# Java Fundamental (JSE)

# Materi

- Pengenalan Java
  - Sejarah, Karakteristik dan Kelebihan
  - Install
  - Java Byte Code
  - Write, Compile, and Run
  - Java API

- Sintaks Bahasa Java
  - Variabel dan Tipe Data
  - Identifier dan Naming Convention
  - Operator
  - Kontrol Eksekusi Program

# Materi

- Object-Oriented Programming
  - Konsep OOP
  - Inheritance
  - Polymorphism
  - Encapsulation
  - Property dan Method
  - Method Overloading

- Constructor
- ‘this’ keyword
- ‘static’ keyword
- Modifier
- Method overriding
- ‘final’ keyword
- Inner class
- Class Abstract

- Interface
- Package
- Exception handling

- Mengakses Database (JDBC)
- Netbeans Matisse

# Sejarah Java

- Mulai dikembangkan sejak 1991 oleh "*Stealth Project*" bertujuan untuk menciptakan suatu sistem *software* yang mampu berjalan pada alat-alat elektronik (*small devices*).
- *James Gosling* berkonsentrasi pada ide pembuatan bahasa pemrograman.
- Juni 1991, muncullah bahasa interpreter "Oak" yang menjadi cikal bakal dari Java.

- Java secara resmi diperkenalkan oleh SUN pada dunia pada tanggal 23 Mei 1995 bersama dengan browser HotJava.
- Javapun merambah ke dunia web/internet.
- Saat ini Java terbagi menjadi 3 teknologi:
  - JSE – untuk aplikasi desktop
  - JME – untuk aplikasi small device (HP, PDA,etc).
  - JEE – untuk aplikasi *enterprise* (web, Distribute Programming: web service/SOAP, RMI, EJB, JNDI, etc.

# Kelebihan dan Karakteristik

- Dari segi sintaks bahasa mirip dengan C/C++
- Karakteristik Java sesuai dengan *white paper* dari Sun:
  - Berorientasi Object (OOP)
  - Robust, java mendorong pemograman yang bebas dari kesalahan dengan bersifat *strongly typed.*

- Portable, dapat berjalan pada SO manapun
- Multithreading, sudah terintegrasi dengan pemrograman multithreading.
- Dinamis, program Java dapat melakukan sesuatu tindakan yang ditentukan pada saat eksekusi program dan bukan pada saat kompilasi

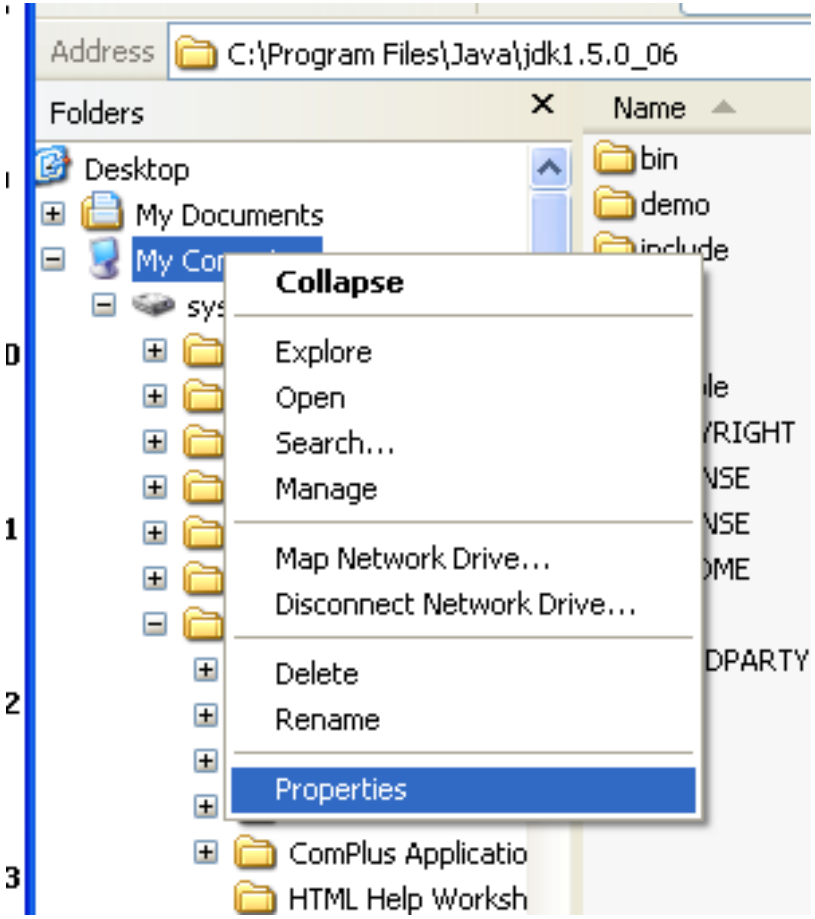
- Sederhana, Java menggunakan bahasa yang sederhana dan mudah dipelajari
- Terdistribusi, Java didesain untuk berjalan pada lingkungan yang terdistribusi seperti halnya internet
- Aman, aplikasi yang dibuat dengan bahasa java lebih dapat dijamin keamanannya terutama untuk aplikasi internet.

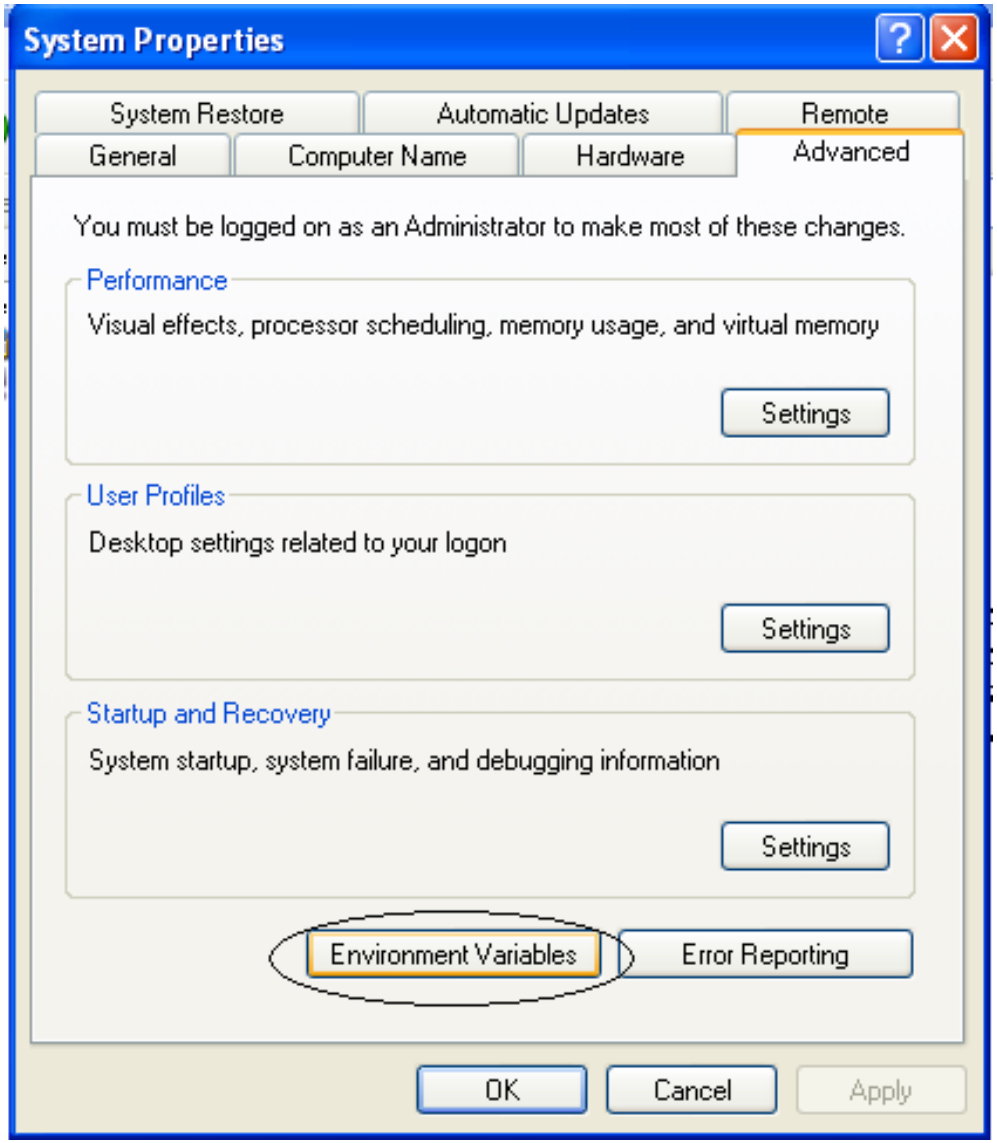
- Netral secara arsitektur, Java tidak terikat pada suatu mesin atau sistem operasi tertentu.
- Interpreted, aplikasi Java bisa dieksekusi pada platform yang berbeda-beda karena melakukan interpretasi pada bytecode
- Berkinerja Tinggi, bytecode Java telah teroptimasi dengan baik sehingga eksekusi program dapat dilakukan secara cepat

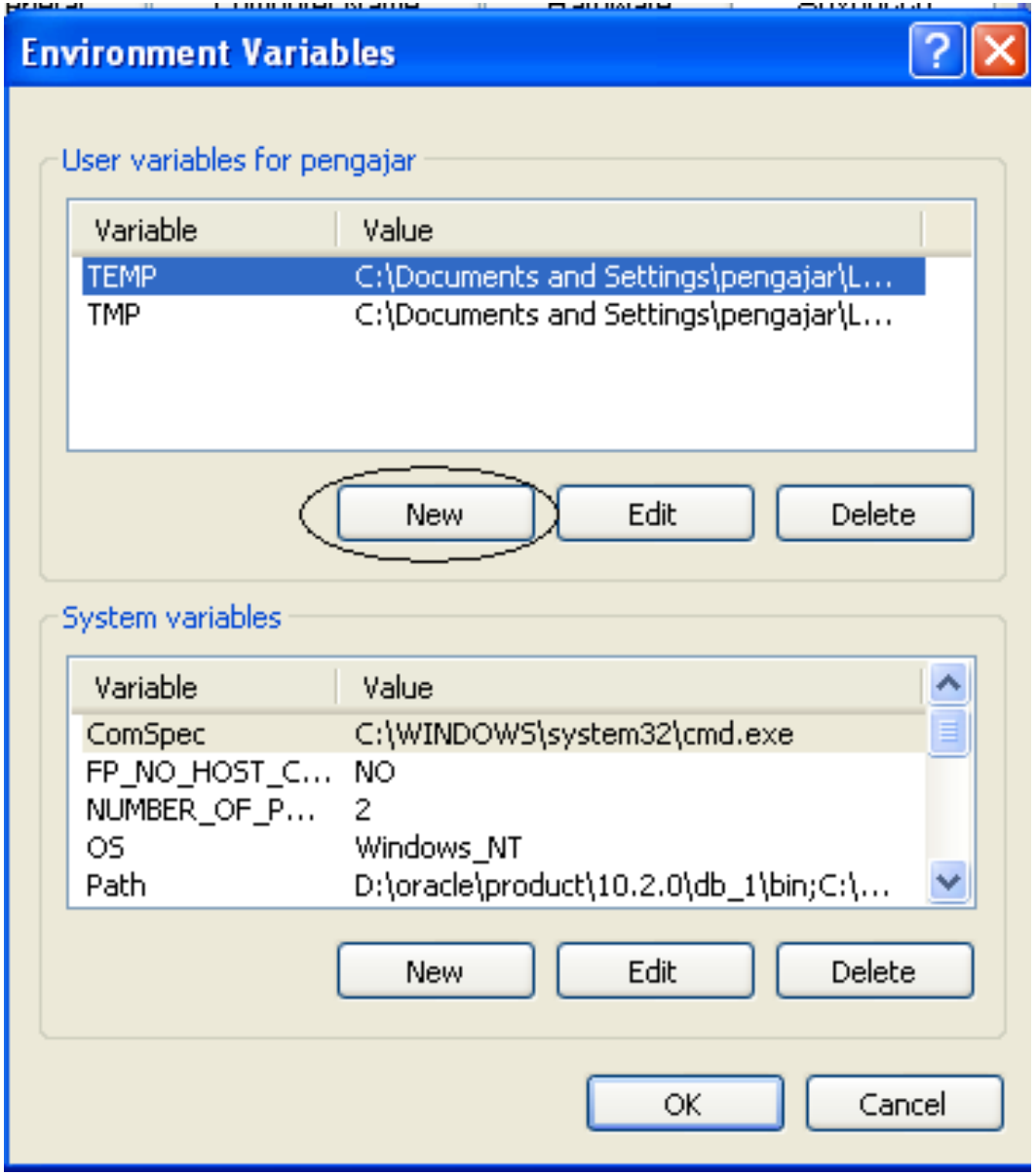
# Menginstal Java

- Apa yang dibutuhkan:
  - JDK : bila kita ingin membuat aplikasi menggunakan Java.
  - JRE : bila kita hanya ingin menjalankan aplikasi Java yang sudah ada.
  - Download : http://java.sun.com GRATISSS

- Setting Environment
  - JAVA_HOME = {dir istalasi}
  - PATH = {dir instalasi}/bin
  - CLASSPATH = .;{dir istalasi}/lib/tools.jar


Address
C:\Program Files\Java\jdk1.5.0_06
Folders
Name
Desktop
My Documents
bin
demo
Collapse
Explore
Open
Search...
Manage
Map Network Drive...
Disconnect Network Drive...
Delete
Rename
Properties
ComPlus Applicatio
HTML Help Worksh


System Properties
System Restore
Automatic Updates
Remote
General
Computer Name
Hardware
Advanced
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings
User Profiles
Desktop settings related to your logon
Settings
Startup and Recovery
System startup, system failure, and debugging information
Settings
Environment Variables
Error Reporting
OK
Cancel
Apply


Environment Variables
User variables for pengajar
Variable
Value
TEMP
C:\Documents and Settings\pengajar\L...
TMP
C:\Documents and Settings\pengajar\L...
New
Edit
Delete
System variables
Variable
Value
ComSpec
C:\WINDOWS\system32\cmd.exe
FP_NO_HOST_C...
NO
NUMBER_OF_P...
2
OS
Windows_NT
Path
D:\oracle\product\10.2.0\db_1\bin;C:\...
New
Edit
Delete
OK
Cancel

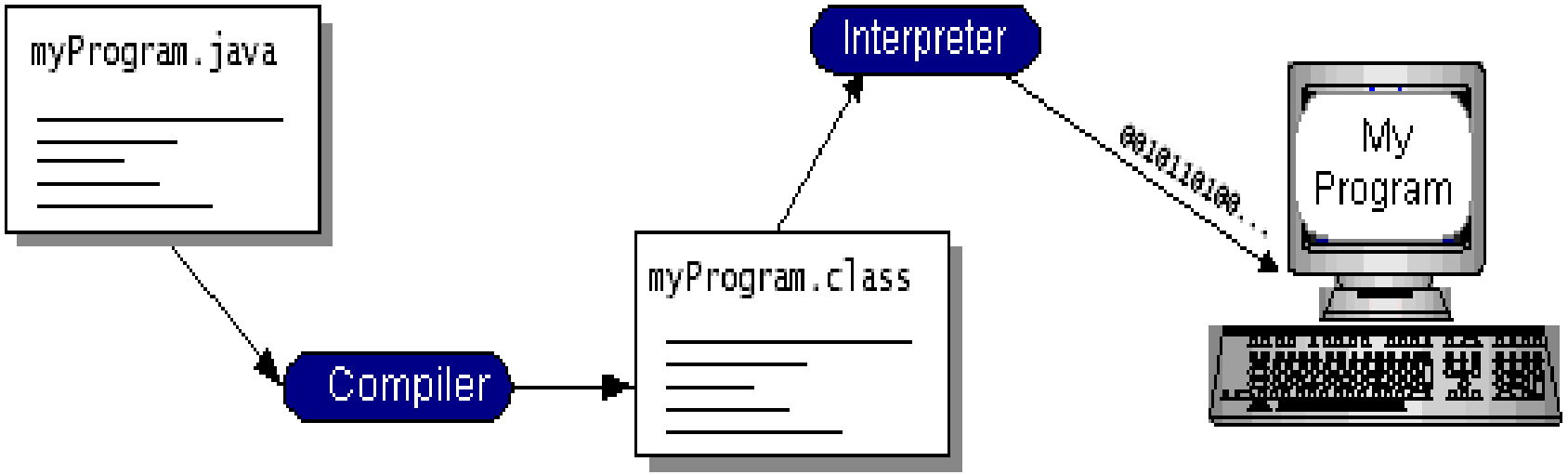
# NOTE:

- JAVA_HOME, untuk java mengenalkan ke Windows
- PATH, untuk mengarahkan path system Windows agar kita bisa menggunakan Java dari console.
- CLASSPATH, digunakan Java untuk mencari library yang dibutuhkan untuk compile atau saat runtime.

# Java Bytecode

- Java menggunakan compiler sekaligus Interpreter agar dapat berjalan pada platform yang berbeda.
- Java compiler mengcompile source code menjadi Java *Bytecode.*
- *Java Bytecode* merupakan instruksi mesin yang tidak spesifik terhadap suatu sistem mesin.

- Bytecode inilah yang akan dieksekusi oleh JVM yang ada didalam JRE.

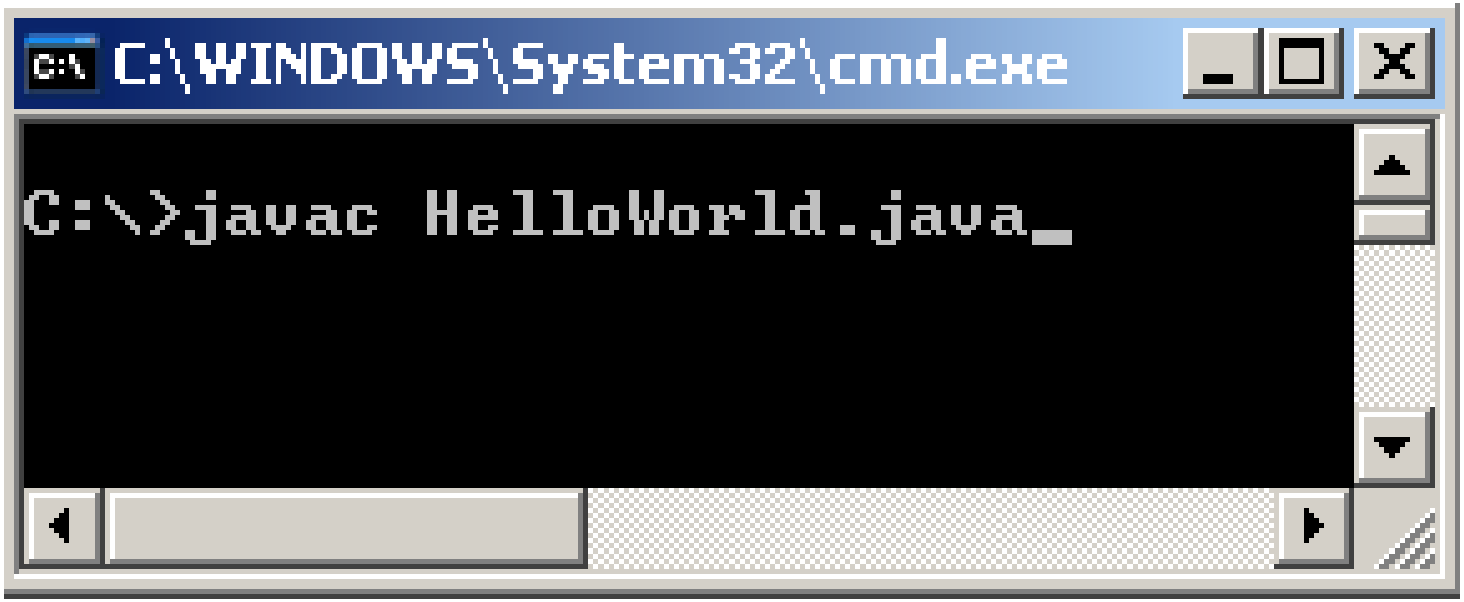


# Write, Compile and Run

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

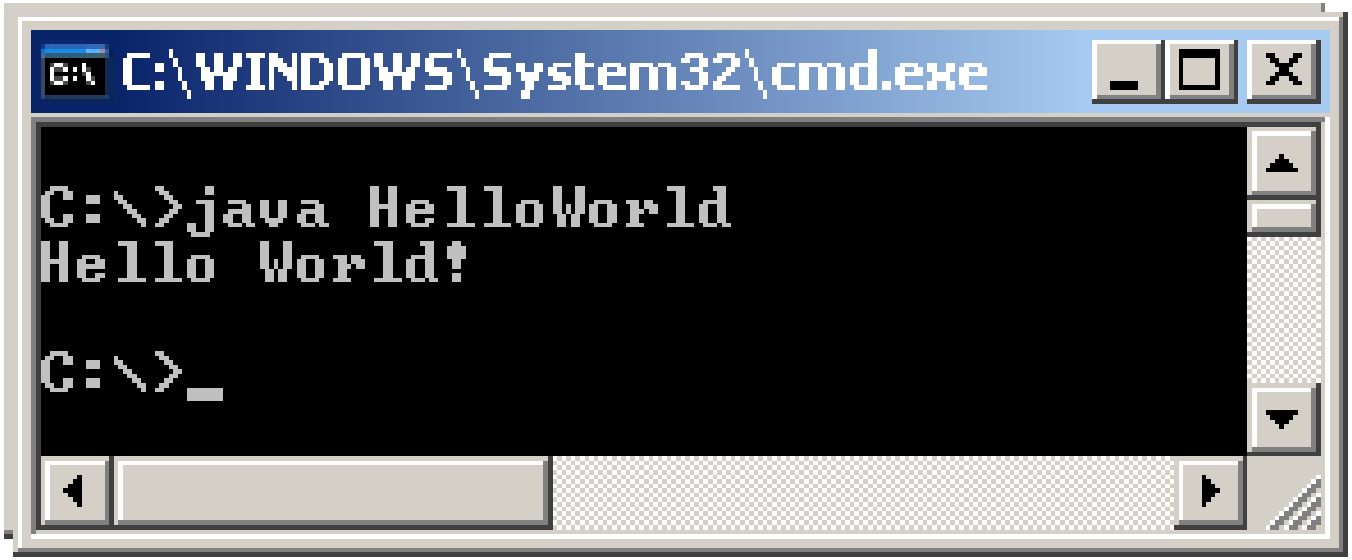
Simpan dengan nama HelloWorld.java

# Compile




Akan menghasilkan file .class → bytecode

# Run



# Memberi Komentar

- double slash ( // ) untuk komentar satu baris
- /* dan */ untuk komentar satu baris atau lebih

# Contoh:

```
public class HelloWorld{
    //baris ini tidak akan dieksekusi
    public static void main(String args[]){
         System.out.println("Selamat Belajar Java");
    }
    /* Baris ini adalah komentar
        yang lebih dari satu baris
        juga tidak akan dieksekusi
    */
}
```

# Keywords / Reserved Words

- Keywords / Reserved Words adalah kata-kata yang memiliki arti yang spesifik bagi kompiler dan tak bisa dipakai untuk kegunaan lainnya pada program. Perlu diingat bahwa java adalah ***case-sensitive***.

## Java Keywords and Reserved Words

| | | | | | | |
|---|---|---|---|---|---|---|
| abstract | class | false | import | package | super | try |
| assert | const | final | instanceof | private | switch | void |
| boolean | continue | finally | int | protected | synchronized | volatile |
| break | default | float | interface | public | this | while |
| byte | do | for | long | return | throw | |
| case | double | goto | native | short | throws | |
| catch | else | if | new | static | transient | |
| char | extends | implements | null | strictfp | true | |

# Beberapa konsep

- Modifier
  - *Modifiers* digunakan untuk menentukan penggunaan dari suatu data,methods dan *class.* Contoh *modifiers* adalah public, static, private, final, abstract dan protected.

- Statements
  - *Statements* merupakan baris perintah atau kumpulan perintah. Setiap *statements* pada java selalu diakhiri dengan titik koma ( ; ).

- Blocks
  - *Blocks* digunakan untuk membentuk suatu grup *statements*. *Blocks* diawali dengan kurung kurawal buka ( { ) dan dan diakhiri dgn kurung kurawal tutup ( } ). Blocks dapat digunakan secara *nested* (*blocks* didalam *blocks*).

- Classes
  - *Classes* merupakan inti dari program Java. Suatu *class* merupakan “blueprint” untuk menciptakan suatu object.

- Methods
  - Methods merupakan kumpulan *statements* yang berfungsi melakukan tugas tertentu di dalam program.

# Identifiers

- Merupakan penamaan pada pemograman untuk variabel, konstanta, method, class dan package
- **INGAT** : Java merupakan bahasa pemograman yang *case-sensitive* (membedakan huruf besar dan kecil)

- Aturan penamaan *identifiers* dalam java adalah:
  - Dapat dimulai dengan huruf, underscore( _ ), atau tanda $
  - Tidak boleh menggunakan simbol operator seprti : + - * / dan lain-lain.
  - Tidak boleh menggunakan *key word*
  - Panjang identifier boleh berapa saja

# Naming Convention

- Selain mengikuti aturan penulisan indentifiers di atas, penulisan program pada Java juga memiliki bentuk – bentuk tertentu yang sudah dibakukan dan diakui secara internasional.
- Bentuk penulisan program java ini biasa disebut *Java Naming Convention*.

- Penamaan Class
  - Huruf pertama setiap kata harus huruf besar. Contoh:
    - HelloWorld
    - Employee
    - BankAccount

- Penamaan Method
  - Huruf pertama setiap kata harus huruf besar, kecuali kata pertama. Contoh:
    - getEmployeeName()
    - setSpeedLimit()
    - accelerate()

- Penamaan Field atau variabel
  - Huruf pertama setiap kata harus huruf besar, kecuali kata pertama. Contoh:
    - employeeName
    - employeeAccountNumber
    - address

- Penamaan Konstatanta / Constant
  - Semua huruf harus huruf besar. Apabila lebih dari satu kata, gunakan *underscore* ( _ ) sebagai pemisah. Contoh:
    - PI
    - MIN_RATE
    - MAX_HEIGHT

# Variabel, Konstanta dan Tipe Data

- **Variabel**
  - Variabel merupakan lokasi penyimpanan yang ada di memori. Setiap variabel memiliki kemampuan menyimpan suatu informasi sesuai dengan tipe data yang didekalrasikan untuk variabel tersebut saja

- Sintaks pendekalrasian variabel secara umum adalah sebagai berikut:
  - `tipe-data nama-variabel`
- Tipe-data meliputi semua tipe data yang dikenal oleh Java, sedangkan nama-variabel adalah *identifier* yang digunakan untuk merujuk ke variabel tersebut di dalam program.

- Contoh kode:

```
int counter;
```

- Kode di atas mendeklarasikan suatu variabel yang bernama **counter** dengan tipe data **int**

# Scope Variabel

- Scope dari variabel dapat dibedakan menjadi dua bagian yaitu : variabel yang dideklarasikan di dalam blok class dan variabel yang dideklarasikan di dalam blok kode.

- Variabel yang dideklarasikan pada blok class, akan dikenali oleh seluruh bagian class. Variabel ini juga dapat diakses dari luar class tetapi tergantung dari *access specifier*nya / modifier. Mengenai hal *access spesifier* akan dibahas pada pertemuan selanjutnya.

- Variabel yang dideklararsikan di dalam blok kode tertentu hanya akan dikenali di dalam blok kode tersebut.

Contoh:

```
public class Scope{
   static int x=10;
   public static void main(String[] args){
      int a = 5;
         // variabel x dapat diakses
         System.out.println("Nilai x="+x);
System.out.println("Nilai a="+a);
         //nested blok
         {
            int b = 8;
            // variabel x masih dikenali
            System.out.println("Nilai x="+x);
            // variabel a juga dikenali
            System.out.println("Nilai a="+a);
            System.out.println("Nilai b="+b);

         }
         //variabel tidak dikenali
         System.out.println("Nilai b="+b);
   }
}
```

## Konstanta

- Konstanta merupakan data yang tidak berubah nilainya selama program berjalan. Pendekalrasian konstanta menggunakan bentuk:

```
final tipe-data NAMA_CONSTANTA = value;
```

Contoh:

```
final int MAX_WIDTH = 100;
```

# Tipe Data

- Tipe data diperlukan agar compiler tahu operasi apa yang valid dan seberapa banyak memori yang diperlukan oleh sebuah nilai yang akan disimpan atau dioperasikan.
- Di dalam Java terdapat tiga tipe data yaitu tipe **data primitif**, **tipe data referensi** dan **array**.

# Tipe Data Primitif

| Tipe Data | Besar (bits) | Jangkauan |
|---|---|---|
| long | 64 | $-2^{63}$ s/d $2^{63} - 1$ |
| int | 32 | $-2^{31}$ s/d $2^{31} - 1$ |
| short | 16 | $-2^{15}$ s/d $2^{15} - 1$ |
| byte | 8 | $-2^{7}$ s/d $2^{7} - 1$ |
| double | 64 | Negatif:<br>-1.79769313486231570E+308 s/d<br>-4.94065645841246544E-324<br>Positif:<br>4.94065645841246544E-324 s/d<br>1.79769313486231570E+308 |
| float | 32 | Negatif:<br>-3.40282346638528860E+38 s/d<br>-1.40129846432481707E-45<br>Positif:<br>1.40129846432481707E-45 s/d<br>3.40282346638528860E+38 |
| char | 16 | ‘\u0000’ s/d ‘\uFFFF’<br>(0 s/d 65535) |
| boolean | 8 | true atau false |

- Ke delapan tipe data di atas dapat dikelompokan menjadi empat kolompok:
  - Integer, merupakan tipe data bilangan bulat yang terdiri atas **byte**,**short**, **int**, dan **long**.
  - Floating-Point, merupakan tipe data bilangan pecahan yang terdiri ata **float** dan **double**

- Karakter, mewakili simbol dari sebuah karakter yang terdiri ata **char**.
- Boolean, merupakan tipe data yang menunjukan nilai true atau false, yang terdiri atas **boolean**

# Tipe Data Referensi

- Tipe data referensi digunakan untuk memegang referensi dari suatu object (instance dari class). Pendeklarasian tipe data ini sama dengan tipe data primitif, namun penggunaanya agak sedikit berbeda.

## Contoh:

```
public class Segitiga{
   int alas;
   int tinggi;
   public static void main(String args[]){
         /*
         Pendeklarasian variabel dengan tipe data
         class Segitiga
         */
         Segitiga s3;

         /*
         Instantiate class Segitiga menjadi object
         */
         s3 = new Segitiga();
         /*
         Setelah proses instantiate ini, anda dapat
         mengakses object Segitiga melalui variabel
         s3
         */
         s3.alas=10;
         s3.tinggi=2;
         System.out.println("Alas = "+s3.alas);
         System.out.println("Tinggi = "+s3.tinggi);
   }
}
```

# Nilai Literal

- Literal adalah suatu nilai yang terlihat secara eksplisit. Perhatikan contoh berikut ini:

```
int index = 10;
```

Pada kode di atas, angka 10 merupakan nilai literal.

# Literal Integer

- Semua bilangan bulat akan dianggap bertipe int oleh Java.
- Nilai literal integer bisa ditulis dengan bilangan basis 10 (desimal), basis 8 (oktal) dan basis 16 (heksadesimal).
- Untuk basis 8 tambahkan angka nol (0) di depan. Contoh : 01,034, 0425, …
- Untuk basis 16 tambahkan 0x (nol dan x), contoh: 0x1, 0x234, 0xA, …

- Untuk menuliskan bilangan bulat yang sangat besar dimana hanya tipe data long (64 bit) yang dapat menampungnya, anda perlu menambahkan karakter L atau l untuk memberi tahu Java bahwa nilai literal tersebut adalah bertipe data long. Contoh: 9234857892347523L

# Literal Floating-Point

- Bilangan *floating-Point* merupakan bilangan desimal yang berupa pecahan.
- Di dalam Java bilangan desimal secara default bertipe double.
- Agar bisa ditampung pada tipe float tambahkan karakter F atau f diakhir literal desimal. Contoh: `375834753897345.48374F`

- Bilangan ini dapat ditulis menggunakan notasi standar biasa atau menggunakan notasi ilmiah (*scientific notation*)
- Notasi standar menggunakan titik untuk menandakan pecahan. Contohnya : 10.2, 11.3, …
- Notasi ilmiah dapat menggunakan lambang E**n** (eksponensial). Contohnya:

  7.02345E3 = 7.02345 x $10^{3}$ = 7023.45

# Literal Boolean

- Untuk tipe data **boolean**, Java hanya mengenal dua nilai literal yaitu **true** dan **false**. Contoh penggunaan :

```
boolean isEmpty = true;
```

- Defaultnya adalah **false**.

# Literal Karakter

- Java didesain untuk menjadi bahasa yang portable dan universal. Karena itu, Java mendukung penggunaan *Unicode Characters,* yang mencakup hampir semua karakter yang dikenal oleh manusia dalam berbagai bahasa seperti *Arabic*, *Latin*, *Greek*, dan lain – lain.

- Untuk itu lebar data tipe data **char** adalah 16 bit agar dapat merepresentasikan semua karakter yang ada.

# Daftar *escape sequence* yang dikenal oleh Java

| **Escape Sequence** | **Keterangan** |
|---|---|
| \’ | Tanda petik satu |
| \” | Tanda petik dua |
| \\ | Backslash |
| \r | Carriage return |
| \n | Pindah baris |
| \f | Form feed |
| \t | Tab |
| \b | backspace |

# Literal String

- Literal untuk String dalam Java ditulis di antara tanda petik ganda, dan sama halnya seperti literal untuk karakter, escape sequence juga dapat digunakan di sini.

# Array

- Array adalah sebuah set variabel yang diberi nama tertentu yang memiliki tipe data yang sama.  Tiap variabel di dalam array disebut elemen, dimana tiap elemen memiliki indeks dengan tipe integer

# Array Satu Dimensi

- `int[] nilaiSiswa; //cara ini lebih dianjurkan`
- `int nilaiSiswa[];`
- `float[] jumlahPanen;`

- Array yang sudah dideklarasikan perlu didefinisikan, seperti contoh berikut

```
nilaiSiswa = new int[10];
jumlahPanen = new float[100];
```

- Untuk memberikan nilai kepada sebuah elemen array caranya dengan menyebutkan nama array yang diikuti indeks dan nilai yang diberikan, seperti contoh berikut:

```
nilaiSiswa[0] = 8;
jumlahPanen[2] = 223.66;
```

# Contoh:

```
public class ArrayAverage{
   public static void main(String[] args){
        int[] values = {1, 3, 5, 7, 9};
        int aveValue;
        aveValue = getAverage(values);
        System.out.println("Jumlah total: " + aveValue);
   }
   public static int getAverage(int[] intArray){
        int sum = 0;         // variabel lokal
        for(int x = 0; x < intArray.length; x++)
          sum = sum + intArray[x];
        return sum / intArray.length;
   }
}
```

# Array Multidimensi

- Cara pendeklarasian array multidimensi ini pada dasarnya sama dengan array satu dimensi di mana anda cukup menambahkan `[]` sesuai dengan dimensi yang anda inginkan.

# Contoh:

```
int[][] arr2;              //array 2 dimensi
int[][][] arr3;            //array 3 dimensi
int[][][][] arr4;          //array 4 dimensi

int[][] arr2 = new int[3][4];
```

Kode di atas akan mengalokasikan memori untuk menampung nilai tipe data int sebanyak 3*4.

# Konversi Tipe Data dan Casting

- Java akan melakukan konversi tipe data secara otomatis jika kedua tipe data tersebut kompatibel. Misalnya dari tipe data **int** ke tipe data **long**. Contoh:

```
int data1 = 10;
long data2 = data1;
```

- Tidak semua tipe data kompatibel satu dengan yang lainnya, misalnya tipe float dengan tipe data int. Float merupakan tipe data pecahan sedangkan int adalah tipe data bilangan bulat. Hal yang sama juga terjadi apabila kita mengkonversi tipe data yang lebih besar ke tipe data yang lebih kecil ukurannya. Misalnya dari tipe data int ke tipe data short.

- Konversi tipe data yang tidak kompatibel dapat kita lakukan secara eksplisit yaitu dengan *casiting*. Sintaks kode:

```
(target tipe-data) nilai;
```

- `float data1 = 10.2F;`
- `int data2 = (int)data1;` **//casting dari float ke int**
- `int data3 = 257;`
- `byte data4 = (byte)data3.` **//casting dari int ke byte**

- Yang perlu diperhatikan di sini adalah jika Anda mengubah tipe data yang berbeda jenis, seperti dari tipe data pecahan ke tipe data bilangan bulat maka akan terjadi pemotongan. Pada contoh di atas maka data2 akan bernilai 10.

- Untuk tipe data yang lebih kecil jika digunakan untuk menampung tipe data yang lebih besar dari daya tampungnya maka yang akan tertampung adalah nilai modulusnya(sisa bagi).

- Pada contoh di atas, tipe data variabel data4 adalah byte (jumlah maksimum yang dapat ditampung oleh byte adalah 256), sedangkan nilai yang hendak ditampung adalah 257. Dari perhitungan 257/256 dipeoleh modulus=1 maka data4 akan bernilai 1

- Konveri tipe data terkadang dapat menimbulkan kebingungan dan kesalahan logic yang sulit dicari penyebabnya. Perhatikan contoh berikut ini:

```
int nilai = 26;
double hasil = nilai/4;
```

- Jika kode di atas di eksekusi maka isi dari variabel hasil adalah 6.0 dan bukan 6.5

- Ini disebabkan karena variabel *nilai* bertipe int sehingga hasil operasi *nilai*/4 adalah juga bertipe data int yang hanya dapat menampung bilangan bulat saja. Sehingga hasil yang didapat dari nilai/4 adalah 6 dan bukan 6.5.

- Untuk menghindari kejadian seperti ini, sebaiknya kita menggunakan tipe data pecahan (double dan float) untuk operasi yang bisa mengahasilkan bilangan pecahan
- Atau anda dapat melakukan casting ke tipe data pecahan pada salah satu operan yang terlibat pada operasi tersebut. Contoh kode:

```
int nilai = 26;
double hasil = (double)nilai/4;
```

# Operator

- ## Operator Assignment

| Operator | Deskripsi | Contoh |
|---|---|---|
| = | Pemberian nilai | nilai=total; |
| ++ | Increment , kemudian beri nilai | jumlah++; |
| -- | Decrement , kemudian beri nilai | Jumlah--; |
| += | Tambah kemudian beri nilai | Jumlah+=total; |
| -= | Kurangi kemudian beri nilai | Jumlah-=total; |
| *= | Kalikan kemudian beri nilai | Jumlah*=total; |
| /= | Bagi kemudian beri nilai | Jumlah/=total; |
| %= | Ambil hasil pembagian kemudian beri nilai | Jumlah%=total; |

# Operator Aritmatika

| Operator | Deskripsi | Contoh |
|---|---|---|
| + | Penjumlahan | nilai=ujian+ulangan; |
| - | Pengurangan | Harga=total-diskon; |
| * | Perkalian | Total=jumlah*harga; |
| / | Pembagian | Persen=nilai/100; |
| % | Hasil bagi | Ganjil=nilai%2; |

# Operator Relasi

| Operator | Deskripsi | Contoh |
|---|---|---|
| > | Lebih besar dari | Nilai>rerata |
| < | Lebih kecil dari | Nilai<rerata |
| >= | Lebih besar atau sama dengan | Nilai>=rerata |
| <= | Lebih kecil atau sama dengan | Nilai<=rerata |
| != | Tidak sama dengan | Nilai!=rerata |
| == | Tepat sama dengan | Status==’A’ |

# Operator Logika

| Operator | Deskripsi | Contoh |
|---|---|---|
| ! | NOT | !jikaAngka |
| && | AND | JikaAngka && jikaTgl |
| \|\| | OR | JikaAngka \|\| jikaTgl |
| == | Sama dengan | x == 3 |

# Operator Bitwise (Manipulasi Bit)

| Operator | Deskripsi | Contoh |
|---|---|---|
| ~ | complement | ~a |
| & | AND | a&b |
| \| | OR | a\|b |
| ^ | Ekslusif OR | a^b |
| >> | Geser kanan | a>>b |
| << | Geser kiri | a<<b |

# Operator ?: (if-then-else)

- Operator ini dapat digunakan untuk menggantikan beberapa instruksi yang menggunakan **if-then-else** (akan dibahas pada bab selanjutnya). Secara umum penggunaannya mengikuti bentuk berikut:

```
ekspresi1?ekspresi2:ekspresi3
```

- Di mana ekspresi1 harus menghasilkan nilai true atau false. Jika ekspresi1 bernilai true maka ekspersi2 akan dieksekusi oleh Java, dan sebaliknya jika ekspresi1 bernilai false maka ekspresi3 yang akan dieksekusi oleh Java. Baik ekspresi2 dan ekspresi3 harus mengembalikan tipe data yang sama dan tida boleh mengembalikan void/tidak ada nilai kembalian.

- Contoh penggunaan:

```
boolean akhir = true;
int x = 20;
int hasil = akhir?x+10:x*20;
```

# Kontrol Eksekusi Program

- **Percabangan (if)**
  - Pernyataan if digunakan untuk menguji suatu kondisi kemudian mengerjakan pernyataan yang lain sesuai hasil pengujian

```
if(ekspresi)
    pernyataan;
```

# Contoh:

```
if(angka % 2 != 0) ++angka;


if(nilai<10)
   System.out.println (“kurang dari 10”);


if(nilai<51){
   System.out.println(“Tidak lulus”);
   status=false;
}
```

- Untuk menambahkan pernyataan yang akan dijalankan jika kondisi uji salah, maka dapat ditambahkan klausa **else**, seperti contoh berikut:

```
if(nilai<51){
    System.out.println("Tidak lulus");
    status=false;
}else{
    System.out.println("Lulus");
    status=true;
}
```

- Untuk melakukan pengujian lebih dari satu kali, dapat ditambahkan klause else if, seperti contoh berikut

```
if(nilai<51){
   System.out.println("Tidak lulus");
   status=false;
}else if((nilai>51) && (nilai<71)){
   System.out.println("Lulus, status=cukup");
   status=true;
}else {
   System.out.println("Lulus, status=memuaskan");
   status=true;
}
```

# Percabangan (switch)

- Pernyataan switch digunakan untuk menguji beberapa pilihan berdasarkan beberapa nilai tertentu
- Ekspresi yang digunakan harus menghasilkan data dengan tipe char, byte, short dan int

- Bentuk dari pernyataan switch adalah sebagai berikut:

```
switch(ekspresi){
    case a:
        pernyataan;
        break;
    .....
    default:
        pernyataan;
        break;
}
```

# Contoh

```
switch(status){
    case 1:
        gaji = 500000;
        break;
    case 2:
        gaji = 750000;
        break;
    default:
        gaji = 300000;
        break;
}
```

# Perulangan (for)

- Pernyataan for digunakan untuk melakukan perulangan dengan menentukan kondisi perulangan dan pernyataan increment / decrement.

- Bentuk pernyataan for adalah sebagai berikut:

```
for(inisialisasi ; kondisi; increment){
   pernyataan1;
   pernyataan2;
   ....
}
```

# contoh

```
byte nilai=1;
for (int i=0; i<8; i++){
   nilai *= 2;
   System.out.println(nilai);
}
```

# Perulangan (**while**)

- Pernyataan while digunakan untuk melakukan perulangan dengan menentukan kondisi yang menyebabkan perulangan dihentikan

- Bentuk pernyataan while adalah sebagai berikut :

```
while(ekspresi){
   pernyataan1;
   pernyataan2;
   ....
}
```

# contoh

```
int i = 0;
while (i < 10){
    nilai += 10;
    i++;
}
```

# Perulangan (**do – while)**

- Penggunaan **do-while** ini mirip dengan bentuk while di atas. Perbedaan utamanya yaitu:
  - Pengecekan kondisi (true atau false) dilakukan pada bagian akhir sehingga pernyataan yang ada di dalam blok perulangan akan dieksekusi minimal satu kali, sekalipun eksekusi **do-while** pertama kali menemukan kondisi bernilai false.

- Penggunaan bentuk do-while mengikuti bentuk berikut ini

```
do{
    Pernyataan1;
    Pernyataan2;
    ....
}while(kondisi);
```

# Jump (break)

- Penggunaan **break** yang utama adalah untuk menghentikan proses perulangan di dalam **while**, **do-while** atau di dalam **for**.

# Contoh:

```
class TestBreak{
   public static void main(String[] args){
      System.out.println("Sebelum for");
      for(int x=0;x<10;x++){
         if(x==4)
       break;
         System.out.pritnln("Nilai x : "+x);
      }
      System.out.println("Setelah For");
  }
}
```

# Jump (continue)

- Terkadang dalam mengeksekusi suatu kode dalam perulangan anda ingin terus melakukan perulangan, tetapi karena kondisi tertentu anda ingin kode setelah posisi tertentu tersebut tidak dieksekusi. Di sinilah kita dapat menggunakan **continue**

# Contoh:

```
class TestContinue{
   public static void main(String[] args){
      int x=10;
      System.out.println("Sebelum while");
      while(x<=50){
       x++;
       if(x%2==0)
          continue;
       System.out.println("Nilai x : "+x);
      }
      System.out.println("Sesudah while");
   }
}
```

# Jump (return)

- Perintah dalam Java terakhir yang dapat dikategorikan sebagai jump adalah perintah **return** yang digunakan di dalam method.
- Method dipanggil oleh bagian lain dari program. Dengan menggunakan perintah **return**, alur eksekusi dikembalikan ke bagian program yang memanggil method tersebut.

# Contoh:

```
public class Orang{
    public String cetakNama(){
        return Hello Nama Saya Hendro!”;
    }

    public statc void main(String[] args){
        Orang org = new Orang();
        System.out.println(“Sebelum panggil method”);
        System.out.println(org.cetakNama());
        System.out.println(“Sesudah panggil method”);
    }
}
```

# Latihan:

- Buatlah aplikasi yang akan menampilkan bilangan prima antara 0 – 100.
- Buatlah aplikasi yang akan menampilkan:

```
*
**
***
****
*****
******
*******
```

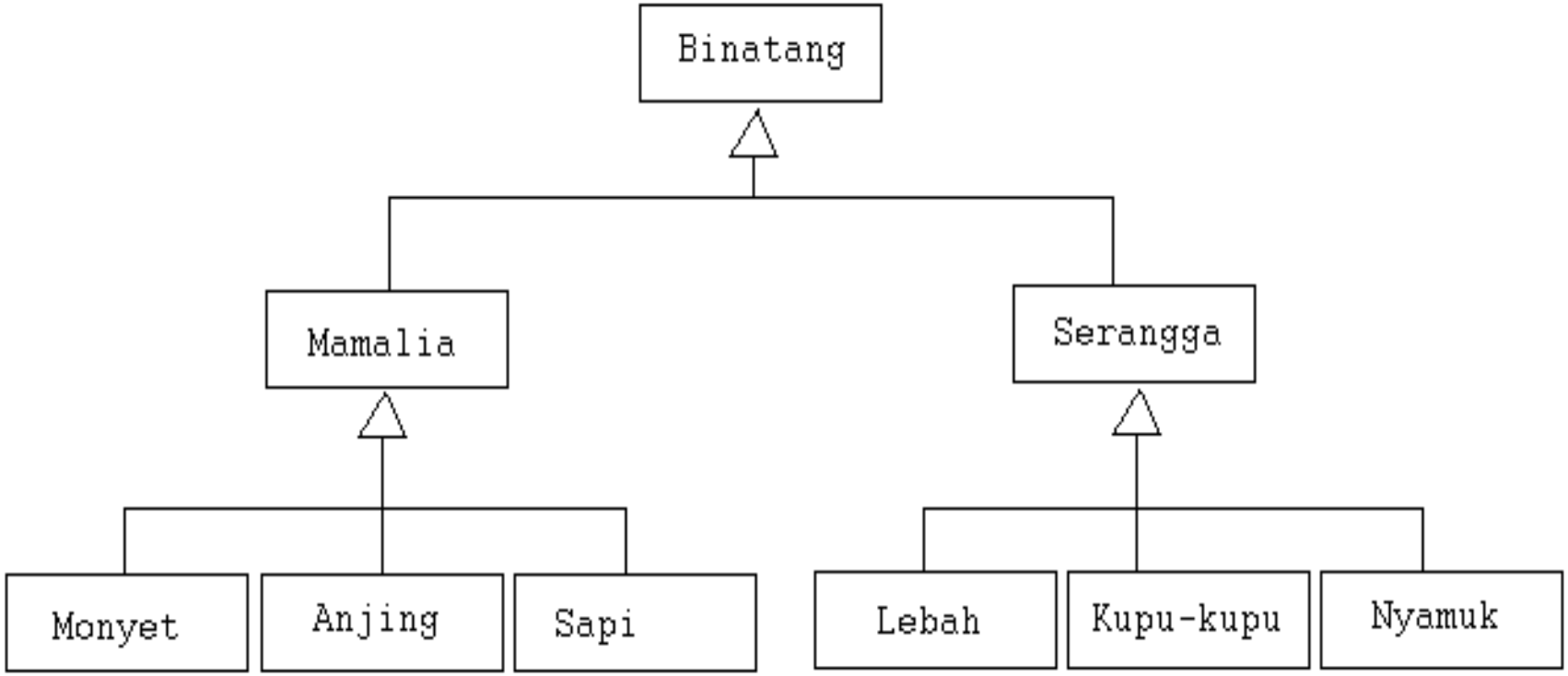
# Object-Oriented Programming 1

# Object-oriented Programming (OOP)

- Pusat dari Java adalah OOP, tanpa OOP Java bukan apa-apa.
- OOP adalah metodologi yang sangat powerful dalam pemrograman
- Class adalah blueprint/bentuk dasar dari object.
- Object adalah realisasi/instance dari class

- Encapsulasi: mekanisme pemrograman yang menggabungkan fungsi dan datanya bersama-sama serta menjaganya dari interfensi luar.
- Polymorphism (banyak bentuk): kemampuan satu interface untuk mengakses action dari class-classnya.

- Inheritance: kemapuan suatu object untuk mewariskan miliknya (property dan method) ke object lain.



# Contoh Encapsulasi

```
public class Person{
    String nama;
    String alamat;
    String pekerjaan;
    int usia;
    boolean sudahNikah;
    void setNama(String var){
        nama = var;
    }
    String getNama(){
        return nama;
    }
    // buat seter dan geter method untuk property lainnya.
}

public class DemoPerson{
    public static void main(String[] args){
        Person p1 = new Person();
        Person p2 = new Person();

        p1.setNama("Budi");
        p2.setNama("Andi");
        System.out.println(p1.getNama());
        System.out.println(p2.getNama());
    }
}
```

**Contoh Polymorphism dan Inheritance**

```
public class Senjata{
    void tembak(){
            System.out.println("dor..dor..dor…");
    }
}

public class M16 extends Senjata{
    void tembak(){
            System.out.println("trrrrrrt…ttrrrrrt…trrrrt….");
    }
}

public class Bazoka extends Senjata{
    void tembak(){
            System.out.println("Boom…Bomm..BOOMM…");
    }
}
```

```
public class Prajurit{
    Senjata senjata;
    void setSenjata(Senjata var){
            senjata =  var;
     }
    void menembak(){
            senjata.tembak();
    }
}

public class DemoSenjata{
   public static void main(String[] args){
            Senjata s = new Senjata();
            Prajurit p = new Prajurit();
            p.setSenjata(s);
            p.menembak;
            s = new M16();
            p.setSenjata(s);
            p.menembak;
   }
}
```

# Variabel atau Property

- Property pada suatu class dapat berupa instace variabel, class variabel, constant variabel (konstanta) yang telah dibahas sebelumnya.

# Method

- Pada bahasa – bahasa pemograman yang lain method disebut juga *function* atau *procedure*
- Dalam pemograman berorientasi object method adalah suatu operasi atau kegiatan yang dapat dilakukan suatu object
- Misalnya Manusia memiliki kegiatan seperti makan, tidur, minum dan lain – lain. Kegiatan – kegiatan inilah yang dapat dijadikan method

```
class Manusia{
   String nama;
   boolean isKenyang;
   int jumPiring;

   void setNama(String _nama){
      nama = _nama;
   }
   String getNama(){
      return nama;
   }
   void makan(){
         cekJumPiring();
         if(isKenyang==false){
          System.out.println(nama+" lapar makan ahhh...");   }else{
             System.out.println(nama+" sudah makan tadi...");
         }
   }

   void cekJumPiring(){
         jumPiring = jumPiring + 1;
         if(this.jumPiring>3)
          isKenyang = true;
   }
}
```

```
class DemoManusia{
        public static void main(String args[]){
                Manusia m1 = new Manusia();
                m1.setNama(“Hendro”);
                m1.makan();
                m1.makan();
                m1.makan();
                m1.makan();
        }
}
```

- Dari contoh di atas dapat kita lihat bentuk umum pendeklarasian method adalah sebagai berikut:

```
tipe-data namaMethod(daftar-parameter){
        //implementasi
}
```

# Method Overloading

- Dalam Java kita boleh memiliki lebih dari satu method yang memiliki nama sama. Inilah yang disebut *method overloading*
- Walaupun Java mengizinkan  memiliki nama method sama lebih dari satu, tetapi daftar parameter yang digunakan haruslah berbeda untuk masing – masing method
- Karena Java akan menggunakan parameter – parameter ini untuk menentukan method mana yang akan dieksekusi

```
class Manusia{
  String nama;
  String jenkel;

  void setNilai(String param1){
         nama = param1;
  }

  void setNilai(String param1,String param2){
         nama = param1;
         jenkel = param2;
  }

  void cetak(){
         System.out.println(nama+" adalah "+jenkel);
  }
}
```

```
class DemoManusia{
        public static void main(String args[]){
                Manusia m1,m2;
                m1 = new Manusia();
                m2 = new Manusia();

                m1.setNilai("Hendro");
                m2.setNilai("Hendro","Laki-laki");

                m1.cetak();
                m2.cetak();

        }
}
```

# Konstruktor

- Konstruktor merupakan method khusus yang digunakan untuk menginisialisasi objek dan tiap class boleh memiliki lebih dari satu konstruktor.
- Perbedaan method biasa dengan konstruktor adalah bahwa konstruktor harus memiliki nama yang sama dengan nama classnya dan tidak memiliki nilai kembalian (tipe-data).
- Konstruktor dijalankan pada saat sebuah object diinisialisasi (menggunakan kata **new**)
- Pada konstruktor juga berlaku overloading, artinya boleh mendeklarasikan lebih dari satu konstruktor, asalkan memiliki parameter yang berbeda – beda.

```
class Manusia{
  String nama;
  String jenkel;

  Manusia(){  //konstruktor
          nama = "unknown";
          jenkel = "unknown";
  }

  Manusia(String param1,String param2){ //konstruktor
          nama = param1;
          jenkel = param2;
  }
  void cetak(){
          System.out.println("Nama : "+nama);
          System.out.println("Jenis Kelamin : "+jenkel +"\n");
  }
}
```

```
class DemoManusia{
        public static void main(String args[]){
                Manusia m1,m2;
                m1 = new Manusia();
                m2 = new Manusia("Hendro","Laki-laki");


                m1.cetak();
                m2.cetak();
        }
}
```

- Yang perlu diperhatikan adalah apabila anda tidak mendeklarasikan satu pun konstruktor, maka Java secara otomatis menambahkan konstruktor default ke dalam class yang kita buat walaupun tidak kelihatan pada kode program. Apabila kita mendeklarasikan satu atau lebih konstruktor maka java tidak akan menambahkan kostruktor default.

# Penggunakan Keyword ‘this’

- Terkadang dalam suatu method kita ingin menunjuk ke objek di mana method ini berada. Untuk itu kita menggunakan keyword ***this***. Perhatikan Contoh berikut.

```
class Manusia{
    String nama;
    void setNama(String nama){
        this.nama = nama;
    }
}
```

# Penggunaan Keyword ‘static’

- Apabila kata kunci static kita tempatkan pada pendeklarasian member (variabel dan method) dari suatu class, maka member class tersebut dapat diakses tanpa harus menciptakan objek class tersebut. Untuk lebih jelasnya perhatikan contoh berikut ini.

```
class Manusia{
  static String nama;
  static String jenkel;

  static void cetak(){
         System.out.println("Nama : "+nama);
         System.out.println("Jenis Kelamin : "+jenkel +"\n");
  }
}

class DemoManusia{
       public static void main(String args[]){
              Manusia.nama = "Hendro";
              Manusia.jenkel = "Laki - laki";
              Manusia.cetak();
       }
}
```

# Modifier

- Penggunaan *modifier* berfungsi untuk melakukan enkapsulasi (membungkus data) pada objeck.
- Dengan menggunakan *modifier* kita dapat menentukan siapa saja yang boleh menggunakan atau mengakses member dari suatu objek.
- Ada empat macam modifier yang dikenal oleh Java, yaitu **private**, **protected**, **public** dan **tanpa *modifier***.

# Class Modifier

- Bentuk penggunaan modifier pada class:

```
modifier class NamaClass{
    …
    …
}
```

| Modifier | Keterangan |
|---|---|
| (default) | Class visible atau dapat digunakan hanya pada package yang sama |
| public | Class visible terhadap semua package yang berbeda – beda |
| final | Class tidak dapat diturunkan lagi / extends |

```
public class Manusia{
  ....
  ....
}
```

- Apabila sebuah class menggunakan modifier public maka class tersebut harus disimpan dengan nama file yang sama dengan nama classnya. Seperti pada contoh di atas, maka class Manusia harus disimpan dengan nama Manusia.java.

# Property dan Method Modifier

- **Bentuk penggunaan modifier** pada property:

```
modifier tipe-data namaProperty;
```

**Contoh:**

```
private int usia;
```

- Bentuk penggunaan modifier pada method:

```
modifier tipe-data namaMethod(parameter){
    …
    …
}
```

- Berikut ini adalah daftar modifier yang dapat digunakan pada method

| Modifier | Keterangan |
|---|---|
| (default) | Method visible atau dapat digunakan hanya pada package yang sama |
| public | Method visible pada semua package |
| private | Method visible hanya di dalam class itu sendiri |
| protected | Method visible didalam package dan sub classnya |
| static | Lihat sub bab sebelumnya |
| final | Method tidak dapat diubah / dioverride pada subclass |
| abstract | Method harus dioverride / didefinisikan pada subclassnya |

```
public class Manusia{
  private String nama;
  private String jenkel;

  public void setNama(String nama){
          this.nama=nama;
  }
  public void setJenkel(String jenkel){
          this.jenkel=jenkel;
  }
  public void cetak(){
          System.out.println("Nama : "+nama);
          System.out.println("Jenis Kelamin : "+jenkel);
  }
}


public class DemoManusia{
        public static void main(String args[]){
                Manusia m = new Manusia();
                m.setNama("Hendro");
                m.setJenkel("Laki-laki");
                m.cetak();
        }
}
```

# Pewarisan pada Java

- Untuk menerapkan konsep pewarisan, Java menyediakan keyword **extends** yang dapat dipakai pada waktu mendeklarasikan suatu class
- Di dalam Java semua class yang kita buat sebenarnya adalah turunan atau subclass dari class **Object**. Class Object merupakan class tertinggi dari semua hierarki class dalam Java.

# Method Overriding

- Jika pada subclass kita menulis ulang method yang ada pada super classnya, maka method yang ada di subclass tersebut disebut meng-override method super classnya
- Jadi ketika kita memanggil method tersebut dari objek subclassnya maka yang akan dijalankan adalah method yang berada di subclass tersebut

```
public class A {
        public void cetak(){
                System.out.println("Dicetak oleh class A");
        }
}
public class B extends A {
        public void cetak(){
                System.out.println("Dicetak oleh class B");
        }
}

public class TestOverride {
        public static void main(String[] args){
                B objB = new B();
                objB.cetak();
        }
}
```

# Keyword ‘final’ dalam Pewarisan

- Pada pertemuan sebelumnya penggunaan kata ***final*** adalah untuk mendeklarasikan suatu konstanta.
- Apabila kata **final** digunakan pada pendeklarasian method, maka method tersebut tidak dapat dioverride
- Selanjutnya apabila kita menggunakan kata final pada deklarasi class, akan menyebabkan class tersebut tidak dapat diturunkan atau membuat subclassnya menggunakan keyword **extends**.

# Inner Class

- Di dalam Java kita dapat mendifinisikan nested class.
- Nested class/inner class adalah class yang dideklarasikan di dalam class lain.
- Kemampuan ini ada sejak Java versi 1.1
- Nested class hanya dikenali pada outer classnya.
- Inner class dapat mengakses semua property dan method outer classnya termasuk yang private.
- Sebaliknya Outer class tidak dapat mengakses member dari inner classnya secara langsung.
- Untuk mengakses member dari inner class, outer classnya harus menciptakan object inner classnya.

```
// Use an inner class.
public class Outer {
	int nums[];
	Outer(int n[]) {
		nums = n;
	}
	void Analyze() {
		Inner inOb = new Inner();
		System.out.println("Minimum: " + inOb.min());
		System.out.println("Maximum: " + inOb.max());
		System.out.println("Average: " + inOb.avg());
	}
	// This is an inner class.
	class Inner {
	   int min() {
		int m = nums[0];
				for(int i=1; i < nums.length; i++)
				if(nums[i] < m) m = nums[i];
				return m;
				    }
```

```
        int max() {
                int m = nums[0];
                for(int i=1; i < nums.length; i++)
                if(nums[i] > m) m = nums[i];
                return m;
        }
        int avg() {
                int a = 0;
                for(int i=0; i < nums.length; i++)
                a += nums[i];
                return a / nums.length;
        }
    }
}

class NestedClassDemo {
        public static void main(String args[]) {
                int x[] = { 3, 2, 1, 5, 6, 9, 7, 8 };
                Outer outOb = new Outer(x);
                outOb.Analyze();
        }
}
```

# Latihan

- Definisikan/identifikasikan object-object beserta role-role yang ada pada business proses pada sebuah perpustakaan (buat class beserta member-membernya).

# Object Oriented Programming 2

# Abstract Class

- Class Abstrak tidak berbeda dengan class-class lainnya yaitu memiliki class members (method dan property)
- Sebuah class adalah abstrak jika salah satu methodnya dideklarasikan abstrak.
- Method abstrak adalah method yang tidak memiliki implementasi.

- Contoh deklarasi method abstrak:

```
abstract public void cetak();
```

- Beberap hal yang perlu diperhatikan adalah sebagai berikut:
  - Class abstrak tidak dapat dibuatkan instan atau objeknya menggunakan keyword **new**
  - Sebuah class dapat dideklarasikan sebagai class abstrak walaupun tidak memiliki method abstrak.

- Variabel dengan tipe class abstrak tetap bisa diciptakan, tetapi harus *refer* ke subclass dari class abtrak tersebut yang tentunya tidak abstrak

# Contoh:

```
abstrak public class Mobil {
      abstract public void injakPedalGas();
      public void injakRem(){
        System.out.println(“Mobil berhenti!”);
      }
}
public class Kijang extends Mobil{
      public void injakPedalGas(){
              System.out.println("Mobil Melaju dengan kecepatan 80 Km/jam…");
      }
}
public class BMW extends Mobil {
   public void injakPedalGas(){
    System.out.println("Mobil Melaju dengan kecepatan 100 Km/jam...");
      }
}
```

- Objek class abtrak tidak dapat diciptakan menggunakan keyword **new** secara langsung.
- Apabila kita terpaksa ingin menciptakan object class abtrak tanpa membuat subclass kongkritnya, maka kita harus mengimplementasikan method – method abstraknya secara langsung saat deklarasi.

# Contoh:

```
public class TestMobil {
	public static void main(String[] args){
		Mobil mobil = new Mobil(){
			public void injakPedalGas(){
				System.out.println("Mobil berjalan...");
			}
		};

		Kijang kijang = new Kijang();
		Escudo escudo = new Escudo();
		BMW bmw = new BMW();

		mobil.injakPedalGas();

		mobil = kijang;
		mobil.injakPedalGas();

		mobil = escudo;
		mobil.injakPedalGas();

		mobil = bmw;
		mobil.injakPedalGas();
	}
}
```

# Interface

- Interface adalah class yang hanya mengandung deklarasi method tanpa memiliki implementasi dan semua property yang dimilikinya bersifat final.
- Interface mirip dengan class abstrak, tetapi interface tidak terikat dengan class hierarki.

- Berikut ini adalah aturan yang harus kita ingat tentang pendeklarasian interface:
  - Modifier yang digunakan hanya **public** atau tidak sama sekali. Jika tidak menggunakan modifier maka interface tersebut hanya dapat diakses dalam package yang sama.
  - Semua variabel yang dideklarasikan dalam interface secara otomatis adalah static final. Karena itu waktu pendeklarasian harus diberikan nilai.

- Semua method adalah abstrak. Bedanya dengan class abstrak adalah kita tidak perlu menuliskan *keyword* **abstract** pada saat mendeklarasikan method dalam interface.
- Kita dapat mengimplementasikan lebih dari satu interface (multiple inheritance) dengan memisahkan nama dari setiap interface dengan tanda koma.

- Dapat terjadi saat kita mengimplementasikan lebih dari satu interface ternyata interface – interface tersebut memiliki method yang sama. Dalam hal ini method yang akan diimplementasi adalah method yang berada pada posisi pertama.
- Semua method yang diimplemetasikan harus **public**.

- Jika kita tidak mengimplementasikan semua method yang ada pada interface, maka class tersebut harus dideklarasikan sebagai **abstract** class.

# contoh

```
public interface Control {
	public void pindahChannel(int channel);
	public void perbesarVolume(int intensitas);
	public void perkecilVolume(int intensitas);
}
```

```
public class TVPolitron implements Control{
   String[] channel = {"RCTI","SCTV","INDOSIAR","ANTV","TV7"};
   public void pindahChannel(int channel) {
      System.out.println("Pindah channel pada TV Politron ke "+
this.channel[channel]);
   }

   public void perbesarVolume(int intensitas) {
      System.out.println("Perbesar volume pada TV Politron sebanyak "+
intensitas);
   }

   public void perkecilVolume(int intensitas) {
      System.out.println("Perkecil volume pada TV Politron sebanyak "+
intensitas);
   }
}
```

```
public class TVSamsung implements Control{
   String[] channel = {"RCTI","SCTV","INDOSIAR","ANTV","TV7"};
   public void pindahChannel(int channel) {
     System.out.println("Pindah channel pada TV Samsung ke "+
this.channel[channel]);
   }
   public void perbesarVolume(int intensitas) {
     System.out.println("Perbesar volume pada TV Samsung sebanyak "+
intensitas);
   }
   public void perkecilVolume(int intensitas) {
     System.out.println("Perkecil volume pada TV Samsung sebanyak "+
intensitas);
   }
}
```

```
public class RemoteControl {
 public void kirimPerintahKeTv(int aksi,Control tv,int tombol){
   switch(aksi){
   case 1:
         tv.pindahChannel(tombol);
         break;
   case 2:
         tv.perbesarVolume(tombol);
         break;
   case 3:
         tv.perkecilVolume(tombol);
         break;
   }
 }
}
```

```
public class TestRemoteControl {
        public static void main(String[] args){
                TVPolitron tvp = new TVPolitron();
                TVSamsung tvs = new TVSamsung();
                RemoteControl rc = new RemoteControl();

                rc.kirimPerintahKeTv(1,tvp,1);

        }
}
```

# Package

- Package adalah cara untuk mengelompokan class dan interface yang ada ke dalam kelompoknya (*name space*) masing – masing sehingga lebih mudah diatur dan memungkinkan penggunaan nama yang sama.
- Untuk mendefinisikan suatu package digunakan *keyword* **package**
- Pendefinisian nama package harus terletak di bagian paling atas dari source program kita.

- Sintaks pendefinisian nama package adalah sebagai berikut:
  - **package namaPackage;**
- Contoh:
  - **package siswa;**
- Java menggunakan package seperti struktur direktori. Oleh karena itu semua class atau interface yang memiliki definisi package seperti contoh di atas, harus disimpan pada direktori bernama siswa.

- Kita juga dapat membuat package secara hierarki layaknya struktur direktori.
- Contoh:
  - **package hen.com.contoh;**
- Pada contoh di atas menunjukan bahwa semua class atau interface yang menggunkan deklarasi package ini harus disimpan pada direktori hen -> com -> contoh
- Apabila program kita akan menggunakan sebuah class yang terletak pada package yang berbeda, maka kita harus meng***import***nya agar dapat digunakan

# Exception

- Exception dalam Java didefinisikan sebagai sebuah obyek yang muncul ketika terjadi kondisi tidak normal dalam sebuah program.
- Untuk menangani exception dalam sebuah program, dapat digunakan pernyataan try, catch, dan finally.

- try atau block try digunakan untuk menjaga pernyataan yang memungkinkan timbulnya exception.
- catch atau block catch digunakan untuk menangani exception ketika muncul.
- finally digunakan untuk menutup proses penanganan exception, dimana pernyataan ini boleh untuk tidak digunakan. Block finally pasti akan dieksekusi saat terjadi error atau tidak.

# Contoh:

```
public class TestTryCatch{
   public static void main(String[] args){
      int i = 1;
      int j = 0;
      try{
          System.out.println("Try block entered " + "i = "+ i
+ " j = "+j);
        System.out.println(i/j);
          System.out.println("blok try berakhir");
      }catch(ArithmeticException e) {
       System.out.println("terjadi exception");
      }
      System.out.println("setelah blok try");
      return;
  }
}
```

# contoh

```
import java.io.IOException;
public class TryBlockTest{
   public static void main(String[] args) throws IOException{
      int[] x = {10, 5, 0};
      try{
         System.out.println("Blok try pertama");
         System.out.println("hasil = " + divide(x,0));
         x[1] = 0;
         System.out.println("hasil = " + divide(x,0));
         x[1] = 1;
         System.out.println("hasil = " + divide(x,1));
      } catch(ArithmeticException e){
         System.out.println("Arithmetic exception”);
      } catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("Index-out-of-bounds exception");
      }
         System.out.println("\nTekan Enter untuk keluar");
      try{
         System.out.println("blok try kedua");
         System.in.read();
         return;
      }catch(IOException e){
         System.out.println("I/O exception ");
      }finally{
         System.out.println("blok finally");
      }
```

```
   public static int divide(int[] array, int index){
     try{
         System.out.println("\nblok try pertama");
         array[index + 2] = array[index]/array[index + 1];
         return array[index + 2];
     }catch(ArithmeticException e) {
        System.out.println("Arithmetic exception");
     }catch(ArrayIndexOutOfBoundsException e) {
         System.out.println("Index-out-of-bounds exception");
      }finally{
          System.out.println("blok finally");
     }
      return array[index + 2];
   }
}
```

# String

- Pada bahasa pemrograman lain String adalah array of characters, tetapi pada Java String adalah Object.
- Sebenarnya kita telah menggunakan class String sejak pertemuan pertama, yaitu saat kita definisikan literal String.
  - System.out.println(“Hello World”);
- Kalimat “Hello World” otomatis akan dijadikan object String oleh Java.

# Constructing String

- Kita dapat menciptakan object String sama seperti menciptakan object-object yang lain.
  - String str = new String(“Hello”);
- Kita juga dapat menciptakan object String dari object String yang lain.
  - String str1 = new String(“Hello”);
  - String str2 = new String(str1);

- Cara lain yang lebih mudah dan paling banyak digunakan.
  - String str = “Hello World”;

# Operating on Strings

- Di dalam class String terdapat beberapa method yang dapat kita gunakan untuk memanipulasi object String.

| ***Method*** | ***Keterangan*** |
|---|---|
| boolean equals(String str) | Mengembalikan true jika String yang dipanggil memiliki karakter yang sama dengan parameter str. |
| int length() | mengembalikan panjang sebuah String |
| char charAt(int index) | Mengambil karakter posisi tertentu pada sebuah String |
| toLowerCase() | Mengubah semua karater menjadi huruf kecil |
| toUpperCase() | Mengubah semua karakter menjadi huruf kapital. |
| substring(int beginindex,int endindex) | Mengambil string pada posisi tertentu dari sebuah string |
| trim() | Menghapus spasi |

# Latihan

- Buatlah aplikasi sederhana yang dapat melakukan operasi-operasi *Change Case* pada sebuah String

# Java Collection Framework

- Java Collection adalah class-class di dalam java yang digunakan untuk menampung object-object.
- Class-class yang ada di java collection framework cukup banyak, bisa dilihat dalam package java.util dari dokumentasi sdk.

- List
  - ArrayList
  - Method-method yang digunakan antara lain
    - add(object o) : Menambahkan object keakhir sebuah list
    - add(int index,Obejct o): Menambahkan object ke posisi tertentu.
    - clear() : menghapus semua object yang ada dalam list
    - get(int index): Mengambil object pada posisi tertentu.
    - size() : mendapatkan jumlah object pada sebuha list
    - etc…

# Contoh List

```
public class DemoList {
    public static void main(String[] args){
      List list = new ArrayList();
      list.add("Selamat");
      list.add(" Datang");
      list.add(" di dunia Java");

      System.out.println("Jumlah element: "+list.size());
      for(int x=0;x<list.size();x++){
          System.out.println("Element ke "+x+": "+list.get(x));
      }
      for(int x=0;x<list.size();x++){
          System.out.print(list.get(x));
      }
    }
}
```

- Map
  - Hashtable
  - Hasmap

    Map adalah collection yang menampung pasangan object key dan value. Map tidak dapat menampung *duplicate keys*.

- Method-method yang dapat digunakan antara lain:
  - put(Object key, Object value) menyimpan object dengan key pada sebuah map.
  - get(Obejct key) mengambil Object untuk key tertentu.
  - clear() menghapus semua objec yang ada dalam map.
  - containsValue(Object value) jika map menyimpan key untuk value ini.
  - size() mengembalikan jumlah key-value yang ada dalam sebuah map

```
public class Demo {
    public static void main(String[] args){
      Hashtable table = new Hashtable();
      table.put("1","Satu");
       //table.put("1",null); // ini pasti error
      System.out.println(table.get("1"));
      HashMap map = new HashMap();
      map.put("1","Satu");
      map.put("1",null);
      System.out.println(map.get("1"));
    }
}
```

- Perbedaan utama dari Hashtable dan HashMap adalah bahwa HashMap dapat menampung null)

# Mengakses Database

# Apa itu JDBC?

- JDBC adalah API java yang digunakan untuk eksekusi perintah SQL ke database.
- JDBC bukan merupakan singkatan tetapi adalah nama teknologi
- JDBC menyediakan API standar bagi developer untuk dapat membuat aplikasi database berbasis Java
- Class-class JDBC berada dalam package java.sql

# Apa yang dilakukan JDBC?

- Menciptakan koneksi ke database
- Mengirim perintah-perintah SQL
- Memproses hasil eksekusi (proses result)

# Apa yang dibutuhkan?

- Database engine (mysql,sql server, oracle,ms.access,db2,hsql,point base, etc)
- JDBC Driver

# 7 step using JDBC

- Load the Driver
- Define the Connection URL
- Establish The Connection
- Create a statement object
- Execute a query
- Process the results
- Close the connection

# Load the Driver

```
try{

   Class.forName("com.mysql.jdbc.Driver").newInstance()
   ;
}catch(ClassNotFoundException e){
   System.out.println(“Error loading driver..”);
}
```

# Define the connection URL

```
String dbURL = "jdbc:mysql://localhost:3306/dbname";
String dbUser = “user";
String dbPass = “password";
```

# Establish the Connection

```
conn = DriverManager.getConnection(dbURL,dbUser,dbPass);
```

# Create Statement

```
Statement statement =
   conn.createStatement();
```

# Execute a Query

```
String sql = "SELECT * FROM table";
ResultSet rs = statement.executeQuery(sql);
```

- *Use statement.executeUpdate if you want to modify the database. These modifying operations includes INSERT, UPDATE, and DELETE.*
- String sgl = "INSER INTO table VALUES(val1,val2,val3)";
- ResultSet rs = statement.executeUpdate(sql);

# Process the Result

```
String nama=“”;
String alamat=“”;
Float nilai=0.0;
Int noUrut = 0;
While(rs.next()){
   nama = rs.getString(1);
   alamat = rs.getString(2);
   nilai = rs.getFloat(3);
   noUrut = rs.getInt(4);
}
```

# Close the statement

- statement.close();
- connection.close();

- Use this statement only if you are sure that there is no more database operation will be proceed.

# Example

- Try it…

# GUI Design With Java Swing

# Materi

- Mengenal Java Swing
- Komponen Dasar Swing
- Membuat Window dengan JFrame
- Menambahkan TextField dan Tombol
- Menambahkan Event pada tombol
- Kotak Dialog

# Mengenal Java Swing

- Java Swing adalah librari java yang digunakan untuk menciptakan *Grafik User Interface* (GUI)
- Dengan Java Swing kita dapat membuat user interface yang *cross platform* atau *OS independent.* User interface yang kita buat dapat dijalankan pada system operasi apa saja (OS yang suport Java) dengan tampilan yang relative sama
- Bahkan kita dapat membuat user interface yang menyerupai Windows XP, Mac OS atau Linux tanpa tergantung dari OS yang kita gunakan.

# Komponen Dasar Swing

- ***Top-level Container***, merupakan container dasar di mana komponen lainnya diletakan. Contoh Top-level container ini adalah Frame, Dialog dan Applet yang diimplementasi dalam class JFrame, Jdialog, dan JApplet.

- ***Intermediate Container***, merupakan komponen perantara di mana komponen lainnya akan diletakan. Salah satu contoh container ini adalah class JPanel

- ***Atomic Component***, merupakan komponen yang memiliki fungsi spesifik dan biasanya user berinteraksi langsung dengan komponen jenis ini. Contohnya adalah JButton, JLabel, JTextField, dan JTextArea

- ***Layout Manager***, berfungsi untuk mengatur bagaimana posisi dari komponen – komponen yang diletakan pada container. Secara default terdapat 5 macam layout yaitu berupa class BorderLayout, BoxLayout, FlowLayout, GridBagLayout, dan GridLayout

- ***Event Handling***, untuk menangani event yang dilakukan oleh user misalnya menekan tombol, mengkilik mouse dan lain – lain

# Window dengan JFrame

```
import javax.swing.*;
public class TestFrame{
   public static void main(String[] args){
        JFrame frame = new JFrame("Contoh JFrame");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400,150);
        frame.setVisible(true);
   }
}
```

# Menambahkan TextField dan tombol

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SimpleForm {
   public static void main(String[] args){
         JFrame myFrame = new JFrame("Form Sederhana");
    FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
         layout.setVgap(10);
         layout.setHgap(10);

         JTextField txtPesan = new JTextField(20);
         JButton cmdTampil = new JButton("Tampil");
         JButton cmdClose = new JButton("Keluar");

         myFrame.getContentPane().setLayout(layout);
         myFrame.getContentPane().add(txtPesan);
         myFrame.getContentPane().add(cmdOK);
         myFrame.getContentPane().add(cmdClose);

myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
         myFrame.pack();
         myFrame.setVisible(true);
   }
}
```

# Menambahkan event pada tombol

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SimpleForm {
   public static void main(String[] args){
      JFrame myFrame = new JFrame("Form Sederhana");
         FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
         layout.setVgap(10);
      layout.setHgap(10);

JTextField txtPesan = new JTextField(20);
         JButton cmdTampil = new JButton("Tampil");
         JButton cmdClose = new JButton("Keluar");
         cmdClose.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent evt) {
                                    System.out.println("Keluar");
                                    System.exit(0);
                           }
                  });
         myFrame.getContentPane().setLayout(layout);
         myFrame.getContentPane().add(txtPesan);
         myFrame.getContentPane().add(cmdOK);
         myFrame.getContentPane().add(cmdClose);

         myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                  myFrame.pack();
                  myFrame.setVisible();
         }
}
```

## Kotak Pesan

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SimpleForm {
   public static void main(String[] args){
           JFrame myFrame = new JFrame("Form Sederhana");
           FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
           layout.setVgap(10);
           layout.setHgap(10);

           final JTextField txtPesan = new JTextField(20);
           JButton cmdTampil = new JButton("Tampil");
           cmdTampil.addActionListener(new ActionListener(){
              public void actionPerformed(ActionEvent evt){
     JOptionPane.showMessageDialog(null,txtPesan.getText());
              }
           });
           JButton cmdClose = new JButton("Keluar");
           cmdClose.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                                 System.out.println("Keluar");
                                 System.exit(0);
              }
           });

           myFrame.getContentPane().setLayout(layout);
           myFrame.getContentPane().add(txtPesan);
           myFrame.getContentPane().add(cmdTampil);
           myFrame.getContentPane().add(cmdClose);

           myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
           myFrame.pack();
           myFrame.setVisible(true);
   }
}
```

# Netbeans Matisse

**Hands On Labs**

# Author


- Nama : Hendro Steven Tampake, S.Kom
- Email : hendro.steven@gmail.com
- Blog : http://hendro.swbtc.net