Bab 8. Dasar-Dasar OOP

Pemrograman Berorientasi Obyek

Politeknik Elektronika Negeri Surabaya

2007

Overview

- Introduction
- Encapsulation
  - Information Hiding
  - Interface to access data
- Constructor
- Overloading Constructor
- Package
- Import
- Keyword *this*

Introduction

- Ada 2 macam tipe data dalam Java :
  - Primitive type :
    - int, short, byte, long, float, double, char, boolean
    - Cukup dilakukan *variable declaration* (deklarasi) :

```
int bilangan;
char huruf;
```

  - Reference type :
    - Array, String, class
    - Harus dilakukan *variable declaration & object creation* (deklarasi variabel & pembuatan objek)
    - Deklarasi untuk menyiapkan variabel yang akan menyimpan *reference address*-nya, sedangkan pembuatan objek menggunakan keyword *new* untuk mengalokasikan memori yang akan digunakan untuk menyimpan value, misal :

```
int nilai[];
nilai = new int[30];
```

→

```
int nilai[] = new int[30];
```

Encapsulation

- Tiga fitur yang dimiliki oleh OOP, yaitu :
  - Encapsulation
  - Inheritance
  - Polymorphism
- Encapsulation adalah suatu cara untuk menyembunyikan implementasi detil dari suatu class dalam rangka menghindari akses yang ilegal.
- Encapsulation mempunyai dua hal mendasar, yaitu :
  - *information hiding*
  - *interface to access data*

Encapsulation

- Misalnya saja kita mempunyai sebuah class seperti dibawah ini :

```
public class MyDate {
  public int day;
  public int month;
  public int year;
}
```

- Selanjutnya kita punya object d yang merefer ke class `MyDate`

```
MyDate d = new MyDate();
```

- Maka kita bisa mengakses semua atribut maupun method yang dimiliki oleh class `MyDate` secara langsung dari objek `d` dengan menggunakan operator titik. Namun hal ini berpeluang membuat kesalahan, misalnya :

```
d.day = 32;                    //invalid day
d.month = 2; d.day = 30;       //plausible but wrong
d.day = d.day + 1;             //no check for wrap around
```

Encapsulation

- Untuk menghindari kasus seperti di atas, kita bisa menyembunyikan infomasi dari sebuah class sehingga anggota-anggota class tersebut (dalam hal ini atribut-atributnya) tidak dapat diakses dari luar, caranya dengan :
  - memberikan akses control `private` ketika mendeklarasikan atribut atau method → *information hiding*
  - menyediakan akses pembacaan melalui method `getXXX()` yang sering disebut sebagai *getters* dan akses penyimpanan melalui method `setXXX()` yang sering disebut sebagai *setter*. Method-method ini akan memungkinkan class untuk memodifikasi data internal, namun yang lebih penting lagi, untuk memverifikasi perubahan yang diinginkan adalah dengan nilai yang valid → *interface to access data*

Encapsulation

```
public class MyDate {
  private int day;
  private int month;
  private int year;

public void setDay (int dd) {
  if(month==2) {
    if (dd > 28)
      System.out.println(“Invalid day..”);
    else
      day = dd;
  }else {
    if(dd > 31)
       System.out.println(“Invalid day..”);
    else
      day = dd;
  }
}
```

```
public int getDay() {
        return day;
}
```

```
MyDate d = new MyDate();
d.setDay(32);
     //invalid day, messages invalid
d.setMonth(2); d.setDay(30);
     //plausible but wrong, setDay messages invalid
d.setDay(getDay() + 1);
     //this will check first if wrap around needs to occur
```

Encapsulation

```
public class Siswa {
  public int nrp;
  public String nama;

  public void info() {
   System.out.println(nrp+" "+nama+" "+"adl siswa PENS");
  }
```

```
public class isiData {
       public static void main(String args[ ]) {
              Siswa s = new Siswa();
              s.nrp=50;
              s.nama="Andi";
              s.info();
         }
}
```

Encapsulation

- Pada kasus program untuk pengisian NRP dalam `class Siswa`, dimisalkan nilai NRP berada dalam range 1-10.
- Jika kita tidak melakukan encapsulation pada `class Siswa`, maka data `nrp` yang kita masukkan tentunya akan diperbolehkan nilai dalam range tipe data `int`.
- Oleh karena itu, *information hiding* terhadap atribut `nrp` sangat diperlukan, sehingga `nrp` tidak bisa diakses secara langsung.
- Selanjutnya bagaimana cara mengakses atribut nrp itu untuk memberikan atau mengubah nilai? Nah, di saat ini kita memerlukan suatu *interface to access data*, yang berupa method dimana di dalamnya terdapat implementasi untuk mengakses data `nrp`.

Encapsulation

Hasil modifikasi tampak sbb:

```
public class Siswa {
   private int nrp;
   public String nama;

   public void setNrp(int n) {
      if(n>=1 && n<=10)
         nrp=n;
      else
         System.out.println("Di luar range....";
   }

   public int getNrp() {
      return nrp;
   }

   public void info() {
      System.out.println(nrp+" "+nama+" "+"adl siswa PENS");
   }
}
```

Constructor

- Contructor adalah bagian dari class yang mirip dengan method (memiliki *parameter list*, tapi tidak memiliki *return value*).
- Constructor merupakan bagian yang pertama kali dijalankan pada saat pembuatan suatu obyek.
- Constructor mempunyai ciri yaitu :
  - mempunyai nama yang sama dengan nama class-nya
  - tidak mempunyai return value (seperti void, int, double dll)

Constructor

- **Setiap class pasti mempunyai constructor**.
- Jika kita membuat suatu class tanpa menuliskan constructornya, maka kompiler dari Java akan menambahkan sebuah constructor kosong., misalnya sebuah class Siswa seperti dibawah ini:

```
public class Siswa {

}
```

- Programmer tidak mendeklarasikan constructornya secara eksplisit, maka ketika proses kompilasi, kompiler Java akan menambahkan constructor kosong sehingga class Siswa tersebut akan tampak sebagai berikut :

```
public class Siswa {
        public Siswa() {
        }
}
```

Constructor

- Karena constructor adalah method yang pertama kali dijalankan pada saat suatu obyek dibuat, maka constructor sangat berguna untuk **menginisialisasi data member**.
- Misalnya saja pada class Siswa diatas dapat dilakukan inisialisasi nrp di dalam constructor yang dideklarasikan secara ekslipit, seperti yang tampak di samping ini

```
public class Siswa {
   private int nrp;
   public Siswa() {
      nrp=0;
   }
}
```

- Kita juga dapat menginisialisasi suatu data member dengan nilai yang diinginkan oleh user dengan cara memasukkannya pada parameter constructor. Misalnya class Siswa diatas dapat dimodifikasi sbb :

```
public class Siswa {
   private int nrp;
   public Siswa(int n) {
      nrp=n;
   }
}
```

- Dengan mendeklarasikan constructor seperti itu, user dapat membuat obyek dengan menginisialisasi `nrp` sesuai yang ia kehendaki, misalnya saja seperti berikut :

```
Siswa t = new Siswa();
```

```
Siswa s = new Siswa(5);
```

Overloading Constructor

- Suatu class dapat mempunyai lebih dari 1 constructor dengan syarat **daftar parameternya tidak boleh ada yang sama.**
- Misalnya saja kita ingin menginisialisasi data member nrp dengan 2 cara, pertama, jika user tidak memberikan nilai inisialisasi nrp, maka nrp akan diset dengan nilai 0; kedua, jika user ingin menginisialisasi nrp sesuai dengan nilai yang diinginkan, maka nrp akan diisi sesuai nilai yang diinginkan oleh user. Sehingga class Siswa diatas dapat kita deklarasikan 2 buah constructor seperti yang tampak sebagai berikut :

```
public class Siswa {
        private int nrp;

        public Siswa() {
                nrp=0;
        }

        public Siswa(int n) {
                nrp=n;
        }
}
```

package

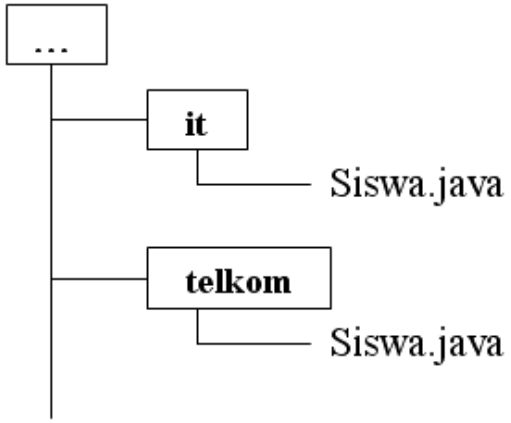
- Package adalah suatu cara untuk memenej class-class yang kita buat. Package akan sangat bermanfaat jika class-class yang kita buat sangat banyak sehingga perlu dikelompokkan berdasarkan kategori tertentu.
- Misalnya saja kita mempunyai 2 buah class Siswa, dimana yang pertama adalah class Siswa untuk mahasiswa jurusan IT dan yang kedua adalah class Siswa untuk mahasiswa Telkom. Kita tetap dapat mendeklarasikan 2 class tersebut dengan nama Siswa, dengan cara mendeklarasikannya package masing-masing class seperti yang tampak di bawah ini:

```
package it;

public class Siswa {
…
}
```

```
package telkom;

public class Siswa {
…
}
```

package

- Yang perlu kita perhatikan pada saat deklarasikan package, bahwa class tersebut harus disimpan pada suatu direktori yang sama dengan nama package-nya.
- Berkenaan dengan class Siswa diatas, class Siswa pada package it harus disimpan pada direktori it, dan class Siswa pada package telkom harus disimpan pada direktori telkom.



import

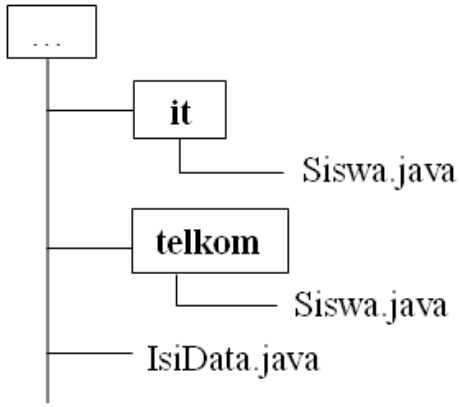
- Suatu class dapat meng-import class lainnya sesuai dengan nama package yang dipunyainya. Misalnya saja kita dapat meng-import class Siswa.java dalam package it dengan mendeklarasikan kata kunci import.

```
import it.Siswa;

public class IsiData {
        …
}
```

- Jika kita ingin meng-import semua class yang ada pada package it, maka kita dapat mendeklarasikannya :

```
import it.*;
```

- Satu hal yang perlu kita ketahui, pada saat kita ingin meng-import suatu class dalam suatu package, pastikan letak package tersebut satu direktori dengan class yang ingin meng-import. Dalam contoh diatas, representasi direktori akan tampak seperti berikut :
- Jika letak package tersebut tidak satu direktori dengan class yang ingin meng-import, maka letak direktori package itu haruslah terdaftar dalam CLASSPATH.

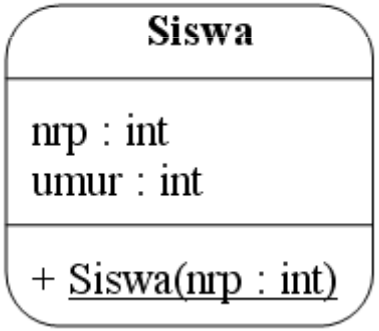


Keyword *this*

- Keyword *this* sangat berguna untuk menunjukkan suatu member dalam class-nya sendiri.
- *This* dapat digunakan baik untuk data member maupun untuk function member, serta dapat juga digunakan untuk constructor.
- *this* ➔ nama class ybs
- Adapun format penulisannya adalah :
  - `this.data_member` → merujuk pada data member
  - `this.function_member()` → merujuk pada function member
  - `this()` → merujuk pada constructor

Keyword *this*

- Misalnya sebuah class diagram sbb :
- Pada saat membaca class diagram diatas, maka kita merasa kesulitan untuk memahami sesungguhnya nilai variabel `n` pada parameter constructor itu akan dipakai untuk menginisialisasi `nrp` atau `umur`.



- Untuk lebih memudahkan, kita dapat menuliskan class diagram yang lebih mudah dimengerti seperti yang tampak di samping ini :
- Dengan class diagram diatas, kita lebih mudah memahami bahwa nilai variabel `nrp` pada parameter constructor tersebut akan dipakai untuk menginisialisasi data member `nrp` pada class Siswa.

Keyword *this*

- Sehingga programnya akan tampak sbb:

```
1  public class Siswa {
2        private int nrp;
3        private int umur;
4
5        public Siswa(int nrp) {
6              nrp = nrp;
7        }
8  }
```

- Pada saat kita menulis baris ke-6, maka yang kita inginkan adalah :

```
nrp = nrp;
```

merujuk pada parameter konstruktor

merujuk pada data member nrp pada class Siswa

Keyword *this*

- Namun dengan cara menulisan seperti itu, maka semua nrp yang ada disana akan merujuk pada nrp yang terdekat, yaitu nrp pada parameter konstuktor, sehingga data member nrp tidak akan diinisialisasi.
- Untuk memberitahu kompiler Java bahwa yang kita maksud adalah `nrp` pada member class Siswa, kita dapat memakai kata kunci this, sehingga penulisannya baris-6 tersebut yang benar adalah seperti ini :

```
this.nrp = nrp
```

akan merujuk pada suatu member yang bernama nrp pada class yang bersangkutan (class Siswa)

Keyword *this*

- *this* dapat juga dipakai untuk memanggil constructor yang lain pada class yang bersangkutan.
- Misalnya saja contoh class Siswa pada overloading constructor dapat kita modifikasi sbb :
- Pada saat kita menuliskan

```
this(0);
```

- kompiler Java akan merujuk pada suatu constructor di class tersebut yang mempunyai daftar parameter yang sesuai, yaitu :

```
public Siswa(int n)
```

- Adapun nilai parameter yang dikirim adalah nilai 0.

```
public class Siswa {
        private int nrp;

        public Siswa() {
                this(0);
        }

        public Siswa(int n) {
                nrp=n;
        }
}
```