MODUL - OBJECT ORIENTED PROGRAMMING 1

Oleh :

Priyanto Tamami, S.Kom.

POLITEKNIK HARAPAN BERSAMA

PROGRAM STUDI D IV TEKNIK INFORMATIKA

Jl. Mataram No. 9 Tegal

Daftar Isi

Bab 1

Kelas dan Objek

1.1 Tujuan

Pada Bab ini diharapkan mahasiswa memahami pengertian dan perbedaan Kelas dan Objek dan mampu mengimplementasikan konsep tersebut pada bahasa pemrograman Java.

1.2 Pengantar

Dalam paradigma pemrograman berorientasi objek, untuk membangun sebuah sistem atau aplikasi yang lengkap, sistem tersebut akan dipecah menjadi bagian-bagian kecil yang disebut dengan objek. Tiap-tiap objek yang terbentuk akan dapat saling berinteraksi membentuk sebuah sistem yang dapat digunakan.

Untuk mempermudah pembentukan objek-objek yang akan digunakan, maka diperlukan klasifikasi-klasifikasi tertentu berdasarkan kesamaan ciri dan fitur, yang disebut dengan kelas. Dengan kata lain bahwa kelas itu sebetulnya adalah deklarasi dari beberapa objek yang nantinya akan digunakan dalam membangun sebuah sistem.

Bagaimana implementasi kedua istilah tersebut dalam bahasa pemrograman Java, mari kita lanjutkan ke bagian **Praktek**.

1.3 Praktek

1.3.1 Kelas

Apabila kita menggunakan bahasa pemrograman Java, aturan yang harus kita ikuti adalah pada saat pembentukan deklarasi sebuah kelas, nama kelas dan nama berkas yang dibuat harus sama persis sampai ke besar dan kecilnya huruf.

Sebagai contoh, apabila kita ingin membuat sebuah kelas `Mahasiswa`, maka kita akan membuat sebuah berkas dengan nama `Mahasiswa.java`, yang didalamnya akan berisi kode berikut :

```
1 public class Mahasiswa {}
```

Agar sebuah aplikasi dapat dijalankan dan dilihat hasilnya, maka kita perlu menambahkan sebuah *method* dengan nama `main` di dalam kelas tersebut, sehingga isi kodenya akan menjadi seperti berikut :

```
1 public class Mahasiswa{
2   public static void main(String args[]) {
3   }
4 }
```

Seluruh aksi program yang dijalankan akan dimulai dari *method* `main` ini. Misalkan kita coba agar aplikasi dapat menampilkan tulisan selamat datang apabila dijalankan, kodenya akan kita ubah menjadi seperti berikut :

```
2 public class Mahasiswa {
3   public static void main(String args[]) {
4     System.out.println("Selamat datang pada mata kuliah OOP");
5   }
```

```
}
```

Agar kode tersebut dapat berjalan, maka kita harus melakukan *compile* terlebih dahulu pada kode sumber dengan cara berikut :

```
$ javac Mahasiswa.java
```

Dari hasil *compile* tersebut, akan terbentuk sebuah berkas dengan nama yang sama, yaitu `Mahasiswa` namun dengan ekstensi `.class`, bila sudah tersebut berkas ini, kita dapat menjalankannya dengan perintah berikut :

```
$ java Mahasiswa
```

Hasil keluaran dari perintah tersebut seharusnya akan menampilkan teks seperti ini :

```
Selamat datang pada mata kuliah OOP
```

1.3.2 Objek

Dari contoh kode sebelumnya, deklarasi kelas `Mahasiswa` sudah memiliki sebuah fitur atau *method* dengan nama `main`. Sekarang kita akan coba menambahkan ciri atau atribut lain pada kelas `Mahasiswa`.

Seorang Mahasiswa tentunya akan memiliki **nama** dan **NIM** (Nomor Induk Mahasiswa), untuk mengimplementasikan atribut ini, kita akan ubah kodenya menjadi seperti ini :

```
public class Mahasiswa {

  String nama;
  String nim;

}
```

Kita hapus terlebih dahulu *method* `main`, agar kita fokus pada kelas `Mahasiswa`. Kelas ini memiliki atribut `nama` dan `nim`, pada kelas ini akan kita tambahkan sebuah fitur atau *method* untuk menampilkan informasi dari Mahasiswa yang bersangkutan, kodenya akan kita tambahkan sehingga terlihat seperti berikut :

```
1  public class Mahasiswa {
2
3    String nama;
4    String nim;
5
6    public void cetakInfo() {
7      System.out.println("Nama : " + nama);
8      System.out.println("NIM : " + nim);
9    }
10
11 }
```

Kelas `Mahasiswa` kita anggap sudah lengkap untuk sementara, kita akan coba membuat sebuah objek dari kelas `Mahasiswa` ini. Buatlah sebuah kelas baru, kita beri nama untuk berkasnya adalah `Aplikasi.java` yang isinya seperti berikut :

```
1  public class Aplikasi {
2    public static void main(String args[]) {
3      Mahasiswa ami = new Mahasiswa();
4      ami.nama = "tamami";
5      ami.nim = "19001";
6      ami.cetakInfo();
7    }
8  }
```

Perhatikan pada baris ke-3, bahwa objek `ami` telah kita buat dengan tipe data berupa kelas `Mahasiswa`, ini artinya, objek `ami` merupakan instan dari kelas `Mahasiswa`.

Pembentukan objek, agar data di dalamnya dapat kita ubah, kita perlu me-

lakukan inisiasi dengan pemanggilan konstruktor `Mahasiswa` dengan kode `new Mahasiswa()`. Konstruktor ini akan kita bahas di bagian lain, namun secara *default*, setiap kelas pasti memiliki 1 (satu) konstruktor tanpa parameter walau tidak dideklarasikan secara eksplisit.

Baris ke-4 dan baris ke-5 mengisikan nilai ke properti `nama` dan `nim` milik objek `ami`. Kemudian pada baris ke-6, *method* `cetakInfo()` milik objek `ami` dipanggil.

Untuk melakukan kompilasi, seperti langkah sebelumnya, kita dapat melakukannya dengan perintah `javac` dari konsol atau *command prompt* seperti berikut :

```
1 $ javac Aplikasi.java
```

Kemudian jalankan dengan perintah berikut :

```
1 $ java Aplikasi
```

Hasil yang dikeluarkan seharusnya akan terlihat seperti berikut :

```
1 Nama : Tamami
2 NIM : 19001
```

1.4 Kesimpulan

Bahwa kelas dan objek itu adalah dua hal yang berbeda, dimana kelas adalah deklarasi sebuah unit yang memiliki atribut dan fitur tertentu, sementara objek adalah instan dari suatu kelas.

1.5 Tugas

Buatlah sebuah kelas `Anggota`, yang di dalamnya terpada atribut **nomor anggota** dan **nama**. Kemudian buat sebuah objek yang merupakan instan dari kelas

`Anggota` dan isikan *nama* dan *nomor anggota*nya. Kemudian cetak hasilnya dalam format `no. anggota : nama` seperti contoh berikut :

```
19001 : tamami
```

Bab 2

Konstruktor, Field, dan Overloading

2.1 Tujuan

Pada Bab ini diharapkan mahasiswa memahami konsep Konstruktor, *Field*, dan *Overloading* pada bahasa pemrograman Java.

2.2 Pengantar

Pada Bab sebelumnya, kita sempat menyinggung sedikit tentang konstruktor, bahwa setiap kelas yang kita deklarasikan, secara implisit akan menyediakan sebuah konstruktor tanpa parameter di dalamnya, konstruktor ini dipanggil pada saat akan membuat instan bagi sebuah objek.

Namun demikian, konstruktor ini pun sebetulnya dapat kita deklarasikan yang biasanya digunakan untuk memberikan nilai-nilai *default* bagi atribut / *field* yang ada di dalamnya.

Lalu apa itu *field*? *Field* atau atribut sebetulnya sudah sangat kita kenal dalam

konsep paradigma pemrograman yang lain dengan nama *variabel*. Biasanya pada sebuah kelas akan memiliki 1 (satu) atau lebih *field* atau atribut, bersama dengan *method* akan menjadi ciri sebuah kelas.

Selain deklarasi konstruktor tanpa parameter, sebetulnya kita masih dapat mendeklarasikan konstruktor lain dengan parameter, dan dapat dideklarasikan lebih dari 1 (satu) konstruktor, implementasi ini disebut *overloading*.

Mari kita lihat implementasi dari ketiga istilah di atas dalam bahasa pemrograman Java.

2.3 Praktek

2.3.1 Konstruktor

Konstruktor sebetulnya adalah fungsi atau *method* yang dipanggil ketika akan membuat sebuah instan dari kelas. Ciri yang terlihat pada konstruktor ini dibanding *method* lain adalah namanya akan sama persis dengan nama kelasnya, dan tidak memiliki nilai balik sama sekali.

Mari kita lihat contoh kelas *Mahasiswa* sebelumnya seperti berikut :

```
1  public class Mahasiswa {
2
3    String nama;
4    String nim;
5
6    public Mahasiswa() {}
7
8    public void cetakInfo() {
9      System.out.println("Nama : " + nama);
10     System.out.println("NIM : " + nim);
11   }
12
```

```
13 }
```

Tampak pada kode tersebut, pada baris ke-6 adalah deklarasi konstruktor tanpa parameter yang apabila tidak dideklarasikan pun, konstruktor tersebut secara implisit sudah ada. Namun sekarang kita akan modifikasi konstruktor tersebut untuk mengisikan nilai *default* ke atribut `nama` dan `nim`. Kodenya akan menjadi seperti berikut :

```
1  public class Mahasiswa {
2
3    String nama;
4    String nim;
5
6    public Mahasiswa() {
7      nama = "tidak ada";
8      nim = "00000";
9    }
10
11   public void cetakInfo() {
12     System.out.println("Nama : " + nama);
13     System.out.println("NIM : " + nim);
14   }
15
16 }
```

Kita akan melihat perubahan pada baris ke-6 sampai ke-9, konstruktor yang tadinya kosong, tanpa deklarasi isi sama sekali, sekarang kita memberikan nilai *default* pada atribut `nama` dan `nim`.

Sekarang kita coba modifikasi kelas `Aplikasi` dari Bab sebelumnya untuk melihat hasilnya, bagaimana bila instan yang terbentuk tidak kita isikan atribut-atributnya. Berikut kodenya :

```
1  public class Aplikasi {
```

```
2   public static void main(String args[]) {
3     Mahasiswa ami = new Mahasiswa();
4     ami.cetakInfo();
5   }
6 }
```

Perhatikan pada baris ke-3 dan ke-4, objek `ami` hanya membentuk instan baru dengan memanggil konstruktor `Mahasiswa` tanpa parameter, kemudian *method* `cetakInfo()` langsung dipanggil.

Untuk melihat hasil keluarannya, pastikan untuk melakukan *compile* terhadap kelas `Mahasiswa` dan `Aplikasi`. Hasil yang didapat pada layar monitor seharusnya akan terlihat seperti berikut :

```
1 Nama : tidak ada
2 NIM : 00000
```

Hal ini disebabkan karena pada saat kita membentuk instan baru dengan memanggil konstruktor `Mahasiswa` tanpa parameter, nilai atribut `nama` dan `nim` sudah terisi secara otomatis dengan nilai *default*, sehingga apabila tidak ada perubahan, maka hasil yang ditampilkan adalah hasil dari pengisian nilai *default* pada konstruktornya.

2.3.2 Field

Seperti dijelaskan pada bagian sebelumnya bahwa istilah *field* ini lebih kita kenal dengan istilah *variabel* atau dalam istilah yang sering disebut dalam beberapa sumber adalah atribut.

Sehingga pada kelas `Mahasiswa`, atribut atau *field* yang dimiliki adalah `nama` dan `nim`.

Namun hendaknya, sesuai aturan pada desain orientasi objek bahwa akses terhadap *field* ini seharusnya terbatas hanya pada kelas yang bersangkutan, apabila

objek lain ingin melakukan akses atau manipulasi data, maka harus dilakukan melalui *method* yang dapat diakses oleh publik.

Jadi idealnya, bentuk kode dari kelas `Mahasiswa` akan menjadi seperti berikut :

```
1  public class Mahasiswa {
2
3    private String nama;
4    private String nim;
5
6    public Mahasiswa() {
7      nama = "tidak ada";
8      nim = "00000";
9    }
10
11   public void cetakInfo() {
12     System.out.println("Nama : " + nama);
13     System.out.println("NIM : " + nim);
14   }
15
16   public void setNama(String nama) {
17     this.nama = nama;
18   }
19
20   public String getNama() {
21     return nama;
22   }
23
24   public void setNim(String nim) {
25     this.nim = nim;
26   }
27
28   public String getNim() {
```

```
29      return nim;
30    }
31
32  }
```

Terlihat sedikit lebih panjang, namun dari baris ke-16 sampai ke bawah sebetulnya adalah deklarasi aksesor untuk *field* atau atribut `nama` dan `nim` yang menjadi `private` pada baris ke-3 dan ke-4.

Dengan kondisi demikian, diharapkan nilai atribut yang berada di dalam kelas lebih dapat dikontrol, dan pengguna berikutnya tidak perlu terlalu pusing memikirkan detail kelasnya, cukup fokus dengan apa fungsinya.

Dengan kondisi ini pula, maka akses terhadap atribut `nama` dan `nim` dari kelas `Aplikasi` tidak bisa dilakukan dengan cara yang lama, perubahan pada kelas `Aplikasi` menjadi seperti berikut :

```
1  public class Aplikasi {
2    public static void main(String args[]) {
3      Mahasiswa ami = new Mahasiswa();
4      ami.setNama("Tamami");
5      ami.setNim("19001");
6      ami.cetakInfo();
7    }
8  }
```

Perhatikan cara akses terhadap atribut `nama` dan `nim` yang dilakukan pada baris ke

Aturan ini berlaku secara umum di Java, dan sebaiknya diikuti, karena banyak *framework* yang dibangun menggunakan standar seperti ini, sehingga kedepannya, saat kita terbiasa dengan skema seperti ini, untuk melakukan integrasi dengan menggunakan *framework* di Java lebih mudah dan dapat dikerjakan dengan hasil yang benar.

2.3.3 Overloading

2.4 Kesimpulan

2.5 Tugas