BAB 2
COLLECTION & THREAD

2.1 Pendahuluan

Pada pembuatan game di java, sering kali para programer mendefinisikan banyak object seperti suara, gambar, dan grafik geometri yang begitu banyak, untuk itu dibutuhkan suatu manajemen yang menangani hal tersebut atau lebih tepatnya sebuah object untuk menangani object-object dari gambar atau grafik yang dibuat. Java telah menyediakan sebuah container atau collection yang digunakan untuk menangani hal tersebut. Selain itu, seiring dengan banyaknya object gambar atau grafik geometri yang dibuat maka proses yang dibutuhkan juga memakan waktu yang cukup lama dimana dalam java penggambaran object itu dilakukan 1000 kali per detik yang sangat memberatkan kinerja dari procesor, oleh sebab itu dibutuhkan thread untuk menangani hal terebut agar komputer berjalan dengan normal sesuai dengan keinginan.

2.2 Tujuan

Setelah mempelajari modul ini peserta diharapkan dapat:

- Memahami tentang interface Collection pada Java
- Mengerti tentang pembuatan Vector, ArrayList dan LinkedList.
- Membuat dan mengelolah Thread
- Memahami proses sinkronisasi data
- Menggunakan method Wait dan Notify

2.3 Collection

Collection atau sering juga disebut sebagai container adalah sebuah object sederhana yang menampung lebih dari satu elemen di dalam satu kesatuan. Collection digunakan untuk menyimpan,mengambil dan memanipulasi data, juga mentransmisikan data dari satu method ke method lain. Collection khusus merepresentasikan data item dari bentuk yang sebenarnya seperti Poker (kumpulan dari kartu), direktori (kumpulan dari file atau folder), kotak surat (kumpulan dari surat-surat), dll.

SDK tidak menyediakan implementasi built-in yang lain dari interface ini tetapi mengarahkan subinterfaces, *Set* interfaces dan *List* interfaces diperbolehkan. Sekarang, apa perbedaan dari kedua interface tersebut. Set merupakan collection yang tidak dipesan dan tidak ada penggandaan didalamnya. Sementara itu, list merupakan collection yang dipesan dari elemen-elemen dimana juga diperbolehkannya penggandaan. *HashSet, LinkedHashSet* dan *TreeSet* adalah implementasi class dari *Set* interfaces. *ArrayList, LinkedList* dan *Vector* merupakan implementasi class dari *List* interfaces.

<table>
<tr><td colspan="6"><root interface>
Collection</td></tr>
<tr><td colspan="3"><interface> Set</td><td colspan="3"><interface> List</td></tr>
<tr><td colspan="3"><implementing classes></td><td colspan="3"><implementing classes></td></tr>
<tr><td>HashSet</td><td>LinkedHashSet</td><td>TreeSet</td><td>ArrayList</td><td>LinkedList</td><td>Vector</td></tr>
</table>

Gambar 2.1 Implementasi Interface Collection

2.3.1 Vektor

Vektor adalah implementsi dari dinamis array, yang hampir sama dengan ArrayList perbedaanya adalah vector mempunyai method yang bukan turunan dari collection framework. Pada Java 2 vektor diperluas lagi dengan mengextends AbstractList dan mengimplementasikan List interface yang sangat mendukung collection. Dibawah ini adalah konstruktor dari Vector.

```
Vector();
Vector(int size);
Vector(int size,int incr);
```

Konstruktor pertama adalah konstruktor default yang menginialisasikan ukuran sebanyak 10 elemen. Konstruktor kedua yaitu menentukan jumlah elemen yang akan digunakan. Konstruktor yang ketiga yaitu menentukan ukuran awal dan apabila ukurannya full maka akan ditambah lagi sebanyak incr. Dibawah ini beberapa method yang disediakan oleh class Vector.

Method	Keterangan
addElement(<object>)	Digunakan untuk menambahkan data object ke dalam Vektor.
elementAt(<index>)	Method ini berfungsi untuk mengambil elemen berdasarkan nomor index yang dimasukan.
lastElement()	Berfungsi untuk mengambil nilai berupa object yang paling terakhir di tambahkan dari sebuah object vektor.
firstElement()	Berfungsi untuk mengambil nilai berupa object pertama yang di tambahkan dari sebuah object vektor.
clear()	Digunakan untuk menghapus seluruh elemen yang tersimpan dalam object vector.
isEmpty()	Memeriksa apakah verktor yang digunakan berisi elemen atau tidak. Jika ada data maka akan mengembalikan nilai boolean berupa false.

2.3.2 ArrayList

ArrayList hampir mirip seperti vektor. Pada JDK 1.1 ArrayList 3-4 kali lebih cepat dari pada Vektor, sedangkan pada JDK 1.1.4 ArrayList 40 kali lebih cepat daripada vektor. Pembuatan Object ArrayList dapat dilihat seperti dibawah ini.

```
ArrayList al=new ArrayList();
```

Tidak seperti array dimana untuk mengambil ukuran menggunakan <array>.length, ArrayList menggunakan al.size(). Dibawah ini adalah beberapa method-method yang dapat digunakan pada ArrayList.

Method	Keterangan
add(<object>)	Digunakan untuk menambahkan data object ke dalam ArrayList.
add(<index>,<object>)	Method ini menyediakan dua parameter untuk menambahkan sebuah object dengan menentukan nomor index elemennya.
get(<index>)	Method get adalah method yang disediakan oleh ArrayList untuk mengambil sebuah object berdasarkan nomor indexnya.
remove(<index>)	Mothod ini berfungsi untuk menghapus elemen ArrayList berdasarkan nomor indexnya.
isEmpty()	Digunakan untuk memeriksa apakah object ArayList yang dibuat kosong atau tidak.
clear()	Menghapus semua elemen yang ada di dalam object ArrayList.

2.3.3 LinkedList

LinkedList mengimplementasi interface List. LinkedList juga memperbolehkan elemennya menggunakan null. Pertama yang harus dilakukan adalah membuat object LinkedList seperti dibawah ini:

```
LinkedList ll=new LinkedList();
```

Ada beberapa method yang disediakan class LinkedList antara lain adalah sebagai berikut:

Method	Keterangan
add(<object>)	Digunakan untuk menambahkan data object ke dalam LinkedList.
size()	Mengambil jumlah elemen pada object LinkedList
get(<index>)	Mengambil nilai elemen berdasarkan nomor index yang ada pada object LinkedList
addFirst()	Menambahkan object pada elemen awal
addLast()	Menambahkan object pada elemen akhir
getFirst()	Mengambil nilai elemen pertama
getLast()	Mengambil nilai elemen terakhir
clear()	Menghapus semua nilai elemen pada object LinkedList
remove()	Method remove tanpa parameter akan menghapus elemen pertama
remove(<index>)	Parameter akan menunjukan elemen mana yang akan dihapus

2.4 Thread

2.4.1 Membuat dan Memulai Thread

Sebuah thread dapat dengan mudah dibuat dan jalankan dengan cara membuat instance dari object Thread dan memanggil method start().

```
Thread myThread= new Thread();
myThread.start();
```

Saat menjalankan kode diatas tentu saja tidak akan ada yang terjadi selain aplikasi jalan dan langsung mati. Ketika menjalankannya JVM akan membuat sebuah thread baru dan secara default akan memanggil method run(), dalam hal ini run() kosong sehingga thread akan langsung mati. Berikut ini adalah cara memberikan tugas kepada thread :

1) meng-extends class Thread
2) meng-implements class Runnable

3) menggunakan innerclass tanpa nama

2.4.2 Menurunkan Class Thread

Cara cepat memberikan tugas kepada thread adalah dengan mengoverride method run().

```
public class MyThread extends Thread {
      public void run(){
            System.out.println("saya adalah thread");
      }
}
```

Berikutnya yang perlu di lakukan adalah membuat instance dari object Thread dan memanggil method start() sama seperti bahasan sebelumnya.

```
Thread myThread = new  MyThread();
myThread.start();
```

2.4.3 Mengimplementasi Class Runnable

Meng-extends class Thread sangat mudah, tetapi tidak menutup kemungkinan seorang programer tidak ingin menulis class baru setiap kali ingin menggunakan thread. Contoh nya ketika sedang mengextends class JFrame dalam aplikasi GUI dan ingin menggunakan thread, solusinya dapat dipecahkan dengan menggunakan interface Runnable().

```
public class MyThread extends JFrame implements Runnable {
      public MyThread(){
            Thread t=new Thread(this);
            t.start();
      }
      public void run(){
            System.out.println("saya adalah thread");
      }
}
```

Dalam kode diatas MyThread membuat instance dari object thread pada konstruktor dengan object Runnable (dalam hal ini class MyThread sendiri), dan selanjutnya memulainya.

2.4.4 Menggunakan Inner Class Tanpa Nama

Sebuah thread dapat dibuat tanpa harus membuat class baru ataupun mengimplements Runnable. Pembuatannya dapat dilakukan dengan dengan cara membuat inner class.

```
new Thread() {
    public void run() {
        System.out.println("saya adalah Thread");
    }
}.start();
```

Cara diatas tidak lazim digunakan karena dapat beresiko terhadap sebuah program apabila kode dalam method run() semakin besar.

2.4.5 Menggunakan Method Sleep

Suatu saat sebuah thread perlu dihentikan untuk sementara waktu untuk keperluan tertentu, untuk itu dapat menggunakan method sleep().

```
try{
      Thread.sleep(1000);
}catch(Exception e){}
```

Kode diatas akan membuat thread berhenti selama 1 detik (1000 milidetik). Selama thread dalam kondisi sleep, thread tidak memerlukan kinerja dari cpu. Kondisi sleep pada sebuah thread akan dimanfaatkan untuk menjalankan thread lainya.

2.4.6 Menghentikan Thread

Ada dua cara yang bisa dilakukan untuk menghentikan sebuah thread. Pertama adalah menggunakan flag (tanda) dan yang kedua adalah menggunakan method interrupt(). Apabila menggunakan method stop(), maka bersiap-siaplah melihat program tak terkendali atau bahkan mati. Method stop() mengakibatkan gangguan terhadap thread lain.

2.4.7 Menggunakan Flag(Tanda)

Cara yang paling banyak digunakan untuk menghentikan sebuah thread adalah menggunakan variabel penanda. Thread dapat memeriksa variabel tersebut secara berkala untuk memastikan kondisi berhenti.

```
public class MyThread extends JFrame implements Runnable {

      private volatile boolean done=false;
      public MyThread(){
            Thread t=new Thread(this);
            t.start();
      }
      public void run(){
            while(!done){
                  System.out.println("saya adalah thread");
            }
      }
      Public void setDone(){
            done=true;
      }
```

Dengan kode diatas secara berkala thread akan memeriksa kondisi penanda, ketika penanda bernilai true maka thread tidak lagi dijalankan.

2.4.8 Menggunakan Interupsi

Cara lain yang dapat digunakan adalah dengan menginterrupsi sebuah thread. Cara ini biasa digunakan pada sistem blocking. Sistem blocking adalah sistem yang mem-block jalannya sistem sampai sebuah kunci di lepaskan. Contoh sistem blocking salah satunya pola penerimaan koneksi jaringan (pembahasan jaringan akan di jelaskan pada bab berikutnya).

Pada teknik penghentian menggunakan flag, terdapat sebuah kondisi yang memungkinkan sistem kehilangan/terlambat menerima informasi. Contohnya ketika sebuah thread sedang memproses suatu perintah, tiba-tiba flag di set. Kondisi ini setidaknya membuat thread harus menyelesaikan proses tersebut dan baru akan menghentikan proses pada loop berikutnya.

```
public class MyThread extends JFrame implements Runnable {
      private volatile boolean done=false;
      Thread t;

      public MyThread(){
            t=new Thread(this);
            t.start();
      }
      public void run(){
            while(!t.isInterrupted()){
                  System.out.println("saya adalah thread");
            }
      }
}
```

Dengan method interrupt(), thread akan di paksa berhenti dan mengabaikan proses yang sedang berjalan. Penggunaan method interrupt() dapat dengan mudah dilakukan dengan memanggil method tersebut.

2.4.9 Sinkronisasi Data

Apa yang terjadi ketika dua thread menggunakan variabel yang sama? Bisa saja salah satu thread mencoba mengkases data yang telah di hapus oleh thread lainya. Dalam hal ini kedua thread harus di sinkronisasi sehingga masing-masing thread tidak kehilangan informasi. Sinkronisasi berfungsi mengunci sebuah object, ketika sebuah object terkunci maka tidak ada yang bisa dilakukan terhadap object tersebut sebelum kunci di lepas.

Asumsikan bahwa buah thread yang mengakses variabel dari object yang sama. Suatu saat thread a mencoba mengambil data dalam variabel tersebut , sementara thread b mencoba mengubah data tersebut .

```
public class GamePlayer{
    public int getLives()
    {  return lives; }
    public void setLives(int l)
    {  lives = l; }
    private int lives;
}
```

Asumsikan kedua buah thread ingin mengakses object gamePlayer yang sama.

```
//Thread A
int lives=gamePlayer.getLives();
gamePlayer.setLives(lives-1);

//Thread B
int lives=gamePlayer.getLives();
gamePlayer.setLives(lives-1);
```

Kondisi yang diinginkan adalah variabel lives mendapatkan pengurangan 2 kali. Namun ketika Thread A dan B mengakses variabel Lives dalam waktu yang bersamaan kemudian menguranginya maka yang terjadi variabel Lives hanya akan mengalami satu pengurangan. Untuk memperbaiki kondisi tersebut dapat menggunakan kata kunci synchronized.

```
//Thread A
Synchronized(gamePlayer){
        int lives=gamePlayer.getLives();
        gamePlayer.setLives(lives-1);
}
//Thread B
Synchronized(gamePlayer){
        int lives=gamePlayer.getLives();
        gamePlayer.setLives(lives-1);
}
```

2.4.10 Menggunakan Method Wait dan Notify

Sebelum bahasan ini, telah dipelajari bagaimana thread bekerja secara mandiri. Pada kenyataan suatu saat seorang programer perlu mengkomunikasikan dua atau lebih thread. Katakanlah dua buah thread digunakan untuk komunikasi jaringan, thread A dan thread B. Thread A akan mengirimkan pesan ketika thread B selesai mengisi pesan.

```
// Thread A
public void waitForMessage() {
    while (hasMessage == false) {
        Thread.sleep(100);
    }
}
// Thread B
public void setMessage(String message) {.....
    hasMessage = true;
}
```

Perhatikan kode diatas. Secara berkala thread A akan mengecek kondisi hasMessage. Jika hasMessage tidak di set pada pengecekan terkini maka berikutnya thread akan sleep selama 100 ms. Kode tersebut memang bekerja, tetapi apa yang terjadi jika hasMessage di set ketika Thread A dalam keadaan sleep? Thread A akan kehilangan atau setidaknya terlambat menerima informasi. Dengan method wait dan notify hal ini dapat di selesaikan. Thread a

cukup menunggu sampai dengan thread B mengirimkan sinyal bahwa pesan telah siap.

```
// Thread A
public synchronized void waitForMessage() {
    try {
        wait();
    }
    catch (InterruptedException ex) { }
}
// Thread B
public synchronized void setMessage(String message) {
    ...
    notify();
}
```

Seperti yang terlihat diatas apabila method waitForMessage() di panggil maka secara otomatis program akan berhenti secara mendadak untuk menunggu method setMessage() dipanggil atau dijalankan.