Pertemuan VII

Class (Lanjutan)

7.1. Deklarasi *Constructor*

Telah tersirat pada pembahasan sebelumnya, *constructor* sangatlah penting pada pembentukan sebuah *object*. *Constructor* adalah *method* dimana seluruh inisialisasi *object* ditempatkan.

Berikut ini adalah *property* dari *constructor* :

a. *Constructor* memiliki nama yang sama dengan *class*
b. Sebuah *constructor* mirip dengan *method* pada umumnya, namun hanya informasi – informasi berikut yang dapat ditempatkan pada *header* sebuah *constructor*, *scope* atau identifikasi pengaksesan (misal: *public*), nama dari konstuktor dan parameter.
c. *Constructor* tidak memiliki *return value*
d. *Constructor* tidak dapat dipanggil secara langsung, namun harus dipanggil dengan menggunakan operator *new* pada pembentukan sebuah objek.

Untuk mendeklarasikan *constructor*, kita dapat menggunakan struktur penulisan secara umum seperti berikut ini :

```
<modifier> <className> (<parameter>*) {
    <statement>
    <statement>
    :
    :
}
```

7.1.1. Default Constructor

Setiap class memiliki default constructor. Sebuah default constructor adalah constructor yang tidak memiliki parameter apapun. Jika sebuah *class* tidak memiliki constructor apapun, maka sebuah default constructor akan dibentuk secara implisit :

Sebagai contoh, pada *class* StudentRecord, bentuk default constructor akan terlihat seperti dibawah ini :

```
public StudentRecord()
{
    //area penulisan kode
}
```

7.1.2. Overloading Constructor

Seperti telah kita bahas sebelumnya, sebuah *constructor* juga dapat dibentuk menjadi *overloaded*. Dapat dilihat pada 4 contoh sebagai berikut :

```
public StudentRecord(){
    //area inisialisasi kode;
}

public StudentRecord(String temp){
    this.name = temp;
}
```

```
public StudentRecord(String name, String address){
    this.name = name;
    this.address = address;
}

public StudentRecord(double mGrade, double eGrade, double sGrade){
    mathGrade = mGrade;
    englishGrade = eGrade;
    scienceGrade = sGrade;
}
```

7.1.3. Menggunakan Constructor

Untuk menggunakan constructor, kita gunakan kode – kode sebagai berikut :

```
public static void main( String[] args )
{
   //membuat 3 objek
   StudentRecord annaRecord=new StudentRecord("Anna");
   StudentRecord beahRecord=new StudentRecord("Beah","Philippines");
   StudentRecord crisRecord=new StudentRecord(80,90,100);

   //area penulisan kode selanjtunya
}
```

Sebelum kita lanjutkan, mari kita perhatikan kembali deklarasi variabel static studentCount yang telah dibuat sebelumnya. Tujuan deklarasi studentCount adalah untuk menghitung jumlah *object* yang dibentuk pada *class* StudentRecord. Jadi, apa yang akan kita lakukan selanjutnya adalah menambahkan nilai dari studentCount setiap kali setiap pembentukan *object* pada *class* StudentRecord. Lokasi yang tepat untuk memodifikasi dan menambahkan nilai studentCount terletak pada constructor-nya, karena selalu dipanggil setiap kali objek terbentuk.

Sebagai contoh :

```
public StudentRecord(){
   //letak kode inisialisasi
   studentCount++; //menambah student
}

public StudentRecord(String temp){
   this.name = temp;
   studentCount++; //menambah student
}

public StudentRecord(String name, String address){
   this.name = name;
   this.address = address;
   studentCount++; //menambah student
}

public StudentRecord(double mGrade, double eGrade, double sGrade){
   mathGrade = mGrade;
   englishGrade = eGrade;
   scienceGrade = sGrade;
   studentCount++; //menambah student
}
```

7.1.4. Pemanggilan Constructor dengan this()

Pemanggilan constructor dapat dilakukan secara berangkai, dalam arti Anda dapat memanggil constructor di dalam constructor lain. Pemanggilan dapat dilakukan dengan referensi *this()*. Perhatikan contoh kode sebagai berikut :

```
1:  public StudentRecord(){
2:  this("some string");
3:
4:  }
5:
6:  public StudentRecord(String temp){
7:  this.name = temp;
8:  }
9:
10: public static void main( String[] args )
11: {
12:
13: StudentRecord annaRecord = new StudentRecord();
14: }
```

Dari contoh kode diatas, pada saat baris ke 13 dipanggil akan memanggil constructor dasar pada baris pertama. Pada saat baris kedua dijalankan, baris tersebut akan menjalankan constructor yang memiliki parameter String pada baris ke-6.

Beberapa hal yang patut diperhatikan pada penggunaan *this()* :

a. Harus dituliskan pada baris pertama pada sebuah constructor
b. Hanya dapat digunakan pada satu definisi constructor. Kemudian metode ini dapat diikuti dengan kode – kode berikutnya yang relevan

7.2. Package

Packages dalam JAVA berarti pengelompokan beberapa *class* dan *interface* dalam satu unit. Fitur ini menyediakan mekanisme untuk mengatur *class* dan *interface* dalam jumlah banyak dan menghindari konflik pada penamaan.

7.2.1. Mengimport Package

Supaya dapat meggunakan *class* yang berada diluar *package* yang sedang dikerjakan, Anda harus mengimport *package* dimana *class* tersebut berada. Pada dasarnya, seluruh program JAVA mengimport *package java.lang.*,* sehingga Anda dapat menggunakan *class* seperti String dan Integer dalam program meskipun belum mengimport *package* sama sekali.

Penulisan import *package* dapat dilakukan seperti dibawah ini :

```
import <namaPaket>;
```

Sebagai contoh, bila Anda ingin menggunakan *class* Color dalam *package* awt, Anda harus menuliskan import *package* sebagai berikut :

```
import java.awt.Color;
import java.awt.*;
```

Baris pertama menyatakan untuk mengimport *class* Color secara spesifik pada *package*, sedangkan baris kedua menyatakan mengimport seluruh *class* yang terkandung dalam *package java.awt*.

Cara lain dalam mengimport *package* adalah dengan menuliskan referensi *package* secara eksplisit. Hal ini dilakukan dengan menggunakan nama *package* untuk mendeklarasikan *object* sebuah *class* :

```
java.awt.Color color;
```

7.2.2. Membuat Package

Untuk membuat *package*, dapat dilakukan dengan menuliskan :

```
package <packageName>;
```

Anggaplah kita ingin membuat *package* dimana *class* StudentRecord akan ditempatkan bersama dengan *class – class* yang lain dengan nama *package* schoolClasses.

Langkah pertama yang harus dilakukan adalah membuat folder dengan nama schoolClasses. Salin seluruh *class* yang ingin diletakkan pada *package* dalam folder ini.

Kemudian tambahkan kode deklarasi *package* pada awal file. Sebagai contoh :

```
package schoolClasses;
public class StudentRecord
{
    private String name;
    private String address;
    private int age;
}
```

Package juga dapat dibuat secara bersarang. Dalam hal ini Java Interpreter menghendaki struktur direktori yang mengandung *class* eksekusi untuk disesuaikan dengan struktur *package*.

7.2.3. Pengaturan CLASSPATH

Diasumsikan *package* schoolClasses terdapat pada direktori C:\. Langkah selanjutnya adalah mengatur classpath untuk menunjuk direktori tersebut sehingga pada saat akan dijalankan, JVM dapat mengetahui dimana *class* tersebut tersimpan.

Sebelum membahas cara mengatur classpath, perhatikan contoh dibawah yang menandakan kejadian bila kita tidak mengatur classpath.

Asumsikan kita mengkompilasi dan menjalankan *class* StudentRecord :

```
C:\schoolClasses>javac StudentRecord.java
C:\schoolClasses>java StudentRecord
Exception in thread "main" java.lang.NoClassDefFoundError: StudentRecord
(wrong name: schoolClasses/StudentRecord)
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(Unknown Source)
at java.security.SecureClassLoader.defineClass(Unknown Source)
at java.net.URLClassLoader.defineClass(Unknown Source)
at java.net.URLClassLoader.access$100(Unknown Source)
at java.net.URLClassLoader$1.run(Unknown Source)
at java.security.AccessController.doPrivileged(Native Method)
at java.net.URLClassLoader.findClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
at java.lang.ClassLoader.loadClass(Unknown Source)
at java.lang.ClassLoader.loadClassInternal(Unknown Source)
```

Kita akan mendapatkan pesan kesalahan berupa NoClassDefFoundError yang berarti JAVA tidak mengetahui dimana posisi *class*. Hal tersebut disebabkan oleh karena *class* StudentRecord berada pada *package* dengan nama studentClasses. Jika kita ingin menjalankan class tersebut, kita harus memberi informasi pada JAVA bahwa nama lengkap dari *class* tersebut adalah schoolClasses.StudentRecord. Kita juga harus menginformasikan kepada JVM dimana posisi pencarian *package*, yang dalam hal ini berada pada direktori C:\. Untuk melakukan langkah – langkah tersebut, kita harus mengatur *classpath*.

Pengaturan classpath pada Windows dilakukan pada *command prompt* :

```
C:\schoolClasses> set classpath=C:\
```

dimana C:\ adalah direktori dimana kita menempatkan *package* (Buatlah *folder*/direktori sesuai nama *package* didalam *classpath*). Setelah mengatur *classpath*, kita dapat menjalankan program di mana saja dengan mengetikkan :

```
C:\schoolClasses> java schoolClasses.StudentRecord
```

Pada UNIX, asumsikan bahwa kita memiliki *class - class* yang terdapat dalam direktori /usr/local/myClasses, ketikkan :

```
export classpath=/usr/local/myClasses
```

Perhatikan bahwa Anda dapat mengatur *classpath* dimana saja. Anda juga dapat mengatur lebih dari satu *classpath*, kita hanya perlu memisahkannya dengan menggunakan **;** (Windows), dan **:** (UNIX). Sebagai contoh :

```
set classpath=C:\myClasses;D:\;E:\MyPrograms\Java
```

dan untuk sistem UNIX :

```
export classpath=/usr/local/java:/usr/myClasses
```

7.3. *Access Modifiers*

Pada saat membuat, mengatur *properties* dan *class methods*, kita ingin untuk mengimplementasikan beberapa macam larangan untuk mengakses data. Sebagai contoh, jika Anda ingin beberapa atribut hanya dapat diubah hanya dengan *method* tertentu, tentu Anda ingin menyembunyikannya dari *object* lain pada *class*. Di JAVA, implementasi tersebut disebut dengan *access modifiers.*

Terdapat 4 macam *access modifiers* di JAVA, yaitu : *public, private, protected* dan *default*. 3 tipe akses pertama tertulis secara ekplisit pada kode untuk mengindikasikan tipe akses, sedangkan yang keempat yang merupakan tipe *default*, tidak diperlukan penulisan *keyword* atas tipe*.*

7.3.1. Akses *Default*

Tipe ini mensyaratkan bahwa hanya *class* dalam *package* yang sama yang memiliki hak akses terhadap variabel dan *methods* dalam *class*. Tidak terdapat *keyword* pada tipe ini.

Sebagai contoh :

```
public class StudentRecord
{
    //akses dasar terhadap variabel
    int name;
    //akses dasar terhadap metode
    String getName(){
        return name;
    }
}
```

Pada contoh diatas, variabel nama dan *method* getName() dapat diakses dari *object* lain selama *object* tersebut berada pada *package* yang sama dengan letak dari file StudentRecord.

7.3.2. Akses Public

Tipe ini mengijinkan seluruh *class member* untuk diakses baik dari dalam dan luar *class*. *Object* apapun yang memiliki interaksi pada *class* memiliki akses penuh terhadap *member* dari tipe ini.

Sebagai contoh :

```
public class StudentRecord
{
    //akses dasar terhadap variabel
    public int name;
    //akses dasar terhadap metode
    public String getName(){
        return name;
    }
}
```

Dalam contoh ini, variabel name dan *method* getName() dapat diakses dari *object* lain.

7.3.3. Akses Protected

Tipe ini hanya mengijinkan *class member* untuk diakses oleh *method* dalam *class* tersebut dan elemen – elemen *subclass*. Sebagai contoh :

```
public class StudentRecord
{
    //akses pada variabel
    protected int name;
    //akses pada metode
    protected String getName(){
        return name;
    }
}
```

Pada contoh diatas, variabel name dan *method* getName() hanya dapat diakses oleh *method* internal *class* dan *subclass* dari *class* StudentRecord. Definisi *subclass* akan dibahas pada bab selanjutnya.

7.3.4. Akses *Private*

Tipe ini mengijinkan pengaksesan *class* hanya dapat diakses oleh *class* dimana tipe ini dibuat. Sebagai contoh :

```
public class StudentRecord
{
    //akses dasar terhadap variabel
    private int name;
    //akses dasar terhadap metode
    private String getName(){
        return name;
    }
}
```

Pada contoh diatas, variabel name dan *method* getName() hanya dapat diakses oleh *method* internal *class* tersebut.

Instance variable dari *class* secara *default* akan bertipe *private* sehingga *class* tersebut hanya akan menyediakan *accessor* dan *mutator methods* terhadap variabel ini.

Contoh program :

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


  public Segitiga(){
    tinggi=0;
     alas=0;
  }

  /*public Segitiga(double initTinggi, double initAlas){
    tinggi=initTinggi;
     alas=initAlas;
  }*/

  public Segitiga(double tinggi, double alas){
    this.tinggi=tinggi;
     this.alas=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[4];
     Byte i;
     String s="";

     S[0] = new Segitiga(6,8);
     S[1] = new Segitiga(5,3);
     S[2] = new Segitiga();
     S[3] = new Segitiga();

     S[2].settinggi(12.0);
     S[2].setalas(8.0);

     S[3].settinggi(11.23);
     S[3].setalas(7.7);

     for (i=0;i<4;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();
     }
  }
}
```

Contoh program yang memanggil Class program sebelumnya :

```
public class HitungSegitiga{

  public static void main (String args[]){
     Segitiga S1 = new Segitiga();
     Segitiga S2 = new Segitiga(10,5);

     S1.settinggi(12.0);
     S1.setalas(9.0);

     System.out.println("Tinggi = " + S1.gettinggi());
     System.out.println("Alas = " + S1.getalas());
     System.out.println("Luas Segitiga = " + S1.getluas());
     System.out.println();
     System.out.println("Tinggi = " + S2.gettinggi());
     System.out.println("Alas = " + S2.getalas());
     System.out.println("Luas Segitiga = " + S2.getluas());
  }
}
```

Referensi:

1. Hariyanto, Bambang, (2007), *Esensi-esensi Bahasa Pemrograman Java*, Edisi 2, Informatika Bandung, November 2007.
2. Utomo, Eko Priyo, (2009), *Panduan Mudah Mengenal Bahasa Java*, Yrama Widya, Juni 2009.
3. Tim Pengembang JENI, JENI 1-6, Depdiknas, 2007