BAB

7

ENKAPSULASI (ENCAPSULATION)

Tujuan

Praktikan dapat memahami dan menjelaskan konsep enkapsulasi dalam pemrograman Java.

ENKAPSULASI

Enkapsulasi atau pembungkusan adalah suatu cara untuk menyembunyikan detail informasi dari user (pengakses kelas) terhadap suatu obyek. Terdapat dua hal mendasar dari enkapsulasi, yaitu:

- Information hidding: penyembunyian detail dari atribut dan method pada suatu kelas.
- Interface untuk pengaksesan data: suatu method untuk mengambil, memberikan, atau mengubah suatu nilai.

Pada pemrograman berorientasi obyek, kebutuhan akan enkapsulasi muncul karena adanya proses sharing data antar method. Dengan adanya enkapsulasi, maka keamanan dari data dan method yang ada didalamnya akan terjaga.

Enkapsulasi dapat dilakukan dengan terlebih dahulu memahami **access modifier** yang mendefinisikan bagaimana suatu data atau method dapat diakses. Terdapat empat macam access modifier pada PBO, yaitu:

- Private : hanya dapat diakses pada kelas itu sendiri
- Public : dapat diakses dari manapun
- Protected : hanya dapat diakses dari package (satu folder) dan subclass (turunan)
- Default : tanpa modifier, hanya dapat diakses dari package dan kelas itu sendiri

Dengan menggunakan enkapsulasi kita dapat membatasi akses langsung suatu kelas atau program kecuali melalui method yang sudah diberikan. Berikut adalah contoh enkapsulasi:

Program 7-1

```
// MahasiswaTest.java
class Mahasiswa{
    //deklarasi private attribute
    private String nama;
    private String nim;
    private String alamat;

    public Mahasiswa() {

    }
```

```
    //Constructor dengan dua input parameter
    public Mahasiswa(String nama, String nim) {
        this.nama = nama;
        this.nim = nim;
    }
    //Constructor dengan tiga input parameter
    public Mahasiswa(String nama, String nim, String alamat) {
        this.nama = nama;
        this.nim = nim;
        this.alamat = alamat;
    }
    //mengakses atribut alamat
    public String getAlamat() {
        return alamat;
    }
    //mengakses atribut nama
    public String getNama() {
        return nama;
    }
    //mengakses attribut nim
    public String getNim() {
            return nim;
    }
    //mengisi atribut alamat
    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }
}

public class MahasiswaTest{
    public static void main(String[] args){
        Mahasiswa objMhs;
        String localNama, localNim;
        objMhs=new Mahasiswa("Andriani","123456");
        objMhs.setAlamat("Watugong 212 Malang");
        localNama=objMhs.getNama();
        localNim=objMhs.getNim();
    }
}
```

Berdasarkan pada kode program di atas, perhatikan cara deklarasi atribut dan method-method yang terdapat dalam kelas Mahasiswa. Sebenarnya akan lebih sederhana jika pada kelas Mahasiswa atribut nama, nim, dan alamat dideklarasikan dengan access modifier public, sehingga tidak perlu membuat method-method khusus (setAlamat, getAlamat, getNim, getNama) untuk mengaksesnya. Demikian pula pada kelas MahasiswaTest, mengambil dan mengubah isi data objek Mahasiswa akan lebih sederhana. Berikut contoh jika kelas Mahasiswa di atas diubah access modifier atributnya:

```
class Mahasiswa{
    //deklarasi private attribute
    private String nama;
    private String nim;
```

```
    private String alamat;
    ...
}

public class MahasiswaTest{
    public static void main(String[] args){
        Mahasiswa objMhs;
        String localNama, localNim;
        objMhs=new Mahasiswa("Andriani","123456");
        objMhs.alamat = "Watugong 212 Malang";
        localNama=objMhs.nama;
        localNim=objMhs.nim;
    }
}
```

Saat membuat object objMhs yang bisa dianggap sebagai proses “membuat seorang mahasiswa”, maka data nama dan nim tidak boleh diubah selama mahasiswa tersebut ada sehingga kita harus membuat data nama dan nim sebagai data read only.

Jika data-data ini dibuat tidak readonly yang dapat memberikan peluang akses penuh, maka kemungkinan akan terdapat masalah tersendiri. Dengan demikian kita perlu melindungi data nama dan nim dari perubahan-perubahan dari luar class, sehingga perlu diberikan modifier private. Kita akan buktikan pada kasuskasus berikutnya.

Method getNama, getNim, dan getAlamat disebut sebagi accesor method karena method tersebut memberikan kembalian nilai data yang sederhana. Sedangkan setAlamat disebut sebagai mutator method karena digunakan untuk memberikan suatu nilai yang sederhana.

Langkah-langkah di atas selanjutnya dapat kita gunakan untuk menerapkan sifat encapsulation atau information hiding. Jadi pada dasarnya encapsulation dapat kita lakukan dengan:

- Mendeklarasikan data atau attribut dengan modifier private
- Memberikan accessor method (getter) untuk mengakses data dari atribut private
- Memberikan mutator method (setter) untuk memberikan nilai ke suatu data dari atribut private.

Keuntungan menerapkan enkapsulasi adalah:

- Bersifat independen
  Suatu modul yang terencapsulasi dengan baik akan bersifat independen dari modul yang lainnya sehingga dapat digunakan pada bagian manapun dari program. Ia tidak akan terikat pada bagian tertentu dari program.
- Bersifat transparan
  Bila melakukan modifikasi pada suatu modul, maka perubahan tersebut akan dirasakan juga oleh bagian program yang menggunakan modul tersebut.
- Menghindari efek diluar perencanaan
  Modul yang terencapsulasi dengan baik hanya akan berinteraksi dengan bagian program lainnya melalui variable-variabel input/output yang telah didefinisikan sebelunnya. Dengan demikian, akan mereduksi kemungkinan adanya hasil imbas pemrosesan yang di luar perencanaan semula.
- Melindungi listing program

Saat program didistribusikan pada khalayak, untuk melindungi listing program. Anda dapat menerapkan prinsip enkapsulasi. Di sini pengguna hanya dapat menggunakan program melalui variable input atau output yang didefinisikan tanpa disertai bagimana proses yang terjadi di dalam modul tersebut.

KEYWORD THIS

Keyword this adalah objek yang langsung digunakan tanpa didahului proses instansiasi. Penggunaan keyword ini yaitu bila ada attribute (non static) dari suatu kelas akan digunakan method yang berada dalam kelas yang sama, namun nama attribute tersebut dan nama parameter yang dilewatkan pada method tersebut SAMA.

Keyword ini dapat digunakan secara implisit maupun eksplisit. Berikut contoh penggunaan yang eksplisit:

Program 7-2

```
// RectangleToy.java
public class RectangleToy {
    private double width, height;
    public void setRectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
}
```

Pemanggilan attribute yang eksplisit, biasa digunakan untuk mengatasi panamaan yang sama. Pada contoh ini parameter di method setRectangle menggunakan nama yang sama dengan attribute di kelas RectangleToy.

Berikut contoh pemanggilan implisit:

Program 7-3

```
// RectangleToy2.java
public class RectangleToy2 {
    private double width, height;
    public void setRectangle(double new width, double new height) {
        width = new width;
        height = new height;
    }
}
```

PANDUAN MENDESAIN KELAS

Berikut adalah beberapa panduan dalam mendesain kelas (melakukan enkapsulasi):

1. Selalu gunakan modifier private untuk data atau attribut. Selain menerapkan sifat encapsulation, langkah ini akan memberikan keuntungan antara lain:
    - Desain yang lebih modular
    - Perubahan representasi class tidak terlalu berpengaruh pada class tersebut
    - Lebih mudah mendeteksi kesalahan

2. Selalu buat nilai inisial (nilai awal) pada data. Inisialisasi data dapat dilakukan dengan menggunakan constructor atau dengan memberikan nilai default.
3. Hindari penggunaan banyak tipe data pada sebuah class. Kumpulkan multiple tipe data menjadi sebuah kelas. Misalkan jika pada class Mahasiswa terdapat:
   `private String jalan; private string kota; private string propinsi; private string kodePos;`
   dapat diganti dengan sebuah **class Alamat** yang didalamnya terdapat data-data di atas.
4. Tidak semua data memerlukan setter dan getter. Untuk data yang hanya perlu di set satu kali, cukup manfaatkan constructor.
5. Biasakan menggunakan standar penulisan yang sama pada setiap class. Manfaatkan penulisan indent untuk menjaga kerapian code sehingga mempermudah trace kesalahan.
6. Buatlah class sesederhana mungkin, hindari membuat class yang kompleks.
7. Gunakan nama class dan method sesuai fungsinya. Ini akan mempermudah kita (dan tim) dalam menggunakan class dan trace error. Ada saran untuk penulisan class dan method:
   - Gunakan kata benda (Noun) untuk sebuah nama class: Mahasiswa, MataKuliah, JadwalKuliah.
   - Gunakan kata kerja (Verb) untuk method : getNama, hitungIPK, setNilai.

Lembar Kerja Praktikum: Modul 7	
NPM:	Asisten:
Nama:	Nilai:
Kelas:	Tanggal:

Soal

[Score: 100] Berdasarkan kode program di bawah ini, tentukanlah:

- Atribut dari Kelas Kapsul
- Method dari Kelas Kapsul
- Accessor Method dari Kelas Kapsul
- Mutator Method dari Kelas Kapsul
- Objek pada kelas Enkapsulasi

```
public class Kapsul{
    private double panjang;
    private double lebar;
    private double tinggi;
    public Kapsul(){
        double panjang=0;
        double lebar=0;
    }
    private double luas(double p, double l){ return p*l; }
    public void setPanjang(double panjang){ this.panjang=panjang; }
    public void setLebar(double lebar){ this.lebar=lebar; }
    public double getPanjang(){ return panjang; }
    public double getLebar(){ return lebar; }
    public double getLuas(){ return luas(panjang, lebar); }
}

public class Enkapsulasi{
    public static void main(String[]args){
        Kapsul pp= new Kapsul();
        pp.setPanjang(50);
        pp.setLebar(100);
        System.out.println("panjang: "+ pp.getPanjang());
        System.out.println("lebar: "+pp.getLebar());
        System.out.println("luasnya adalah: "+ pp.getLuas());
    }
}
```