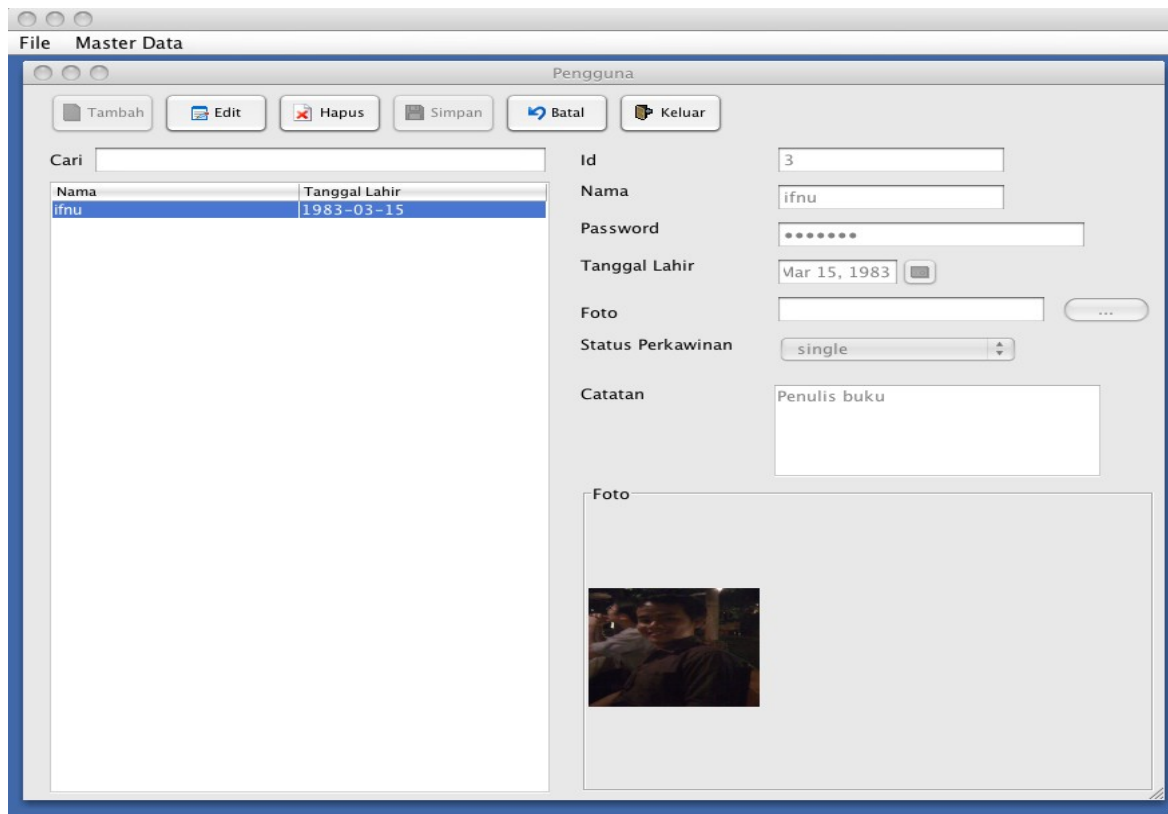


Class diagram pertama adalah data pengguna, role dan menu yang digunakan untuk menyimpan data login user sekaligus mendefinisikan ACL untuk user tersebut. Biasanya ada beberapa role dalam aplikasi POS, seperti Kasir, Administrator dan Super User. Modul kedua adalah data Transaksi untuk mencatat pembelian dan penjualan.

Dilihat dari sisi jenis datanya, data-data di atas dibagi menjadi dua jenis: Master dan Transaksi. Data Master adalah data-data yang dipersiapkan sebelum data Transaksi bisa dibuat, Data master ini relatif statis dan berubah sedikit-demi sedikit selama aplikasi berjalan, sedangkan data Transaksi selalu berubah ketika ada proses pembelian atau penjualan barang. Data pengguna, produk dan menu tergolong dalam jenis data Master, sedangkan pembelian dan penjualan tergolong dalam jenis data Transaksi. Dari sisi tampilan aplikasi, biasanya dibedakan antara screen untuk data master dan data transaksi, tampilan screen untuk data Master biasanya cukup sederhana, sedangkan untuk data transaksi dibuat sebaik mungkin agar proses input data cepat dan akurat. Di bagian berikutnya kita akan membuat screen untuk masing-masing data tersebut.

Data Master

Screen pertama yang akan kita buat adalah screen untuk input data Person (pengguna) yang tergolong data Master. Screen untuk data Master akan terlihat sama untuk data-data lainnya, sehingga cukup kita bahas satu saja di buku ini, screen-screen lain bisa dilihat di kode yang menyertai buku ini.



Di bagian paling atas terdapat tombol-tombol operasi data seperti : Tambah, Edit, Hapus, Simpan, Batal dan Keluar. Kemudian di sebelah kiri ada textfield untuk melakukan pencarian terhadap data Pengguna yang ditampilkan dalam table di bawahnya. Sebelah kanan adalah form untuk memanipulasi data Pengguna.

Cara menggunakan screen id dimulai dari membuka menu untuk menampilkan screen, kemudian user bisa menekan tombol tambah, berikutnya melengkapi semua data di form sebelah kanan, termasuk memilih foto yang akan ditampilkan. Setelah semua form selesai diisi, tombol simpan ditekan untuk menyimpan data ke dalam database. Data pengguna yang baru saja disimpan akan ditampilkan di dalam table yang ada di sebelah kiri, setiap kali baris dalam table dipilih, maka detail informasi Pengguna akan ditampilkan di sebelah kanan, tombol Edit

dan Hapus menjadi bisa diklik. Kalau tombol Edit ditekan, maka form sebelah kanan menjadi bisa diubah-ubah isinya. Setelah selesai melakukan edit terhadap data Pengguna, tombol save ditklik untuk menyimpan data ke dalam database.

Tombol Delete digunakan untuk menghapus data Pengguna dari database, tombol ini hanya menjadi bisa diklik kalau ada data pengguna dalam table yang dipilih. Proses penghapusan data tidak selalu bisa dilakukan kalau data Pengguna ini masih digunakan di tempat lain, misalnya data Role masih menyimpan data Pengguna yang akan dihapus, maka proses penghapusan akan gagal. Hal ini terjadi karena ada constraint foreign key data Pengguna di dalam data relasi antara Role-Pengguna.

Nah setelah mengetahui bagaimana cara menggunakan screen Pengguna di atas, mari kita bahas bagaimana cara membuat screen di atas.

Mendesign Screen

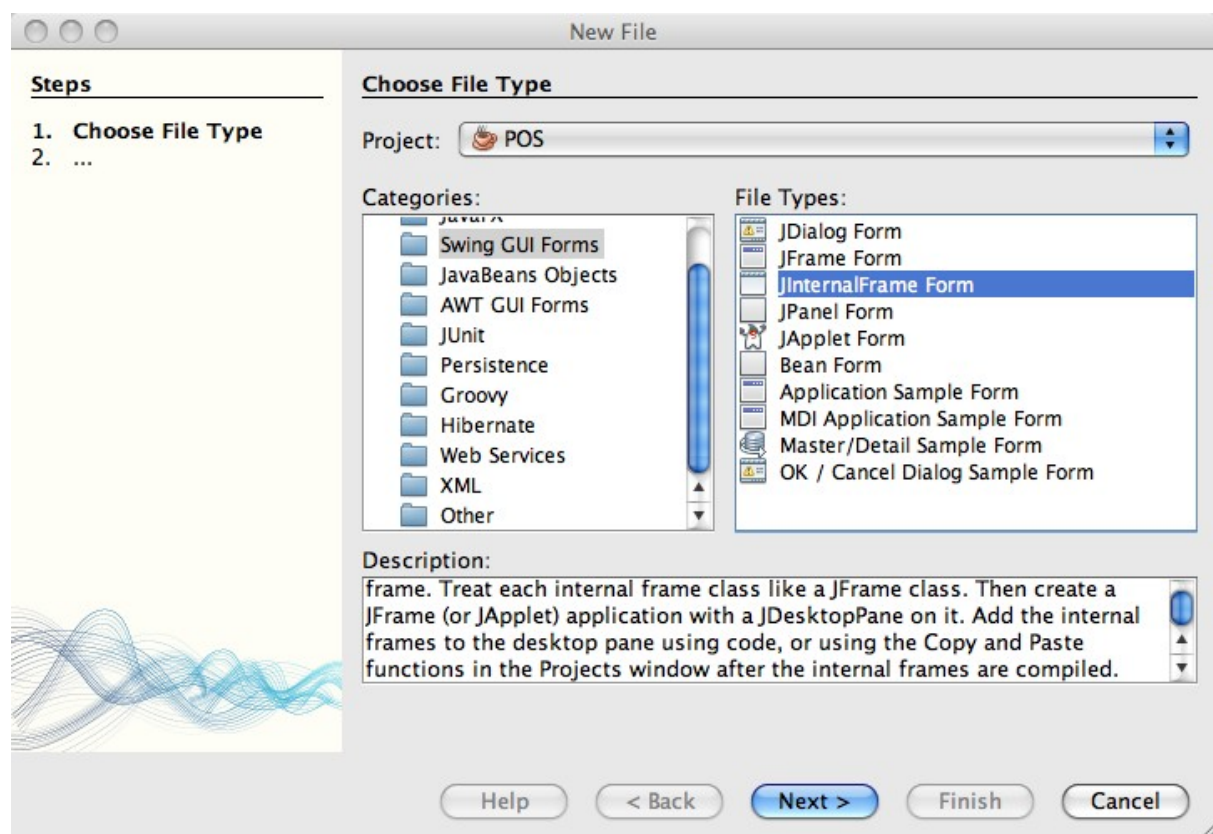
Langkah pertama adalah buat class PersonPanel dengan tipe JInternalFrame. Langkah-langkahnya adalah sebagai berikut :

1. Pilih menu untuk membuat file baru

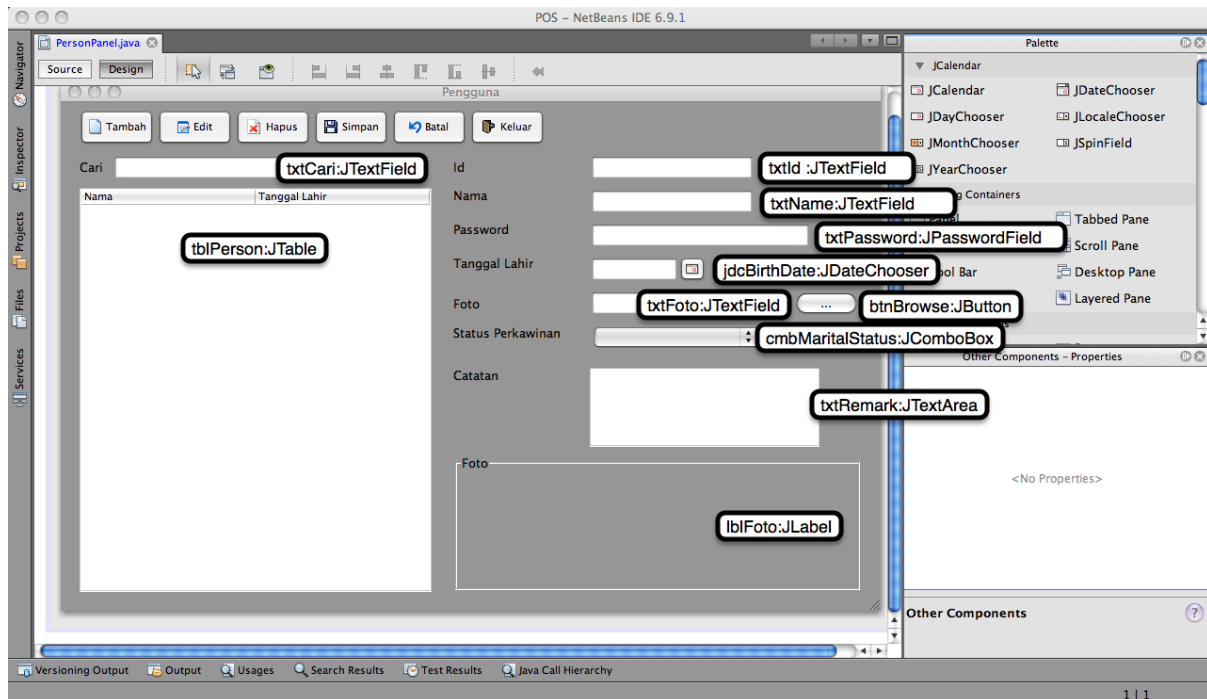
File > New File

Kemudian akan muncul jendela untuk memilih jenis file apa yang akan dibuat, pilih

Swing GUI Forms > JInternalFrame Form

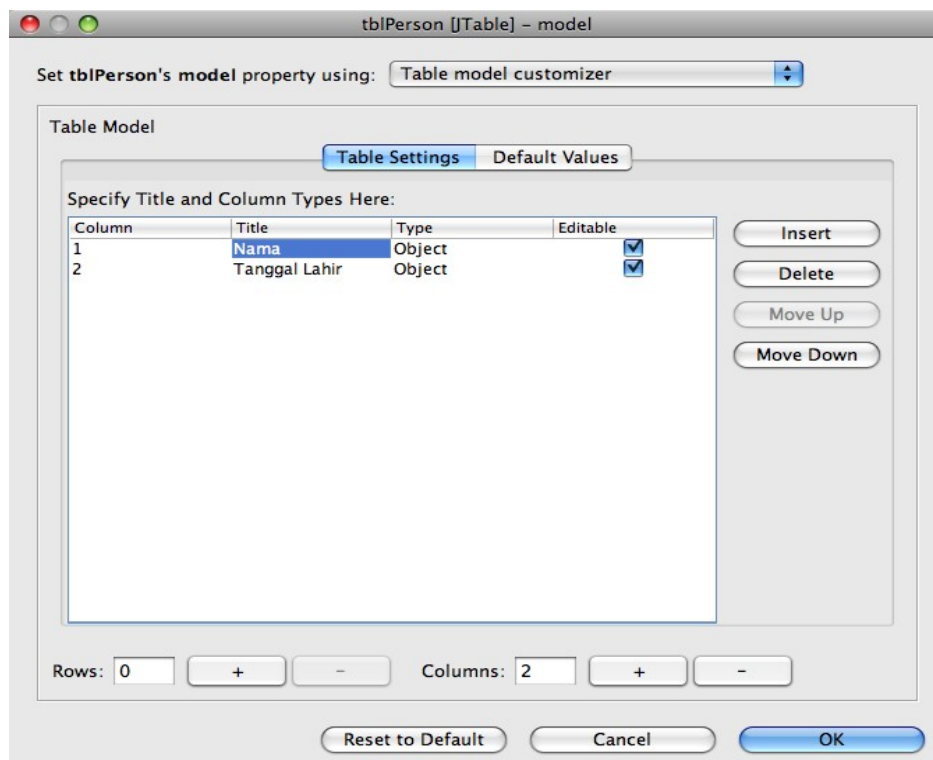


Beri nama PersonPanel dan letakkan di package com.googlecode.projecttemplate.pos.ui.security



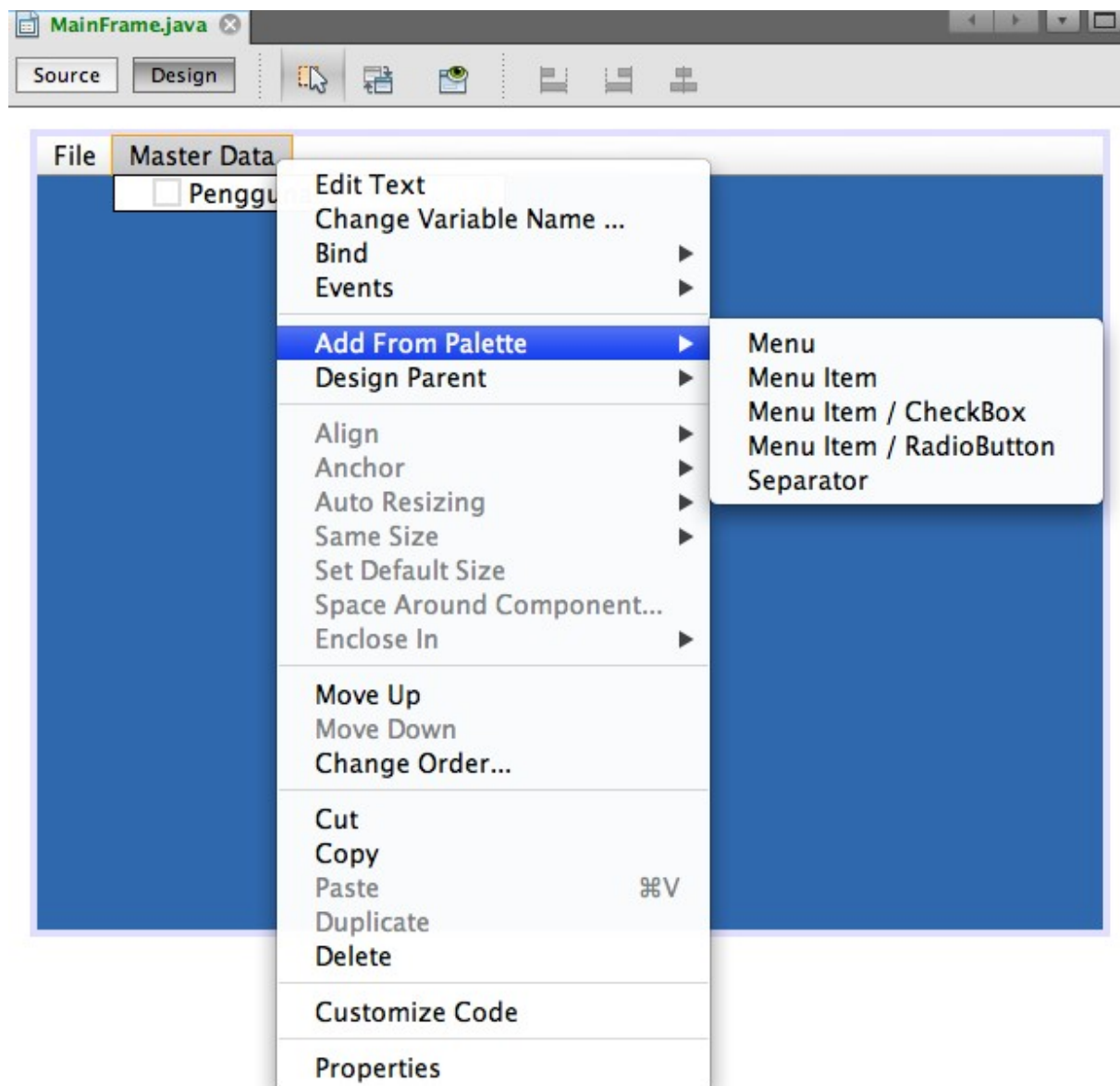
Buat satu per satu komponen untuk menampilkan data person seperti dalam gambar di atas menggunakan komponen-komponen yang bisa didrag-n-drop dari jendela palette

2. Kemudian beri nama untuk satu per satu untuk setiap tombol di atas : btnAdd, btnEdit, btnDelete, btnSave, btnCancel dan btnKeluar
3. Komponen JdateChooser diambil dari library Jcalendar, di bab sebelumnya sudah dijelaskan bagaimana caranya mendownload library ini dan meletakkanya dalam palette. Silahkan merujuk ke Bagian 2 : Netbeans bab Menambahkan Library
4. Untuk mengedit komponen Jtable agar mempunyai 2 buah kolom tanpa baris sama sekali, silahkan pilih Jtable tersebut kemudian klik pada baris Model di jendela Properties, akan muncul jendela untuk mengedit tampilan dari Jtable seperti di bawah ini :



Setelah selesai mempersiapkan screen PersonPanel, langkah berikutnya adalah membuat frame utama yang akan berisi menu dan JdesktopPane yang nantinya akan digunakan sebagai container dari semua JinternalFrame yang ada dalam aplikasi. Ikuti langkah-langkah berikut ini untuk membuat frame utama :

1. Membuat class dengan tipe JFrame, pilih menu File > New untuk membuat file baru dan pilih file dengan tipe Swing GUI Forms > JFrame form
2. Beri nama MainFrame letakkan di package com.googlecode.projecttemplate.pos.ui kemudian tekan tombol Finish.
3. Langkah berikutnya adalah menambahkan MenuBar untuk menampilkan menu-menu aplikasi. Klik item Menu Bar dari Pallet dan drag n drop ke atas MainFrame. Ganti nama variabelnya menjadi mnuBar.
4. Selanjutnya tambahkan Jmenu dengan melakukan klik kanan di Menu pilih item Add Menu. Ganti textnya menjadi "Master Data" dan ganti variabelnya menjadi mnuMasterData
5. Berikutnya adalah menambahkan JMenuItem ke dalam mnuMasterData. Klik kanan di mnuMasterData kemudian pilih Add From Palette > Menu Item. Ganti textnya menjadi "Pengguna" dan ganti nama variabelnya menjadi mnuPengguna.
6. Hasil langkah-langkah di atas bisa dilihat di gambar di bawah ini



- Setelah menu selesai dibuat, langkah terakhir adalah meletakkan JdesktopPane ke dalam MainFrame sebagai tempat dimana nanti JInternalFrame seperti PersonPanel akan ditampilkan. Pilih item JdesktopPane dari Pallete kemudian drag n drop di atas MainFrame beri nama variabelnya desktopPane, sesuaikan ukurannya agar memenuhi seluruh halaman MainFrame.

Setelah PersonPanel dan MainFrame selesai didesign, sekarang waktunya menggabungkan keduanya agar kalau mnuPanel diklik PersonPanel akan ditampilkan dalam desktopPane.

Klik kanan mnuPerson kemudian pilih menu Event > Action > actionPerformed, tampilan Source akan dibuka untuk class MainFrame, silahkan tambahkan kode berikut ini di dalam jendela Source

```
public PersonPanel personPanel;
private void mnuPersonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        if (personPanel == null) {
            personPanel = new PersonPanel();
            desktopPane.add(personPanel);
        } else {
            personPanelToFront();
        }
        personPanel.setVisible(true);
        personPanel.setSelected(true);
        personPanel.setSize(desktopPane.getSize());
    } catch (PropertyVetoException ex) {
        log.error("error ketika menampilkan person panel", ex);
    }
}
```

Perhatikan kode `public PersonPanel personPanel;` adalah variabel dari PersonPanel yang digunakan untuk menampung PersonPanel yang sedang aktif. Setiap kali PersonPanel ditutup dengan menekan tombol X atau tombol Keluar maka variabel ini harus diset menjadi null, hal ini dimaksudkan agar hanya satu jendela PersonPanel saja yang aktif. Kalau pengecekan ini tidak dilakukan maka jendela PersonPanel akan ditampilkan berkali-kali kalau menu mnuPerson ditekan.

Di dalam blok catch terdapat object log yang digunakan untuk menyimpan / menampilkan log aplikasi. Class object log ini berasal dari library Log4j yang bisa didownload dari log4j.apache.org, kemudian tambahkan log4j jar ke dalam library project. Kalau anda masih bingung untuk mensetup log4j, anda bisa menghapus kode log di dalam statement catch di atas dan ganti dengan `ex.printStackTrace()`. Kalau anda berhasil mendownload log4j jar dan menambahkan ke dalam library project, lanjutkan dengan meneklarasikan object log tepat di bawah statement class MainFrame sebagai berikut ini :

```
private static final Logger log = Logger.getLogger(MainFrame.class);
```

Kita ingin agar setiap kali user menjalankan aplikasi, jendela aplikasi ini nantinya akan ditampilkan memenuhi layar, kode berikut ini diletakkan di dalam constructor MainFrame untuk memaksimalkan tampilan MainFrame

```
public MainFrame() {
    initComponents();
    setExtendedState(JFrame.MAXIMIZED_BOTH);
}
```

Setiap kali NetBeans membuat class JFrame baru, maka secara otomatis NetBeans juga membuat method main di dalam class JFrame tersebut. Sampai dengan langkah ini anda bisa mencoba menjalankan class JFrame dengan mengklik kanan MainFrame dan memilih menu Run File. Coba tekan menu Pengguna dan pastikan PersonPanel tampil di dalam DesktopPane.

Membuat class Main dan Inisialisasi Spring Application Context

Sampai di langkah di bab ini kita sudah membuat kode backend untuk melakukan koneksi ke database, membuat Entity, membuat Dao, membuat Service dan mempersiapkan UI aplikasi.

Langkah berikutnya adalah membuat sebuah class yang merupakan class utama yang menggabungkan kode backend dan kode UI.

Buat class baru dan beri nama Main, letakkan di dalam package com.googlecode.projecttemplate.pos . Class Main ini nantinya akan menginisialisasi Spring dan Hibernate, kemudian menyimpan service dari Spring untuk digunakan dalam aplikasi. Setelah inisialisasi Spring selesai, baru kemudian MainFrame akan diinisialisasi dan ditampilkan. Nanti kita juga akan membahas bagaimana caranya membuat splash screen agar proses inisialisasi Spring terlihat oleh user, dan user tidak merasa bingung selama inisialisasi Spring karena tampilan MainFrame belum terlihat.

Berikut ini adalah isi dari class Main :

```
public class Main {
    private static SecurityService securityService;
    private static MainFrame frame;
    public static SecurityService getSecurityService() {
        return securityService;
    }
    public static MainFrame getFrame() {
        return frame;
    }
    public static void main(String[] args) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                ApplicationContext applicationContext =
                    new ClassPathXmlApplicationContext("applicationContext.xml");
                securityService = (SecurityService)
                    applicationContext.getBean("securityService");
                frame = new MainFrame();
                frame.setVisible(true);
            }
        });
    }
}
```

Perhatikan kode di atas, di dalam class ini ada object securityService yang berisi method-method yang digunakan untuk melakukan berbagai operasi database terhadap Entity Person, Menu dan Role. Object ini ditandai dengan static dan dibuatkan method getternya, tujuan penggunaan static ini adalah agar object securityService bisa dengan mudah diakses oleh object lain dari aplikasi POS yang membutuhkan object securityService. Kita nanti akan melihat lebih banyak bagaimana cara menggunakan object securityService ini di dalam PersonPanel.

Kode di atas juga berisi kode EventQueue.invokeLater, method ini digunakan untuk mengeksekusi kode aplikasi di dalam thread baru. Tujuan utamanya adalah agar Thread utama aplikasi yang disebut sebagai Event Dispatcher Thread (EDT) tetap bebas mendengarkan aksi dari user terhadap UI aplikasi. Praktek penggunaan thread baru untuk aplikasi dan membebaskan EDT mendengarkan event dari user sangat penting agar aplikasi bisa berjalan dengan halus dan tidak terasa ada lag antara aksi yang dilakukan user dengan response dari aplikasinya.

Hingga bab ini kita belum membuat interface SecurityService dan class SecurityServiceImp, berikut ini adalah class-class yang diperlukan untuk membuat SecurityService:

Class MenuDao dan RoleDao yang diletakkan dalam package com.googlecode.projecttemplate.pos.dao

```
@Repository
public class MenuDao extends BaseDaoHibernate<Menu>{
}
```

```
@Repository
public class RoleDao extends BaseDaoHibernate<Role>{
```

```
}
```

Karena kita sudah membuat class BaseDaoHibernate maka membuat MenuDao dan RoleDao menjadi sangat ringkas.

Interface SecurityService yang diletakkan dalam package com.googlecode.projecttemplate.pos.service

```
public interface SecurityService {  
  
    public Person save(Person person);  
    public Person delete(Person person);  
    public Person getPerson(Long id);  
    public List<Person> getPersons();  
  
    public Menu save(Menu menu);  
    public Menu delete(Menu menu);  
    public Menu getMenu(Long id);  
    public List<Menu> getMenus();  
  
    public Role save(Role role);  
    public Role delete(Role role);  
    public Role getRole(Integer id);  
    public List<Role> getRoles();  
  
}
```

Dan terakhir class SecurityServiceImpl di dalam package com.googlecode.projecttemplate.pos.service.impl yang mengimplementasikan interface SecurityService

```
@Service("securityService")  
@Transactional(readOnly=true)  
public class SecurityServiceImpl implements SecurityService{  
    @Autowired private PersonDao personDao;  
    @Autowired private MenuDao menuDao;  
    @Autowired private RoleDao roleDao;  
    @Transactional  
    public Person save(Person person) {  
        return personDao.save(person);  
    }  
    @Transactional  
    public Person delete(Person person) {  
        return personDao.delete(person);  
    }  
    public Person getPerson(Long id) {  
        return personDao.getById(id);  
    }  
    public List<Person> getPersons() {  
        return personDao.getAll();  
    }  
    @Transactional  
    public Menu save(Menu menu) {  
        return menuDao.save(menu);  
    }  
    @Transactional  
    public Menu delete(Menu menu) {  
        return menuDao.delete(menu);  
    }  
}
```

```

public Menu getMenu(Integer id) {
    return menuDao.getById(id);
}
public List<Menu> getMenus() {
    return menuDao.getAll();
}
@Transactional
public Role save(Role role) {
    return roleDao.save(role);
}
@Transactional
public Role delete(Role role) {
    return roleDao.delete(role);
}
public Role getRole(Long id) {
    return roleDao.getById(id);
}
public List<Role> getRoles() {
    return roleDao.getAll();
}
}

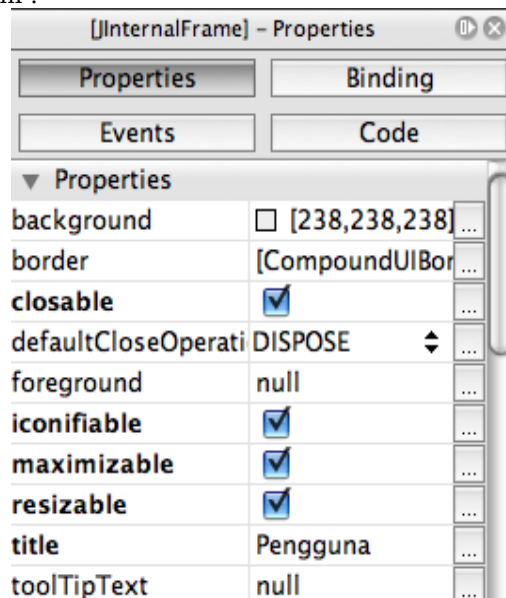
```

Untuk mengetes apakah class Main ini sudah ditulis dengan benar, coba klik kanan class Main kemudian pilih Run File. MainFrame akan ditampilkan sekaligus anda akan melihat adanya log yang mengindikasikan Spring-Hibernate di load dengan benar, pastikan tidak ada pesan error terlihat di jendela Output NetBeans.

Setelah kerangka aplikasi sudah siap, kita kembali lagi ke PersonPanel untuk melengkapi kodenya.

Melengkapi kode PersonPanel

Langkah pertama untuk melengkapi kode PersonPanel adalah mensetting property dari PersonPanel agar JFrame-nya bisa diminimize, dimaximize dan diclose. Lihat tampilan jendela property di bawah ini :



Dibagian sebelumnya ketika membuat MainFrame dijelaskan bahwa ada object personPanel yang harus diset null setiap kali PersonPanel ditutup menggunakan tombol X atau tombol Keluar. Untuk melakukan hal tersebut kita bisa memasang kodenya di event Internal Frame Closed, caranya dengan memilih PersonPanel kemudian klik kanan dan pilih menu Events > Internal Frame >

Internal Frame Closed, di jendela Source yang terbuka silahkan tambahkan kode berikut ini :

```
private void formInternalFrameClosed(javax.swing.event.InternalFrameEvent evt) {  
    Main.getFrame().personPanel = null;  
}
```

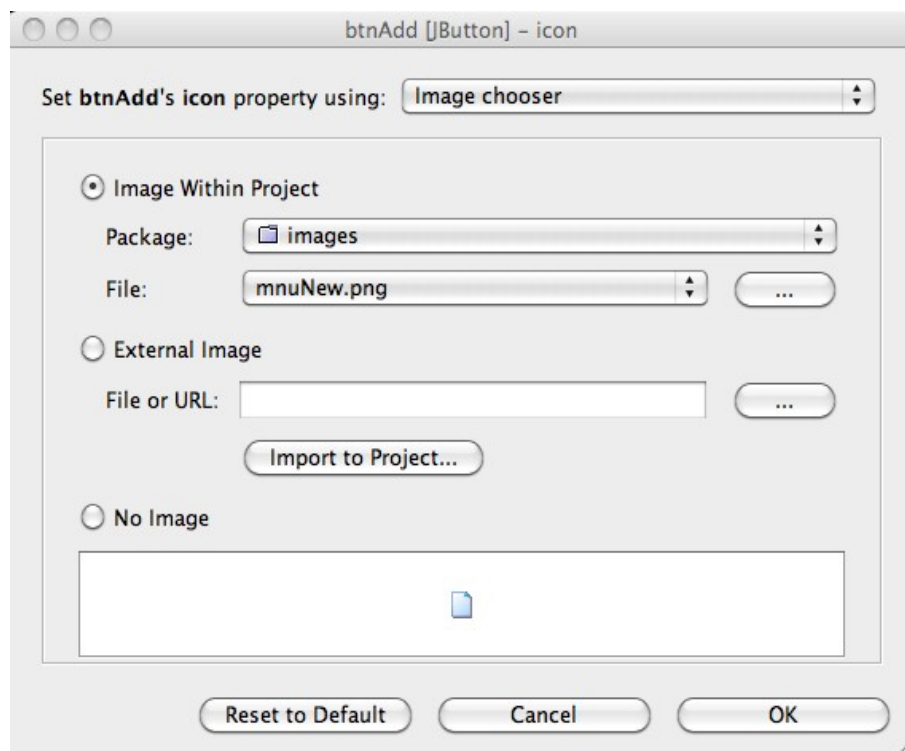
Langkah berikutnya kita akan menambahkan event ke tombol-tombol yang ada di dalam PersonPanel. Sebelum kita mulai, kita bahas satu persatu apa saja yang behaviour setiap tombol ketika user mengklik tombol tersebut, setelah dipahami bagaimana behaviournya baru kita bahas bagaimana cara mengimplementasikan behaviour itu di dalam kode. Pertahankan tombol-tombol di bawah ini :



Tombol-tombol di atas adalah JButton biasa yang ditambahkan icon agar kelihatan fungsionalitasnya. Untuk menambahkan icon ke dalam JButton caranya gampang, pertama download dulu icon-icon di atas dari sini :

<http://code.google.com/p/project-template/source/browse/#svn/trunk/java-desktop-book/code/src/images>

Kemudian buat package dengan nama images dan letakkan semua image yang didownload dari URL di atas ke dalam package tersebut. Setelah itu pilih misalnya tombol tambah kemudian buka jendela properties bagian icon, akan terlihat jendela seperti di bawah ini :



Setelah tampilan tombol-tombol disiapkan, mari kita bahas fungsionalitasnya satu per satu.

Tombol Tambah :

- Aktif ketika PersonPanel baru dibuka
- Aktif setelah tombol Hapus, Simpan dan Batal ditekan
- Non Aktif kalau tombol Tambah atau Edit ditekan

- Ketika ditekan, tombol tambah akan melakukan :
 - Mengaktifkan semua komponen dalam PersonPanel
 - Membersihkan PersonPanel dan mereset variabel dalam PersonPanel
 - Menonaktifkan tombol Edit, Hapus dan Tambah
 - Mengaktifkan tombol Simpan dan Batal

Tombol Edit

- Non Aktif ketika PersonPanel baru dibuka
- Aktif ketika user memilih data Person dari table di sebelah kiri
- Non Aktif kalau tombol Edit atau Batal ditekan
- Ketika ditekan, tombol edit akan melakukan :
 - Mengaktifkan semua komponen dalam PersonPanel
 - Mengaktifkan tombol Simpan dan Batal
 - Menonaktifkan tombol Edit, Hapus, dan Tambah

Tombol Hapus

- Non Aktif ketika PersonPanel baru dibuka
- Aktif ketika user memilih data dari table di sebelah kiri
- Non aktif ketika tombol Hapus atau Batal ditekan
- Ketika ditekan, tombol hapus akan melakukan :
 - Mencoba menghapus object Person dari database
 - Kalau proses penghapusan gagal, misalnya karena data Person digunakan sebagai foreign key di table lain, maka popup ditampilkan untuk memberi keterangan error.
 - Kalau proses penghapusan berhasil, maka lakukan hal-hal berikut ini :
 - Bersihkan nilai-nilai dari komponen di dalam PersonPanel
 - Reset variabel di dalam PersonPanel
 - Refresh nilai dalam tblPerson
 - Aktifkan tombol Tambah
 - Non Aktifkan tombol Edit, Hapus, Simpan dan Batal
 - Non Aktifkan semua komponen dalam PersonPanel

Tombol Simpan

- Non Aktif ketika PersonPanel baru dibuka
- Aktif ketika user menekan tombol Tambah dan Edit
- Non Aktif ketika tombol Simpan atau Batal ditekan
- Ketika ditekan, tombol Simpan akan melakukan :
 - Validasi PersonPanel apakah sudah benar apa belum. Misalnya apakah text field nama sudah diisi atau belum dan seterusnya. Tampilkan popup error kalau proses validasi gagal.
 - Instansiate object baru dari class Person
 - Ambil semua nilai dari komponen-komponen di dalam PersonPanel dan masukkan nilainya ke dalam object Person.
 - Mencoba menginsert data Person ke database kalau user menekan tombol Tambah

sebelumnya.

- Mencoba mengupdate data Person ke database kalau user menekan tombol Edit sebelumnya.
- Penanda untuk menentukan apakah Person akan diinsert atau diupdate ke database adalah nilai id dari Person. Kalau nilai id null maka hibernate lewat method saveOrUpdate class Session akan menginsert ke database, sebaliknya kalau nilai id tidak sama dengan null maka Person akan coba diupdate.
- Kalau proses insert atau update gagal maka tampilkan popup untuk menampilkan error
- Kalau proses insert atau update berhasil maka lakukan hal-hal berikut ini :
 - Bersihkan nilai-nilai dari komponen dalam PersonPanel
 - Non Aktifkan semua komponen dalam PersonPanel
 - Reset variabel di dalam PersonPanel
 - Refresh tblPerson dengan menampilkan data terbaru dari database
 - Aktifkan tombol Tambah
 - Non Aktifkan tombol Edit, Hapus, Simpan dan Batal
 - Non Aktifkan semua komponen dalam PersonPanel

Tombol Batal

- Non Aktif ketika PersonPanel baru saja dibuka
- Aktif ketika user menekan tombol Tambah atau Edit
- Non Aktif ketika user menekan tombol Batal
- Ketika ditekan, tombol Batal akan melakukan :
 - Membersihkan nilai-nilai dari komponen dalam PersonPanel
 - Menonaktifkan semua komponen dalam PersonPanel
 - Aktifkan tombol Tambah
 - Menonaktifkan tombol Edit, Hapus, Simpan dan Batal

Tombol Keluar

- Selalu aktif
- Memanggil method dispose() dari JinternalFrame untuk menutup JinternalFrame dan menandai objectnya agar dibersihkan garbage collector di cycle berikutnya
- Mengeset variabel personPanel di dalam MainFrame sebagai penanda bahwa PersonPanel sudah ditutup. Hal ini dilakukan untuk menghindari PersonPanel dibuka berkali-kali kalau user memilih menu Person Data dari Menu Bar.

User memilih satu baris di dalam tblPerson

- Meload Entity Person dari database, hal ini dilakukan karena data picture ditandai Lazy sehingga perlu diloat secara manual dari database. Data picture ditandai Lazy dimaksudkan untuk menghemat bandwidth ketika semua data Person ditampilkan dalam tblPerson
- Set variabel person di dalam PersonPanel dengan Entity Person yang baru saja diloat
- Masukkan nilai-nilai dari variabel person ke dalam komponen-komponen di dalam PersonPanel
- Aktifkan tombol Edit dan Hapus
- Nonaktifkan komponen-komponen dalam PersonPanel

Kalau kita lihat algoritma di atas terdapat beberapa fungsionalitas yang sama yang digunakan

berulang-ulang disetiap tombol di atas, sehingga kita bisa membuat satu method di dalam PersonPanel untuk setiap fungsionalitas dalam algoritma di atas. Berikut ini daftar method-methodnya :

- private void refreshTable() digunakan untuk merefresh tblPerson agar menampilkan data terbaru dari database
- private void enableForm(boolean status) digunakan untuk mengaktifkan atau menonaktifkan komponen-komponen dalam PersonPanel. Parameter status bernilai boolean untuk menandakan apakah kita ingin mengaktifkan atau menonaktifkan komponen-komponen tersebut
- private void clearForm() digunakan untuk membersihkan nilai-nilai komponen-komponen di dalam PersonPanel sekaligus mereset variabel person menjadi null.
- private boolean validateForm() digunakan untuk memvalidasi komponen-komponen dalam PersonPanel apakah sudah mempunyai nilai yang ditentukan
- private void loadFormToModel() digunakan untuk memasukkan nilai komponen-komponen dalam PersonPanel ke dalam variabel person
- private void loadModelToForm() digunakan untuk memasukkan nilai dalam variabel person ke dalam komponen-komponen PersonPanel

Method-method di atas akan membentuk kerangka dari PersonPanel dan menyediakan fungsionalitas untuk digunakan oleh tombol-tombol.

Class PersonPanel juga memerlukan beberapa variabel untuk menampung data yang diperlukan, seperti image, person dan lain-lainya. Variabel-variabel tersebut diletakkan tepat di bawah deklarasi class PersonPanel dan tidak berada di dalam method apapun.

```
public class PersonPanel extends javax.swing.JInternalFrame {  
  
    private List<Person> persons;  
    private Person person;  
    private JFileChooser chooser;  
    private final int maxPictureSize = 128;  
    private ImageIcon image;  
    private static final Logger log = Logger.getLogger(PersonPanel.class);  
  
    //kode lain di sini  
}
```

Selain method-method dan variabel-variabel di atas, kita memerlukan dua buah class di dalam PersonPanel. Class pertama adalah PersonTableModel yang menextends AbstractTable model, class ini digunakan untuk menampilkan List<Person> ke dalam tblPerson. Class kedua adalah PersonTableSelectionListener yang mengimplementasikan interface ListSelectionListener, class ini digunakan untuk mendengarkan event pemilihan baris di dalam tblPerson. Setiap kali user memilih satu baris dalam tblPerson maka data lengkap Person akan ditampilkan dalam PersonPanel. Agar management kode lebih simpel, kedua class ini adalah inner class yang deklarasinya dilaksanakan di dalam class PersonPanel.

Class PersonTableModel adalah sebagai berikut :

```
private class PersonTableModel extends AbstractTableModel{  
    private List<Person> listPersons;  
    public PersonTableModel(List<Person> listPersons) {  
        this.listPersons = listPersons;  
    }  
    public int getRowCount() {  
        return listPersons.size();  
    }  
    public int getColumnCount() {  
        return 2;  
    }  
}
```



```

    public Object getValueAt(int rowIndex, int columnIndex) {
        Person p = persons.get(rowIndex);
        switch(columnIndex){
            case 0 : return p.getName();
            case 1 : return p.getBirthDate();
            default: return "";
        }
    }
}

```

Class di atas hanya mengimplementasikan 3 buah method abstract dari AbstractTableModel, yaitu `getRowCount` untuk menentukan berapa banyak baris yang akan ditampilkan, `getColumnCount` menentukan berapa kolom yang akan ditampilkan dan `getValueAt` digunakan untuk mendapatkan nilai dari setiap cell dalam `tblPerson`.

Class `PersonTableSelectionListener` adalah sebagai berikut ini :

```

private class PersonTableSelectionListener implements ListSelectionListener{
    public void valueChanged(ListSelectionEvent e) {
        if(tblPerson.getSelectedRow()>=0){
            person = persons.get(tblPerson.getSelectedRow());
            person = Main.getSecurityService().getPerson(person.getId());
            loadModelToForm();
            btnDelete.setEnabled(true);
            btnAdd.setEnabled(false);
            btnCancel.setEnabled(true);
            btnEdit.setEnabled(true);
            btnSave.setEnabled(false);
        }
    }
}

```

Class `PersonTableSelectionListener` hanya mengimplementasikan satu method saja yaitu `valueChanged` dari interface `ListSelectionListener`. Di dalam method tersebut terdapat method `loadModelToForm` yang belum dibuat, di bagian berikutnya kita akan membuat method-method tersebut.

Berikutnya kita akan implementasikan satu persatu method-method di atas beserta kode-kode yang akan dieksekusi ketika setiap tombol di atas diklik oleh user.

Method `refreshTable`

```

private void refreshTable(){
    persons = Main.getSecurityService().getPersons();
    tblPerson.setModel(new PersonTableModel(persons));
}

```

Method `enableForm`

```

private void enableForm(boolean status){
    txtName.setEnabled(status);
    txtPassword.setEnabled(status);
    jdcBirthDate.setEnabled(status);
    btnBrowse.setEnabled(status);
    txtFoto.setEnabled(false);
    txtRemark.setEnabled(status);
    cmbMaritalStatus.setEnabled(status);
}

```

Method `clearForm`

```

private void clearForm(){
    txtId.setText("");
    txtName.setText("");
}

```

```

        txtPassword.setText("");
        txtRemark.setText("");
        txtFoto.setText("");
        tblPerson.getSelectionModel().clearSelection();
        jdcBirthDate.setDate(null);
        person = null;
        lblFoto.setIcon(null);
    }

```

Method validateForm

```

private boolean validateForm(){
    if(txtName.getText().length()>0 &&
        txtPassword.getPassword()!=null &&
        txtPassword.getPassword().length>0){
        return true;
    } else {
        JOptionPane.showMessageDialog(Main.getFrame(),
            "Isi semua field!", "Error", JOptionPane.ERROR_MESSAGE);
        return false;
    }
}

```

Method loadFormToModel

```

private void loadFormToModel(){
    person.setName(txtName.getText());
    if(person.getPassword()==null
        || !person.getPassword().equals(
            new String(txtPassword.getPassword()))){
        //encrypt kata sandi dengan MD5
        String kataSandi =
            new MD5(new String(txtPassword.getPassword())).asHex();
        person.setPassword(kataSandi);
    }
    person.setRemark(txtRemark.getText());
    person.setBirthDate(jdcBirthDate.getDate());
    person.setStatus((MaritalStatus) cmbMaritalStatus.getSelectedItem());
    if(!txtFoto.getText().equals("")){
        ObjectOutputStream dataOutputStream = null;
        try {
            ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
            dataOutputStream = new ObjectOutputStream(outputStream);
            dataOutputStream.writeObject(image);
            dataOutputStream.flush();
            person.setPicture(outputStream.toByteArray());
        } catch (IOException ex) {
            log.error("gagal mengubah Image ke bytearray", ex);
        } finally {
            try {
                dataOutputStream.close();
            } catch (IOException ex) {
                Exceptions.printStackTrace(ex);
            }
        }
    }
}

```

Di dalam method ini terdapat class MD5 yang digunakan untuk membuat hash dari password user untuk disimpan ke database. Hal ini penting untuk menghindari password user disimpan sebagai

palain text di dalam database. Class ini berasal dari project fast-md5 (fast-md5.jar) yang bisa didownload dari twmacinta.com/myjava/fast_md5.php kemudian tambahkan ke dalam library project. Class MD5 ini kompatibel dengan method md5 dari php sehingga tidak ada kekhawatiran table person ini nanti dibaca dari PHP.

Kemudian terdapat kode untuk menyimpan gambar ke dalam database. Setelah berkulat dengan beberapa teknik penyimpanan data image ke database, akhirnya salah satu tekniknya berhasil. Teknik ini tidak langsung membaca image menjadi byte array karena nantinya proses merubah byte array ke image memerlukan informasi format image tersebut, karena setiap format image seperti jpg, gif atau png mempunyai codec yang berbeda. Teknik yang digunakan di atas adalah dengan membaca image dari file sebagai object ImageIO kemudian ImageIO disimpan ke dalam DataObjectStream sebagai serializable object kemudian dibungkus oleh ByteArrayOutputStream sebelum dirumah menjadi byte array. Dengan cara ini tidak diperlukan informasi tentang format image-nya, karena nanti pada waktu membaca dari byte array di database langsung didapatkan object ImageIO yang langsung bisa ditampilkan sebagai icon dari JLabel, sehingga gambarnya tampil di PersonPanel.

Method loadModelToForm

```
private void loadModelToForm(){
    txtId.setText(String.valueOf(person.getId()));
    txtName.setText(person.getName());
    txtPassword.setText(person.getPassword());
    jdcBirthDate.setDate(person.getBirthDate());
    cmbMaritalStatus.setSelectedItem(person.getStatus());
    txtRemark.setText(person.getRemark());
    if(person.getPicture()!=null){
        try {
            ObjectInputStream objectInputStream =
                new ObjectInputStream(
                    new ByteArrayInputStream(person.getPicture()));
            image = (ImageIcon) objectInputStream.readObject();
            lblFoto.setIcon(image);
        } catch (ClassNotFoundException ex) {
            log.error("Image gagal dibaca", ex);
        } catch (IOException ex) {
            log.error("Image gagal dibaca", ex);
        }
    } else {
        lblFoto.setIcon(null);
    }
}
```

Di dalam method di atas terdapat kode untuk merubah byte array dari database menjadi ImageIO. Kebalikan dari method sebelumnya yang merubah file image menjadi byte array.

Sekarang kita lihat kode di konstruktor PersonPanel, di dalam konstruktor-nya tedapat kode-kode untuk meload Person dari database dan diletakkan dalam tblPerson, kemudian kode untuk membatasi panjang txtNama karena panjang kolom di database terbatas, kalau nama-nya terlalu panjang akan mengakibatkan SQL Exception.

```
public PersonPanel() {
    initComponents();

    TextComponentUtils.setMaximumLength(100, txtName);

    tblPerson.setAutoCreateColumnsFromModel(false);
    tblPerson.getSelectionModel().addListSelectionListener(
        new PersonTableSelectionListener());
    refreshTable();

    enableForm(false);
}
```

```

cmbMaritalStatus.setModel(new DefaultComboBoxModel(MaritalStatus.values()));

//pengaturan tombol ketika person panel dibuka
btnDelete.setEnabled(false);
btnAdd.setEnabled(true);
btnCancel.setEnabled(false);
btnEdit.setEnabled(false);
btnSave.setEnabled(false);
}

```

Di dalam kode di atas terdapat class `TextComponentUtils` yang berisi utility untuk membatasi panjang `txtName`. Class ini bisa didownload dari :

<http://code.google.com/p/project-template/source/browse/trunk/java-desktop-book/code/src/com/googlecode/projecttemplate/pos/util/TextComponentUtils.java>

Setelah selesai mempersiapkan fungsi-fungsi dan variabel yang diperlukan, sekarang kita buat event handling untuk tombol-tombol. Untuk menambahkan event ketika tombol diklik, pilih tombolnya kemudian klik kanan dan pilih menu `Event > Action Performed`. Setelah itu jendela `Source` akan dibuka dan sebuah method akan dibuat, di dalam method inilah kode untuk mengimplementasikan algoritma yang kita bahas di awal bagian ini.

Tombol Tambah

```

private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {
    clearForm();
    enableForm(true);
    //pengaturan tombol
    btnDelete.setEnabled(false);
    btnAdd.setEnabled(false);
    btnCancel.setEnabled(true);
    btnEdit.setEnabled(false);
    btnSave.setEnabled(true);
}

```

Tombol Edit

```

private void btnEditActionPerformed(java.awt.event.ActionEvent evt) {
    if(person!=null){
        enableForm(true);
        btnDelete.setEnabled(false);
        btnAdd.setEnabled(false);
        btnCancel.setEnabled(true);
        btnEdit.setEnabled(false);
        btnSave.setEnabled(true);
    }
}

```

Tombol Hapus

```

private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
    if(person!=null){
        try{
            Main.getSecurityService().delete(person);
            clearForm();
            person = null;
            refreshTable();
            enableForm(false);
            //pengaturan tombol
            btnDelete.setEnabled(false);
            btnAdd.setEnabled(true);
            btnCancel.setEnabled(false);
        }
    }
}

```

```

        btnEdit.setEnabled(false);
        btnSave.setEnabled(false);
    } catch (Exception ex) {
        log.error(ex);
        JOptionPane.showMessageDialog(this,
            "Data masih digunakan tidak bisa dihapus!",
            "Error", JOptionPane.ERROR_MESSAGE);
    }
}
}

```

Tombol Simpan

```

private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {
    if(validateForm()){
        if(person == null){
            person = new Person();
        }
        loadFormToModel();
        try{
            Main.getSecurityService().save(person);
            clearForm();
            refreshTable();
            enableForm(false);
            //pengaturan tombol
            btnDelete.setEnabled(false);
            btnAdd.setEnabled(true);
            btnCancel.setEnabled(false);
            btnEdit.setEnabled(false);
            btnSave.setEnabled(false);
        } catch (Exception ex) {
            log.error(ex);
            JOptionPane.showMessageDialog(this, "Data gagal disimpan!"
                , "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}
}

```

Tombol Batal

```

private void btnCancelActionPerformed(java.awt.event.ActionEvent evt) {
    clearForm();
    enableForm(false);
    //pengaturan tombol
    btnDelete.setEnabled(false);
    btnAdd.setEnabled(true);
    btnCancel.setEnabled(false);
    btnEdit.setEnabled(false);
    btnSave.setEnabled(false);
}

```

Tombol Keluar

```

private void btnExitActionPerformed(java.awt.event.ActionEvent evt) {
    Main.getFrame().personPanel = null;
    dispose();
}

```

Selain kelima tombol di atas, ada satu lagi tombol browse yang digunakan untuk memilih file gambar.

Tombol Browse

```

private void btnBrowseActionPerformed(java.awt.event.ActionEvent evt) {
    if(chooser == null){
        chooser = new JFileChooser(System.getProperty("user.home"));
        chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
        chooser.setFileFilter(
            new FileNameExtensionFilter("jpg|png|gif", "jpg","png","gif"));
    }
    int ret = chooser.showOpenDialog(this);
    if(ret == JFileChooser.APPROVE_OPTION){
        File f = chooser.getSelectedFile();
        try {
            Image img = ImageIO.read(new FileInputStream(f));
            //cek ukuran foto, kalau terlalu besar resize ke maxPictureSize
            if(img.getHeight(this)>maxPictureSize || img.getWidth(this)>maxPictureSize){
                if(img.getHeight(this) > img.getWidth(this)){
                    float scale = img.getHeight(this)/maxPictureSize;
                    int widthSize = (int) (img.getWidth(this) / scale);
                    img = img.getScaledInstance(maxPictureSize, widthSize,
                        Image.SCALE_SMOOTH);
                } else {
                    float scale = img.getWidth(this)/maxPictureSize;
                    int heightSize = (int) (img.getHeight(this) / scale);
                    img = img.getScaledInstance(maxPictureSize, heightSize,
                        Image.SCALE_SMOOTH);
                }
            }
            image = new ImageIcon(img);
            lblFoto.setIcon(image);
            txtFoto.setText(f.getAbsolutePath());
        } catch (IOException ex) {
            log.error("error membuka file foto",ex);
        }
    }
}
}

```

Di pojok kiri atas ada satu buah text field yang digunakan sebagai pencarian data Person di tblPerson. Proses pencariannya berdasarkan nilai dari kolom di sebelah kiri, Jtabel mempunyai feature untuk memindah-mindahkan urutan kolom dengan cara mendrag-n-drop kolom, misalnya kita bisa memindahkan kolom Tanggal Lahir ke sebelah kiri atau sebaliknya memindahkan kolom Nama ke sebelah kanan. Pencariannya menggunakan model pencocokan startsWith dengan parameter huruf yang diketik di dalam text field, misalnya diketik huruf "i" maka akan dicari nama yang diawali dengan string "i", proses pencarian ini akan memilih nama dengan awalan "i" yang ditemukan pertama kali dan tblPerson akan discroll ke baris tersebut, jadi hasil pencarian tidak difilter cuma nama dengan awalan "i" saja. Proses pencariannya dilakukan di client, jadi tidak ada database hit selama pencarian.

Klik kanan di txtCari kemudian pilih menu Events > Key > Key Released kemudian tambahkan kode di bawah ini :

```

private void txtSearchKeyReleased(java.awt.event.KeyEvent evt) {
    for(int i=0;i<tblPerson.getRowCount();i++){
        if(tblPerson.getValueAt(i, 0).toString().startsWith(txtSearch.getText())){
            //select baris yang ditemukan
            tblPerson.getSelectionModel().setSelectionInterval(i, i);
            //scroll ke baris tersebut kalau ada di bawah atau bagian atas
            ComponentUtils.scrollToRect(tblPerson, i);
            break;
        }
    }
}
}

```

```
}
```

Kode di atas memerlukan class utility ComponentUtils yang dapat didownload di :

<http://code.google.com/p/project-template/source/browse/trunk/java-desktop-book/code/src/com/googlecode/projecttemplate/pos/util/ComponentUtils.java>

Kode-kode di atas adalah template untuk data master, untuk latihan silahkan buat panel untuk data Role, Menu dan Barang. Di bagian berikutnya kita akan bahas bagaimana membuat data transaksi penjualan.

Data Transaksi

Pencatatan data transaksi di aplikasi POS adalah bagian paling penting dari aplikasi. Dari jendela inilah transaksi penjualan dilaksanakan. Tampilan jendela transaksi penjualan bisa dilihat di bawah ini:

Aplikasi POS Toko Bangkit

File Master Data Transaksi

Penjualan

Tambah Edit Hapus Simpan Batal Cari Keluar

ID Transaksi Tanggal Nov 16, 2010

Kode Barang **Rp. 2,020,950**

Kode Barang	Nama Barang	Harga Satuan	Kuantitas	Sub Total
1	indomie	1050	1	1050
2	kecap abc	4950	1	4950
3	training java	2014950	1	2014950

Bagian atas terdapat tombol-tombol untuk operasi data, ada satu lagi tambahan tombol Cari untuk melakukan pencarian data transaksi sebelumnya. Jika diklik, tombol Cari akan menampilkan jendela pencarian transaksi, kemudian user bisa memilih salah satu baris transaksi dari jendela ini. Setelah dipilih dan ditekan tombol OK, maka transaksi akan ditampilkan di dalam jendela penjualan, kemudian user bisa mengedit data penjualan tersebut.

The screenshot shows a window titled 'Cari' with a search bar. Below the search bar is a table with three columns: 'ID Penjualan', 'Tgl Penjualan', and 'Total Harga'. The table contains three rows of data. The second row is highlighted. At the bottom of the window are two buttons: 'OK' and 'Batal'.

ID Penjualan	Tgl Penjualan	Total Harga
1	2010-11-17 14:3...	13650
2	2010-11-17 14:3...	69300
3	2010-11-17 14:3...	16950

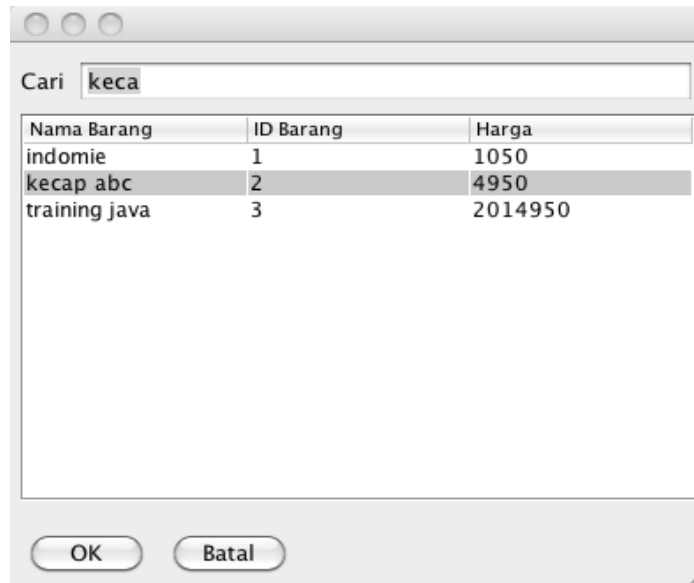
Kalau POS ini digunakan bersama dengan barcode scanner, maka kursor bisa diletakkan dalam kode barang, kemudian gunakan barcode scanner untuk menscan kode barang. Barcode scanner akan menampilkan kode barang plus karakter enter (karakter kode 13) ke dalam text field kode barang, kita akan memasang actionPerformed listener di dalam textfield tersebut agar setiap ada karakter enter dari barcode scanner langsung dilakukan query untuk mencari kode barangnya dan dibuatkan data transaksi penjualannya.

Di sebelah kanan textfield kode barang terdapat tombol lookup untuk melakukan pencarian terhadap barang secara manual, misalnya barang yang dibawa pembeli tidak ada kode barangnya karena lepas atau rusak. Kasir bisa menggunakan fasilitas ini untuk mencari kode barang berdasarkan nama barangnya. Fasilitas pencarian barang ini juga berguna untuk mencari harga barang jika ada pembeli yang ingin mengetahui harga barang tetapi harganya tidak tercantum di dalam rak.

The screenshot shows a window titled 'Cari' with a search bar. Below the search bar is a table with three columns: 'ID Barang', 'Nama Barang', and 'Harga'. The table contains three rows of data. At the bottom of the window are two buttons: 'OK' and 'Batal'.

ID Barang	Nama Barang	Harga
1	indomie	1050
2	kecap abc	4950
3	training java	2014950

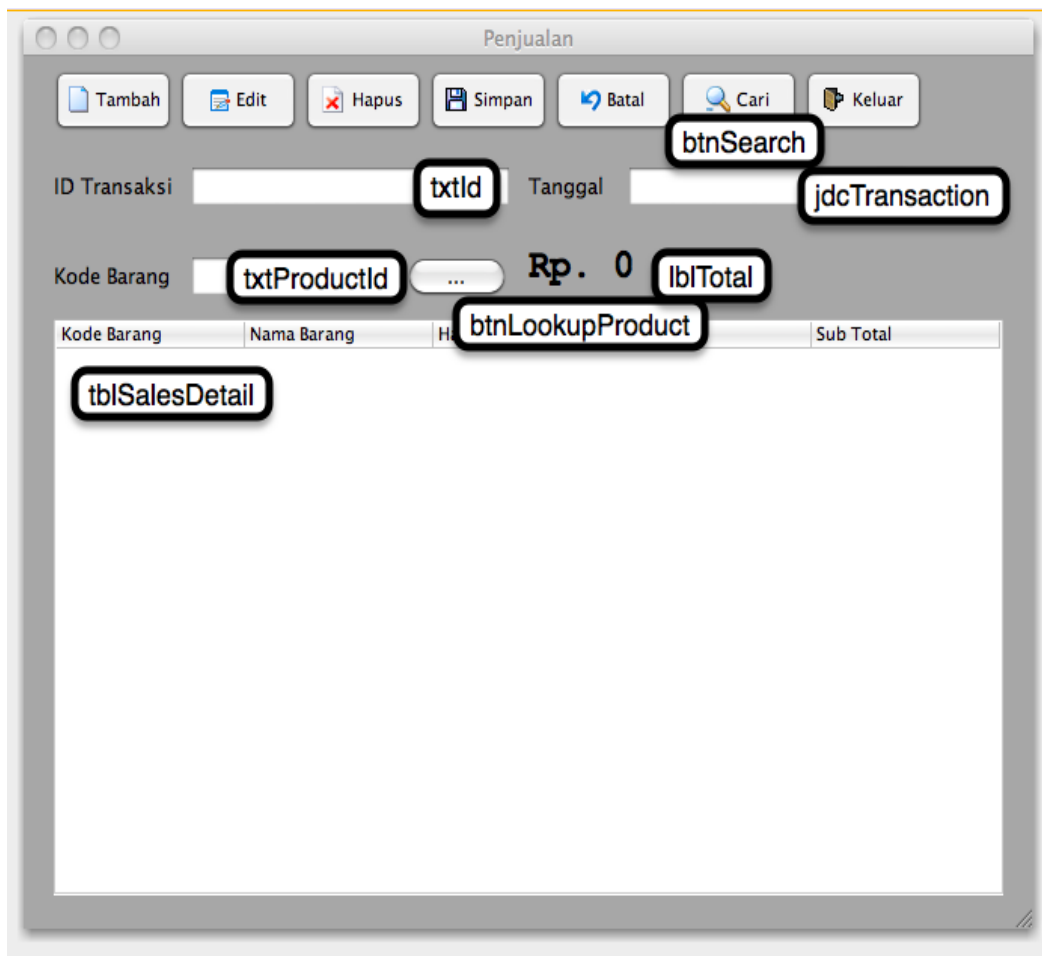
Textfield cari di dalam jendela ini bisa digunakan untuk melakukan pencarian terhadap barang, proses pencarian dilakukan terhadap kolom yang ada di sebelah kiri. Kolom di sebelah kiri secara default akan menampilkan kode barang, kalau misalnya ingin melakukan pencarian berdasarkan nama barang, cukup drag kolom Nama Barang ke sebelah kiri menggantikan kolom ID Barang.



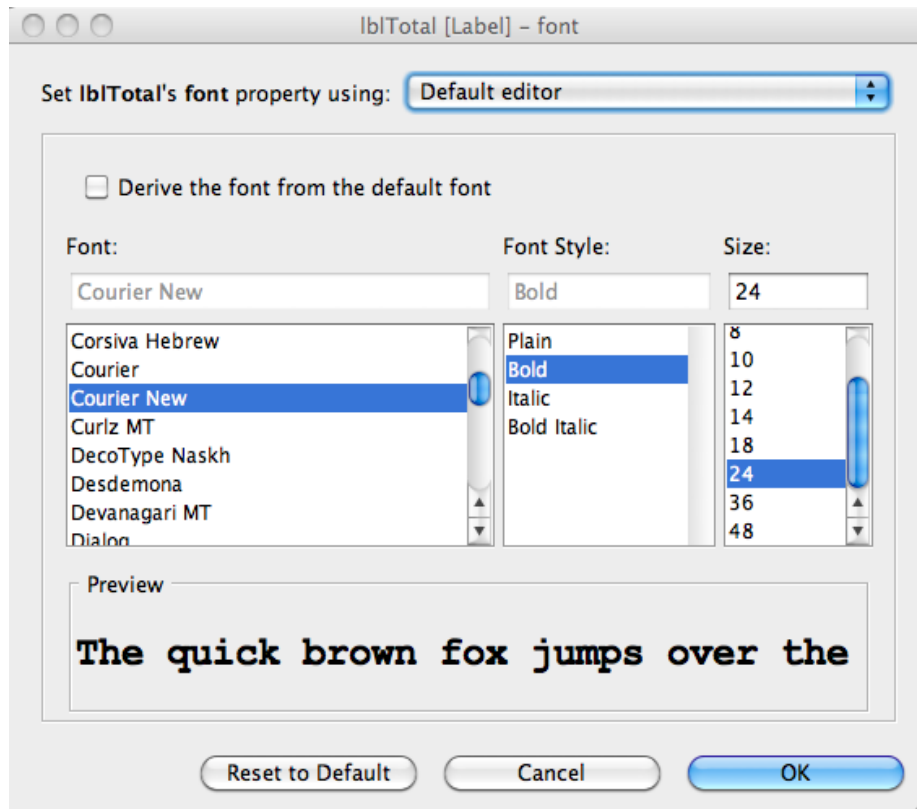
Penggeseran kolom ini bisa dilakukan terhadap semua Jtable, feature ini sudah build in di dalam Jtable, tidak perlu kode tambahan. Setelah mengetahui fungsionalitas jendela transaksi penjualan di atas, di bagian berikutnya kita akan membahas bagaimana menyiapkan ketiga screen di atas kemudian kita bahas satu per satu bagaimana membuat kodenya.

Mempersiapkan Screen Transaksi Penjualan

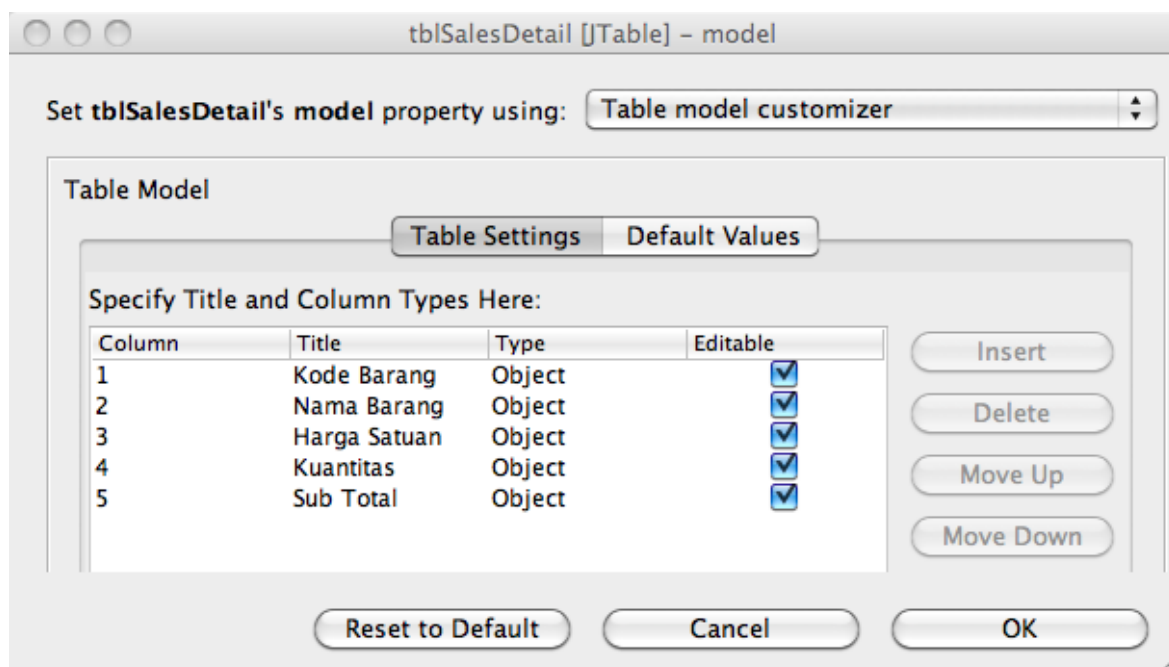
Screen pertama yang kita siapkan adalah JinternalFrame untuk menampilkan halaman penjualan. Screen ini terdiri dari tujuh buah tombol di sisi atas, satu tombol tambahan dibanding screen master adalah tombol Cari. Kemudian ada dua buah text field untuk menampilkan id transaksi dan id barang, satu buah tombol lookup untuk melakukan pencarian barang, sebuah date chooser dari Jcalendar untuk menampilkan tanggal transaksi dan sebuah JLabel untuk menampilkan total transaksi.



Untuk membuat JFrame di atas, pilih menu File > New File kemudian pilih Swing GUI Form > JFrame Form. Beri nama SalesPanel dan letakkan di package com.googlecode.projecttemplate.pos.ui.transaction. Beberapa konfigurasi di properties yang perlu dilakukan adalah mengubah ukuran dan jenis font dari lblTotal. Seperti terlihat dari gambar di bawah ini. Cara menampilkan jendela font tersebut adalah dengan memilih baris Font dari jendela properties lblTotal.



Kemudian tblSalesDetail juga perlu diubah sedikit properties-nya, yang pertama adalah Model. Pilih tblSalesDetail kemudian buka jendela properties dan pilih Model. Jendela editor untuk mengedit TableModel akan terlihat seperti gambar di bawah ini :

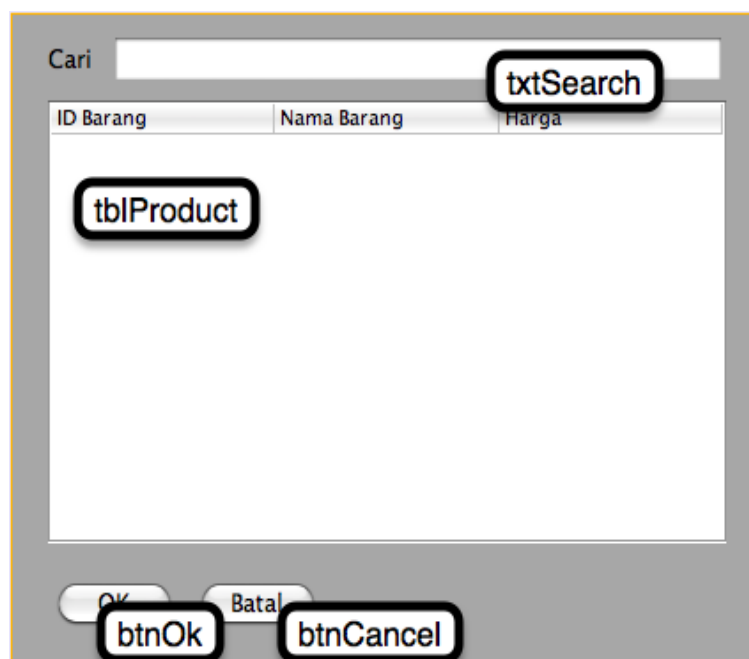


Selain properties model, ada satu lagi properties dari tblSalesDetail yang perlu diubah, yaitu

cellSelectionEnabled. Centang properties cellSelectionEnabled agar user bisa memilih cell dari tblSalesDetail, kalau tidak dicentang maka tblSalesDetail hanya bisa dipilih baris per baris.

Setelah selesai menyiapkan screen transaksi penjualan, sekarang kita siapkan screen untuk menampilkan pencarian barang yang nantinya akan ditampilkan kalau tombol btnLookupProduct ditekan. Berbeda dengan screen transaksi penjualan yang merupakan JInternalFrame, screen pencarian barang adalah turunan dari Jdialog. Jdialog dipilih untuk menampilkan screen pencarian barang karena sifatnya yang "modal" alias berada di atas screen lainnya, dan selama screen pencarian barang ini tampil maka komponen lainnya di bawahnya tidak bisa dipilih. User juga diwajibkan untuk memilih salah satu barang di dalam tblProduct sebelum menekan tombol OK, atau menekan tombol Cancel.

Untuk membuat screen pencarian barang ini, pilih File > New File kemudian pilih Swing GUI Form > Jdialog Form. Beri nama ProductLookupDialog dan letakkan di dalam package yang sama dengan SalesPanel. Setelah selesai dibuat, kemudian tambahkan komponen-komponen ke dalam Jdialog seperti gambar di bawah ini. Jangan lupa untuk mengedit properties model dari tblProduct seperti di atas.



Setelah menyelesaikan screen pencarian barang, screen berikutnya yang akan kita buat adalah screen pencarian transaksi. Screen pencarian transaksi akan ditampilkan kalau user menekan tombol Cari. Tampilan screen pencarian transaksi ini sama persis dengan tampilan screen pencarian barang, hanya datanya saja yang berbeda. Agar pembuatan screen pencarian transaksi lebih cepat, kita bisa lakukan copy class ProductLookupDialog kemudian paste di package yang sama dan ganti nama class-nya menjadi SalesLookupDialog.

Setelah proses copy paste selesai, ubah tampilan dan nama-nama komponen dalam SalesLookupDialog seperti gambar di bawah ini :

The image shows a Java Swing window titled "Cari". At the top, there is a search bar with a label "txtSearch". Below the search bar is a table with three columns: "ID Penjualan", "Tgl Penjualan", and "Total Harga". The table is currently empty. Below the table, there is a label "tblSales". At the bottom of the window, there are two buttons: "btnOk" and "btnCancel". Below these buttons are the labels "OK" and "Batal" respectively.

Setelah ketiga screen di atas selesai dibuat, berikutnya kita akan mulai menulis kode-kode untuk mengimplementasikan fungsionalitas dari ketiga screen di atas. Kita mulai dari kode-kode backend untuk mengakses entity Product dan entity Sales.

Membuat Service dan DAO untuk Sales dan Product

Seperti yang sudah kita bahas sebelumnya, membuat DAO sangat mudah dengan adanya BaseDaoHibernate, cukup buat sebuah class dan extends BaseDaoHibernate maka semua fungsionalitas DAO sudah terpenuhi. Kedua class DAO di ini diletakkan di dalam package yang sama dengan class-class DAO yang sudah dibuat. Class ProductDao adalah sebagai berikut :

```
@Repository
public class ProductDao extends BaseDaoHibernate<Product>{

}
```

Kemudian SalesDao adalah sebagai berikut :

```
@Repository
public class SalesDao extends BaseDaoHibernate<Sales>{

}
```

Bisa dilihat bahwa kode kedua DAO di atas sangat sederhana, kalau tidak ada query yang berbeda dengan BaseDaoHibernate maka class DAO-nya akan kosong seperti di atas.

Berikutnya kita akan membuat ProductService dan ProductServiceImpl. Sebenarnya kita tidak perlu membuat sebuah Service untuk setiap Entity/DAO, satu Service bisa digunakan untuk beberapa Entity/DAO, hanya saja perlu diingat agar class Service jangan sampai terlalu besar ukurannya untuk menghindari kode yang terlalu berantakan. Service dari Entity yang tidak termasuk dalam transaction seperti Product bisa disatukan dengan Entity lain, tapi karena tidak ada lagi Entity lain selain Product yang tidak termasuk dalam transaction atau security, maka dibuatlah PersonService. Kalau misalnya ada Entity lain selain Person yang tidak termasuk kedua kategori Entity di atas, bisa juga dibuat MasterService untuk mengakomodasi semua Entity yang sejenis ini.

Class ProductService adalah sebagai berikut :

```
public interface ProductService {

    Product save(Product product);
    Product delete(Product product);
    Product getProduct(Long id);
    List<Product> getProduct();
    List<Product> getProduct(int start, int num);

}
```

Class ProductServiceImpl :

```
@Service("personService")
@Transactional(readOnly=true)
public class PersonServiceImpl implements PersonService{
    @Autowired private PersonDao personDao;
    @Transactional(readOnly=false)
    public void save(Person person) {
        personDao.save(person);
    }
    @Transactional(readOnly=false)
    public void delete(Person person) {
        personDao.delete(person);
    }
    public Long count() {
        return personDao.count();
    }
    public Person getPerson(Long id) {
        return personDao.getById(id);
    }
    public List<Person> getPersons() {
        return personDao.getAll();
    }
    public List<Person> getPersons(int start, int num) {
        return personDao.getAll(start, num);
    }
}
```

Kemudian kita buat class SalesService :

```
public interface SalesService {

    Sales save(Sales sales);
    Sales delete(Sales sales);
    Sales getSales(Long id);
    List<Sales> getSales();
    List<Sales> getSales(int start, int num);

}
```

Terakhir adalah class SalesServiceImpl:

```
@Service(value="salesService")
@Transactional(readOnly=true)
public class SalesServiceImpl implements SalesService{
    @Autowired private SalesDao salesDao;
    @Transactional
    public Sales save(Sales sales) {
        //agar yang digunakan adalah Date server bukan date client kalau
        //dijalankan dengan arsitektur three tier
    }
}
```

```

        sales.setSalesDate(new Date());
        return salesDao.save(sales);
    }
    @Transactional
    public Sales delete(Sales sales) {
        return salesDao.delete(sales);
    }
    public Sales getSales(Long id) {
        Sales s = salesDao.getById(id);
        Hibernate.initialize(s.getSalesDetails());
        return s;
    }
    public List<Sales> getSales() {
        return salesDao.getAll();
    }
    public List<Sales> getSales(int start, int num) {
        return salesDao.getAll(start, num);
    }
}

```

Setelah selesai membuat DAO dan Service, sekarang kita edit class Main dan meletakkan instance dari PersonService dan SalesService agar bisa diakses dari class lain yang memerlukan. Proses penambahan instance Service ke dalam class Main dilakukan setiap kali ada Service baru yang dibuat. Berikut ini class Main yang sudah diedit :

```

public class Main {
    private static SecurityService securityService;
    private static SalesService salesService;
    private static ProductService productService;
    private static MainFrame frame;
    public static SecurityService getSecurityService() {
        return securityService;
    }
    public static SalesService getSalesService() {
        return salesService;
    }
    public static ProductService getProductService() {
        return productService;
    }
    public static MainFrame getFrame() {
        return frame;
    }
    public static void main(String[] args) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                ApplicationContext applicationContext =
                    new ClassPathXmlApplicationContext("applicationContext.xml");
                securityService =
                    (SecurityService) applicationContext.getBean("securityService");
                salesService =
                    (SalesService) applicationContext.getBean("salesService");
                productService =
                    (ProductService) applicationContext.getBean("productService");
                frame = new MainFrame();
                frame.setVisible(true);
            }
        });
    }
}

```

```
}
```

Selesai sudah kode untuk mengakses data Product dan Sales. Langkah berikutnya adalah membuat kode untuk UI-nya. Pertama kita implementasikan dahulu kode untuk screen ProductLookupDialog.

Melengkapi Kode di Class ProductLookupDialog

Class ProductLookupDialog adalah class turunan dari JDialog, class ini termasuk kategori “lookup” untuk melakukan lookup (pencarian) data lain. Screen jenis ini hanya digunakan untuk menampilkan data dan pencarian, tidak ada proses perubahan data. Kode untuk membuat screen lookup juga kurang lebih sama, perbedaannya hanya pada Entity apa yang akan dicari. Misalnya antara Sales dan Product, perbedaan screen lookup dari kedua entity ini hanya pada Entity saja. Dengan melakukan copy paste dari ProductLookupDialog menjadi SalesLookupDialog, mereplace Product menjadi Sales dan mengganti kode untuk melakukan query ke database.

Berikut ini kita bahas satu per satu bagian dari class ProductLookupDialog. Setelah class-nya dibuat, hapus method public void main yang digenerate dari netbeans, kemudian ubah deklarasi class, property dan constructor

```
public class ProductLookupDialog extends javax.swing.JDialog {
    private Product product;
    private List<Product> products;
    public ProductLookupDialog() {
        super(Main.getFrame(), true);
        initComponents();
        setLocationRelativeTo(null);
        tblProduct.setAutoCreateColumnsFromModel(false);
        tblProduct.getSelectionModel().addListSelectionListener(
            new ProductSelectionListener());
        products = Main.getProductService().getProduct();
        tblProduct.setModel(new ProductTableModel(products));
    }
    //kode lain di sini
}
```

Dalam constructor class ProductLookupDialog terdapat kode-kode untuk menginisialisasi Jdialog ini agar bersifat “modal”, kode untuk melakukan setting modal ada di baris pertama, dengan memanggil constructor dari Jdialog dan mengeset parameter kedua dengan nilai true. Di baris berikutnya ada method initComponents yang digunakan oleh NetBeans mengenerate kode-kode layout. Baris berikutnya ada method setLocationRelativeTo dengan parameter null, method ini dipanggil agar Jdialog diletakkan tepat di tengah-tengah dari layar. Apapun ukuran layarnya setLocationRelativeTo(null) pasti bisa menletakkan Jdialog tepat di tengah tengah layar, jadi tidak perlu ada kalkulasi ukuran layar user untuk meletakkan Jdialog di tengah-tengah.

Setelah itu ada kode-kode tblProduct.setAutoCreateColumnsFromModel(false); yang digunakan untuk mencegah Jtable membuat ulang kolomnya kalau model-nya diset. Kode ini wajib ditulis agar kolom dari Jtable yang sudah diubah secara visual di langkah sebelumnya tidak dioverride ketika model dari Jtable diset. Tiga baris berikutnya digunakan untuk mendaftarkan ProductSelectionListener, mengambil data product dari database dan mengeset table model ke dalam tblProduct.

Seperti halnya screen data master yang sebelumnya sudah kita buat, screen ini mempunyai sebuah table, oleh karena itu perlu dibuat class yang mengimplementasikan TableModel dan ListSelectionListener. Kode di atas terdapat class ProductTableModel dan class ProductSelectionListener, kedua class tersebut adalah inner class dari ProductLookupDialog, jadi letakkan kodenya di dalam class ProductLookupDialog. Berikut ini kode untuk kedua class tersebut.

```
private class ProductTableModel extends AbstractTableModel{
    private List<Product> products;
    public ProductTableModel(List<Product> products) {
        this.products = products;
    }
}
```



```

    }
    public int getRowCount() {
        return products.size();
    }
    public int getColumnCount() {
        return 3;
    }
    public Object getValueAt(int rowIndex, int columnIndex) {
        Product p = products.get(rowIndex);
        switch(columnIndex){
            case 0: return p.getId();
            case 1: return p.getName();
            case 2: return p.getPrice();
            default: return "";
        }
    }
}

```

Class ProductSelectionListener :

```

private class ProductSelectionListener implements ListSelectionListener{

    public void valueChanged(ListSelectionEvent e) {
        if(tblProduct.getSelectedRow()>=0){
            product = products.get(tblProduct.getSelectedRow());
        }
    }
}

```

Kemudian kita akan membuat kode untuk action listener dari setiap komponen dalam ProductLookupDialog, yang pertama ada lah ketika user mengetikkan huruf di dalam txtSearch. Event yang digunakan adalah keyReleased, cara membuat event ini adalah dengan mengklik kanan txtSearch kemudian pilih Events > Key > key Released, di jendela Code yang terbuka tambahkan kode berikut ini :

```

private void txtSearchKeyReleased(java.awt.event.KeyEvent evt) {
    for(int i =0;i<tblProduct.getRowCount();i++){
        String val = tblProduct.getValueAt(i, 0).toString();
        if(val!=null && val.startsWith(txtSearch.getText())){
            tblProduct.getSelectionModel().setSelectionInterval(i, i);
            ComponentUtils.scrollToRect(tblProduct, i);
            break;
        }
    }
}

```

Kode di atas mengandung class ComponentUtils yang digunakan untuk menscroll Jtable kalau hasil pencariannya berada di bagian bawah atau bagian atas. Di bab sebelumnya terdapat link untuk mendownload class ComponentUtils ini.

Berikutnya adalah kode ketika tombol OK ditekan, untuk menangkap event penekanan tombol OK digunakan listener untuk mendengarkan event actionPerformed terhadap tombol OK. Klik kanan btnOk, kemudian pilih menu Event > actionPerformed, dan tambahkan kode berikut ini :

```

private void btnOkActionPerformed(java.awt.event.ActionEvent evt) {
    if(product!=null){
        dispose();
    } else {
        JOptionPane.showMessageDialog(this,"Pilih satu barang terlebih dahulu!",
            "error",JOptionPane.ERROR_MESSAGE);
    }
}

```

```
}
}
```

Listerner yang terakhir perlu diimplementasikan adalah pada waktu tombol Cancel ditekan, tambahkan listener untuk event actionPerformed di tombol Cancel kemudian ubah kodenya menjadi seperti di bawah ini :

```
private void btnCancelActionPerformed(java.awt.event.ActionEvent evt) {
    product = null;
    dispose();
}
```

Tombol Cancel hanya akan mengeset variabel product menjadi null dan memanggil method dispose dari Jdialog agar menutup dirinya.

Kode terakhir adalah method yang digunakan untuk memanggil ProductLookupDialog dari class lain, kode ini sangat penting dan sedikit berbeda perilakunya dari kode-kode lainnya. Mari kita lihat kodenya :

```
public Product getProduct(){
    setVisible(true);
    return product;
}
```

Baris pertama dari method di atas memanggil method setVisible dari Jdialog, method ini bersifat blocking, alias cursor eksekusi kode akan berhenti di baris tersebut hingga ada kode lain yang memanggil method dispose() dari Jdialog. Kalau kita lihat-lihat lagi ke atas, method dispose() akan dipanggil ketika user menekan tombol OK atau tombol Cancel. Nah setelah method dispose() dipanggil barulah method getProduct ini akan melanjutkan eksekusinya ke baris berikutnya yaitu mereturn nilai dari property product. Kalau tombol OK yang ditekan property product harus berisi product yang sekarang sedang dipilih di tblProduct, kalau tombol Cancel yang ditekan maka property product akan diset menjadi null baru method dispose() dipanggil. Bagaimana cara menggunakan method ini? Kita akan bahas di bagian berikutnya sekaligus menerangkan kode-kode yang ada dalam class SalesPanel.

Untuk bahan latihan, silahkan buat SalesLookupDialog dengan menggunakan kode dari ProductLookupDialog di atas, ganti class Product menjadi Sales, kemudian ubah class yang mengimplementasi TableModel-nya dan terakhir ubah cara mengambil data dari database. Kalau ada yang bingung silahkan lihat kode SalesLookupDialog dari URL ini :

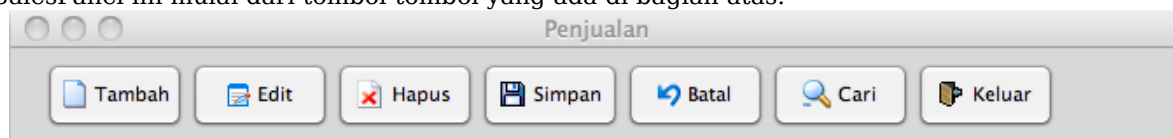
<http://code.google.com/p/project-template/source/browse/trunk/java-desktop-book/code/src/com/googlecode/projecttemplate/pos/ui/transaction/SalesLookupDialog.java>

Setelah selesai membuat dua buah screen lookup, sekarang kita bahas bagian utama buku ini: SalesPanel.

Melengkapi Kode Class SalesPanel

SalesPanel adalah bagian utama POS yang akan digunakan oleh kasir melakukan input data penjualan di toko. Sebagian besar kode dalam SalesPanel sama dengan kode PersonPanel, strukturnya pun nyaris sama, perbedaannya ada lah SalesPanel digunakan untuk memange data Sales-SalesDetail yang mempunyai patter Master-Detail, sehingga perlu sedikit penyesuaian untuk mengakomodasi SalesDetail.

Sebelum kita mulai listing kode-nya, kita bahas dulu fungsionalitas dari setiap komponen dalam SalesPanel ini mulai dari tombol-tombol yang ada di bagian atas.



Tombol Tambah :

- Aktif ketika SalesPanel baru dibuka

- Aktif setelah tombol Hapus, Simpan dan Batal ditekan
- Non Aktif kalau tombol Tambah atau Edit ditekan
- Ketika ditekan, tombol tambah akan melakukan :
 - Mengkatifkan semua komponen dalam SalesPanel
 - Membersihkan SalesPanel dan mereset variabel dalam SalesPanel
 - Menonaktifkan tombol Edit, Hapus dan Tambah
 - Mengaktifkan tombol Simpan dan Batal

Tombol Edit

- Non Aktif ketika SalesPanel baru dibuka
- Aktif ketika user memilih data Sales dengan memilih satu baris dalam jendela pencarian data transaksi ketika user menekan tombol Cari.
- Non Aktif kalau tombol Edit atau Batal ditekan
- Ketika ditekan, tombol edit akan melakukan :
 - Mengaktifkan semua komponen dalam SalesPanel
 - Mengaktifkan tombol Simpan dan Batal
 - Menonaktifkan tombol Edit, Hapus, dan Tambah

Tombol Hapus

- Non Aktif ketika SalesPanel baru dibuka
- Aktif ketika user memilih data Sales dengan memilih satu baris dalam jendela pencarian data transaksi ketika user menekan tombol Cari.
- Non aktif ketika tombol Hapus atau Batal ditekan
- Ketika ditekan, tombol hapus akan melakukan :
 - Mencoba menghapus object Sales dari database
 - Kalau proses penghapusan gagal, maka popup ditampilkan untuk memberi keterangan error.
 - Kalau proses penghapusan berhasil, maka lakukan hal-hal berikut ini :
 - Bersihkan nilai-nilai dari komponen di dalam SalesPanel
 - Reset variabel di dalam SalesPanel
 - Refresh nilai dalam tblSalesDetail
 - Aktifkan tombol Tambah
 - Non Aktifkan tombol Edit, Hapus, Simpan dan Batal
 - Non Aktifkan semua komponen dalam SalesPanel

Tombol Simpan

- Non Aktif ketika SalesPanel baru dibuka
- Aktif ketika user menekan tombol Tambah dan Edit
- Non Aktif ketika tombol Simpan atau Batal ditekan
- Ketika ditekan, tombol Simpan akan melakukan :
 - Validasi SalesPanel apakah sudah benar apa belum. Misalnya apakah sudah ada SalesDetail yang diinput. Tampilkan popup error kalau proses validasi gagal.
 - Instansiasi object baru dari class Person

- Ambil semua nilai dari komponen-komponen di dalam SalesPanel dan masukkan nilainya ke dalam object Sales.
- Mencoba menginsert data Sales dan SalesDetail ke database kalau user menekan tombol Tambah sebelumnya.
- Mencoba mengupdate data Sales dan SalesDetail ke database kalau user menekan tombol Edit sebelumnya.
- Penanda untuk menentukan apakah Sales dan SalesDetail akan diinsert atau diupdate ke database adalah nilai id dari Sales. Kalau nilai id null maka hibernate lewat method saveOrUpdate dari class Session akan menginsert ke database, sebaliknya kalau nilai id tidak sama dengan null maka Sales dan SalesDetail akan coba diupdate. Cascade All dan Cascade Delete Orphan di dalam mapping class Sales membantu proses update kalau kasir menambah, mengedit jumlah transaksi atau menghapus salah satu data SalesDetail. Tidak perlu kode tambahan untuk melakukan pengecekan mana SalesDetail yang ditambah, diedit atau dihapus, sangat praktis.
- Kalau proses insert atau update gagal maka tampilkan popup untuk menampilkan error
- Kalau proses insert atau update berhasil maka lakukan hal-hal berikut ini :
 - Bersihkan nilai-nilai dari komponen dalam SalesPanel
 - Non Aktifkan semua komponen dalam SalesPanel
 - Reset variabel di dalam SalesPanel
 - Bersihkan isi tblSalesDetail dengan memberikan List kosong
 - Aktifkan tombol Tambah
 - Non Aktifkan tombol Edit, Hapus, Simpan dan Batal
 - Non Aktifkan semua komponen dalam SalesPanel

Tombol Batal

- Non Aktif ketika SalesPanel baru saja dibuka
- Aktif ketika user menekan tombol Tambah atau Edit
- Non Aktif ketika user menekan tombol Batal
- Ketika ditekan, tombol Batal akan melakukan :
 - Membersihkan nilai-nilai dari komponen dalam SalesPanel
 - Menonaktifkan semua komponen dalam SalesPanel
 - Aktifkan tombol Tambah
 - Menonaktifkan tombol Edit, Hapus, Simpan dan Batal

Tombol Cari

- Selalu Aktif
- Ketika ditekan, tombol Cari akan melakukan :
 - Memanggil SalesLookupDialog dan menunggu user menekan tombol OK atau Cancel
 - Kalau object Sales yang dikembalikan oleh SalesLookupDialog maka tampilkan data Sales tersebut.
 - Query ulang data Sales dari database dengan mem-fetch SalesDetail, hal ini harus dilakukan karena pada waktu SalesLookupDialog mengambil data Sales, data SalesDetailnya tidak ikut diquery alias lazy fetch. Kalau tidak dilakukan query ulang ke database maka LazyInitializationException akan terjadi.
 - Set nilai object sales dan salesDetail yang baru saja diambil dari database.
 - Aktifkan tombol Hapus dan Edit. Non Aktifkan tombol Tambah dan Simpan.

- Kalau object Sales yang dikembalikan SalesLookupDialog bernilai null, artinya user menekan tombol Cancel, tidak perlu melakukan apa-apa kalau kondisi ini terjadi.

Tombol Keluar

- Selalu aktif
- Memanggil method dispose() dari JInternalFrame untuk menutup JInternalFrame dan menandai objectnya agar dibersihkan garbage collector di cycle berikutnya
- Mengeset variabel salesPanel di dalam MainFrame sebagai penanda bahwa SalesPanel sudah ditutup. Hal ini dilakukan untuk menghindari PersonPanel dibuka berkali-kali kalau user memilih menu Transaksi > Penjualan dari Menu Bar.

User memilih satu baris di dalam tblSalesDetail

- tblSalesDetail bisa dipilih cell-per-cell oleh user, hanya kolom Kuantitas saja yang bisa diedit. Jika user mengganti nilai kuantitas, maka sub total dari SalesDetail yang sedang dipilih berubah dan update lblTotal agar menampilkan total dari penjualan

User mengetikkan id barang kemudian menekan enter di txtProductId

- Query ke database untuk mendapatkan Product berdasarkan id yang diinput user
- Cek ke dalam daftar SalesDetail dalam tblSalesDetail apakah barang tersebut sudah ada, kalau sudah ada maka tambah kuantitasnya dengan satu, update property subTotal dari SalesDetail dan update lblTotal untuk menampilkan total penjualan
- Kalau Product tersebut belum ada dalam tblSalesDetail, maka buat object SalesDetail, set property price dari SalesDetail dari price yang ada di Product. Set property quantity menjadi satu, hitung total penjualan dan update lblTotal untuk menampilkan total penjualan.

User menekan tombol btnLookupProduct

- Tampilkan instansiasi ProductLookupDialog dan tampilkan ke user.
- Kalau user memilih salah satu product dan menekan tombol OK yang ditandai return dari ProductLookupDialog tidak bernilai null, maka:
 - Cek ke dalam daftar SalesDetail dalam tblSalesDetail apakah barang tersebut sudah ada, kalau sudah ada maka tambah kuantitasnya dengan satu, update property subTotal dari SalesDetail dan update lblTotal untuk menampilkan total penjualan
 - Kalau Product tersebut belum ada dalam tblSalesDetail, maka buat object SalesDetail, set property price dari SalesDetail dari price yang ada di Product. Set property quantity menjadi satu, hitung total penjualan dan update lblTotal untuk menampilkan total penjualan.
- Kalau user menekan tombol Cancel di ProductLookupDialog maka nilai kembalinya akan null dan tidak perlu dilakukan apa-apa

Kalau kita lihat algoritma di atas terdapat beberapa fungsionalitas yang sama yang digunakan berulang-ulang disetiap tombol di atas, sehingga kita bisa membuat satu method di dalam PersonPanel untuk setiap fungsionalitas dalam algoritma di atas. Berikut ini daftar method-methodnya :

- private void refreshTable() digunakan untuk merefresh tblSalesDetail agar menampilkan data SalesDetail yang diinput user
- private void enableForm(boolean status) digunakan untuk mengaktifkan atau menonaktifkan komponen-komponen dalam SalesPanel. Parameter status bernilai boolean untuk menandakan apakah kita ingin mengaktifkan atau menonaktifkan komponen-komponen tersebut
- private void clearForm() digunakan untuk membersihkan nilai-nilai komponen-komponen di dalam SalesPanel sekaligus mereset variabel sales menjadi null, dan salesDetails menjadi List kosong.
- private boolean validateForm() digunakan untuk memvalidasi komponen-komponen dalam

SalesPanel apakah sudah ada transaksi SalesDetail yang diinput oleh user.

- private void loadFormToModel() digunakan untuk memasukkan nilai komponen-komponen dalam SalesPanel ke dalam variabel sales
- private void loadModelToForm() digunakan untuk memasukkan nilai dalam variabel sales ke dalam komponen-komponen SalesPanel
- private void refreshTotalLabel() digunakan untuk menghitung total transaksi dan mengeset nilai total transaksi-nya ke lblTotal.
- private void addSalesDetail(Product p) digunakan untuk membuat object SalesDetail dari Product yang diset dalam parameter method tersebut. Kemudian object SalesDetail ditambahkan ke List SalesDetail.

Setelah kita bahas algoritma dari setiap komponen dalam SalesPanel, sekarang kita listing satu per satu kodenya.

Seperti biasa, kita listing dulu deklarasi class SalesPanel dan property/variabel yang dimiliki oleh class SalesPanel.

```
public class SalesPanel extends javax.swing.JInternalFrame {  
    private Sales sales;  
    private List<SalesDetail> salesDetails = new ArrayList<SalesDetail>();  
    private static Logger log = Logger.getLogger(SalesPanel.class);
```

Berikutnya kita listing kode untuk method-method yang diterangkan di atas.

Method clearForm :

```
private void clearForm(){  
    txtProductId.setText("");  
    lblTotal.setText("Rp. 0");  
    jdcTransaction.setDate(new Date());  
    salesDetails = new ArrayList<SalesDetail>();  
    sales = null;  
    tblSalesDetail.setModel(new SalesDetailTableModel(salesDetails));  
}
```

Method enableForm :

```
private void enableForm(boolean status){  
    txtProductId.setEnabled(status);  
    btnLookupProduct.setEnabled(status);  
    tblSalesDetail.setEnabled(status);  
}
```

Method validateForm :

```
private boolean validateForm(){  
    if(salesDetails==null || salesDetails.isEmpty()){  
        JOptionPane.showMessageDialog(this, "Transaksi tidak boleh kosong!"  
            , "Error", JOptionPane.ERROR_MESSAGE);  
        return false;  
    }  
    return true;  
}
```

Method loadFormToModel:

```
private void loadFormToModel(){  
    sales.setSalesDetails(salesDetails);  
    sales.setSalesDate(new Date());  
    BigDecimal total = BigDecimal.ZERO;  
    for (SalesDetail salesDetail : salesDetails) {  
        total = total.add(salesDetail.getSubtotal());  
        salesDetail.setSales(sales);  
    }
```

```

    }
    sales.setTotalSales(total);
}

```

Method loadModelToForm :

```

private void loadModelToForm(){
    salesDetails = sales.getSalesDetails();
    tblSalesDetail.setModel(new SalesDetailTableModel(salesDetails));
    lblTotal.setText(TextComponentUtils.formatNumber(sales.getTotalSales()));
}

```

Method refreshTable :

```

private void refreshTable(){
    tblSalesDetail.setModel(new SalesDetailTableModel(salesDetails));
}

```

Method refreshTotalLabel :

```

private void refreshTotalLabel(){
    if(salesDetails!=null && !salesDetails.isEmpty()){
        BigDecimal total = BigDecimal.ZERO;
        for (SalesDetail salesDetail : salesDetails) {
            total = total.add(salesDetail.getSubtotal());
        }
        lblTotal.setText("Rp. " + TextComponentUtils.formatNumber(total));
    }
}

```

Method addSalesDetail :

```

private void addSalesDetail(Product p){
    if(p!=null){
        SalesDetail salesDetail = new SalesDetail();
        salesDetail.setProduct(p);
        salesDetail.setPrice(p.getPrice());
        salesDetail.setQuantity(1);
        if(salesDetail.getSubtotal() != null){
            salesDetail.setSubtotal(salesDetail.getSubtotal().add(p.getPrice()));
        } else {
            salesDetail.setSubtotal(p.getPrice());
        }
        salesDetails.add(salesDetail);
        refreshTable();
        refreshTotalLabel();
    } else {
        JOptionPane.showMessageDialog(this, "Barang tidak ada!"
            , "Error", JOptionPane.ERROR_MESSAGE);
    }
}

```

Dalam SalesPanel ini, yang ditampilkan dalam Jtable adalah SalesDetail, sehingga class yang akan dibuat adalah SalesDetailTableModel. Class ini adalah inner class dari SalesPanel, jadi tambahkan di dalam class SalesPanel, jangan buat file .java lagi.

```

private class SalesDetailTableModel extends AbstractTableModel{
    private List<SalesDetail> salesDetails;
    SalesDetailTableModel(List<SalesDetail> salesDetails) {
        this.salesDetails = salesDetails;
    }
    public int getRowCount() {
        return salesDetails.size();
    }
}

```

```

    }
    public int getColumnCount() {
        return 5;
    }
    public Object getValueAt(int rowIndex, int columnIndex) {
        SalesDetail s = salesDetails.get(rowIndex);
        switch(columnIndex){
            case 0: return s.getProduct().getId();
            case 1: return s.getProduct().getName();
            case 2: return s.getPrice();
            case 3: return s.getQuantity();
            case 4: return s.getSubtotal();
            default: return "";
        }
    }
    @Override
    public Class<?> getColumnClass(int columnIndex) {
        if(columnIndex == 2 || columnIndex == 4){
            return BigDecimal.class;
        } else if(columnIndex == 3){
            return Integer.class;
        }
        return String.class;
    }
    @Override
    public boolean isCellEditable(int rowIndex, int columnIndex) {
        if(columnIndex == 3) {
            return true;
        }
        return false;
    }
    @Override
    public void setValueAt(Object aValue, int rowIndex, int columnIndex) {
        SalesDetail s = salesDetails.get(rowIndex);
        if(columnIndex == 3){
            s.setQuantity((Integer) aValue);
            s.setSubtotal(s.getPrice().multiply(
                new BigDecimal(s.getQuantity())));
            refreshTotalLabel();
        }
    }
}

```

Berbeda dari TableModel yang sudah-sudah, class SalesDetailTableModel mengoverride beberapa method lain dari AbstractTableModel. Hal ini dikarenakan ada satu kolom yaitu Kuantitas yang sifatnya editable, user bisa mengganti nilai dari kolom Kuantitas ini langsung di dalam tablenya, sehingga method getColumnClass, isCellEditable dan setValueAt harus dioverride.

Method getColumnClass digunakan untuk mendefinisikan tipe data dari setiap kolom. Hal ini penting terutama agar kolom Kuantitas hanya bisa diinput angka saja, kalau user menginput String maka Jtable akan mengubah warna huruf dalam cell tersebut menjadi merah, perubahan warna ini mengindikasikan ke user bahwa inputan yang dimasukkan salah.

Method isCellEditable digunakan untuk mengetahui cell apa saja yang bisa diedit oleh user, dalam hal ini semua cell di kolom dengan index = 3 (Kuantitas) bisa diedit.

Method setValueAt akan dipanggil oleh Jtable kalau user mengedit cell dalam Jtable. Karena hanya cell-cell di kolom Kuantitas saja yang bisa diedit maka terdapat pengecekan terhadap parameter columnIndex. Parameter aValue berisi data yang diinput user ke dalam cell tersebut,

karena dalam method getColumnClass sudah di tentukan bahwa kolom Kuantitas adalah Integer maka kita bisa melakukan casting langsung terhadap aValue sebagai class Integer. Setelah itu dilakukan perhitungan sub total, total dan mengeset lblTotal untuk menampilkan total dari transaksi.

Setelah property, method dan TableModel selesai dibuat, silahkan edit konstruktor dari class SalesPanel menjadi seperti berikut ini :

```
public SalesPanel() {
    initComponents();
    tblSalesDetail.setAutoCreateColumnsFromModel(false);
    jdcTransaction.setDate(new Date());
    enableForm(false);
    btnDelete.setEnabled(false);
    btnAdd.setEnabled(true);
    btnCancel.setEnabled(false);
    btnEdit.setEnabled(false);
    btnSave.setEnabled(false);
}
```

Berikutnya adalah kode-kode yang dieksekusi oleh setiap tombol-tombol di bagian atas SalesPanel :

Tombol Tambah :

```
private void btnAddActionPerformed(java.awt.event.ActionEvent evt) {
    clearForm();
    enableForm(true);
    //pengaturan tombol
    btnDelete.setEnabled(false);
    btnAdd.setEnabled(false);
    btnCancel.setEnabled(true);
    btnEdit.setEnabled(false);
    btnSave.setEnabled(true);
}
```

Tombol Edit :

```
private void btnEditActionPerformed(java.awt.event.ActionEvent evt) {
    if(sales!=null){
        enableForm(true);
        btnDelete.setEnabled(false);
        btnAdd.setEnabled(false);
        btnCancel.setEnabled(true);
        btnEdit.setEnabled(false);
        btnSave.setEnabled(true);
    }
}
```

Tombol Hapus :

```
private void btnDeleteActionPerformed(java.awt.event.ActionEvent evt) {
    if(sales!=null){
        try{
            Main.getSalesService().delete(sales);
            clearForm();
            sales = null;
            refreshTable();
            enableForm(false);
            //pengaturan tombol
            btnDelete.setEnabled(false);
            btnAdd.setEnabled(true);
            btnCancel.setEnabled(false);
        }
    }
}
```

```

        btnEdit.setEnabled(false);
        btnSave.setEnabled(false);
    } catch (Exception ex) {
        log.error(ex);
        JOptionPane.showMessageDialog(this,
            "Data masih digunakan tidak bisa dihapus!"
            , "Error", JOptionPane.ERROR_MESSAGE);
    }
}
}
}

```

Tombol Simpan :

```

private void btnSaveActionPerformed(java.awt.event.ActionEvent evt) {
    if(validateForm()){
        if(sales == null){
            sales = new Sales();
        }
        loadFormToModel();
        try{
            Main.getSalesService().save(sales);
            clearForm();
            refreshTable();
            enableForm(false);
            //pengaturan tombol
            btnDelete.setEnabled(false);
            btnAdd.setEnabled(true);
            btnCancel.setEnabled(false);
            btnEdit.setEnabled(false);
            btnSave.setEnabled(false);
        } catch (Exception ex) {
            log.error(ex);
            JOptionPane.showMessageDialog(this, "Data gagal disimpan!"
            , "Error", JOptionPane.ERROR_MESSAGE);
        }
    }
}
}
}

```

Tombol Batal :

```

private void btnCancelActionPerformed(java.awt.event.ActionEvent evt) {
    clearForm();
    enableForm(false);
    //pengaturan tombol
    btnDelete.setEnabled(false);
    btnAdd.setEnabled(true);
    btnCancel.setEnabled(false);
    btnEdit.setEnabled(false);
    btnSave.setEnabled(false);
}

```

Tombol Cari :

```

private void btnSearchActionPerformed(java.awt.event.ActionEvent evt) {
    Sales s = new SalesLookupDialog().getSales();
    if(s!=null){
        sales = Main.getSalesService().getSales(s.getId());
        loadModelToForm();
        //edit mode
        btnDelete.setEnabled(true);
    }
}

```

```

        btnAdd.setEnabled(false);
        btnCancel.setEnabled(true);
        btnEdit.setEnabled(true);
        btnSave.setEnabled(false);
    }
}

```

Tombol cari di atas menggunakan class SalesLookupDialog yang sudah dibuat sebelumnya, terlihat ada kode Sales s = new SalesLookupDialog().getSales(); yang digunakan untuk memanggil SalesLookupDialog, return dari method getSales bisa null atau ada nilainya. Kalau ada nilainya berarti tombol OK yang ditekan, kalau nilainya null artinya tombol Cancel yang ditekan.

Tombol Keluar :

```

private void btnExitActionPerformed(java.awt.event.ActionEvent evt) {
    Main.getFrame().salesPanel = null;
    dispose();
}

```

Tombol btnLookupProduct :

```

private void btnLookupProductActionPerformed(java.awt.event.ActionEvent evt) {
    Product p = new ProductLookupDialog().getProduct();
    if(p!=null){
        boolean isProductInSalesDetails = false;
        for (SalesDetail salesDetail : salesDetails) {
            if(salesDetail.getId()!=null
                && salesDetail.getProduct().getId().equals(p.getId())){
                salesDetail.setQuantity(salesDetail.getQuantity() + 1);
                salesDetail.setSubtotal(salesDetail.getPrice().multiply(
                    new BigDecimal(salesDetail.getQuantity())));
                isProductInSalesDetails = true;
                break;
            }
        }
        if(isProductInSalesDetails){
            refreshTable();
            refreshTotalLabel();
        } else {
            addSalesDetail(p);
        }
    }
}

```

Ketika user menekan enter di txtProductId, atau ketika user melakukan barcode scan di txtProductId maka event actionPerformed dari txtProductId akan dipanggil. Untuk mendengarkan event actionPerformed klik kanan di txtProductId kemudian pilih Event > actionPerformed dan tambahkan kode berikut ini dalam jendela Code yang terbuka :

```

private void txtProductIdActionPerformed(java.awt.event.ActionEvent evt) {
    String productId = txtProductId.getText();
    try {
        //cari apakah barang sudah ada di dalam tblSalesDetail
        Long pId = Long.valueOf(productId);
        boolean isProductInSalesDetails = false;
        for (SalesDetail salesDetail : salesDetails) {
            if(salesDetail.getId()!=null
                && salesDetail.getProduct().getId().equals(pId)){
                salesDetail.setQuantity(salesDetail.getQuantity() + 1);
                salesDetail.setSubtotal(salesDetail.getPrice().multiply(
                    new BigDecimal(salesDetail.getQuantity())));
            }
        }
    }
}

```

```

        isProductInSalesDetails = true;
        break;
    }
}
if(isProductInSalesDetails){
    refreshTable();
    refreshTotalLabel();
} else {
    Product p = Main.getProductService().getProduct(pId);
    if(p!=null){
        addSalesDetail(p);
    }
}
txtProductId.setText("");
} catch (NumberFormatException numberFormatException) {
    JOptionPane.showMessageDialog(this, "Id barang harus berupa angka!"
        , "Error", JOptionPane.ERROR_MESSAGE);
}
}
}

```

Yang terakhir adalah menambahkan event JinternalFrame closing dari class SalesPanel. Caranya dengan mengklik kanan SalesPanel kemudian pilih Events > InternalFrame > internalFrameClosing kemudian ubah kodenya seperti di bawah ini :

```

private void formInternalFrameClosing(javax.swing.event.InternalFrameEvent evt) {
    Main.getFrame().salesPanel = null;
}

```

Kode di atas digunakan untuk mencegah SalesPanel dibuka berulang-ulang ketika menu Transaksi > Penjualan dipilih user.

Sebelum mencoba menjalankan aplikasi POS ini, perlu dibuat data Product terlebih dahulu, karena tidak ada screen untuk menginput data Product maka bisa kita lakukan insert langsung ke database. Jalankan query ini di dalam console RDBMS yang digunakan :

```

insert into T_PRODUCT(name,price) values ('indomie',1050);
insert into T_PRODUCT(name,price) values ('kecap abc',4950);
insert into T_PRODUCT(name,price) values ('training java',2014950);

```

Sampai di sini kode untuk Data Transaksi selesai dibuat, sebagai bahan latihan silahkan buat PurchasePanel yang kodenya mirip dengan SalesPanel hanya saja data yang diolah adalah Purchase.

Bagian berikutnya kita akan membahas bagaimana membuat report dengan JasperReport. Report ini sangat penting untuk aplikasi bisanis, jenis report bisa bermacam-macam, bisa berupa report transaksi per harian yang berisi list data transaksi atau bisa juga berupa laporan perkembangan penjualan hari per hari yang ditampilkan dalam chart. Laporan pertama bisa dikategorikan sebagai laporan transaksi sedangkan yang kedua adalah jenis laporan Data Mining.

BAGIAN 5

JasperReports

Pengenalan

Konsep report sangat penting dalam aplikasi bisnis perusahaan, bahkan salah satu alasan kenapa semua transaksi harus dicatat adalah untuk membuat laporan keuangan. Ada pertanyaan mungkin kenapa kok harus menggunakan library report seperti jasper ini? Apakah tidak cukup menggunakan Jtable saja? Jawabannya adalah dengan menggunakan library report seperti jasper kita bisa mendesain report dengan bantuan tools visual seperti iReport. Alasan kedua adalah report bisa langsung di print karena formatnya sudah ready for print selain itu setelah report jadi, kita bisa export hasilnya ke berbagai format dokumen seperti PDF, OpenOffice, Excell dan Words, bahkan bisa juga diexport menjadi HTML.

JasperReports adalah library reporting paling populer di Java, kepopuleran JasperReports sangat ditunjang dengan adanya tools visual designer yaitu iReport. Dengan menggunakan iReports membuat report menjadi sangat cepat dan akurat.

Feature-feature JasperReports juga sudah sangat lengkap, berikut ini adalah daftar featurenya yang diambil dari <http://jasperforge.org/projects/jasperreports> :

- Pixel-perfect page-oriented atau continuous output untuk web atau print
- Dashboards, tables, crosstabs, charts dan gauges
- Web-based and pixel-perfect reports
- Report output dalam format PDF, XML, HTML, CSV, XLS, RTF, TXT
- Sub-reports dapat dengan mudah digunakan untuk menampilkan layout yang kompleks
- Support barcode
- Dapat melakukan rotasi posisi text secara visual
- Styles library
- Drill-down / hypertext link, termasuk support untuk PDF bookmarks
- Tidak ada batasan ukuran report
- Conditional printing
- Multi datasource dalam satu report
- Internationalized dan Localizable

Persiapan

Untuk mulai bekerja dengan JasperReports kita perlu mendownload iReport terlebih dahulu, installer iReport dapat didownload dari :

<http://sourceforge.net/projects/ireport/files/iReport/>

Setelah proses download selesai, kemudian silahkan install iReport. Proses instalasi sangat sederhana tinggal menekan tombol next, next, next. Setelah instalasi selesai, jalankan iReport dan anda akan melihat tampilan iReport. Tampilan iReport mirip dengan NetBeans, hal ini dikarenakan iReport dikembangkan menggunakan NetBeans Platform. Kita akan bahas lebih banyak mengenai iReport di bab-bab selanjutnya ketika membahas bagaimana caranya membuat laporan transaksi harian.

Persiapan berikutnya adalah menambahkan library JasperReports ke dalam project POS. Pada waktu buku ini ditulis, versi JasperReports terbaru adalah 3.7.6 dan Jar-jar yang dibutuhkan untuk mencompile dan menjalankan JasperReports 3.7.6 ini antara lain :

- commons-beanutils-1.8.2.jar
- commons-collections-3.2.1.jar
- commons-digester-1.7.jar
- commons-javaflow-20060411.jar
- commons-logging-1.1.jar
- groovy-all-1.7.5.jar
- jasperreports-3.7.6.jar

Anda bisa mendownload JasperReports dari link di bawah ini :

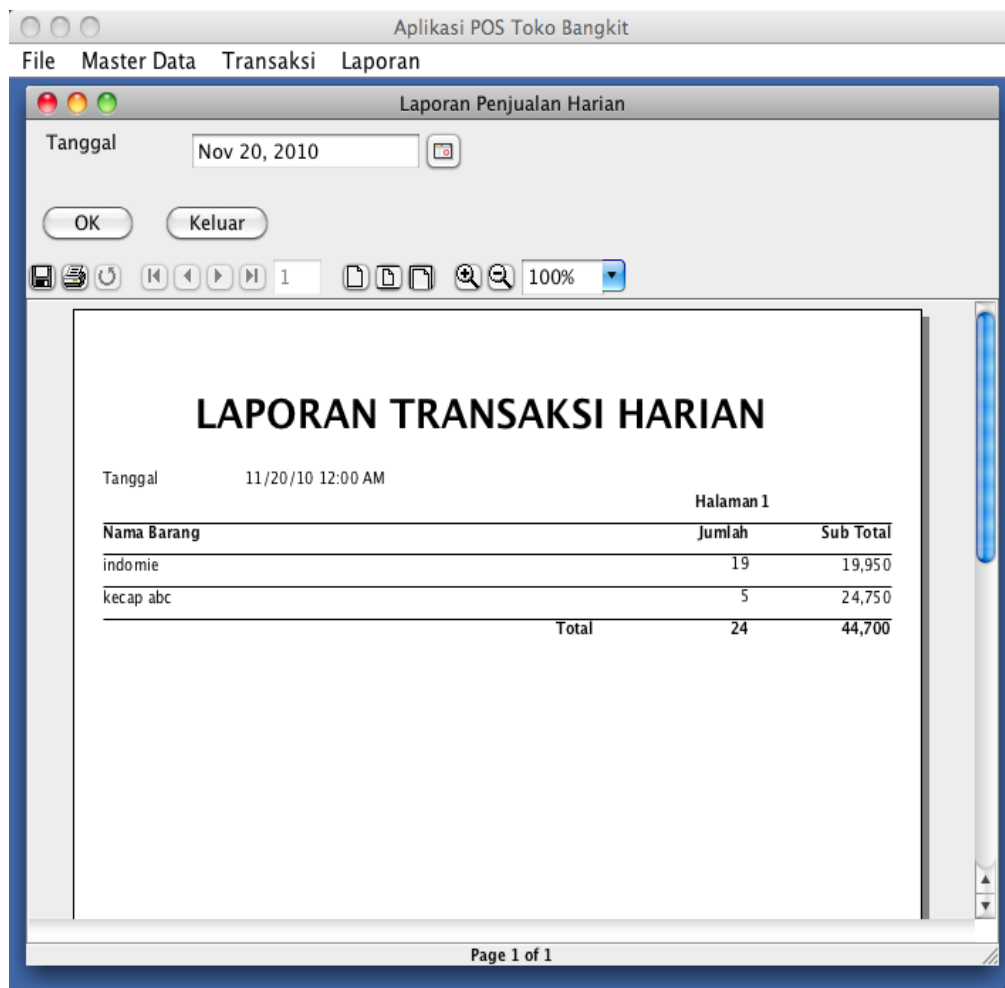
<http://sourceforge.net/projects/jasperreports/>

atau mencari jar-jar di atas dari folder instalasi iReport. Kemudian tambahkan ke dalam folder lib dari project NetBeans, dan terakhir tambahkan semua jar di atas ke dalam library project.

Setelah proses persiapan selesai kita siap untuk membuat Laporan Transaksi Harian.

Laporan Transaksi Harian

Dalam buku ini kita akan belajar membuat laporan transaksi harian, laporan ini ditampilkan dalam menu yang ada fasilitas pemilihan tanggal transaksi yang akan dibuatkan reportnya. Laporan transaksi harian terlihat seperti gambar di bawah ini :



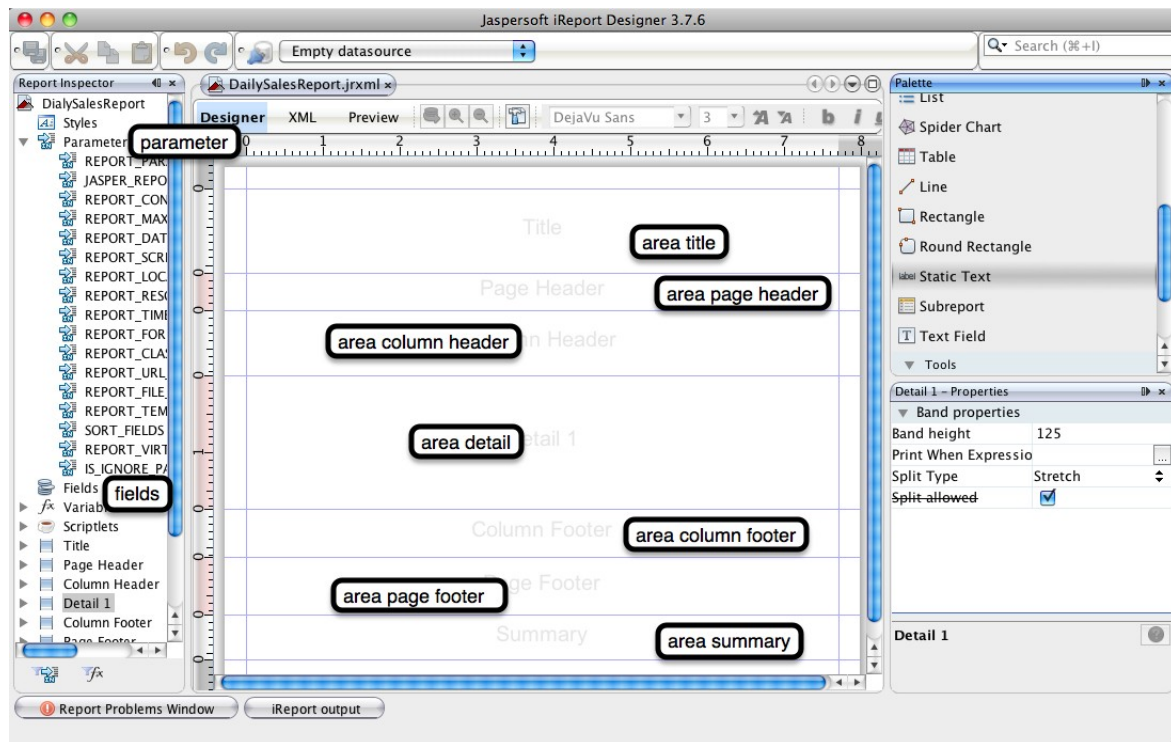
Terlihat di bagian atas terdapat date chooser yang digunakan untuk memilih tanggal kemudian ada tombol OK untuk menampilkan report dan report yang sudah dibuat ditampilkan di bawahnya menggunakan komponen JasperReports yaitu JRViewer.

Untuk membuat laporan di atas, diperlukan file reportnya yang dibuat menggunakan iReport, kemudian JInternalFrame untuk menampilkan report dalam aplikasi dan ReportService yang digunakan untuk melakukan akses ke database dan mendapatkan data reportnya. Mari kita bahas satu per satu cara membuatnya.

Membuat JasperReports Menggunakan iReport Visual Designer

Di bab sebelumnya kita sudah membahas instalasi iReport, sekarang kita bahas lebih lanjut bagaimana membuat report menggunakan iReport ini. Langkah pertama adalah dengan membuat report baru dari iReport. Pilih menu File > New, kemudian di dialog yang muncul pilih Report > Blank A4, simpan file reportnya dengan nama DailySalesReport.jrxml, letakkan di dalam folder src/reports dari project anda.

Setelah proses pembuatan file report baru selesai, tampilan awal iReport bisa dilihat seperti gambar di bawah ini.



Sebelah Kiri ada jendela Report Inspector yang berisi komponen-komponen yang ada dalam report. Bagian terpenting yang berhubungan dengan kode kita nanti adalah Parameters, Fields dan Variables, kita akan bahas lebih banyak tentang ketiganya di bagian selanjutnya. Kemudian di bagian tengah ada Band atau area dari report, ada title, page header, column header, detail, column footer dan summary.

Sebelah kanan ada palette yang berisi komponen-komponen report seperti garis, text, subreport, kotak, gambar dan lain-lainya. Biasanya untuk laporan keuangan komponen yang diletakkan dalam report cukup minimalis, hal ini dimaksudkan untuk menghemat tinta ketika diprint di kertas. Di bawah Palette ada jendela Properties yang digunakan untuk mengedit property dari komponen report.

Band Title digunakan untuk meletakkan judul dari report, komponen yang digunakan untuk judul adalah static text yang diubah font-nya agar kelihatan lebih besar.

Band Page Header digunakan kalau akan menampilkan sesuatu yang selalu ada di setiap halaman, misalnya nomer halaman report dan tanggal transaksi.

Band Column Header digunakan untuk menampilkan header column dari reportnya.

Band Detail berisi field-field yang akan ditampilkan, jadi nanti kita akan membuat tiga buah field yaitu nama barang, jumlah dan subtotal untuk ditampilkan di report ini.

Band Column Footer digunakan untuk menampilkan sesuatu setiap kolom selesai, dalam report ini tidak digunakan.

Band Page Footer digunakan untuk menampilkan sesuatu di bagian bawah page, bisa digunakan untuk menampilkan halaman atau sub total setiap halaman report.

Band Summary digunakan untuk menampilkan sesuatu di akhir dari report, misalnya menampilkan grand total atau misalnya tempat untuk tanda tangan.

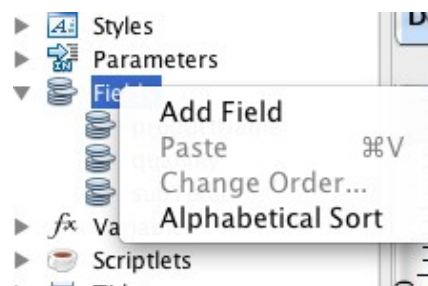
Setelah mengetahui bagian-bagian dari iReport kita bisa mulai untuk menyusun reportnya, dimulai dari membuat Field.

Membuat Field

Pada bagian sebelumnya di bab tentang Hibernate, kita sempat membahas class DailySalesReport yang digunakan untuk kepentingan report. Field yang harus dibuat dalam report harus sama persis dengan property yang dimiliki oleh class DailySalesReport. Mari kita lihat class DailySalesReport :

```
public class DailySalesReport {
    private String productName;
    private Long quantity;
    private BigDecimal subTotal;
    public String getProductName() {
        return productName;
    }
    public void setProductName(String productName) {
        this.productName = productName;
    }
    public Long getQuantity() {
        return quantity;
    }
    public void setQuantity(Long quantity) {
        this.quantity = quantity;
    }
    public BigDecimal getSubTotal() {
        return subTotal;
    }
    public void setSubTotal(BigDecimal subTotal) {
        this.subTotal = subTotal;
    }
}
```

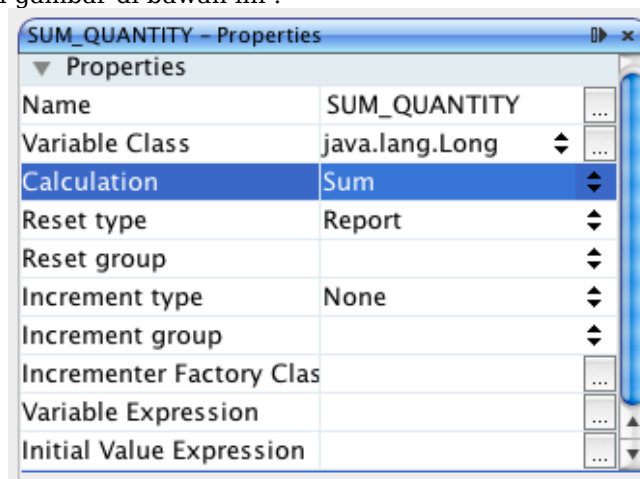
Kita lihat di dalam class DailySalesReport di atas mempunyai tiga buah property yaitu productName, quantity dan subTotal. Mari kita buat ketiga buah field tersebut di dalam report, caranya dengan klik kanan node Fields > Add Field seperti gambar di bawah ini



Setelah field di buat jangan lupa untuk merubah tipe dari field di jendela properties sebelah kanan. Tipe field harus sama persis dengan tipe property dalam class DailySalesReport, kalau tidak sama akan mengakibatkan ClassCastException. Kemudian kita drag field-field tersebut ke jendela sebelah kanan di bagian details, tambahkan kolom header yang sesuai dengan fieldnya. Hasilnya bisa dilihat di gambar di bawah ini :

Column Header		
Nama Produk	Jumlah	Sub Total
<code>{product}</code>	<code>{quantity}</code>	<code>{subTotal}</code>

Seperti kita lihat di atas, untuk field angka gunakan Horizontal Align kanan agar posisi digit bisa dibandingkan dengan baris atasnya. Berikutnya kita akan menambahkan sum dari quantity dan subTotal yang diletakkan di Band Summary agar ditampilkan sekali saja di akhir dari report. Untuk membuat sum kita akan menambahkan Variables, caranya dengan mengklik kanan Variables > Add Variable, kemudian ganti nama variabelnya menjadi SUM_QUANTITY dan setting properties-nya seperti gambar di bawah ini :



Terlihat dari gambar di atas bahwa variabel name adalah SUM_QUANTITY, class-nya sama persis dengan field quantity yaitu java.lang.Long dan calculation Sum. Buat satu lagi variabel dengan nama SUM_SUBTOTAL dengan tipe BigDecimal dan calculation Sum. Kemudian letakkan kedua variabel tersebut di Band Summary, seperti gambar di bawah ini :

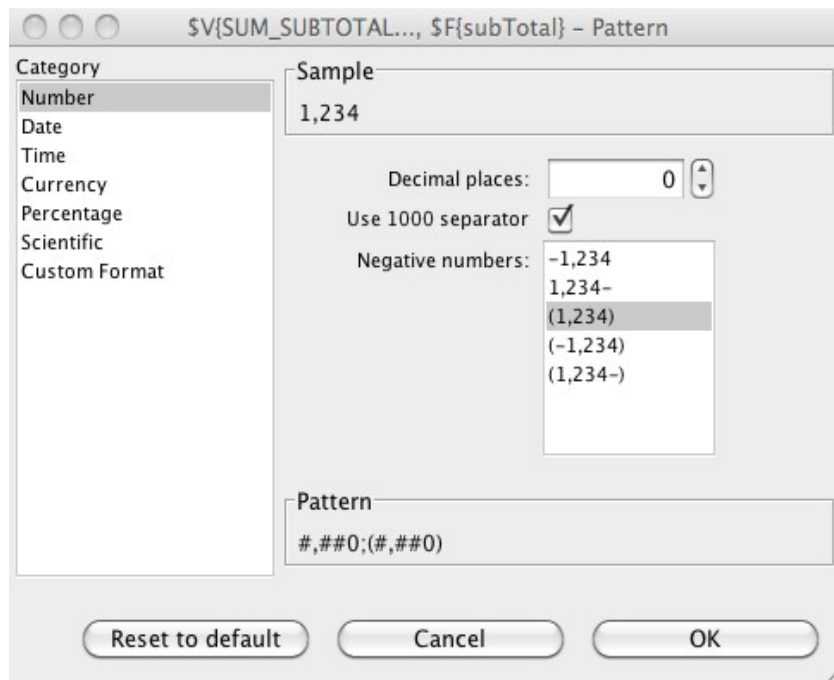
Column Header		
Nama Produk	Jumlah	Sub Total
<code>{product}</code>	<code>{quantity}</code>	<code>{subTotal}</code>
Detail 1		
Summary		
Total	<code>{SUM_QUANTITY}</code>	<code>{SUM_SUBTOTAL}</code>

Terlihat bahwa setelah Band Detail langsung ada Band Summary sedangkan Band lain tidak terlihat, kita bisa mendrag Band-band tersebut hingga tingginya menjadi nol dan tidak terlihat lagi. Kita bisa mensetting field dan variabel yang bertipe uang seperti subTotal dan SUM_SUBTOTAL agar menggunakan format ribuan, caranya dengan mengganti Pattern dari keduanya. Pilih subTotal dan SUM_SUBTOTAL di jendela Designer, kemudian edit properties pattern dari jendela Properties, akan muncul dialog Pattern seperti gambar di bawah ini. Pilih Decimal places menjadi 0 atau angka yang dikehendaki, kemudian check combo box Use 1000 Separator dan pilih pattern untuk Negative numbers.

Langkah selanjutnya adalah menambahkan nomor halaman di setiap halaman report, caranya dengan mendrag ke variabel PAGE_NUMBER yang sudah disediakan oleh JasperReports. Letakkan variabel PAGE_NUMBER di Band Page Header atau Band Page Footer, tergantung design dari report yang dikehendaki.

Berikutnya adalah menambahkan tanggal dari laporan, misalnya kita ingin menampilkan laporan penjualan tanggal tertentu maka harus ada parameter tanggal yang dilempar ke report. Caranya adalah dengan membuat sebuah Parameters dengan tipe java.util.Date, klik kanan di node

Parameters > Add Parameter dan beri nama date untuk parameter tersebut, kemudian drag parameter date dan letakkan di Band Page Header. Format tanggal bisa dirubah dengan mengedit properties pattern.



Langkah terakhir adalah menambahkan judul dari laporan dengan mendrag sebuah Static Text, beri judul "LAPORAN TRANSAKSI HARIAN" kemudian ubah font agar terlihat lebih besar. Agar tampilan laporan menjadi sedikit rapi, bisa ditambahkan garis untuk membatasi Band Column Header, Band Detail dan Band Summary. Hasil akhir dari report ini bisa dilihat seperti gambar di bawah ini :

LAPORAN TRANSAKSI HARIAN		
Tanggal	\$P{date}	Halaman\$V
Nama Barang	Jumlah	Sub Total
\$F{productName}	\$F{quantity}	\$F{subTotal}
Total	\$V	\$V{SUM_TOTAL}

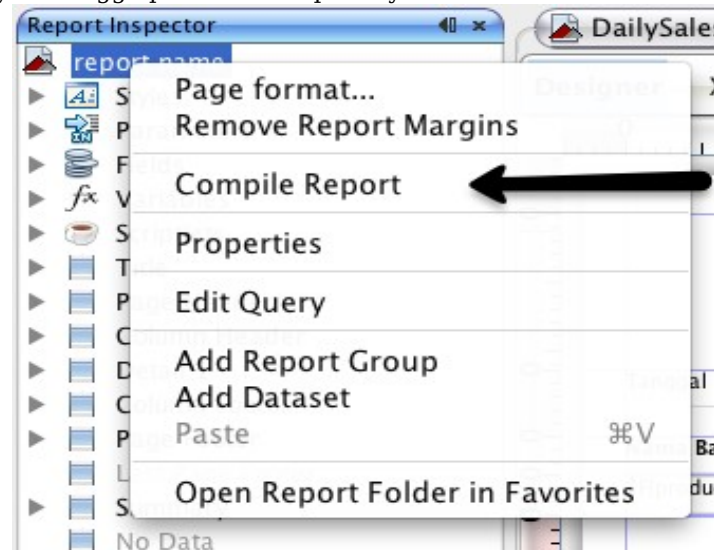
Proses pembuatan DailySalesReport.jrxml sudah selesai, berikutnya kita tambahkan utility untuk mengcompile file .jrxml menjadi file .jasper

Mencompile File jrxml Menjadi jasper Menggunakan Ant Task

Proses pembentukan report di JasperReport dimulai dengan membuat design report, hasilnya adalah file jrxml. Setelah itu file jrxml akan dicompile menjadi file jasper; pada waktu aplikasi berjalan file jasper ini akan di load dan diisi dengan data menjadi object JasperPrint, nah object ini sudah berisi data-data yang diperlukan untuk diubah menjadi berbagai format dokumen seperti PDF, Excell atau Words menggunakan JasperExporter. Atau bisa juga JasperPrint ditampilkan ke dalam komponen Swing yaity JRViewer.

Proses kompilasi dari jrxml ke jasper bisa menggunakan iReport dengan mengklik kanan node Report-nya dan pilih menu Compile Report seperti gambar di halaman selanjutnya. Hasil kompilasi berupa file jasper bisa diambil di folder yang ditampilkan di jendela Report Output. Tetapi proses ini harus dilakukan setiap kali reportnya di ubah, ada kemungkinan lupa

mengcompile file-nya sehingga perubahan report-nya tidak terlihat.



Cara yang lebih baik adalah dengan membuat sebuah Ant Task untuk mengcompile semua file report setiap kali project NetBeans di-Clean And Build. Sebelum kita lanjutkan untuk membuat kode-nya, sedikit kita bahas apa itu Ant.

Ant adalah build script tools yang digunakan untuk melakukan compile file .java, kemudian membuat jar dan juga bisa digunakan untuk menjalankan aplikasi yang sedang didevelop. Bahkan NetBeans di belakang layar menggunakan Ant ini untuk semua kegiatannya, dari proses build hingga menjalankan application server atau menjalankan emulator Java ME. Semuanya dilaksanakan menggunakan Ant script. Dalam Ant script ada "target" dimana target adalah satu blok script yang bisa dieksekusi. Setiap kali ingin membuat sebuah "task" yang dibuat adalah target ini.

Bentuk Ant Script adalah file XML, anda bisa membuka Ant Script dari project NetBeans yang sekarang sedang kita kerjakan dengan membuka folder nbproject dan cari file build-impl.xml atau file build.xml yang ada di root folder dari project NetBeans. File build-impl.xml adalah jantung dari project NetBeans, jadi tidak boleh diedit secara manual sama sekali, sedangkan file build.xml bisa diedit dengan memasang task yang ingin kita eksekusi. Dalam build.xml ada beberapa target dengan nama spesial, target-target ini dieksekusi setelah perintah tertentu di dalam NetBeans selesai dilaksanakan. Misalnya target "-post-compile" akan dilaksanakan setelah proses compile di NetBeans selesai.

Satu lagi istilah dalam Ant yang perlu diketahui, yaitu Taskdef. Setiap jenis task dalam Ant pada dasarnya adalah kode Java, nah taskdef ini digunakan untuk memapping antara istilah dalam Ant dan kode Java sebenarnya. Misalnya kita nanti akan mendefinisikan sebuah task yang tugasnya mengcompile file jrxml menjadi file jasper, yang kita lakukan pertama kali adalah mendefinisikan task apa yang akan digunakan dan nama class untuk task tersebut. Lebih jauh mengenai Ant bisa dibaca dari website ant.apache.org

Mari kita lihat kode Ant script yang ditambahkan dalam build.xml untuk mengcompile file jrxml menjadi jasper di bawah ini :

```
<taskdef name="jrc"
  classname="net.sf.jasperreports.ant.JRAntCompileTask">
  <classpath>
    <fileset dir="./lib">
      <include name="**/*.jar"/>
    </fileset>
  </classpath>
</taskdef>
<target name="-post-compile">
  <mkdir dir="./build/reports"/>
```

```

<delete>
  <fileset file="./src/reports/*.jasper"/>
</delete>
<jrc srcdir="./src/reports" destdir="./src/reports"
  tempdir="./build/reports"
  keepjava="true" xmlvalidation="true">
  <include name="**/*.jrxml"/>
</jrc>
</target>

```

Seperti kita lihat di atas, Ant script dimulai dengan definisi taskdef jrc yang ditambahkan classpath dimana file-file jar yang dibutuhkan untuk mencompile jrxml berada. Di bawahnya ada target dengan nama -post-compile, target ini dieksekusi setiap kali NetBeans selesai meng-compile file-file .java, di dalamnya ada script untuk membuat folder (mkdir), kemudian menghapus semua file jasper dan terakhir menggunakan taskdef jrc.

Coba klik kanan project di NetBeans kemudian pilih Clean and Build, script di atas akan dieksekusi dan output dari script ini bisa dilihat di jendela Output :

```

Created dir: /Users/ifnu/NetBeansProjects/java-desktop-book/code/build/reports
Compiling 1 report design files.
log4j:WARN No appenders could be found for logger
(net.sf.jasperreports.engine.xml.JRXmlDigesterFactory).
log4j:WARN Please initialize the log4j system properly.
File : /Users/ifnu/NetBeansProjects/java-desktop-
book/code/src/reports/DailySalesReport.jrxml ... OK.

```

Report sudah siap, Ant Script sudah bisa mengcompile file jrxml, langkah berikutnya adalah menyiapkan ReportService untuk mengambil data dari database dan memasukkan datanya ke dalam jasper.

Mempersiapkan ReportService

Seperti halnya modul-modul lain dalam aplikasi, kita membutuhkan service untuk mengambil data dari database. Dalam hal ini adalah ReportService, berbeda dengan service-service yang lain ReportService tidak memerlukan DAO, hal ini dikarenakan query untuk report selalu custom dan kita hanya memerlukan kolom-kolom yang ditampilkan dalam report saja, sedangkan kolom-kolom lain tidak diperlukan. Dengan cara seperti ini, proses query ke database menjadi lebih efisien.

Berikut ini kode untuk interface ReportService yang di letakkan dalam package yang sama dengan interface service lainnya.

```

public interface ReportService {
    JasperPrint getDailySalesReport(Date date);
}

```

Kode implementasi dari ReportService, yaitu ReportServiceImpl

```

@Service("reportService")
@Transactional(readOnly=true)
public class ReportServiceImpl implements ReportService{

    private static final Logger log = Logger.getLogger(ReportServiceImpl.class);
    @Autowired private SessionFactory sessionFactory;

    public JasperPrint getDailySalesReport(Date date) {
        try{
            List<DailySalesReport> dailySalesReports =
                sessionFactory.getCurrentSession()
                    .createQuery("select s.product.name as productName,"
                        + " sum(s.quantity) as quantity, "
                        + " sum(s.subtotal) as subTotal from SalesDetail s ")

```

```

        + " where day(s.sales.salesDate) = day(:date) "
        + " group by s.product.name order by s.product.name")
        .setParameter("date", date)
        .setResultTransformer(
            Transformers.aliasToBean(DailySalesReport.class))
        .list();

    InputStream is = ReportServiceImpl.class
        .getResourceAsStream("/reports/DailySalesReport.jasper");

    Map<String, Object> parameters = new HashMap<String, Object>();
    parameters.put("date", date);

    return JasperFillManager.fillReport(is, parameters,
        new JRBeanCollectionDataSource(dailySalesReports));
} catch (JRException ex) {
    log.error("error generate DailySalesReport", ex);
}
return null;
}
}

```

Mari kita bahas satu per satu kode di atas. Di bagian paling atas terdapat annotation Spring yaitu @Service dan @Transactional, kita sudah membahas panjang lebar mengenai kedua annotation ini. Berikutnya ada annotation dari Spring juga yaitu @Autowired untuk menginject SessionFactory ke dalam ReportServiceImpl ini. Ada juga kode untuk menginisialisasi Logger dari log4j.

Bagian berikutnya ada kode untuk mengeksekusi HQL, query HQL yang dieksekusi adalah :

```

select s.product.name as productName, sum(s.quantity) as quantity,
       sum(s.subtotal) as subTotal
from SalesDetail s
where day(s.sales.salesDate) = day(:date)
group by s.product.name
order by s.product.name

```

HQL di atas melakukan query ke database untuk mendapatkan nama barang, jumlah dari barang tersebut dan jumlah subtotal-nya. Fungsi day digunakan untuk membandingkan hanya tanggal saja tanpa memperhatikan waktu-nya. Group by digunakan untuk menjumlahkan jumlah barang dan subtotal berdasarkan nama barangnya. Order by digunakan untuk mengurutkan data berdasarkan nama barang.

HQL di atas menyederhanakan SQL cukup signifikan, hal ini dikarenakan kita tidak perlu melakukan join antara tiga buah table: T_PRODUCT, T_SALES dan T_SALES_DETAIL. Cukup select dari SalesDetail dan gunakan operator . (titik) untuk mengakses kolom lain yang berelasi dengan Entity SalesDetail.

Method setParameter digunakan untuk memasukkan parameter tanggal transaksi ke dalam HQL. Sedangkan method setResultTransformer digunakan untuk mentransformasi hasil query menjadi sebuah List of Class. Kalau tidak menggunakan setResultTransformer maka hasil query di atas adalah List<Object[]>, dimana proses memasukkan data dengan struktur tersebut ke dalam report menjadi sedikit lebih susah. Karena Transformer yang digunakan adalah aliasToBean maka setiap kolom yang diselect **harus** ditambahkan as dan diikuti dengan nama properties dari class DailySalesReport. Walaupun nama kolom yang diselect dalam hql sama persis dengan nama properties dari class DailySalesReport, as tetap harus didefinisikan.

Kode berikutnya digunakan untuk meload file jasper dari filesystem, kenapa harus menggunakan kode seperti di bawah ini ?

```

InputStream is =
    ReportServiceImpl.class.getResourceAsStream("/reports/DailySalesReport.jasper");

```

Hal ini dikarenakan nantinya file jasper di atas akan diletakkan dalam jar. Kode di bawah ini bisa juga digunakan untuk meload file jasper, **tetapi hanya bisa berjalan kalau aplikasi dijalankan dari NetBeans, kalau aplikasi dijalankan tanpa netbeans dlangsung dari jar yang dihasilkan, maka kodenya menjadi error :**

```
InputStream is = new FileInputStream("src/reports/DailySalesReport.jasper");
```

Error yang muncul adalah FileNotFoundException dikarenakan pada waktu aplikasi sudah diinstall, tidak ada lagi filder src dan file jasper berada di dalam jar.

Kode berikutnya digunakan untuk melempar Parameters dari kode java ke dalam report. Struktur yang digunakan adalah Map, dimana key bertipe String dan value bertipe Object. Nilai key harus sama dengan nama parameter di dalam report dan tipe dari value-nya harus sama persis dengan tipe value di dalam Parameter report. Kalau kita lihat kembali di bagian sebelumnya, report DailySalesReport hanya mempunyai satu Parameters bernama "date" dengan tipe java.util.Date.

Kode terakhir digunakan untuk mengisi reprot dengan data yang sudah diambil dari database. Return dari method ini adalah JasperPrint yang bisa ditampilkan dalam komponen JRViewer.

Kita juga perlu memodifikasi class Main untuk memegang instance dari ReportService ini. Kode class Main menjadi seperti berikut ini :

```
public class Main {

    private static SecurityService securityService;
    private static SalesService salesService;
    private static ProductService productService;
    private static ReportService reportService;
    private static MainFrame frame;

    public static SecurityService getSecurityService() {
        return securityService;
    }
    public static SalesService getSalesService() {
        return salesService;
    }
    public static ReportService getReportService() {
        return reportService;
    }
    public static ProductService getProductService() {
        return productService;
    }
    public static MainFrame getFrame() {
        return frame;
    }

    public static void main(String[] args) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                ApplicationContext =
                    new ClassPathXmlApplicationContext("applicationContext.xml");
                securityService =
                    (SecurityService) applicationContext.getBean("securityService");
                salesService =
                    (SalesService) applicationContext.getBean("salesService");
                productService =
                    (ProductService) applicationContext.getBean("productService");
                reportService =
                    (ReportService) applicationContext.getBean("reportService");
                frame = new MainFrame();
            }
        });
    }
}
```

```

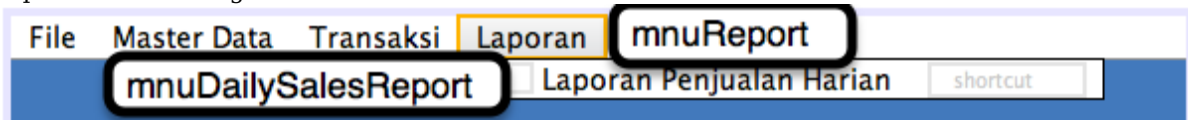
        frame.setVisible(true);
    }
    });
}
}

```

Langkah berikutnya adalah membuat UI untuk menampilkan report di atas. MainFrame perlu ditambahkan satu menu dan membuat class DailySalesReportPanel yang bertipe JinternalFrame.

Membuat UI Report

Langkah pertama adalah mengubah MainFrame dengan menambahkan satu menu baru untuk report. Perhatikan gambar di bawah ini :



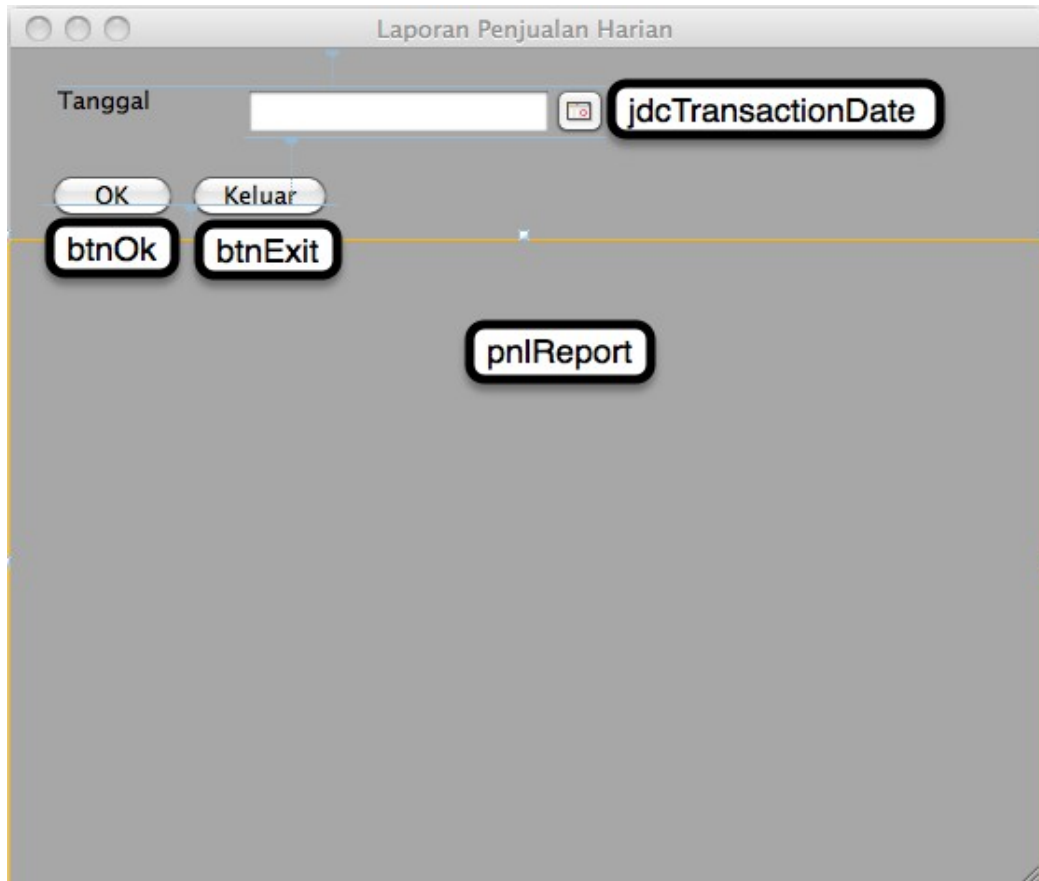
mnuReport adalah Jmenu yang diletakkan dalam menuBar dan mnuDailySalesReport adalah JmenuItem yang diletakkan di bawah mnuReport. Tambahkan event actionPerformed di dalam mnuDailySalesReport, jendela Code akan terbuka dan tambahkan kode di bawah ini :

```

public DailySalesReportPanel dailySalesReportPanel;
private void mnuDailySalesReportActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        if (dailySalesReportPanel == null) {
            dailySalesReportPanel = new DailySalesReportPanel();
            desktopPane.add(dailySalesReportPanel);
        } else {
            dailySalesReportPanel.toFront();
        }
        dailySalesReportPanel.setVisible(true);
        dailySalesReportPanel.setSelected(true);
        dailySalesReportPanel.setSize(desktopPane.getSize());
    } catch (PropertyVetoException ex) {
        log.error("error ketika menampilkan sales panel", ex);
    }
}
}

```

Berikutnya kita akan membuat DailySalesReportPanel, pilih menu File > New kemudian di dialog yang muncul pilih Swing GUI Form > JinternalFrame Form. Letakkan dalam package com.googlecode.projecttemplate.pos.ui.reports



Class `DailySalesReportPanel` di atas terdiri dari sebuah label `Tanggal`, kemudian `JdateChooser` `jdcTransactionDate`, di bawahnya ada dua buah `Jbutton` yaitu `btnOk` dan `btnExit`. Di bagian paling bawah ditandai dengan garis coklat-kuning adalah `Jpanel` `pnlReport`. `Jpanel` di bagian paling bawah ini digunakan untuk menampilkan `JViewer` dari `JasperReport`. Layout dari `pnlReport` harus diganti dengan `BorderLayout` agar `JViewer` yang ditampilkan menempati penuh `pnlReport`. Cara mengubah layoutnya adalah dengan mengklik kanan `pnlReport` kemudian pilih menu `Set Layout > Border Layout`.

Setelah screen siap, mari kita lihat kode apa saja yang ada dalam class ini. Pertama kita lihat konstruktornya :

```
public DailySalesReportPanel() {
    initComponents();
    jdcTransactionDate.setDate(new Date());
}
```

Kode di atas hanya melakukan satu hal yaitu mengeset tanggal dari `jdcTransactionDate` dengan tanggal hari ini.

Kedua adalah kode ketika tombol `Keluar` ditekan, untuk memasang listener ketika tombol `Keluar` ditekan bisa dilakukan dengan mengklik dua kali tombol `Keluar` atau klik kanan tombol `keluar` kemudian pilih `Events > actionPerformed`.

```
private void btnExitActionPerformed(java.awt.event.ActionEvent evt) {
    Main.getFrame().dailySalesReportPanel = null;
    dispose();
}
```

Kode di atas tidak jauh berbeda dengan semua kode untuk tombol `Keluar` yang sudah kita tulis di bagian-bagian sebelumnya.

Berikutnya kita harus memasang listener ketika user menekan icon `X` dan menutup `JinternalFrame`, kode yang harus dieksekusi hanya mengeset variabel `dailySalesReport` di

dalam frame menjadi null. Hal ini diperlukan untuk mencegah JFrame dibuka double ketika user menekan menu Laporan Penjualan Harian ditekan berkali-kali. Klik kanan DailySalesReportPanel di jendela Design kemudian pilih Events > JFrame > JFrameClosing

```
private void formJFrameClosing(javax.swing.event.JFrameEvent evt){
    Main.getFrame().dailySalesReportPanel = null;
}
```

Kode terakhir adalah listener ketika tombol OK ditekan. Di dalamnya ada kode untuk memanggil ReportService kemudian membuat object JViewer yang dipasang di dalam pnlReport dan update UI-nya agar tampilan pnlReport direfresh :

```
private void btnOkActionPerformed(java.awt.event.ActionEvent evt) {
    JasperPrint print = Main.getReportService().getDailySalesReport(
        jdcTransactionDate.getDate());
    JViewer viewer = new JViewer(print);
    pnlReport.removeAll();
    pnlReport.add(viewer, BorderLayout.CENTER);
    pnlReport.updateUI();
}
```

Nah sekarang kode-nya sudah selesai, silahkan jalankan aplikasi-nya dan pastikan tidak ada error pada waktu menampilkan report.

Proses pembuatan report selanjutnya sama dengan langkah-langkah yang sudah kita kerjakan, awali dengan mempersiapkan report, usahakan report yang dibuat cukup sederhana, kemudian lakukan query dan pengolahan data agar mudah ditampilkan di dalam report. Berikutnya adalah mempersiapkan UI dengan parameter-parameter yang diperlukan oleh report.

Best practice yang saya ikuti adalah buat report sesederhana mungkin pada waktu mendesign, kemudian lakukan pengolahan data di dalam kode Java. Hal ini lebih mudah dilakukan karena proses logic yang terlalu rumit di dalam report menjadi susah dipahami. Lebih baik pengolahan data yang rumit dilakukan di dalam kode Java. Tetapi perlu diperhatikan juga bahwa semua data report di-load ke dalam aplikasi menjadi List, kalau ukuran data dalam database sangat besar hingga ribuan baris, maka aplikasi rentan terkena OutOfMemoryException. Jika hal tersebut tidak bisa dihindari, sebaiknya gunakan query atau Stored Procedure yang dipanggil dari file report. Kemudian detail koneksi database harus menggunakan username yang hanya bisa membaca database saja, jangan sampai menyimpan informasi Super User database di dalam report.

BAGIAN 6

ARSITEKTUR THREE TIER

Arsitektur Three Tier

Selama ini aplikasi desktop sebagian besar menggunakan arsitektur Client Server, dimana client akan langsung melakukan koneksi ke database. Konsep ini dipopulerkan oleh Microsoft bersama dengan platform VB. Konsep utamanya adalah meletakkan bisnis proses dan logic aplikasi di dalam server database lewat Stored Procedure dan Trigger.

Belakangan konsep client-server mulai mendapat tantangan dengan berkembangnya jaringan internet. Perusahaan ingin membuat aplikasi desktop namun kantor-kantor cabang tersebar di beberapa tempat, jika masih ingin menggunakan arsitektur client-server lewat jaringan internet, maka banyak sekali pertimbangan keamanan yang perlu diperhatikan, karena membuka port database di jalur internet bukanlah ide yang baik dari sisi security. Masalah lain adalah lisensi, banyak database vendor yang memberlakukan lisensi per user untuk menentukan harga lisensi. Dengan arsitektur client server, setiap client harus menggunakan user yang berbeda sehingga secara drastis menaikkan harga lisensi mana kala ada ratusan client terhubung ke server. Masalah lain yang muncul adalah jika client menggunakan koneksi dialup miskin bandwidth, komunikasi client ke server akan sedikit terganggu (corrupt).

Selain alasan di atas, alasan performance tuning juga menjadi perhatian kalau menggunakan arsitektur client server. Misalnya clientnya sudah membengkak menjadi ratusan, maka server (database) harus melayani semua client tersebut, kalau database sudah tidak sanggup lagi maka harus beli hardware lebih besar dan migrasi dari hardware lama. Arsitektur Three tier dapat mengatasi hal ini dengan membebankan sebagian pekerjaan ke Server Aplikasi, sehingga database server tidak bekerja sendirian.

Teknologi-teknologi baru seperti cache dan Hibernate Search dapat meningkatkan kinerja aplikasi secara keseluruhan dengan meminimalisasi hit ke database. Dimana database connection sering kali menjadi bottleneck dalam aplikasi dengan ukuran data yang sangat besar. Ukuran data yang besar memunculkan masalah serius, yaitu slow query. Hal ini terjadi karena proses select-join dilakukan dalam table yang berukuran sangat besar

Arsitektur aplikasi three tier membagi aplikasi menjadi tiga lapisan. Lapisan pertama berada di sini client, lapisan pertama tidak terhubung langsung ke database melainkan terhubung ke lapisan kedua yaitu Application server, koneksi ke database berada di application server. Keuntungan utamanya adalah setiap client tidak perlu mempunyai user dalam database, koneksi ke database bisa dipool di dalam application server, sehingga bisa dikelola seefisien mungkin. Lapisan terakhir adalah database sebagai tempat penyimpanan data permanen.

Arsitektur three tier tidak menyaratkan semua lapisan harus dijalankan dalam server fisik yang berbeda, bisa saja dari client, application server hingga database server dijalankan dalam satu server. Tetapi arsitektur ini mempunyai fleksibilitas tinggi sehingga kalau load aplikasi menjadi besar, setiap lapisan bisa dipisahkan menjadi beberapa server fisik.

Ada juga alasan keamanan yang menjadikan arsitektur ini sangat strategis, berdasarkan level sensitifitasnya, database berada di level paling dalam dari aplikasi yang disebut dengan area DMZ (demilitarized Zone). Area DMZ ini hanya bisa diakses dari network internal saja dan hanya sedikit sekali orang yang mempunyai akses ke dalam area ini. Sehingga antar setiap lapisan bisa difilter menggunakan firewall yang makin ke dalam makin strict aturannya. Misalnya application server hanya bisa diakses dari IP yang diijinkan saja, kemudian ditambah satu lagi firewall antara application server dengan database dimana hanya IP application server plus IP untuk administrasi saja yang bisa melakukan koneksi.

Koneksi antara client ke application server menggunakan protokol yang disebut remoting. Alternatif remoting di Java sangat banyak, pilih salah satu alternatif yang sesuai dengan infrastruktur dan kebutuhan. Di bawah ini adalah matriks perbandingan antara berbagai macam remoting yang bisa digunakan :

Matriks Perbandingan antar protokol remoting

Remoting	Jenis Remoting	Protokol	Interoperability dengan platform lain	Implementasi	Kebutuhan bandwidth
RMI	Binary	TCP socket	Hanya aplikasi Java	EJB, RMI murni	Sangat besar, cocok di intranet
CORBA	Binary	TCP socket	Java, C++, Python dll	Implementasi sudah jarang, nyaris obsolete	Sangat besar, cocok di intranet
Spring HTTPInvoker	Binary	HTTP	Java dan harus pake spring	Spring remoting	Cukup besar, cocok untuk intranet
Hessian	Binary	HTTP	Java dan library hessian	Hessian	Cukup besar, cocok untuk intranet
Hessian	Text / XML	HTTP	Java dan library hessian	Hessian	Cocok untuk internet, karena berbasis text
Webservice	Text / XML	HTTP	Semua platform bisa	JAX-WS, Spring WS, Axis2	Cocok untuk internet, karena berbasis text
HTTP call with json/XML	Text / json / XML	HTTP	Semua platform bisa. Apache HTTPClient	Jackson, JSON lib,	Cocok untuk internet, karena berbasis text

Jika aplikasi akan di jalankan dalam lingkungan intranet dimana bandwidth dan stabilitas koneksi tidak menjadi masalah maka pilihan remoting bisa diatuhkan kepada Spring Remoting menggunakan protokol HTTPInvoker. Jika aplikasi di sisi server akan diexpose ke internet sehingga bandwidth menjadi sangat penting dan stabilitas koneksi tidak bisa diandalkan maka remoting menggunakan menggunakan HTTP call dengan format data XML/JSON menjadi pilihan yang tepat. Hal ini dikarenakan kalau paket data berupa binary dan urutan datangnya paket tidak beraturan maka pake binary akan menjadi corrupt dan tidak bisa oleh penerima.

Keuntungan HTTP Invoker adalah kode untuk komunikasi antara client dengan application server menjadi sangat sederhana menggunakan feature Spring Remoting. Tidak ada perubahan kode sama sekali, hanya perlu ditambahkan konfigurasi untuk server dan konfigurasi untuk client. Sangat mengesankan.

Implementasi Arsitektur Three Tier

Kode yang sudah kita buat di tujuh bab sebelumnya tidak banyak berubah, hanya saja semua Entity harus mengimplementasikan interface Serializable. Hal ini dikarenakan setiap object yang akan dikirim dari client ke application server atau sebaliknya akan dirubah menjadi binary data kemudian ditransfer lewat jaringan dan di tujuan akan disusun lagi menjadi object di dalam Java. Proses transfer object dari satu tempat ke tempat yang lain menyaratkan bahwa class dari object tersebut harus mengimplementasikan interface Serializable dan semua property di dalam class tersebut harus juga mengimplementasikan interface Serializable.

Lihat package `com.googlecode.projecttemplate.pos.model` dan pastikan semua class di dalam package tersebut implements `Serializable`. Perhatikan pula di semua Service yang ditulis, method `save` dan `delet` akan mengembalikan object Entity, hal ini dimaksudkan agar client dapat memperoleh object yang sudah dimanipulasi dalam database. Misalnya pada waktu `save` property `id` dari Entity bernilai `null` kalau datanya masih baru, `id` tersebut akan diset di server pada waktu proses `save` selesai dan diisi nilai yang digenerate dari database. Object Entity yang ada di client nilai `id`-nya masih `null` sehingga perlu diupdate dengan object yang dikembalikan oleh server.

Implementasi three tier menggunakan Spring sangat mudah, perubahan kode Java sangat minimal dan hanya perlu menambahkan dua buah konfigurasi masing-masing untuk client dan server. Perubahan kode Java hanya membuat satu buah class untuk start server dan sedikit perubahan di class `Main`. Yang pertama akan kita lakukan adalah membuat konfigurasi untuk Server-nya. Buat sebuah xml di dalam folder `src`, namakan dengan `serverContext.xml`, kemudian isikan kode berikut ini di dalam `serverContext.xml` :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd
    http://www.springframework.org/schema/tx
    http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

  <bean
    class="org.springframework.remoting.support.SimpleHttpServerFactoryBean">
    <property name="contexts">
      <map>
        <entry key="/PersonService" value-ref="personServiceHttpInvoker"/>
        <entry key="/ProductService" value-ref="productServiceHttpInvoker"/>
        <entry key="/PurchaseService" value-ref="purchaseServiceHttpInvoker"/>
        <entry key="/ReportService" value-ref="reportServiceHttpInvoker"/>
        <entry key="/SalesService" value-ref="salesServiceHttpInvoker"/>
        <entry key="/SecurityService" value-ref="securityServiceHttpInvoker"/>
      </map>
    </property>
    <property name="port" value="9090"/>
  </bean>
  <bean id="personServiceHttpInvoker"
    class="org.springframework.remoting.httpinvoker.SimpleHttpInvokerServiceExporte
r"
    p:service-ref="personService"
    p:serviceInterface="com.googlecode.projecttemplate.pos.service.PersonService"/>
  <bean id="productServiceHttpInvoker"
    class="org.springframework.remoting.httpinvoker.SimpleHttpInvokerServiceExporte
r"
    p:service-ref="productService"
    p:serviceInterface="com.googlecode.projecttemplate.pos.service.ProductService"/
>
  <bean id="purchaseServiceHttpInvoker"
    class="org.springframework.remoting.httpinvoker.SimpleHttpInvokerServiceExporte
r"
    p:service-ref="purchaseService"
    p:serviceInterface="com.googlecode.projecttemplate.pos.service.PurchaseService"
  />
</beans>
```

```

<bean id="reportServiceHttpInvoker"
    class="org.springframework.remoting.httpinvoker.SimpleHttpInvokerServiceExporter"
    p:service-ref="reportService"
    p:serviceInterface="com.googlecode.projecttemplate.pos.service.ReportService"
/>
<bean id="salesServiceHttpInvoker"
    class="org.springframework.remoting.httpinvoker.SimpleHttpInvokerServiceExporter"
    p:service-ref="salesService"
    p:serviceInterface="com.googlecode.projecttemplate.pos.service.SalesService"
/>
<bean id="securityServiceHttpInvoker"
    class="org.springframework.remoting.httpinvoker.SimpleHttpInvokerServiceExporter"
    p:service-ref="securityService"
    p:serviceInterface="com.googlecode.projecttemplate.pos.service.SecurityService"
/>
</beans>

```

Bagian paling atas file XML ini adalah deklarasi namespace dari Spring, kemudian di bawahnya ada kode untuk mensetting Java 6 Sun Http Server. Semenjak Java 6, Sun menyertakan modul kecil web server yang bisa digunakan sebagai Http Server, dalam arsitektur three tier Http Server ini digunakan sebagai middle tier untuk menjembatani client dengan database. Kalau misalnya project-nya berukuran cukup besar dan user cukup banyak, sebaiknya gunakan application server yang lebih besar semisal tomcat atau glassfish. Konfigurasi di atas hanya bisa digunakan untuk Sun Http Server yang ada di Java 6, kalau ingin menggunakan tomcat atau glassfish maka konfigurasi di atas harus diubah, kemudian harus dibuat project baru bertipe web project, konfigurasi perlu dilakukan di web.xml dan war yang dihasilkan dari web project tersebut bisa di deploy di dalam tomcat atau glassfish.

Spring menyediakan class untuk mengkonfigurasi Sun Http Server yaitu SimpleHttpServerFactoryBean, class tersebut memerlukan port dan map untuk memapping URL (context path) dengan HttpHandler. Seperti contoh di atas, untuk setiap service kita akan membuat HttpInvoker exporter yang bertipe HttpHandler, kemudian dimapping dengan URL. Misalnya konfigurasi di bawah ini :

```

<entry key="/PersonService"
    value-ref="personServiceHttpInvoker"/>

```

Konfigurasi di atas akan memapping bean personServiceHttpInvoker ke dalam Url /PersonService, url tersebut nantinya bisa diakses oleh client dari url :

```

http://localhost:9090/PersonService

```

Bean personServiceHttpInvoker sendiri adalah instansiasi class SimpleHttpInvokerServiceExporter yang mengimplementasikan HttpHandler dan mengekspose personService sebagai http invoker.

Setelah selesai membuat konfigurasi untuk server, sekarang kita buat konfigurasi untuk client-nya. Konfigurasi client akan melakukan sebaliknya dari yang dilakukan server, kalau di server adalah mengekspose Spring Service menjadi sebuah URL, maka konfigurasi client akan mengubah sebuah URL menjadi Spring Service.

Buat sebuah file XML dengan nama clientContext.xml di dalam folder src kemudian isikan kode berikut ini :

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans

```

```

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<context:property-placeholder location="classpath:server.properties"/>
<bean id="personService"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean"
  p:serviceUrl="http://${server.ip}:${server.port}/PersonService"
  p:serviceInterface="com.googlecode.projecttemplate.pos.service.PersonService">
</bean>
<bean id="productService"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean"
  p:serviceUrl="http://${server.ip}:${server.port}/ProductService"
  p:serviceInterface="com.googlecode.projecttemplate.pos.service.ProductService">
</bean>
<bean id="purchaseService"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean"
  p:serviceUrl="http://${server.ip}:${server.port}/PurchaseService"
  p:serviceInterface="com.googlecode.projecttemplate.pos.service.PurchaseService">
</bean>
<bean id="reportService"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean"
  p:serviceUrl="http://${server.ip}:${server.port}/ReportService"
  p:serviceInterface="com.googlecode.projecttemplate.pos.service.ReportService">
</bean>
<bean id="salesService"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean"
  p:serviceUrl="http://${server.ip}:${server.port}/SalesService"
  p:serviceInterface="com.googlecode.projecttemplate.pos.service.SalesService">
</bean>
<bean id="securityService"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean"
  p:serviceUrl="http://${server.ip}:${server.port}/SecurityService"
  p:serviceInterface="com.googlecode.projecttemplate.pos.service.SecurityService">
</bean>
</beans>

```

Seperti biasa, kode XML Spring diawali dengan deklarasi namespace. Kemudian di bawahnya ada kode untuk meload property server.properties yang berisi konfigurasi IP dari server dan port-nya. Isi dari server.properties adalah :

```

server.ip=127.0.0.1
server.port=9090

```

Di dalam XML property di atas bisa diakses menggunakan sintaks \${server.ip} dan \${server.port}. Bagian berikutnya adalah merubah URL menjadi Spring Service, contohnya :

```

<bean id="personService"
  class="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean"
  p:serviceUrl="http://${server.ip}:${server.port}/PersonService"
  p:serviceInterface="com.googlecode.projecttemplate.pos.service.PersonService">
</bean>

```

konfigurasi di atas berusaha membuat bean dengan nama personService di client, bean ini berasal dari URL berikut ini setelah di substitusi dengan nilai dari server.properties :

http://127.0.0.1:9090/PersonService

Interface yang digunakan sebagai tipe service-nya adalah PersonService. Jadi secara sederhana

bisa dikatakan bahwa konfigurasi di atas memindahkan Service yang ada di server supaya bisa diakses dari client.

Membuat Server dan Client Class

Seperti disebutkan di atas, kita perlu membuat sebuah class untuk menjalankan server, class ini fungsinya sama dengan class Main, bedanya hanya pada file Spring Context apa yang dieksekusi. File applicationContext.xml dan serverContext.xml akan dijalankan di server sedangkan client hanya menjalankan clientContext.xml, sehingga perlu dilakukan perubahan sedikit di dalam class Main yang bertindak sebagai client.

Buat sebuah class namakan POSServer di package com.googlecode.projecttemplate.pos.server kemudian tambahkan kode di bawah ini :

```
public class POSServer {
    public static void main(String[] args) {
        AbstractApplicationContext ctx =
            new ClassPathXmlApplicationContext(
                new String[]{"applicationContext.xml", "serverContext.xml"});
        ctx.registerShutdownHook();
    }
}
```

Kemudian lakukan perubahan di dalam class Main untuk mengganti applicationContext.xml dengan clientContext.xml seperti di bawah ini :

```
//ApplicationContext applicationContext =
// new ClassPathXmlApplicationContext("applicationContext.xml");
ApplicationContext applicationContext =
    new ClassPathXmlApplicationContext("clientContext.xml");
```

Terlihat dalam kode di atas, dua baris pertama dimatikan dengan memberi tanda komentar dan digantikan dengan 2 baris berikutnya. Kalau ingin merubah kembali arsitektur three tier ke client-server cukup hilangkan tanda komentar di 2 baris pertama dan tambahkan komentar di 2 baris berikutnya, sangat simple dan tetapi powerfull.

Untuk mencoba arsitektur three tier, anda harus menjalankan class POSServer terlebih dahulu baru kemudian class Main. Tidak boleh ada dua POSServer dijalankan sekaligus karena bisa terjadi bentrok untuk memperebutkan port 9090. Secara fungsional tidak ada yang berbeda antara arsitektur client-server dan three-tier, perbedaannya ada pada middle tier yang bisa mengurangi load di client, karena sekarang client tidak perlu manage SessionFactory dari hibernate. SessionFactory hibernate ada di sisi middle tier (application server), sehingga kita bisa menerapkan caching, kalau SessionFactory ada di client maka caching menjadi tidak optimal karena setiap client mempunyai caching sendiri-sendiri.

Sampai di sini aplikasi sudah siap, langkah berikutnya adalah membuat installer untuk menginstall client maupun server. Kita akan menggunakan library izpack untuk membuat installer.

BAGIAN 7

MEMBUAT INSTALLER DENGAN IZPACK

Izpack

IzPack adalah tools yang membantu memecahkan masalah instalasi aplikasi di berbagai platform. Seperti halnya Java, IzPack dapat membuat instalasi yang bisa berjalan di semua platform, windows, linux maupun Mac OSX. Hal ini dimungkinkan karena IzPack juga berbasis Java. Installer hasil generate dari IzPack adalah executable jar yang ketika diklik akan menjalankan proses instalasi aplikasi. IzPack mempunyai banyak tools dan modul yang dapat dengan mudah dikonfigurasi berdasarkan setiap platform dimana aplikasi akan diinstall, misalkan kita bisa mempunyai kondisi kalau di windows akan diinstall file .bat untuk menjalankan aplikasi sedangkan di *nix yang diinstall adalah file .sh

IzPack didesign sangat modular dan mudah dicustomisasi sesuai dengan kebutuhan kita. Konsep modular di dalam IzPack terlihat dari digunakannya XML sebagai konfigurasi untuk mendefinisikan proses instalasi aplikasi. Di dalam XML konfigurasi tersebut kita bisa mendefinisikan langkah-langkah instalasi dan panel apa saja yang akan ditampilkan dalam langkah-langkah instalasi. Sebagai contoh, kita bisa mendefinisikan langkah pertama instalasi adalah panel untuk memilih folder instalasi, kemudian menampilkan readme, kemudian menampilkan lisensi aplikasi, dan terakhir menampilkan progress bar proses instalasi. Semua panel yang disebutkan tadi sudah disediakan secara default oleh IzPack, kita juga bisa membuat sendiri panel yang kita inginkan dengan mengimplementasikan interface yang disediakan oleh IzPack. Pendekatan ini membuat IzPack sangat modular dan mudah dikustomisasi.

Setelah mendefinisikan XML untuk proses instalasi, IzPack dapat mengenerate installer dengan dua cara berbeda :

- Dengan menggunakan command line yang disediakan oleh IzPack
- Dengan menggunakan Ant Task yang sudah disediakan oleh IzPack

Kita akan mencoba kedua pendekatan di atas, cara pertama kita coba menggunakan sample instalasi yang disediakan oleh IzPack, sedangkan cara kedua kita gunakan untuk membuat instalasi POS server dan POS client.

Berikut ini adalah daftar beberapa feature yang dimiliki oleh IzPack :

- Konfigurasi XML untuk mengenerate installer
- Multi bahasa, sudah tersedia 10 bahasa (sayang bahasa indonesia tidak termasuk)
- Integrasi dengan Ant, atau menggunakan command line
- Kustomisasi yang mudah dengan panel dan API yang sangat lengkap
- Konsep variabel konfigurasi yang sangat powerfull
- Konsep kondisi yang bagus untuk segala kemungkinan proses instalasi
- Dapat mengenerate berbagai macam installer : standard, web-based, multi-volume dll
- Dapat mengeksekusi command line di *nix maupun windows
- Hasil instalasi berupa satu buah file tunggal (jar) sehingga sangat ringkas untuk didistribusikan
- Integrasi dengan kode native platform
- dst

Setelah kita bahas sedikit background IzPack, sekarang waktunya kita coba untuk membuat installer

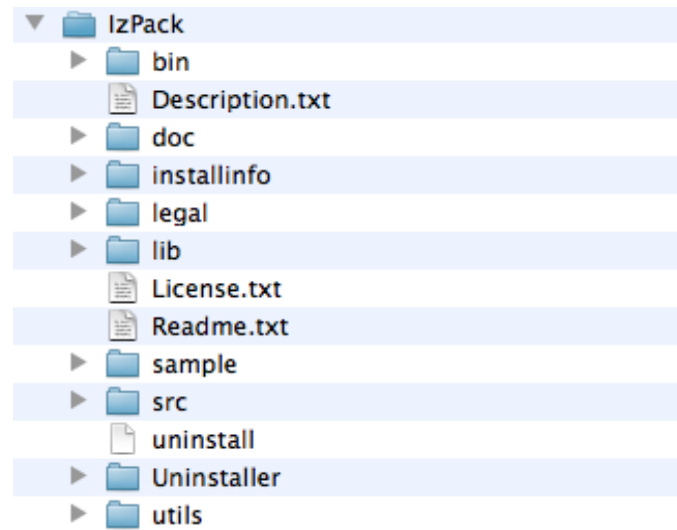
Membuat Installer dari Sample di IzPack

IzPack dapat didownload dari : <http://izpack.org/downloads> , pada waktu buku ini ditulis versi

IzPack stable adalah 4.3.3. Setelah download selesai, IzPack dapat diinstall dengan mudah, buka command prompt dan cd ke dalam folder dimana file IzPack installer berada, kemudian jalankan perintah berikut ini :

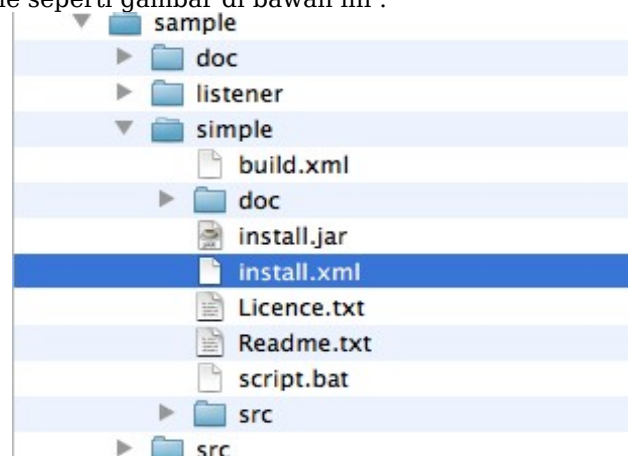
```
$ java -jar izpack-install-4.3.3.jar
```

Atau silahkan double click file izpack-install-4.3.3.jar. Setelah proses instalasi selesai, mari kita periksa satu per satu folder instalasi IzPack.



Folder bin berisi command line untuk membuat installer dari XML konfigurasi. Folder doc berisi semua dokumentasi IzPack, ada versi html dan versi pdf. Folder berikutnya adalah installinfo dan legal yang berisi informasi proses instalasi dan berisi lisensi IzPack. Folder lib berisi library-library yang diperlukan oleh IzPack. Folder sample berisi contoh XML konfigurasi IzPack, di bagian selanjutnya nanti kita akan menggunakan XML konfigurasi yang ada di dalam folder sample ini untuk mencoba membuat installer dengan IzPack. Folder src berisi source code dari IzPack. Uninstaller berisi jar yang digunakan untuk uninstall IzPack dari system. Folder terakhir adalah utils yang berisi utility seperti native launcher (exe) untuk berbagai macam platform.

Nah sekarang mari kita coba membuat installer dengan IzPack dari sample yang disediakan. Folder sample berisi file seperti gambar di bawah ini :



Terdapat file install.xml yang merupakan XML konfigurasi IzPack, kita akan menggunakan install.xml ini untuk membuat installer. Cara yang kita gunakan untuk membuat installer dalam bagian ini adalah dengan menggunakan command line. Silahkan buka command prompt atau console kemudian navigasi (cd) ke dalam folder \$IZPACK_HOME/sample/simple, dimana \$IZPACK_HOME adalah folder dimana IzPack diinstall. Kemudian jalankan perintah berikut ini :

```
$ ../../bin/compile install.xml
```

Akan keluar output di console seperti di bawah ini :

```
::: IzPack - Version 4.3.3 :::

< compiler specifications version: 1.0 >

- Copyright (c) 2001-2008 Julien Ponge
- Visit http://izpack.org/ for the latest releases
- Released under the terms of the Apache Software License version 2.0.

-> Processing : install.xml
-> Output : install.jar
-> Base path : .
-> Kind : standard
-> Compression : default
-> Compr. level: -1
-> IzPack home : /Applications/IzPack

Adding resource: IzPack.uninstaller
Setting the installer information
Setting the GUI preferences
Adding langpack: eng
Adding resource: flag.eng
Adding langpack: fra
Adding resource: flag.fra
Adding resource: LicencePanel.licence
Adding resource: InfoPanel.info
Adding content of jar: file:/Applications/IzPack/lib/standalone-compiler.jar!/bin/panels/HelloPanel.jar
Adding content of jar: file:/Applications/IzPack/lib/standalone-compiler.jar!/bin/panels/InfoPanel.jar
Adding content of jar: file:/Applications/IzPack/lib/standalone-compiler.jar!/bin/panels/LicencePanel.jar
Adding content of jar: file:/Applications/IzPack/lib/standalone-compiler.jar!/bin/panels/TargetPanel.jar
Adding content of jar: file:/Applications/IzPack/lib/standalone-compiler.jar!/bin/panels/PacksPanel.jar
Adding content of jar: file:/Applications/IzPack/lib/standalone-compiler.jar!/bin/panels/InstallPanel.jar
Adding content of jar: file:/Applications/IzPack/lib/standalone-compiler.jar!/bin/panels/FinishPanel.jar
Building installer jar: /Applications/IzPack/sample/simple/install.jar
[ Begin ]

Copying the skeleton installer
Copying 7 files into installer
Merging 7 jars into installer
Writing 3 Packs into installer
Writing Pack 0: Base
Writing Pack 1: Docs
Writing Pack 2: Sources

[ End ]
Build time: Sat Dec 04 14:53:32 SGT 2010
```

Proses pembuatan installer ini akan menghasilkan sebuah file dengan nama install.jar, file ini sudah berisi semua aplikasi plus installer-nya. Jadi cukup satu file ini saja yang dicopy atau

didistribusikan sebagai installer. Nah sekarang mari kita coba untuk menjalankan install.jar ini, jalankan perintah ini dari folder \$IZPACK_HOME/sample/simple :

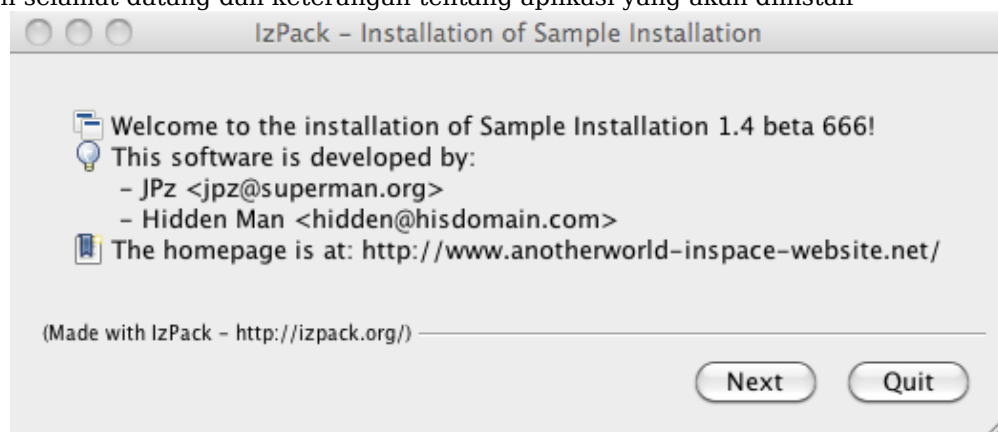
```
$ java -jar install.jar
```

Atau bisa juga langsung double click install.jar karena ini adalah executable jar. Setelah itu akan muncul jendela langkah-langkah instalasi seperti di bawah ini :

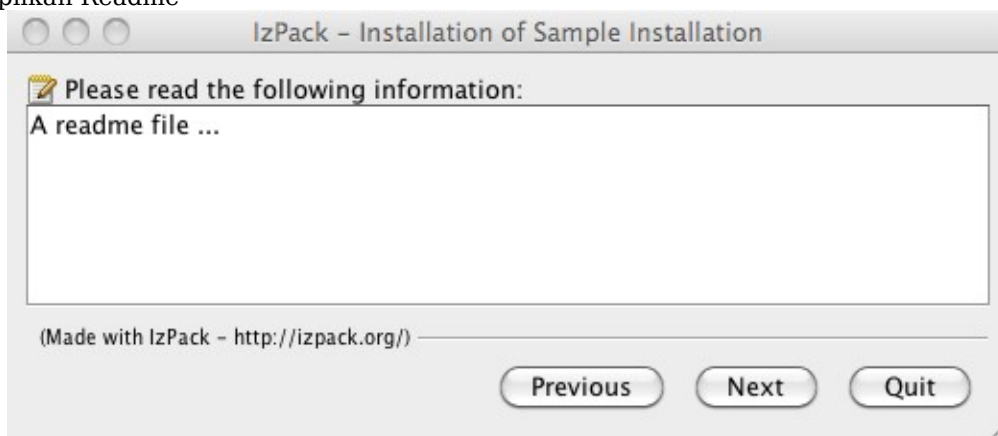
Memilih Bahasa



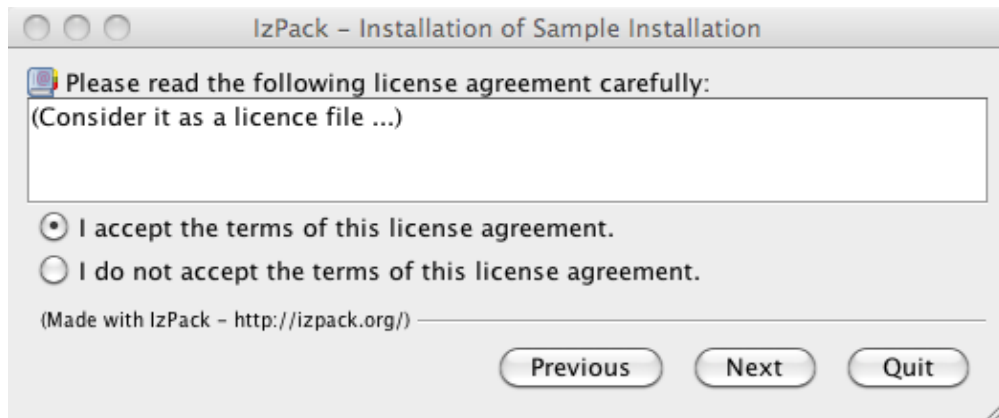
Halaman selamat datang dan keterangan tentang aplikasi yang akan diinstall



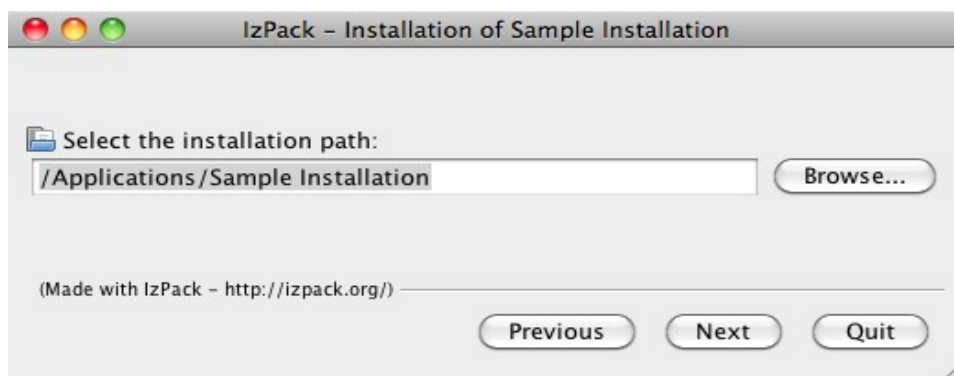
Menampilkan Readme



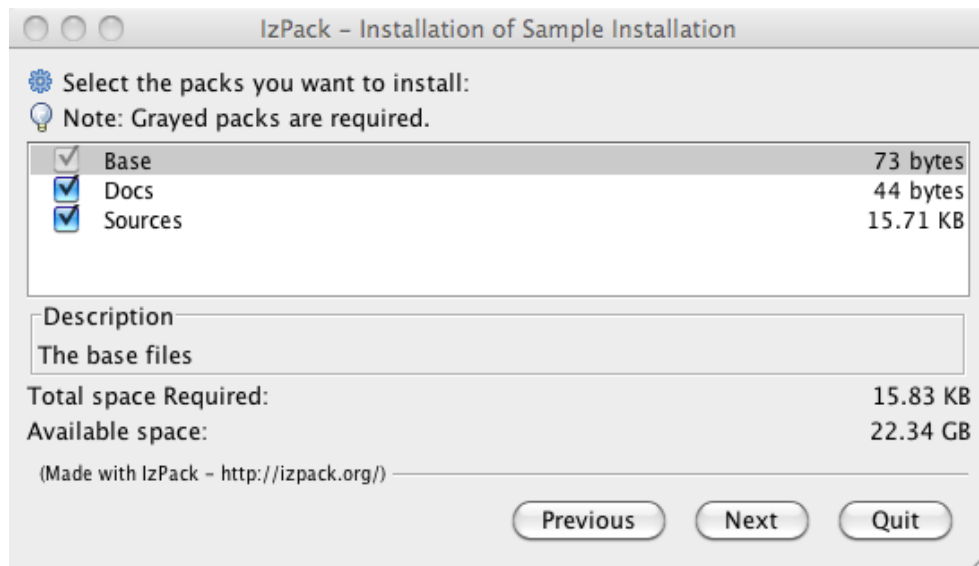
Menampilkan lisensi



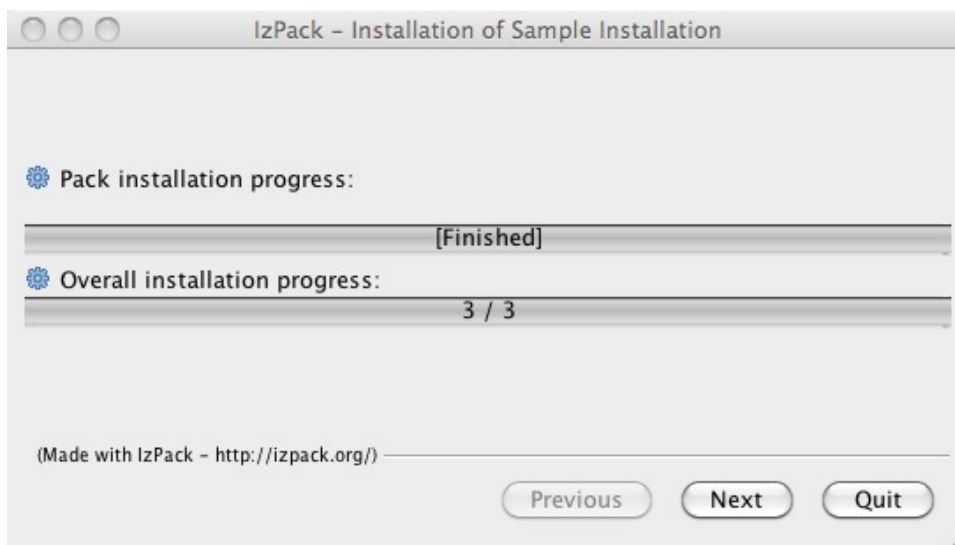
Menampilkan halaman untuk memilih folder instalasi



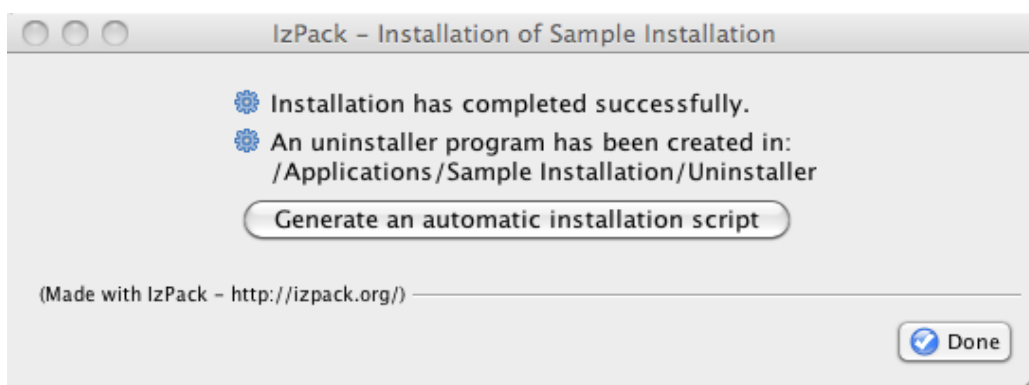
Memilih modul apa saja yang akan diinstall



Progress Bar menampilkan proses instalasi sukses



Rekap proses instalasi sukses dijalankan



Setelah melihat bagaimana mudahnya membuat installer dengan IzPack, sekarang waktunya belajar yang agak susah, yaitu membuat XML konfigurasi. Kita akan menggunakan install.xml dari sampe IzPack kemudian kita bahas satu per satu bagian dari install.xml tersebut.

XML Konfigurasi IzPack

Untuk membuat installer yang kita lihat sebelumnya, diperlukan sebuah file XML konfigurasi. File ini dibuat dengan tangan alias tidak ada aplikasi GUI untuk mengenerate. Sekarang mari kita lihat install.xml dan kita bahas satu per satu bagianya :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>
<installation version="1.0">
  <info>
    <appname>Sample Installation</appname>
    <appversion>1.4 beta 666</appversion>
    <authors>
      <author name="JPz" email="jpz@superman.org"/>
      <author name="Hidden Man" email="hidden@hisdomain.com"/>
    </authors>
    <url>http://www.anotherworld-inspace-website.net/</url>
```



```

</info>
<guiprefs width="640" height="480" resizable="yes"/>
<locale>
  <langpack iso3="eng"/>
  <langpack iso3="fra"/>
</locale>
<resources>
  <res id="LicencePanel.licence" src="Licence.txt"/>
  <res id="InfoPanel.info" src="Readme.txt"/>
</resources>
<panels>
  <panel classname="HelloPanel"/>
  <panel classname="InfoPanel"/>
  <panel classname="LicencePanel"/>
  <panel classname="TargetPanel"/>
  <panel classname="PacksPanel"/>
  <panel classname="InstallPanel"/>
  <panel classname="FinishPanel"/>
</panels>
<packs>
  <pack name="Base" required="yes">
    <description>The base files</description>
    <file src="Readme.txt" targetdir="$INSTALL_PATH"/>
    <file src="Licence.txt" targetdir="$INSTALL_PATH"/>
    <file src="script.bat" targetdir="$INSTALL_PATH"/>
    <parsable targetfile="$INSTALL_PATH/script.bat"/>
    <!-- The file will be parsed -->
  </pack>
  <pack name="Docs" required="no">
    <description>The documentation</description>
    <file src="doc" targetdir="$INSTALL_PATH"/>
    <!-- Recursive adding -->
  </pack>
  <pack name="Sources" required="no">
    <description>The sources</description>
    <file src="src" targetdir="$INSTALL_PATH"/>
  </pack>
</packs>
</installation>

```

Seperti hal-nya file XML, install.xml diawali dengan tag `<?xml>` di baris paling atas. Hal ini wajib dilakukan untuk sebuah file XML yang valid. IzPack XML konfigurasi mempunyai `<installation>` sebagai root tag-nya, attribute version digunakan untuk mendefinisikan versi dari XML konfigurasi, untuk saat ini cukup diisi dengan "1.0".

Tag berikutnya adalah `<info>` yang berisi informasi aplikasi seperti nama aplikasi, pembuat, website dan seterusnya. Di bawahnya ada `<guiprefs>` yang digunakan untuk mendefinisikan ukuran layar dari installer. `<locale>` digunakan untuk mengkonfigurasi bahasa apa saja yang bisa digunakan dalam proses instalasi. `<resources>` digunakan untuk mendefinisikan berbagai macam resource yang diperlukan oleh tag berikutnya yaitu `<panels>`, misalnya dalam contoh di atas ada `LicencePanel` dan `InfoPanel` yang memerlukan resource berupa text file yang perlu ditampilkan.

Bagian terpenting dari XML konfigurasi ini adalah `<packs>` dimana proses mengkopi file yang perlu diinstall ke dalam folder instalasi. Kita bisa membagi file-file yang perlu di copy dalam `<pack>`, misalnya di contoh di atas, ada `Base`, `Docs` dan `Sources`. Ada attribute `required` untuk menentukan apakah `<pack>` tersebut harus diinstall dengan memberi nilai `no` ataukah pengguna bisa memilih untuk tidak menginstall dengan memberi nilai `yes`.

Setelah kita ulas sekilas XML konfigurasi di atas, mari kita bahas lebih mendetail.

Tag Installation<installation>

Tag ini adalah root dari XML konfigurasi IzPack, tidak ada yang istimewa dari tag ini, hanya ada satu attribute yaitu version dan isinya harus 1.0

Tag Information <info>

Berisi informasi informasi umum aplikasi, di dalam tag <info> bisa berisi tag :

- <appname> : nama aplikasi
- <appversion> : versi aplikasi
- <appsubpath> : sub path yang diturunkan dari path default instalasi yang diinput user pada waktu proses instalasi berjalan. Kita bisa meletakkan variabel di dalam appsubpath ini, IzPack akan mengevaluasi variabel dan menggantinya dengan nilai sebenarnya pada waktu proses instalasi berjalan. Kalau tidak diset maka appname akan digunakan sebagai nilai default.
- <url> : website aplikasi
- <authors> : mendaftarkan pembuat aplikasi, bisa lebih dari satu dengan menambahkan <author> di dalamnya. <author> mempunyai dua atribut
 - name : nama dari pembuat aplikasi
 - email : email dari pembuat aplikasi
- <uninstaller> : digunakan untuk mendefinisikan apakah setelah proses instalasi selesai akan dibuat uninstaller atau tidak, serta mendefinisikan nama uninstallernya. Tag ini mempunyai attribute "write" yang nilainya "yes" atau "no". Secara default akan dibuat uninstaller, dan hanya jika uninstaller tag diset serta write = "no" maka uninstaller tidak akan dibuat.
- <javaversion> : mendefinisikan versi minimum Java yang diperlukan untuk instalasi, misalnya: 1.1, 1.2, 1.3, 1.4, 1.5 atau 1.6. Nilai ini akan digunakan untuk dibandingkan dengan nilai yang ada dalam system properties java.version.
- <requiresjdk> : nilai yang diijinkan adalah yes atau no, digunakan untuk menentukan apakah installer ini memerlukan JDK atau hanya memerlukan JRE
- <webdir> : digunakan untuk membuat "web installer", nilai dari tag ini adalah url dimana package instalasinya berada di internet, harus berisi url lengkap, misalnya <http://ifnubima.org/pos/installer>
- <summarylogfilepath> : digunakan untuk mendefinisikan letak logfile untuk instalasi.
- <writeintallationsummary> : mendefinisikan apakah file .installinformation akan ditulis atau tidak. Defaultnya adalah ditulis (yes)
- <pack200/> : tambahkan tag ini jika ingin jar yang ada dalam installer dikompres menggunakan pack200, kecuali signed jar. Ukuran installer nantinya akan menurun drastis dari ukuran sebelum dikompres.
- <run-privileged/> : menambahkan tag ini akan berusaha menjalankan installer dalam mode administrator dan menampilkan popup untuk login sebagai user sebelum installer berjalan.

Silahkan melihat listing file install.xml untuk melihat bagaimana tag <info> ini digunakan dalam XML konfigurasi IzPack.

Variabel, Tag Variabel <variable> dan Tag dynamic variabel <dynamicvariable>

Dalam file install.xml di atas terdapat sebuah variabel yang ditandai dengan simbol \$ yaitu \$INSTALL_PATH. IzPack menyediakan banyak variabel yang dapat digunakan dalam XML konfigurasi, daftar lengkap variabel yang sudah disediakan IzPack adalah sebagai berikut:

- \$INSTALL_PATH : path ke folder instalasi
- \$APPLICATIONS_DEFAULT_ROOT : path default untuk aplikasi
- \$JAVA_HOME : letak instalasi Java (JRE)

- `$CLASS_PATH` : Class path yang digunakan untuk mendefinisikan letak library / class, digunakan oleh Java.
- `$USER_HOME` : direktori home dari user di OS
- `$USER_NAME` : nama user yang digunakan untuk login ke OS
- `$APP_NAME` : nama aplikasi yang didefinisikan di `<info>`
- `$APP_URL` : url aplikasi yang didefinisikan di dalam `<info>`
- `$APP_VER` : versi aplikasi yang didefinisikan di dalam `<info>`
- `$ISO3_LANG` : kode bahasa yang dipilih user di langkah pertama instalasi (lihat gambar di bab sebelumnya)
- `$IP_ADDRESS` : alamat ip komputer user
- `$HOST_NAME` : nama host dari komputer user
- `$FILE_SEPARATOR` : pemisah folder dari OS yang digunakan user. Kalau windows \ kalau *nix /
- `$DesktopShortcutCheckboxEnabled`: variabel ini digunakan sebagai penanda apakah setelah instalasi selesai akan dibuatkan shortcut di desktop user apa tidak. Untuk mengeset variabel ini harus menggunakan tag `<variable>`, hati-hati nama variabelnya case-sensitive.
- `$InstallerFrame.logfilePath` : path untuk log proses instalasi. Kalau variabel ini tidak diset, maka digunakan nilai default : `$INSTALL_PAT/Uninstaller/install.log`

Variabel-variabel di atas bisa digunakan dalam proses instalasi untuk bermacam-macam kebutuhan. Selain variabel yang sudah disiapkan oleh IzPack di atas, ada juga variabel yang bersifat dinamis, dimana kita bisa mendefinisikan sendiri. `<dynamicvariables>` digunakan bersama `<variable>` untuk mendefinisikan dynamic variabel ini, contohnya:

```
<dynamicvariables>
  <variable name="app-version" value="1.0" condition="kondisi1"/>
  <variable name="released-on" value="08/03/2010" condition="!kondisi2"/>
</dynamicvariables>
```

IzPack juga mempunyai feature sangat powerful dengan adanya Parse file, dimana kita bisa mendefinisikan variabel di dalam text file, kemudian IzPack akan memarsing file tersebut untuk disubstitusi dengan nilai variabelnya. Nanti di bagian `<pack>` akan dibahas sedikit bagaimana feature parse ini digunakan.

Kondisi dan Tag Kondisi `<conditions>`

IzPack mempunyai feature percabangan eksekusi dengan menggunakan kondisi. Di sebagian besar tag terdapat attribute `condition` yang digunakan untuk mengetasi suatu kondisi tertentu dipenuhi atau tidak, kalau dipenuhi berarti tag tersebut akan dieksekusi, sebaliknya kalau tidak dipenuhi maka tag tersebut tidak akan dieksekusi. Contohnya adalah di listing kode di atas, terdapat attribute `condition` di dalam tag variabel, misalnya `kondisi1` tidak bernilai benar maka variabel dengan nama `app-version` tidak akan dibuat.

IzPack mengijinkan kondisi ini dioperasikan menggunakan operator logika :

- `+` : operator logika dan (and)
- `|` : operator logika atau (or)
- `\` : operator logika Xor
- `!` : operator negasi (not)

IzPack juga menyediakan kondisi yang langsung bisa digunakan dalam XML konfigurasi. Kondisi-kondisi tersebut antara lain :

- `izpack.windowsinstall` : bernilai benar jika OS yang digunakan adalah windows dan keluarganya

- izpack.windowsinstall.xp : bernilai benar jika OS yang digunakan adalah windows XP
- izpack.windowsinstall.2003 : bernilai benar jika OS yang digunakan adalah windows 2003
- izpack.windowsinstall.vista : bernilai benar jika OS yang digunakan adalah windows Vista
- izpack.windowsinstall.7 : bernilai benar jika OS yang digunakan adalah windows 7
- izpack.macinstall : bernilai benar jika OS yang digunakan adalah Mac OSX
- izpack.solarisinstall : bernilai benar jika OS yang digunakan adalah Solaris
- izpack.solarisinstall.x86 : bernilai benar jika OS yang digunakan adalah Solaris yang berjalan di processor dengan arsitektur x86 seperti Intel dan AMD
- izpack.solarisinstall.sparc : bernilai benar jika OS yang digunakan adalah Solaris yang berjalan di processor sparc

Selain kondisi yang sudah disediakan IzPack, kita juga bisa mendefinisikan kondisi tertentu dengan menggunakan tag <conditions> dan <condition>. Contohnya :

```
<conditions>
  <condition type="variable">
    <name>kondisi1</name>
    <value>>true</value>
  </condition>
  <condition type="variable">
    <name>kondisi2</name>
    <value>>false</value>
  </condition>
</conditions>
```

Tag GUI Preference <guiprefs>

Tag guiprefs digunakan untuk mendefinisikan ukuran jendela installer, tampilan lookandfeel, serta beberapa parameter tampilan . Terdapat beberapa attribute dalam tag ini :

- resizable : digunakan untuk mendefinisikan apakah jendela installer dapat diubah ukuranya. Nilainya adalah yes atau no
- width : lebar dari window
- height : tinggi window

di dalam tag guipref bisa ditambahkan tag laf untuk mendefinisikan lookandfeel installer, lookandfeel yang tersedia antara lain :

- kunststoff
- liquid
- metouia
- jgoodies
- substance

Tag laf memerlukan tag <os> untuk menentukan lookandfeel apa yang digunakan dalam OS tersebut, kalau tidak disebutkan maka lookandfeel default untuk setiap OS akan digunakan. Windows defaultnya adalah windows lookandfeel, Mac : Aqua dan *nix : metal.

Contoh penggunaanya :

```
<guiprefs height="600" resizable="yes" width="800">
  <laf name="metouia">
    <os family="unix" />
  </laf>
  <laf name="looks">
    <os family="windows" />
  </laf>
</guiprefs>
```

```
<param name="variant" value="extwin" />
</laf>
</guiprefs>
```

Tag Localization <locale>

Tag ini digunakan untuk mendaftarkan pilihan bahasa apa saja yang bisa digunakan dalam proses instalasi. Secara default IzPack menyediakan 10 bahasa. Dalam tag <locale> terdapat tag <langpack> untuk mendaftar satu per satu bahasa yang didukung installer anda. Berikut ini contoh penggunaanya :

```
<locale>
  <langpack iso3="eng"/>
  <langpack iso3="fra"/>
</locale>
```

Tag Resources <resources>

Beberapa panel, seperti panel Readme dan Licence memerlukan file text yang akan ditampilkan di masing-masing panel. File text tersebut harus didefinisikan dalam tag <resources> ini agar dikenali oleh panelnya. IzPack sayangnya tidak menyediakan feature untuk mengecek apakah resource yang didefinisikan benar-benar ada pada waktu membuat installer, sehingga terkadang IzPack mengeluarkan pesan bahwa installer berhasil dibuat tetapi ketika dijalankan, terdapat error yang membuat installer gagal.

Di dalam tag ini terdapat tag <res> yang mempunyai beberapa attribute, antara lain:

- src : path dimana resource berada.
- id : identifiers untuk menandai nama resourcenya
- parse : mempunyai nilai yes atau no. Kalau diset yes, IzPack akan memarsing resource ini dan melakukan pencarian apakah ada definisi variabel di dalamnya, kalau ada maka IzPack akan mengganti variabel tersebut dengan nilai sebenarnya.
- type : tipe dari resource. Gunakan attribute ini kalau resource berupa file text. Nilai dari attribute type ini antara lain: javaprop, xml, plain, java, shel, at, ant
- encoding : tipe encoding yang digunakan jika resource berupa file teks

Berikut ini adalah contoh penggunaanya :

```
<resources>
  <res id="LicencePanel.licence" src="Licence.txt"/>
  <res id="InfoPanel.info" src="Readme.txt"/>
</resources>
```

Panel <panels>

Tag ini sangat penting, karena mendefinisikan halaman apa saja yang akan ditampilkan dalam langkah-langkah instalasi aplikasi. IzPack menyediakan beberapa jenis panel yang bisa digunakan dalam installer. Untuk banyak kasus, panel-panel ini sudah mencukupi semua kebutuhan. IzPack juga mengizinkan kita mendefinisikan panel buatan kita sendiri, dalam buku ini kita tidak membahas topik panel hingga mendetail ke pembuatan custom panel, silahkan merujuk ke manual IzPack untuk topik tersebut.

Tag <panel> mempunyai beberapa attribute, antara lain :

- classname : nama class dari panel
- id : identitas atau nama dari panel yang sedang dibuat
- condition : kondisi yang bisa digunakan untuk menentukan apakah panel ini ditampilkan atau tidak
- jar : nama jar dimana class dari panel ini berada.

Berikut ini contoh penggunaanya :

```
<panels>
  <panel classname="HelloPanel"/>
  <panel classname="InfoPanel"/>
  <panel classname="LicencePanel"/>
  <panel classname="TargetPanel"/>
  <panel classname="PacksPanel"/>
  <panel classname="InstallPanel"/>
  <panel classname="FinishPanel"/>
</panels>
```

Tag <panels> mempunyai beberapa tag optional yang bisa digunakan, yang pertama adalah tag <help>, tag ini digunakan untuk mendefinisikan bantuan untuk panel tertentu. Tag <help> mempunyai attribute iso3 yang mendefinisikan bahasa bantuan, kalau misalnya installer mendukung 3 bahasa, sebaiknya tag <help> ini juga menyediakan bantuan untuk ketiga bahasa tersebut. Berikut ini contohnya :

```
<panel classname="HelloPanel">
  <help iso3="deu" src="HelloPanelHelp_deu.html" />
  <help iso3="eng" src="HelloPanelHelp_eng.html" />
</panel>
```

Tag kedua adalah <validator>, tag ini digunakan untuk memvalidasi inputan yang ada dalam panel. Misalnya kita akan membuat panel untuk mengkonfigurasi koneksi ke database, namakan saja panelnya KonfiguasiDatabase, kemudian di dalam panel ini ada text field untuk menginput jdbc url, username dan password. Kemudian kita ingin memvalidasi apakah inputan ini sudah benar dan koneksi ke database berhasil, maka kita buat misalnya JdbcConnectionValidator, class ini harus mengimplemntasi interface com.izforge.izpack.installer.DataValidator. Berikut ini contoh penggunaanya :

```
<panel classname="DatabasePanel">
  <validator classname="JdbcConnectionValidator"/>
</panel>
```

Panel juga mendukung feature action dengan tag <actions>. Tag <actions> ini bisa digunakan kalau kita ingin menjalankan suatu aksi tertentu. Tag <actions> mempunyai dua buah attribute, yaitu :

- classname : berisi nama class dari action yang akan dieksekusi
- stage : tahapan dimana action ini akan dieksekusi, nilai yang bisa digunakan untuk attribute stage ini adalah preconstruct, preactivate, prevalidate, postvalidate.

Berikut ini adalah contoh penggunaan actions

```
<panel id="koneksiDatabase" classname="DatabasePanel">
  <validator classname="JdbcConnectionValidator"/>
  <actions>
    <action stage="preconstruct" classname="JdbcConnectionPreConstruct"/>
    <action stage="preactivate" classname="JdbcConnectionPreActivate"/>
    <action stage="prevalidate" classname="JdbcConnectionPreValidate"/>
    <action stage="postvalidate" classname="JdbcConnectionPostValidate"/>
  </actions>
</panel>
```

Tag Packs <packs>

Tag ini sangat penting karena berfungsi untuk mendefinisikan file apa saja yang akan dicopy / diinstall. Di dalam tag <packs> terdapat beberapa tag lain, yaitu : <pack>, <refpack> dan <refpackset>. Tag <packs> mempunyai beberapa attribute, antara lain :

- name : nama dari packs, nama ini nanti akan muncul pada panel instalasi
- required : nilainya yes atau no. Kalau diisi yes berarti pada waktu panel instalasi menampilkan apa saja pack (modul) yang akan diinstall, pack ini tidak bisa di-uncheck.

Sebaliknya, kalau diisi no berarti pack ini bisa diuncheck dan IzPack tidak akan menginstall file-file yang didefinisikan dalam pack ini.

- **os** : bersifat optional. Attribute ini digunakan untuk menentukan target OS dari pack ini. Nilainya bisa windows, mac, unix dan seterusnya.
- **preselected** : attribute ini bersifat optional, digunakan untuk menentukan apakah pack ini secara default dipilih atau tidak.
- **loose** : attribute ini bisa digunakan jika ingin file-file dalam pack ini tidak diletakkan dalam jar hasil instalasi, hal ini dimaksudkan agar aplikasi bisa dijalankan langsung. Skenario yang mungkin terjadi misalnya: kita ingin aplikasi kita bisa dijalankan langsung dari CD installer, tetapi bisa juga diinstall langsung di PC pengguna. Kalau semua file diletakkan dalam jar maka aplikasi harus diinstall terlebih dulu baru bisa digunakan. Perlu diperhatikan juga relative path dalam aplikasi, karena aplikasi dijalankan langsung dari CD tentu saja berbeda path-nya dengan aplikasi yang diinstall di dalam PC.
- **id** : attribute ini digunakan sebagai identifiers dan bersifat unique. Selain itu attribute ini juga digunakan dalam proses penterjemahan pack ke dalam suatu bahasa tertentu.
- **packImgId** : digunakan kalau kita ingin menampilkan gambar untuk pack ini. Nilainya harus didefinisikan dalam tag `<resources>` dan harus bernilai sama dengan id yang ada dalam pack `<res>`
- **condition** : digunakan untuk mengetes kondisi tertentu sebelum pack ini diinstall. Kalau nilai dari kondisi salah maka pack tidak akan diinstall / ditampilkan dalam proses instalasi.
- **hidden** : dapat bernilai true atau false, digunakan untuk menampilkan atau menyembunyikan pack pada proses instalasi. Attribute ini cukup berguna kalau kita ingin menambahkan pack yang dieksekusi tetapi tidak ingin ditampilkan.

Tag `<refpack>` hanya mempunyai satu attribute saja yaitu `file`, yang berisi nilai relative path dari suatu XML konfigurasi yang dibuat sebelumnya. Jadi kita bisa membuat XML konfigurasi lebih dari satu kemudian memanggil XML tersebut dari XML konfigurasi utama dengan menggunakan `<refpack>` ini, konsepnya mirip sebuah class mengimport class lain. Hal ini memungkinkan XML konfigurasi dipecah menjadi beberapa file terpisah untuk memudahkan maintenance kalau ada beberapa orang yang bertanggung jawab untuk memelihara modulnya masing-masing. XML konfigurasi yang bisa digunakan oleh `<refpack>` hanya boleh berisi tag `<resources>` dan tag `<packs>` saja.

Tag `<refpackset>` digunakan kalau kita tidak ingin mendefinisikan `refpack` apa saja yang akan diikuti, tetapi kita cukup mendefinisikan sebuah folder yang akan discan oleh IzPack untuk menemukan semua XML konfigurasi yang lain. Tag ini mempunyai dua buah attribute, yaitu :

- **dir** : berisi direktori yang akan discan oleh IzPack untuk menemukan semua `refpack`
- **includes** : berisi pattern bagaimana menemukan `refpack`

Contoh penggunaan `refpackset` :

```
<refpackset dir="" includes="**/refpack.xml"/>
```

Sampai di sini semua tag yang kita perlukan untuk membuat XML konfigurasi IzPack sudah cukup diterangkan. Tidak semua tag xml dijelaskan dalam buku ini, kalau ingin membaca referensi lebih lengkap tentang sintaks dari XML konfigurasi ini, silahkan membaca dokumentasi IzPack yang disertakan dalam instalasi IzPack.

Panel-Panel yang Disediakan IzPack

Di bab ini kita akan membahas satu per satu Panel yang disediakan oleh IzPack, selain panel yang disediakan, kita juga bisa membuat panel sendiri, tetapi topik ini tidak dibahas dalam buku ini karena memerlukan penjelasan cukup panjang. Konsep panel dalam IzPack berbeda dengan panel dalam Swing, karena panel dalam IzPack didefinisikan menggunakan XML, tidak menggunakan kode Java untuk membuat layoutnya. Untuk kebutuhan yang umum, IzPack menyediakan cukup lengkap panel-panel yang dibutuhkan. Kita akan membahas panel-panel ini sebelum membahas XML konfigurasi untuk membuat installer POS.

>HelloPanel dan HTMLHelloPanel

Panel ini digunakan untuk menampilkan detail informasi aplikasi, semisal nama, versi, URL, pembuat aplikasi dan seterusnya. Sedangkan HTMLHelloPanel mengijinkan file html digunakan untuk menampilkan detail aplikasi.

CheckedHelloPanel

Fungsinya sama persis dengan HelloPanel, hanya saja panel ini akan melakukan pemeriksaan ke dalam registri windows akan dibuat untuk menandakan aplikasi tersebut sudah diinstall, selain itu juga dilakukan pemeriksaan pada registri yang sama apakah aplikasi sudah pernah diinstall sebelumnya. Kalau menggunakan panel ini jangan lupa untuk mendefinisikan registri apa yang akan digunakan.

InfoPanel dan HTMLInfoPanel

Panel ini digunakan untuk menampilkan README, info lebih lengkap tentang aplikasi yang akan diinstall. Text file yang akan ditampilkan harus didefinisikan di dalam resource dengan nama "InfoPanel.info". IzPack mengijinkan substitusi variabel di dalam file text-nya, sehingga kita bisa mendefinisikan variabel di dalam XML konfigurasi dan menggunakannya di dalam text file.

HTMLInfoPanel digunakan kalau ingin menampilkan file HTML. Gambar bisa dimasukkan dalam HTML, src menunjuk ke nama resource yang harus didefinisikan di dalam XML konfigurasi. Misalnya di dalam HTML ada `` maka harus dibuat resource dengan nama abcd yang menunjuk ke file gambar tertentu.

LicencePanel dan HTMLLicencePanel

Fungsi-nya sama dengan InfoPanel, hanya saja panel ini digunakan untuk menampilkan lisensi dari aplikasi. Text file yang akan ditampilkan harus didefinisikan sebagai resource dengan nama "LicencePanel.licence". Hal yang sama juga berlaku untuk HTMLLicencePanel.

PacksPanel

Panel ini akan menampilkan `<packs>` yang didefinisikan di dalam XML konfigurasi. Tidak perlu ada konfigurasi tambahan untuk mendefinisikan PacksPanel. Semua informasi yang akan ditampilkan diambil dari XML konfigurasi. PacksPanel mempunyai beberapa varian, antara lain ImgPacksPanel dan TreePacksPanel, perbedaanya hanya di cara menampilkan packs saja.

TargetPanel

Panel ini digunakan untuk memilih di mana folder instalasi. User mendefinisikan di mana aplikasi akan diinstall, atau bisa juga menggunakan tombol browse untuk memilih folder instalasi.

InstallPanel

Harus ada di setiap XML konfigurasi, panel ini akan melakukan proses instalasi sebenarnya, mulai dari unpack file, copy file hingga eksekusi semua perintah yang didefinisikan di dalam XML konfigurasi.

FinishPanel

Menampilkan rekap proses instalasi dan menampilkan keterangan kalau instalasi selesai dieksekusi.

Membuat Installer Aplikasi POS

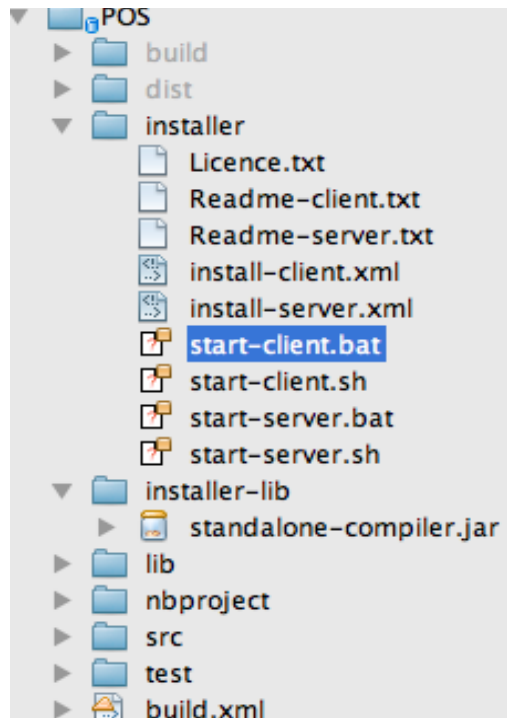
Sampai di sini penjelasan mengenai IzPack sudah cukup panjang lebar, sekarang kita akan membuat installer untuk aplikasi POS. Sebelum XML konfigurasi dibuat, ada beberapa langkah persiapan yang harus dilaksanakan, seperti membuat startup script berupa file bat (windows) dan file sh (*nix). Kemudian menyediakan file readme dan licence, langkah terakhir adalah menyiapkan ant script untuk digabungkan dengan build.xml dari netbeans, sehingga kita bisa membuat installer secara otomatis.

Installer yang akan dibuat ada 2 jenis, yaitu installer untuk POS Server dan installer untuk POS

Client. Kedua installer ini mempunyai file binary yang sama, perbedaannya ada di dalam class main yang dieksekusi. Kalau POS Server akan menjalankan class POSServer sedangkan POS Client akan menjalankan class Main.

Persiapan

Persiapan untuk membuat installer ada beberapa tahap, diawali dengan mempersiapkan struktur folder. Buat dua buah folder yaitu installer dan installer-lib. Kemudian copy file standalone-compiler.jar dari folder instalasi IzPack ke dalam installer-lib. Lihat gambar di bawah ini :



Folder installer digunakan untuk meletakkan semua file yang diperlukan untuk membuat installer, sedangkan installer-lib digunakan untuk meletakkan library IzPack. Library IzPack tidak diletakkan di dalam folder lib karena tidak digunakan ketika aplikasi berjalan, hanya digunakan untuk membuat installer, sehingga sebaiknya dipisahkan dari folder lib.

Seperti terlihat pada gambar di atas, kita perlu membuat Readme-client.txt untuk menampilkan keterangan apa saja yang perlu user baca ketika menginstall POS Client. Klik kanan foldernya pilih New > Other di dalam jendela new file pilih Other > Empty File, kemudian beri nama Readme-client.txt. Isi file tersebut dengan keterangan, misalnya seperti ini :

Aplikasi client POS. Jangan lupa setting server.properties agar IP dan port-nya sesuai dengan yang ada di serverContext.xml

Buat Readme-server.txt dengan langkah yang sama dan isi filenya dengan keterangan, misalnya seperti ini:

Aplikasi server POS. Jangan lupa setting server.properties agar IP dan port-nya sesuai dengan yang ada di serverContext.xml

Berikutnya kita akan membuat file Licence.txt untuk ditampilkan dalam LicencePanel. Langkah membuat file tersebut sama dengan dua file di atas, isi dengan pernyataan tentang lisensi aplikasi, misalnya seperti di bawah ini :

Aplikasi server dan client POS lisensi Apache Licence 2.0

Membuat Startup Script

Startup Script adalah sebuah file text yang berisi kode untuk menjalankan aplikasi Java, jadi

sifatnya executable. Kalau di windows bentuknya adalah file bat, kalau di *nix bentuknya file sh. Kita akan membuat empat buah startup script untuk menjalankan POS Server dan POS Client di windows dan *nix. Untuk menjalankan POS Server, class yang dipanggil adalah POSServer sedangkan POS Client yang dipanggil adalah Main. Startup script yang akan kita buat adalah :

- start-client.bat : digunakan menjalankan POS Client di windows

```
java -cp POS.jar;lib/* com.googlecode.projecttemplate.pos.Main
```

- start-cleint.sh : digunakan menjalankan POS Client di *nix

```
java -cp POS.jar:lib/* com.googlecode.projecttemplate.pos.Main
```

- start-server.bat : digunakan menjalankan POS Server di windows

```
java -cp POS.jar;lib/* com.googlecode.projecttemplate.pos.server.POSServer
```

- start-server.sh : digunakan menjalankan POS Server di *nix

```
java -cp POS.jar:lib/* com.googlecode.projecttemplate.pos.server.POSServer
```

Perbedaan antara file bat dan sh terletak pada tanda pemisah classpath, kalau di windows tanda pemisah classpath adalah ; (titik koma), sedangkan di *nix adalah : (titik dua).

Membuat XML Konfigurasi POS Client dan POS Server

Sebagian besar XML konfigurasi untuk POS Client dan POS Server tidak jauh berbeda. Proses instalasi kedua aplikasi ini juga sangat sederhana, copy POS.jar dan semua jar yang ada di dalam folder dist. Kemudian copy Startup Script ke folder instalasi. Sehingga struktur XML konfigurasi-nya nyaris sama, perbedaanya hanya di setting saja.

XML konfigurasi untuk POS Client adalah installer-client.xml, untuk membuat file ini klik kanan di folder installer kemudian pilih New > Other, di jendela New File pilih node XML kemudian di sebelah kanan pilih XML Document, beri nama installer client. Mari kita lihat XML konfigurasi dari POS Client kemudian kita bahas satu per satu bagianya.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>
<installation version="1.0">
  <info>
    <appname>POS Bangkit Cell</appname>
    <appversion>1.0</appversion>
    <authors>
      <author name="Ifnu Bima" email="ifnubima@gmail.com"/>
    </authors>
    <url>http://project-template.googlecode.com</url>
  </info>
  <guiprefs width="640" height="480" resizable="yes"/>
  <locale>
    <langpack iso3="eng"/>
  </locale>
  <resources>
    <res id="LicencePanel.licence" src="installer/Licence.txt"/>
    <res id="InfoPanel.info" src="installer/Readme-client.txt"/>
  </resources>
  <panels>
    <panel classname="HelloPanel"/>
    <panel classname="InfoPanel"/>
    <panel classname="LicencePanel"/>
    <panel classname="TargetPanel"/>
    <panel classname="PacksPanel"/>
    <panel classname="InstallPanel"/>
    <panel classname="FinishPanel"/>
  </panels>
  <packs>
    <pack name="POS Client" required="yes">
```

```

        <description>POS Client</description>
        <file src="installer/Readme-client.txt" targetdir="$INSTALL_PATH"/>
        <file src="installer/Licence.txt" targetdir="$INSTALL_PATH"/>
        <file src="dist/lib" targetdir="$INSTALL_PATH"/>
        <file src="dist/POS.jar" targetdir="$INSTALL_PATH"/>
        <file src="installer/start-client.bat"
            targetdir="$INSTALL_PATH" condition="izpack.windowsinstall"/>
        <file src="installer/start-client.sh"
            targetdir="$INSTALL_PATH" condition="!izpack.windowsinstall"/>
    </pack>
</packs>
</installation>

```

XML Konfigurasi di atas cukup sederhana, di bagian atas terdapat tag <info> yang berisi detail dari aplikasi. Kemudian di bawahnya terdapat <guiprefs> untuk mendefinisikan lebar dan tinggi jendela installer. Selanjutnya ada <locale> yang berisi bahasa untuk installer ini, yaitu bahasa Inggris (eng).

Tag <resources> berisi dua buah resource yaitu LicencePanel.licence yang berisi file Licence.txt, isi dari file ini nantinya akan ditampilkan dalam LicencePanel. Yang kedua adalah InfoPanel.info yang berisi file Readme-client.txt, file ini nantinya akan ditampilkan pada InfoPanel.

Tag <panels> berisi panel-panel yang akan ditampilkan dalam proses instalasi, ada tujuh buah panel. Semua panel di atas sudah diterangkan di bagian sebelumnya, kalau ingin melihat panel-panel lain silahkan rujuk ke manual IzPack.

Bagian terakhir adalah bagian terpenting dari XML konfigurasi ini, tag <packs> digunakan untuk mendaftarkan semua file-file yang akan disertakan dalam proses instalasi. File pertama adalah readme-client.txt yang diletakkan dalam folder instalasi (\$INSTALL_PATH). File kedua adalah Licence.txt yang juga akan diletakkan dalam folder instalasi. Berikutnya adalah satu buah folder yaitu lib, folder ini berisi semua file jar yang diperlukan oleh POS Client. Kalau kita memasukkan folder dalam src maka semua isi dalam folder tersebut akan ikut dicopy.

File berikutnya adalah POS.jar dimana semua kode yang kita tulis berada, letaknya ada di dalam folder dist. File berikutnya adalah startup script untuk windows, start.cleint.bat, file ini akan dicopy jika installer dijalankan di lingkungan windows. Terlihat dari adanya attribute condition="izpack.windowsinstall". File terakhir adalah startup script untuk *nix, file ini hanya dicopy jika installer dijalankan di lingkungan selain windows, terlihat dari attribute condition yang berisi "!izpack.windowsinstall".

XML konfigurasi untuk POS Server tidak jauh berbeda, seperti yang sudah kita bahas sebelumnya, perbedaannya hanya pada setting, sedangkan struktur XML konfigurasi sama persis. XML konfigurasi untuk POS Server diletakkan dalam installer-server.xml, cara membuat file ini juga sama persis dengan cara membuat file installer-client. Berikut ini adalah isi dari installer-server.xml :

```

<?xml version="1.0" encoding="UTF-8"?>
<installation version="1.0">
    <info>
        <appname>POS Bangkit Cell</appname>
        <appversion>1.0</appversion>
        <appsubpath>POS-server</appsubpath>
        <authors>
            <author name="ifnu bima" email="ifnubima@gmail.com"/>
        </authors>
        <url>http://project-template.googlecode.com</url>
    </info>
    <guiprefs width="640" height="480" resizable="yes"/>
    <locale>
        <langpack iso3="eng"/>
        <langpack iso3="fra"/>
    </locale>

```

```

        <langpack iso3="spa"/>
    </locale>
    <resources>
        <res id="LicencePanel.licence" src="installer/Licence.txt"/>
        <res id="InfoPanel.info" src="installer/Readme-server.txt"/>
    </resources>
    <panels>
        <panel classname="HelloPanel"/>
        <panel classname="InfoPanel"/>
        <panel classname="LicencePanel"/>
        <panel classname="TargetPanel"/>
        <panel classname="PacksPanel"/>
        <panel classname="InstallPanel"/>
        <panel classname="FinishPanel"/>
    </panels>
    <packs>
        <pack name="POS Server" required="yes">
            <description>POS Server</description>
            <file src="installer/Licence.txt" targetdir="$INSTALL_PATH"/>
            <file src="installer/Readme-server.txt" targetdir="$INSTALL_PATH"/>
            <file src="dist/lib" targetdir="$INSTALL_PATH"/>
            <file src="dist/POS.jar" targetdir="$INSTALL_PATH"/>
            <file src="installer/start-server.bat"
                targetdir="$INSTALL_PATH" condition="izpack.windowsinstall"/>
            <file src="installer/start-server.sh"
                targetdir="$INSTALL_PATH" condition="!izpack.windowsinstall"/>
        </pack>
    </packs>
</installation>

```

Setelah selesai membuat XML konfigurasi langkah terakhir adalah membuat ant script di dalam build.xml agar installer dibuat secara otomatis.

Membuat Ant Script Installer

Pada bagian sebelumnya kita mencoba membuat installer dengan sample menggunakan command line, nah sekarang kita akan menjalankan install-server.xml dan install-cleint.xml menggunakan ant script. IzPack menyediakan ant task untuk menjalankan XML konfigurasi dari ant.

Buka file build.xml di NetBeans, kemudian tambahkan script ant berikut ini di dalam tag <project>, sama persis dengan yang kita lakukan pada waktu membuat ant script untuk mengcompile jasper report :

```

<taskdef name="IzPack"
    classname="com.izforge.izpack.ant.IzPackTask">
    <classpath>
        <fileset dir="./installer-lib">
            <include name="**/*.jar"/>
        </fileset>
    </classpath>
</taskdef>
<property name="base.dir"
    value="/Users/ifnu/NetBeansProjects/project-template/java-desktop-book/code"/>
<target name="-post-jar">
    <IzPack input="{base.dir}/installer/install-client.xml"
        output="{base.dir}/dist/POS-client-installer.jar"
        installerType="standard"
        basedir="{base.dir}"/>
    <IzPack input="{base.dir}/installer/install-server.xml"
        output="{base.dir}/dist/POS-server-installer.jar"

```

```
        installerType="standard"
        basedir="${base.dir}"/>
</target>
```

Bagian pertama dari ant script tersebut adalah <taskdef> yang digunakan untuk mendefinisikan task baru dari IzPack. Namanya adalah "IzPack", di dalamnya perlu didefinisikan nama class dari taskdef dan classpath dimana class tersebut bisa ditemukan.

Kemudian ada satu property yang bernama "base.dir" property ini digunakan untuk mendefinisikan di mana folder kerja kita, isinya berupa absolute path dari folder kerja. Kemudian ada target dengan nama "-post-jar", target ini adalah target spesial yang dikenali oleh NetBeans, target dengan nama ini akan dijalankan setelah proses kompilasi dan pembuatan jar dari project POS ini selesai dilaksanakan. Misalnya kita memilih menu Clean and Build dari project, maka target "-post-jar" ini akan dieksekusi.

Isi target "-post-jar" adalah dua buah anttask IzPack, ada tiga buah attribute dari anttask IzPack, yaitu :

- input berisi file XML konfigurasi: install-client.xml atau install-server.xml
- output berisi nama file installer yang dihasilkan oleh IzPack
- installerType kita isi dengan standard
- baseDir berisi folder kerja kita yang sudah didefinisikan dalam property base.dir

Nah sekarang silahkan coba jalankan Clean and Build dari project POS, kemudian silahkan coba eksekusi file installer yang dihasilkan oleh IzPack: POS-client-installer.jar dan POS-server-installer.jar. Setelah proses instalasi dari kedua installer itu berhasil dilaksanakan, coba jalankan startup script dan pastikan aplikasi berjalan sesuai yang diinginkan

Proses instalasi ini masih mempunyai banyak kelemahan, misalnya kita harus melakukan chmod ke startup script untuk *nix agar file sh-nya dapat dieksekusi. Kemudian proses instalasi database juga masih harus dilakukan secara manual dengan menginstall server RDBMS-nya, membuat database / Schema, membuat user database dan terakhir kemudian menjalankan DDL untuk membuat table-table yang diperlukan aplikasi.

Sampai di sini kita sudah mempelajari ilmu dasar dari pembuatan aplikasi Java desktop. Latihan dan hands-on langsung sangat diperlukan untuk mengetahui berbagai macam pemecahan masalah yang dihadapi ketika membuat aplikasi. Tidak ada cara cepat untuk menguasai semua teknik tersebut, dibutuhkan waktu cukup lama agar semua teknik dikuasai dengan baik. Semoga buku ini bisa membuka wawasan anda dalam membuat aplikasi Java Desktop.

Selamat belajar.

Penutup

Dengan ilmu yang sudah diperoleh dari buku ini, anda sudah bisa mulai untuk membuat program Java desktop sederhana. Pada awalnya pasti terasa sulit, sikapi dengan pantang menyerah dan selalu cari cara yang lebih baik dalam membuat aplikasi.

Langkah selanjutnya anda bisa mulai aktif bertanya atau menjawab hal-hal yang berhubungan dengan Java. Media yang bisa digunakan banyak sekali, bisa forum, milis atau diskusi dengan teman. Ini cara terbaik untuk mengetes apakah pemahaman anda mengenai Java sudah cukup lengkap atau anda masih perlu belajar lebih banyak lagi.

Setelah yakin dengan kemampuan anda, berfikirilah untuk mengambil sertifikasi profesional Java. Pelatihan untuk persiapan sertifikasi Java banyak tersedia di lembaga pelatihan. Kalau anda merasa terlalu berat secara finansial mengambil kursus persiapan sertifikasi, berlatihlah sendiri menggunakan materi yang banyak tersedia di internet, misalnya javaranch.com.

Cara belajar Java yang paling efektif adalah dengan melibatkan diri dalam project berbasis Java. Jika di perusahaan anda tidak memungkinkan, di luar sana banyak sekali project opensource yang memerlukan bantuan anda. Berkunjuglah ke website-website open source project hosting seperti sourceforge.net atau dev.java.net

Learn, Try dan Teach adalah formula untuk meningkatkan pengetahuan anda tentang Java. Jika sudah belajar dan sukses mencoba, tularkan ilmu anda pada orang disekeliling anda, dijamin ilmunya bakalan bertambah berlipat-lipat.

Referensi dan Bacaan Lebih Lanjut

1. Java Tutorial : <http://download.oracle.com/javase/tutorial>
2. Java Documentation : <http://download.oracle.com/javase/6/docs/>
3. Netbeans Indonesia : <http://groups.yahoo.com/group/Netbeans-indonesia/>
4. Java User Group Indonesia : <http://groups.yahoo.com/group/jug-indonesia/>
5. Java Design Pattern : <http://www.javacamp.org/designPattern/>
6. Java Desktop : <http://www.javadesktop.org>
7. JGoodies : <http://www.jgoodies.com>
8. SwingX : <http://www.swinglabs.org>
9. Java Persistence : http://en.wikibooks.org/wiki/Java_Persistence
10. BigDecimal : http://blogs.sun.com/CoreJavaTechTips/entry/the_need_for_bigdecimal
11. Joda Time : <http://www.ibm.com/developerworks/java/library/j-jodatime.html>
12. Joda Time : java.ociweb.com/mark/programming/JodaTime.pdf
13. InputStream : <http://tutorials.jenkov.com/java-io/inputstream.html>
14. SCJP Sun Certified Programmer for Java 6 Exam 310-065. Katherine Sierra. Bert Bates.

