UNIVERSITAS KOMPUTER INDONESIA

Java Generic & Collection

Unikom Programming Team

Eko Kurniawan Khannedy

5/1/2010

Java Generic

Kenapa Pemrograman Generic?

Generic Programming artinya kode yang dapat digunakan oleh beberapa objek yang tipenya berlainan. Misal kita memiliki kelas *KoleksiString*, *KoleksiInteger* dan *KoleksiDouble*, dengan menggunakan *Generic Programming*, kita dapat membuat kelas *Koleksi* yang dapat menampung data *String*, *Integer* maupun *Double*.

Kelas Generic

Misal sebelum menggunakan *Generic Programming*, kita akan membuat kelas *Koleksi* untuk menampung *String* seperti ini :

```
package khannedy.unikom.generic.data;

public class KoleksiString {

   private String data;

   public String getData() {
      return data;
   }

   public void setData(String data) {
      this.data = data;
   }
}
```

Untuk menampung data Integer :

```
package khannedy.unikom.generic.data;

public class KoleksiInteger {

   private int data;

   public int getData() {
      return data;
   }

   public void setData(int data) {
      this.data = data;
   }
}
```

Andai kita membutuhkan banyak data yang akan di tampung dalam kelas Koleksi, maka mau tidak mau kita harus membuat setiap kelas Koleksi untuk tipe tertentu. Hal ini sangat melelahkan jika dilakukan :(

Dengan demikian, diperlukan *Generic Programming* untuk mengatasi masalah tersebut. Jika kita menggunakan *Generic Programming*, maka kita hanya cukup membuat satu buat kelas Koleksi, dan kelas tersebut dapat menampung seluruh objek dengan tipe data yang berbeda.

```
package khannedy.unikom.generic.data;

public class Koleksi<T> {

   private T data;

   public T getData() {
      return data;
   }

   public void setData(T data) {
      this.data = data;
   }
}
```

Dengan menggunakan *Generic Programming* diatas, kita dapat membuat Koleksi yang dapat menampung **object** dengan tipe apapun. Namun perlu diingat, karena Generic Programming hanya dapat menggunakan **object**, sehingga **data primitive** harus diimplementasikan dalam objek, misal *int* menjadi *Integer*, *double* menjadi *Double*.

```
package khannedy.unikom.generic;

import
khannedy.unikom.generic.data.Koleksi;

public class Main {

   public static void main(String[] args) {
```

```
        Koleksi<String> a = new Koleksi<String>();
        a.setData("String");

        Koleksi<Integer> b = new
Koleksi<Integer>();
        b.setData(1);
    }
}
```

Pendeklarasian Generic Programming ditandai dengan simbol yang diawali tanda **(<)** dan diakhiri tanda **(>)**, setelah nama Kelas :

```
class Koleksi<T>
```

Dari kode diatas, berarti kita membuat simbol generic dengan simbol **T**, sehingga **T** dianggap tipe data dalam lingkup kelas tersebut. Saat pendeklarasian objek *Koleksi*, maka kita harus menentukan tipe T tersebut :

```
Koleksi<String> a = new Koleksi<String>();
```

Kode diatas berarti kita mengganti simbol **T** dengan tipe data *String*.

Metode Generic

Selain dapat diimplementasikan dalam kelas, Generic Programming juga dapat diimplementasikan dalam sebuah metode.

```
package khannedy.unikom.generic.data;

public class Utilitas {

    public static <T> T ambilTengah(T[] data) {
        return data[data.length / 2];
    }
}
```

Untuk membuat generic dalam metode, simbol dideklarasikan sebelum *return value*, misal :

```
public <T> T ambilTengah(T[] data)
```

Atau :

```
public <T> void kosongkan(T[] data)
```

Perlu diingat jika ***Generic Programming dalam sebuah Metode, hanya berlaku untuk metode tersebut, tidak berlaku untuk metode yang lain dalam kelas yang sama***, sehingga dibawah ini adalah salah :

```
package khannedy.unikom.generic.data;

public class Utilitas {

    public <T> T ambilTengah(T[] data) {
        return data[data.length / 2];
    }

    // simbol T tidak dikenal
    public T ambilAwal(T[] data){
        return data[0];
    }
}
```

Pewarisan

Dalam *Generic Programming* pun dikenal dengan pewarisan, pewarisan ini digunakan untuk membatasi masukkan data yang dapat digunakan dalam kode *generic*. Secara default simbol pada *Generic Programming* merupakan turunan kelas *Object*, sehingga tipe data apapun dapat masuk ke kode generic tersebut.

```
public class Koleksi<T extends Object>
```

Dan jika kita ingin membatasi kode generic yang kita buat untuk **kelas tertentu dan turunannya**, maka kita bisa memanfaatkan fitur pewarisan pada *Generic Programming*.

Misal, kita memiliki kelas *Musisi*, dan kelas *MusisiBerbakat*, dimana kelas *MusisiBerbakat* tersebut adalah turunan dari kelas *Musisi*.

```
package khannedy.unikom.generic.data;

public class Musisi {

   public void nyanyi() {
      // nyanyi
   }
}
```

```
package khannedy.unikom.generic.data;

public class MusisiBerbakat
      extends Musisi{

   public void dansa() {
      // dansa
   }
}
```

Jika kita akan membatasi kode *generic* yang akan kita buat, hanya dapat digunakan untuk kelas *Musisi* dan keturunannya, maka kita dapat membuatnya seperti ini :

```
package khannedy.unikom.generic.data;

public class Koleksi
      <T extends Musisi> {

   private T data;

   public T getData() {
      return data;
   }

   public void setData(T data) {
      this.data = data;
   }
}
```

Dengan demikian, maka kelas *Koleksi* yang baru, hanya dapat digunakan oleh kelas Musisi dan keturunannya. ***Pewarisan tidak hanya dapat diterapkan pada kelas generic, dapat juda di metode generic***.

Selain membatasi, dengan pewarisan kita juga dapat memanggil metode dari kelas yang kita tentukan. Misal pada kode diatas, kelas *Musisi* memiliki metode nyanyi(), dengan mewarisi kelas *Musisi* dalam kode *generic* yang kita buat, maka kita dapat memanggil seluruh metode yang ada pada kelas *Musisi* dalam kode *generic* kita :

```
package khannedy.unikom.generic.data;

public class Koleksi
      <T extends Musisi> {

   private T data;

   public T getData() {
      return data;
   }

   public void setData(T data) {
      this.data = data;
   }

   public void nyanyiEuy(){
      if(this.data != null){
         this.data.nyanyi();
      }
   }
}
```

Jika kita tidak mewarisi sebuah kelas, maka metode yang hanya dapat dipanggil hanyalah metode yang ada pada kelas Object.

Java Collection

Java *Collection* merupakan kumpulan-kumpulan kelas yang digunakan sebagai kelas struktur data, seperti array, linkedlist, stack, tree dan lain-lain. ***Seluruh kelas Java Collection merupakan turunan kelas java.lang.Iterable***. Sehingga dapat digunakan dalam perulangan *foreach*.

```
package khannedy.unikom.generic;

public class Main {

   public static void main(String[] args) {
      String[] data = new String[10];
      for (String a : data) {
         // manipulasi a
      }
   }
}
```

Seluruh kelas-kelas Java *Collection* merupakan kelas *generic*, sehingga dapat digunakan untuk menampung objek dengan berbadai tipe data.

ArrayList

ArrayList merupakan struktur data *Array* yang dapat berkembang kapasitasnya secara otomatis, sehingga berbeda dengan *Array* biasa yang ukuran *Array*-nya terbatas saat dideklarasikan.

```
package khannedy.unikom.generic;

import java.util.ArrayList;

public class Main {

   public static void main(String[] args) {
      ArrayList<String> list =
               new ArrayList<String>();
   }
}
```

Kode diatas, berarti kita membuat *ArrayList* yang digunakan untuk menampung data String. Untuk menambah data yang ada dalam ArrayList, kita dapat menggunakan metode **add(T data)** :

```
list.add("Data Baru");
```

Dan untuk mengubah data yang telah ada dalam ArrayList, kita bisa menggunakan metode **set(int index, T dataBaru)** :

```
list.set(0, "Data Baru");
```

Index dalam *ArrayList*, sama dengan pada Array biasa, dimulai dari 0 dan diakhiri dengan panjangArray-1.

Untuk mendapatkan data yang ada dalam *ArrayList*, kita dapat menggunakan metode **get(int index)** :

```
String data = list.get(10);
```

Artinya kita menggambil data pada *ArrayList* yang ke-11, hal ini dikarenakan *index* diawali dari 0.

Untuk menghapus data dalam *ArrayList*, kita dapat menggunakan metode **remove(int index)** :

```
list.remove(8);
```

Sama seperti metode get() dan set(), index pada metode remove() pun diawali dengan index ke-0.

Untuk mendapatkan total data yang ada dalam *ArrayList*, kita dapat menggunakan metode size() :

```
int total = list.size();
```

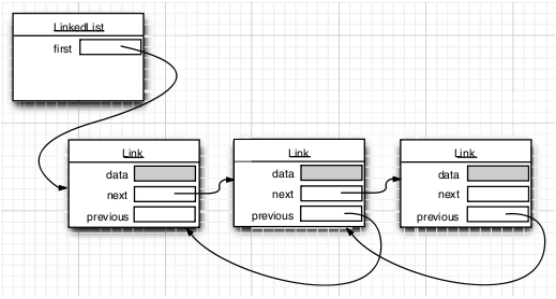
Kekurangan *ArrayList* adalah, saat kita menghapus data maka data pada index setelah data yang dihapus, akan diubah indexnya ke index sebelumnya. Misal jika kita memiliki 100 data dalam ArrayList, lalu kita menghapus data *index* ke 20, maka proses yang terjadi adalah :

1. Hapus data ke 21
2. Tempatkan data ke 22 menjadi ke 21
3. Tempatkan data ke 23 menjadi ke 22
4. Tempatkan data ke 24 menjadi ke 23
5. Dan seterusnya hingga data terakhi

Sehinga ArrayList tidak cocok jika digunakan untuk menampung data yang banyak menlakukan proses penghapusan data.

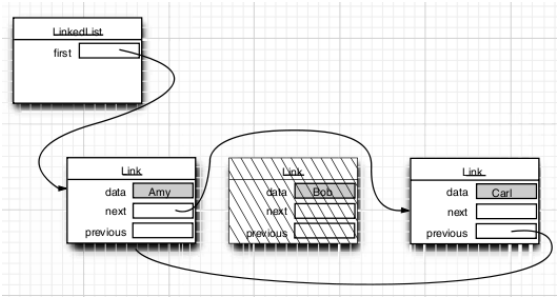
LinkedList

LinkedList merupakan struktur data yang setiap data nya memiliki pointer ke data berikutnya dan data sebelumnya. Berbeda dengan ArrayList, pada LinkedList data tidak disimpan pada index.



LinkedList merupakan sturktur data yang dapat digunakan sebagai solusi kekurangan pada ArrayList, yaitu pada saat penghapusan data. Pada LinkedList, penghapusan data hanya melakukan 3 langkah, misal kita menghapus data ke 5 :

1. Hapus data ke-5
2. Ubah pointer next data ke-4 menjadi mengacu ke data ke-6
3. Ubah pointer prev data ke-6 menjadi mengacu ke data ke-4



Seluruh operasi yang ada pada LinkedList hampir sama dengan yang ada pada ArrayList. Sehingga penggunaanya cukup sama dengan ArrayList.

```
package khannedy.unikom.generic;

import java.util.LinkedList;

public class Main {

   public static void main(String[] args) {
      LinkedList<String> list =
            new LinkedList<String>();

      // tambah data
      list.add("Data Baru");

      // ubah data
      list.set(0, "Data Baru Lagi");

      // hapus data
      list.remove(0);

      // total data
      int total = list.size();
   }
}
```

Namun yang menjadi permasalahan dalam LinkedList yaitu saat proses, pencarian. Dikarenakan pada LinkedList, data tidak memiliki index, maka saat terjadi proses pencarian, maka dilakukan secara *sequensial*, sehingga dapat memperlambat proses pencarian.

Set

Set merupakan struktur data yang digunakan untuk menampung data yang datanya tidak boleh ada yang sama. Jika data yang sama, maka data hanya akan ditampung sekali.

Set merupakan *Interface*, sehingga dibutuhkan implementasi sebuah kelas untuk menggunakan *Set*. Dalam Java Collection, implementasi defaultnya adalah *HashSet*, dimana *HashSet* melakukan pengecekan duplikasi berdasarkan metode **equals()** dan **hashCode()**.

Secara default **hashCode()** setiap objek itu unik dan defaulnya metode **equals()** membandingkan objek pada memori, jika lokasi memorinya sama, maka dianggap sama, namun kita dapat membuat implementasi hashCode() sendiri, misal untuk Mahasiswa, mahasiswa disebut sama jika memiliki Nim yang sama, Mahasiswa tidak dianggap sama jika Nama nya sana, namun Nim nya berbeda, sehingga kita dapat membuat kelas Mahasiswa seperti berikut :

```
package khannedy.unikom.generic.data;

public class Mahasiswa {

   private int nim;

   private String nama;

   public String getNama() {
      return nama;
   }

   public void setNama(String nama) {
      this.nama = nama;
   }

   public int getNim() {
      return nim;
   }

   public void setNim(int nim) {
      this.nim = nim;
   }

   @Override
   public boolean equals(Object obj) {
      Mahasiswa m = (Mahasiswa) obj;
      return m.nim == nim;
   }

   @Override
   public int hashCode() {
      return nim;
   }
}
```

Dengan demikian, maka kelas Mahasiswa, hashCode()–nya akan dicek berdasarkan Nim :

```
package khannedy.unikom.generic;

import java.util.HashSet;
import java.util.Set;
import
khannedy.unikom.generic.data.Mahasiswa;

public class Main {

   public static void main(String[] args) {
      Set<Mahasiswa> set =
            new HashSet<Mahasiswa>();

      Mahasiswa a = new Mahasiswa();
      a.setNim(1);
      a.setNama("Eko Kurniawan Khannedy");
      set.add(a);

      Mahasiswa b = new Mahasiswa();
      b.setNim(1);
      b.setNama("Tukul Arwana");
      set.add(b);

      for (Mahasiswa m : set) {
         System.out.println("Nim : "
               + m.getNim());
         System.out.println("Nama : "
               + m.getNama());       }
   }
}
```

Maka hasilnya adalah :

```
Nim : 1
Nama : Eko Kurniawan Khannedy
```

Artinya hanya mahasiswa dengan nama **Eko Kurniawan Khannedy** saja yang masuk, sedangkan **Tukul Arwana** tidak masuk, kenapa? Hal ini dikarenakan memiliki nim yang sama yaitu 1.

TreeSet

TreeSet merupakan struktur data dimana data yang ditampungnya akan otomatis di urutkan berdasarkan Comparator.

```
Interfaces Comparator<T>() {
  public int compare(T o1, T o2)   {
  }
}
```

Dalam Comparator, kita diwajibkan membandingkan 2 buat objek dengan tipe data yang sama. Jika o1 lebih besar dari o2 maka return nya harus positif, jika lebih kecil maka return nya harus negatif, jika sama maka return nya harus 0.

Contoh, pada kelas Mahasiswa sebelumnya, kita akan membandingkan tiap mahasiswa berdasarkan nim nya, namun dibandingkannya secara terbaik, artinya data akan diurutkan secara **DESC**.

```
package khannedy.unikom.generic;

import java.util.Comparator;
import
khannedy.unikom.generic.data.Mahasiswa;

public class PembandingMahasiswa
      implements Comparator<Mahasiswa> {

   public int compare(Mahasiswa o1,
                      Mahasiswa o2) {
      if (o1.getNim() > o2.getNim()) {
         return -1;
      } else if (o1.getNim() < o2.getNim()) {
         return 1;
      } else {
         return 0;
      }
   }
}
```

Dan jika kita implementasikan dalam TreeSet :

```
package khannedy.unikom.generic;

import java.util.TreeSet;
import
khannedy.unikom.generic.data.Mahasiswa;

public class Main {

   public static void main(String[] args) {
      PembandingMahasiswa pembanding =
            new PembandingMahasiswa();

      TreeSet<Mahasiswa> set =
            new TreeSet<Mahasiswa>
            (pembanding);

      Mahasiswa a = new Mahasiswa();
      a.setNim(1);
      a.setNama("Eko Kurniawan Khannedy");
      set.add(a);

      Mahasiswa b = new Mahasiswa();
      b.setNim(2);
      b.setNama("Tukul Arwana");
      set.add(b);

      for (Mahasiswa m : set) {
         System.out.println("Nim : "
               + m.getNim());
         System.out.println("Nama : "
               + m.getNama());
      }
   }
}
```

Maka hasilny adalah :

```
Nim : 2
Nama : Tukul Arwana
Nim : 1
Nama : Eko Kurniawan Khannedy
```

Collection yang Lain

Ada banyak sekali Collection dalam Java, seperti Stack, Vector, Map, Queue dan lain-lain.

Resource Belajar

- **http://java.sun.com/docs/books/tutorial/extra/generics/index.html**
- **http://java.sun.com/docs/books/tutorial/collections/index.html**

Ayo Berkomunitas!!!

- Milis Unikom Programming Team : **http://tiny.cc/milis-unikom-prog-team**
- Facebook Programming Team : **http://tiny.cc/fb-unikom-prog-team**
- OpenSource University Meetup : **http://osum.sun.com/**
- OpenSource University Meetup UNIKOM : **http://osum.sun.com/group/unikom**