Class

6.1. Pengenalan Class, Object, dan Method

6.1.1. Class

Class merupakan konsep pokok di bahasa pemrograman berorientasi obyek, demikian pula di java. Class mendefinisikan bentuk dan perilaku obyek. Sembarang konsep/abstraksi yang diimplementasikan di java harus dikapsulkan dalam class.

Pemrograman di java tidak mungkin dipisahkan dari class. Pada pemrograman sebelumnya, class hanya mengkapsulkan metode (berbentuk fungsi) main() untuk menunjukkan sintaks dasar bahasa java.

Class adalah struktur dasar dari OOP (Object Oriented Programming). Class terdiri dari dua tipe anggota yang disebut dengan field (attribut/properti) dan method (metode). Field merupakan tipe data yang didefinisikan oleh class, sementara method merupakan operasi.

6.1.2. Object

Sebuah obyek adalah sebuah instance (keturunan) dari class. Pada dunia perangkat lunak, sebuah obyek adalah sebuah komponen perangkat lunak yang stukturnya mirip dengan obyek pada dunia nyata. Setiap obyek dibangun dari sekumpulan data (atribut) yang disebut variabel untuk menjabarkan karakteristik khusus dari obyek, dan juga terdiri dari sekumpulan method yang menjabarkan tingkah laku dari obyek. Bisa dikatakan bahwa obyek adalah sebuah perangkat lunak yang berisi sekumpulan variabel dan method yang berhubungan. Variabel dan method dalam obyek Java secara formal diketahui sebagai variabel instance dan method instance. Hal ini dilakukan untuk membedakan dari variabel class dan method class.

6.1.3. Method

Method adalah bagian-bagian kode yang dapat dipanggil oleh program utama atau dari metode lainnya untuk menjalankan fungsi yang spesifik.

Berikut adalah karakteristik dari metode :

a. Dapat mengembalikan satu nilai atau tidak sama sekali
b. Dapat menerima beberapa parameter yang dibutuhkan atau tidak ada parameter sama sekali. Parameter bisa juga disebut sebagai argumen dari fungsi.
c. Setelah metode selesai dieksekusi, dia akan kembali pada metode yang memanggilnya.

6.2. Membuat Class

Sebelum menulis class Anda, pertama pertimbangkan dimana Anda akan menggunakan class dan bagaimana class tersebut akan digunakan. Pertimbangkan pula nama yang tepat dan tuliskan seluruh informasi atau properti yang ingin Anda isi pada class. Jangan sampai terlupa untuk menuliskan secara urut method yang akan Anda gunakan dalam class.

Dalam mendefinisikan class secara umum dapat ditulis sebagai berikut :

```
<modifier> class <ClassName> {
    <attributeDeclaration>*
    <constructorDeclaration>*
    <methodDeclaration>*
}
```

dimana :

`<modifier>`	adalah sebuah access modifier, yang dapat dikombinasikan dengan tipe modifier lain.
`<ClassName>`	adalah nama class yang akan kita buat.

6.3. Deklarasi Methods

Sebelum kita membahas method apa yang akan dipakai pada class, mari kita perhatikan penulisan method secara umum.

Dalam pendeklarasian method, kita tuliskan :

```
<modifier> <returnType> <name>(<parameter>*) {
    <statement>*
}
```

dimana,

<modifier> dapat menggunakan beberapa modifier yang berbeda
<returnType> dapat berupa seluruh tipe data, termasuk void
<name> identifier atas class
<parameter> terdiri dari <tipe_parameter> <nama_parameter>[,]

6.3.1. Accessor Methods

Untuk mengimplementasikan enkapsulasi, kita tidak menginginkan sembarang object dapat mengakses data kapan saja. Untuk itu, kita deklarasikan atribut dari class sebagai private. Namun, ada kalanya dimana kita menginginkan object lain untuk dapat mengakses data private. Dalam hal ini kita gunakan accessor methods.

Accessor Methods digunakan untuk membaca nilai variabel pada class, baik berupa instance maupun static. Sebuah accessor method umumnya dimulai dengan penulisan get<namaInstanceVariable>. Method ini juga mempunyai sebuah return value.

Sebagai contoh, kita ingin menggunakan accessor method untuk dapat membaca nama, alamat, nilai bahasa Inggris, Matematika, dan ilmu pasti dari siswa.

Mari kita perhatikan salah satu contoh implementasi accessor method.

```
public class StudentRecord
{
    private String name;
    :
    :
    :
    public String getName(){
        return name;
    }
}
```

dimana,

public - Menjelaskan bahwa method tersebut dapat diakses dari object luar class
String - Tipe data return value dari method tersebut bertipe String
getName - Nama dari method
() - Menjelaskan bahwa method tidak memiliki parameter apapun

Pernyataan berikut,

```
return name;
```

dalam program kita menandakan akan ada pengembalian nilai dari nama instance variable ke pemanggilan method. Perhatikan bahwa return type dari method harus sama dengan tipe data seperti data pada pernyataan return. Anda akan mendapatkan pesan kesalahan

sebagai berikut bila tipe data yang digunakan tidak sama :

```
StudentRecord.java:14: incompatible types
found : int
required: java.lang.String
return age;
^
1 error
```

Contoh lain dari penggunaan accessor method adalah getAverage,

```
public class StudentRecord
{
   private String name;
   :
   :
   :
   public double getAverage(){
      double result = 0;
      result = ( mathGrade+englishGrade+scienceGrade )/3;
      return result;
   }
}
```

Method getAverage() menghitung rata – rata dari 3 nilai siswa dan menghasilkan nilai return value dengan nama result.

6.3.2. Mutator Methods

Bagaimana jika kita menghendaki object lain untuk mengubah data? Yang dapat kita lakukan adalah membuat method yang dapat memberi atau mengubah nilai variable dalam class, baik itu berupa instance maupun static. Method semacam ini disebut dengan mutator methods. Sebuah mutator method umumnya tertulis set<namaInstanceVariabel>.

Mari kita perhatikan salah satu dari implementasi mutator method :

```
public class StudentRecord
{
   private String name;
   :
   :
   public void setName( String temp ){
      name = temp;
   }
}
```

Dimana :

public	- Menjelaskan bahwa method ini dapat dipanggil object luar class
void	- Method ini tidak menghasilkan return value
setName	- Nama dari method
(String temp)	- Parameter yang akan digunakan pada method

Pernyataan berikut :

```
name = temp;
```

mengubah nilai instance variable name menjadi sama dengan nilai dari temp.

Perlu diingat bahwa mutator methods tidak menghasilkan return value. Namun berisi beberapa argumen dari program yang akan digunakan oleh method.

6.3.3. Multiple Return Statements

Anda dapat mempunyai banyak return values pada sebuah method selama mereka tidak pada blok program yang sama. Anda juga dapat menggunakan konstanta disamping variabel sebagai return value.

Sebagai contoh, perhatikan method berikut ini :

```
public String getNumberInWords( int num ){
    String defaultNum = "zero";
    if( num == 1 ){
        return "one"; //mengembalikan sebuah konstanta
    }
    else if( num == 2){
        return "two"; //mengembalikan sebuah konstanta
    }
    return defaultNum; // mengembalikan sebuah variabel
}
```

6.3.4. Static Methods

Kita menggunakan static method untuk mengakses static variable studentCount.

```
public class StudentRecord
{
    private static int studentCount;
    public static int getStudentCount(){
        return studentCount;
    }
}
```

Dimana :

public	- Menjelaskan bahwa method ini dapat diakses dari object luar class
static	- Method ini adalah static dan pemanggilannya menggunakan [namaclass].[namaMethod]. Sebagai contoh : studentRecord.getStudentCount
Int	- Tipe return dari method. Mengindikasikan method tersebut harus mempunyai return value berupa integer
getStudentCount()	- Nama dari method - Method ini tidak memiliki parameter apapun

Pada deklarasi di atas, method getStudentCount() akan selalu menghasilkan return value 0 jika kita tidak mengubah apapun pada kode program untuk mengatur nilainya. Kita akan membahas pengubahan nilai dari studentCount pada pembahasan constructor.

6.4. Referensi this

Reference this digunakan untuk mengakses instance variable yang dibiaskan oleh parameter. Untuk pemahaman lebih lanjut, mari kita perhatikan contoh pada method setAge. Dimisalkan kita mempunyai kode deklarasi berikut pada method setAge.

```
public void setAge( int age ){
    age = age; //SALAH!!!
}
```

Nama parameter pada deklarasi ini adalah age, yang memiliki penamaan yang sama dengan instance variable age. Parameter age adalah deklarasi terdekat dari method, sehingga nilai dari parameter tersebut akan digunakan. Maka pada pernyataan :

```
age = age;
```

kita telah menentukan nilai dari parameter age kepada parameter itu sendiri. Hal ini sangat tidak kita kehendaki pada kode program kita. Untuk menghindari kesalahan semacam ini, kita gunakan metode referensi this. Untuk menggunakan tipe referensi ini, kita tuliskan :

```
this.<namaInstanceVariable>
```

Sebagai contoh, kita dapat menulis ulang kode hingga tampak sebagai berikut :

```
public void setAge( int age ){
    this.age = age;
}
```

Method ini akan mereferensikan nilai dari parameter age kepada instance variable dari object StudentRecord.

Kita hanya dapat menggunakan referensi this terhadap instance variable dan bukan static ataupun class variabel.

6.5. Overloading Methods

Dalam class yang kita buat, kadangkala kita menginginkan untuk membuat method dengan nama yang sama namun mempunyai fungsi yang berbeda menurut parameter yang digunakan. Kemampuan ini dimungkinkan dalam pemrograman Java, dan dikenal sebagai overloading method.

Overloading method mengijinkan sebuah method dengan nama yang sama namun memiliki parameter yang berbeda sehingga mempunyai implementasi dan return value yang berbeda pula. Daripada memberikan nama yang berbeda pada setiap pembuatan method, overloading method dapat digunakan pada operasi yang sama namun berbeda dalam implementasinya.

Sebagai contoh, pada class StudentRecord kita menginginkan sebuah method yang akan menampilkan informasi tentang siswa. Namun kita juga menginginkan operasi penampilan data tersebut menghasilkan output yang berbeda menurut parameter yang digunakan. Jika pada saat kita memberikan sebuah parameter berupa string, hasil yang ditampilkan adalah nama, alamat dan umur dari siswa, sedang pada saat kita memberikan 3 nilai dengan tipe double, kita menginginkan method tersebut untuk menampilkan nama dan nilai dari siswa.

Untuk mendapatkan hasil yang sesuai, kita gunakan overloading method di dalam deklarasi class StudentRecord.

```
public void print( String temp ){
    System.out.println("Name:" + name);
    System.out.println("Address:" + address);
    System.out.println("Age:" + age);
}

public void print(double eGrade, double mGrade, double sGrade){
    System.out.println("Name:" + name);
    System.out.println("Math Grade:" + mGrade);
    System.out.println("English Grade:" + eGrade);
    System.out.println("Science Grade:" + sGrade);
}
```

Jika kita panggil pada method utama (main) :

```
public static void main( String[] args )
{
    StudentRecord annaRecord = new StudentRecord();
    annaRecord.setName("Anna");
    annaRecord.setAddress("Philippines");
    annaRecord.setAge(15);
    annaRecord.setMathGrade(80);
    annaRecord.setEnglishGrade(95.5);
    annaRecord.setScienceGrade(100);

    //overloaded methods
    //panggilan metode print pertama
    annaRecord.print( annaRecord.getName() );

    //panggilan metode print kedua
    annaRecord.print( annaRecord.getEnglishGrade(),
    annaRecord.getMathGrade(), annaRecord.getScienceGrade());
}
```

Kita akan mendapatkan output pada panggilan metode print pertama sebagai berikut :

```
Name:Anna
Address:Philippines
Age:15
```

Kemudian akan dihasilkan output sebagai berikut pada panggilan metode print kedua :

```
Name:Anna
Math Grade:80.0
English Grade:95.5
Science Grade:100.0
```

Jangan lupakan bahwa overloaded method memiliki property sebagai berikut :

a. Nama yang sama
b. Parameter yang berbeda
c. Nilai kembalian (return) bisa sama ataupun berbeda

Contoh program :

```
public class Segitiga{
   private double tinggi;
   private double alas;

   public void settinggi(double tinggi){
      this.tinggi = tinggi;
   }

   public void setalas(double alas){
      this.alas = alas;
   }

   public double gettinggi(){
      return tinggi;
   }

   public double getalas(){
      return alas;
   }

   public double getluas(){
      return (this.tinggi * this.alas * 0.5);
   }

   public static void main (String args[]){
      Segitiga S[] = new Segitiga[2];
      Byte i;

      //Membuat objek dari class Segitiga
      S[0] = new Segitiga();
      S[1] = new Segitiga();

      S[0].settinggi(12.0);
      S[0].setalas(8.0);
```

```
      S[1].settinggi(11.23);
      S[1].setalas(7.7);

      for (i=0;i<2;i++){
        System.out.println("Segitiga ke-" + (i+1));
        System.out.println("Tinggi = " + S[i].gettinggi());
        System.out.println("Alas = " + S[i].getalas());
        System.out.println("Luas Segitiga = " + S[i].getluas());
        System.out.println();
      }
   }
}
```