Membuat dan Menggunakan Class

Membuat *class* pada java :

Untuk mendefinisikan suatu *class* pada java digunakan :

```
class ClassName {
...
}
```

suatu *class* dapat terdiri dari
- *attribute* / *data field*
- *method*

Attribute

Attribute pada suatu *class* dapat berupa *instance variable*, *class variable*, *constant variable* (konstanta).

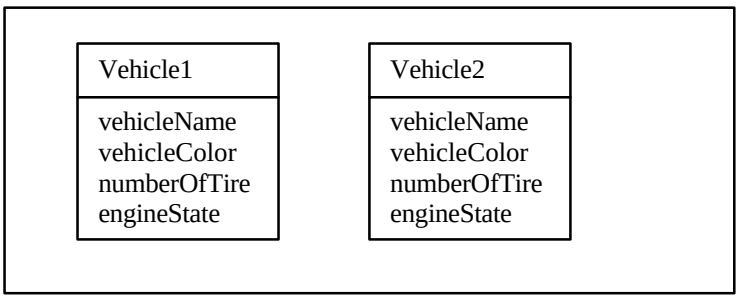
Instance variable merupakan variabel yang terletak didalam pendefinisian suatu *class*. *Instance variable* baru terbentuk apabila suatu *class* dibuat *instance*-nya.

Contoh dari *Instance Variable* :

```
class Vehicle {
     String vehicleName;
     String vehicleColor;
     int numberOfTire;
     boolean engineState;
     ...
}
```

vehicleName, vehicleColor, numberOfTire, dan engineState merupakan *instance variable* dari *class* Vehicle.

jika *class* Vehicle dibuat 2 buah *instance* misalkan Vehicle1, vehicle2 maka kedua-duanya akan memiliki ke empat *instance variable* :



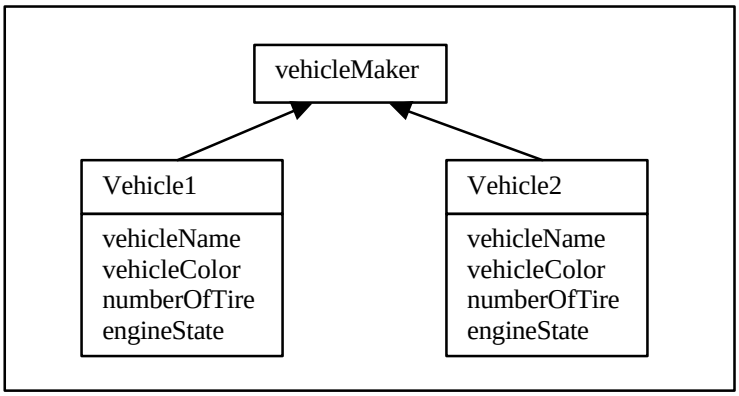
Gambar 3.1. Instance Variable

Class Variable merupakan variabel yang global terhadap suatu *class* dan semua *instance*-nya.
pendefinisian suatu *class variable* adalah dengan menambahkan *keyword* **static** didepan pendeklarasian variabel.

contoh *class variable* :

```
class Vehicle {
     String vehicleName;
     String vehicleColor;
     int numberOfTire;
     boolean engineState;
     static String vehicleMaker;
     ...
}
```

vehicleMaker merupakan *class variable*. jika dibuat *instance*-nya misalkan vehicle1 dan vehicle2 maka akan menjadi :



Gambar 3.2. Class Variable

Constant variable (konstanta) merupakan variabel yang nilainya tidak pernah berubah.
untuk membuat konstanta digunakan *keyword* **final** sebelum pendeklarasian variabel dengan harga awalnya.
contoh :

```
class Vehicle {
     final boolean RUN = true;
     final boolean STOP = false;
     ...
     String vehicleName;
     String vehicleColor;
     int numberOfTire;
     boolean engineState;
     static String vehicleMaker;
     ...
}
```

RUN dan STOP merupakan konstanta. untuk membuat konstanta tersebut menjadi global pada setiap *instance class* dapat ditambahkan *keyword* **static** didepan pendeklarasian konstanta sehingga menjadi :

```
static final boolean RUN = true;
static final boolean STOP = false;
```

Method

menunjukkan operasi dari suatu *class*. *Method* terdiri dari 4 bagian dasar :

- nama method
- tipe pengembalian method (object atau primitive data type)
- parameter
- isi method

Syntaks :

```
return type methodname (type1 arg1, type2 arg2, type3 arg3,...) {
  isi method ...
}
```

Instance Method merupakan *method* yang hanya tersedia apabila *instance* dari suatu *class* dibuat.
contoh :

```
class Vehicle {
     static final boolean RUN = true;
     static final boolean STOP = false;
     ...
     String vehicleName;
     String vehicleColor;
     int numberOfTire;
     boolean engineState;
     static String vehicleMaker;
     ...
     public void startEngine(boolean pEngineState) {
              engineState=pEngineState;
              if(engineState == RUN)
                System.out.println("The engine is run");
              else
          System.out.println("The engine is stop");
     }

    public boolean engineIsStart() {
        return engineState;
    }

}
```

startEngine merupakan *instance method* . returntype yang menggunakan *keyword* **void** berarti *method* tidak mengembalikan suatu nilai.
sedangkan engineIsStart merupakan *method* yang mengembalikan nilai. sehingga harus menggunakan *keyword* **return**.

cara menggunakan *instance method* harus dengan membuat *instance* dari *class* terlebih dahulu contoh :

```
    ...
    Vehicle vehicle1 = new Vehicle();

    vehicle1.startEngine(Vehicle.RUN);
    System.out.println(vehicle1.engineIsStart());
    ...
```

Class Method merupakan method yang tersedia untuk setiap *instance* dari suatu *class*.sehingga *class method* dapat digunakan baik setelah dibuat *instance*-nya ataupun tanpa dibuat *instance*-nya. Untuk membuat suatu class method digunakan *keyword* **static**.
contoh :

```
class Vehicle {
     static final boolean RUN = true;
     static final boolean STOP = false;
     ...
     String vehicleName;
     String vehicleColor;
     int numberOfTire;
     boolean engineState;
     static String vehicleMaker;
     ...
     public static void vehicleVersion {
         System.out.println("Vehicle Version is 0.1");
     }

}
```

vehicleVersion merupakan *class method*. *class method* dapat digunakan seperti contoh berikut :

```
    ...
    Vehicle.vehicleVersion();  //Tanpa membuat instance atau
    ...
    Vehicle vehicle1 = new Vehicle();  // dengan membuat instance
    vehicle1.vehicleVersion();
```

Accessor dan Mutator

Accessor merupakan *method* yang digunakan untuk melakukan membaca suatu variabel *private* di dalam suatu *class*.
contoh :

```
class Vehicle {
     static final boolean RUN = true;
     static final boolean STOP = false;
     ...
     private String vehicleName;
     ...
     public String getVehicleName() {
         return vehicleName;
```

```
    }
}
```

getVehicleName() merupakan *Accessor* untuk melihat isi dari variable *private* vehicleName.

Mutator merupakan *method* yang digunakan untuk mengubah isi suatu variabel *private* didalam suatu class.

Contoh :

```
class Vehicle {
     static final boolean RUN = true;
     static final boolean STOP = false;
     ...
     private String vehicleName;
     ...
     public void setVehicleName(String pVehicleName) {
      vehicleName=pVehicleName;
     }
}
```

setVehicleName merupakan Mutator untuk mengubah isi dari variabel *private* vehicleName.

Constructor

Merupakan *method* khusus yang digunakan untuk menginisialisasi *object*. *Constructor* tidak mempunyai *return type.* dan tiap *class* dapat mempunyai satu atau lebih *constructor*

Constructor mempunyai nama yang sama dengan nama *class* dan mempunyai bentuk umum :

```
    namaClass(type1 arg1, type2 arg2, ...) {
       inisialisasi pada constructor;
    }
```

contoh :

```
class Vehicle {
     static final boolean RUN = true;
     static final boolean STOP = false;
     ...
     private String vehicleName;
     private String vehicleColor;
     private int numberOfTire;
     private boolean engineState;
     static String vehicleMaker;
     ...
   Vehicle(String pVehicleName, String pVehicleColor,int
pNumberOfTire) {
      vehicleName = pVehicleName;
      vehicleColor = pVehicleColor;
      numberOfTire = pNumberOfTire;
   }
   ...
}
```

Vehicle(String pVehicleName, String pVehicleColor,int pNumberOfTire) merupakan *constructor* pada *class* Vehicle. untuk membuat *instance* dari Vehicle digunakan *constructor* tersebut.
contoh :

```
   Vehicle vehicle1 = new Vehicle("Suzuki","blue",4);
   Vehicle vehicle2 = new Vehicle("Mazda","Red",4);
```

bila suatu *class* tidak dibuatkan *constructor* maka java akan membuat suatu default *constructor* yaitu Vehicle()

Modifier

Merupakan *keyword* pada java yang menentukan penggunaan dari suatu *class, data field* dan *method*.

Class Modifier

class modifier diletakkan pada posisi :

```
modifier class ClassName {
 ...
}
```

contoh :

```
public class Vehicle {
   ...
}
```

Modifier	Explanation
(default)	class visible atau dapat digunakan pada package yang sama
public	class visible terhadap semua package yang berbeda-beda
final	class tidak dapat dibuat / di extend menjadi subclass

Method Modifier

Method modifier diletakkan pada posisi :

```
modifier returntype methodName(type1 arg1,type 2 arg2, ...) {
   ...
}
```

contoh :

```
public String getVehicleName() {
  return vehicleName;
}
```

Modifier	Explanation
(default)	method visible atau dapat digunakan pada package yang sama
public	method visible terhadap semua package yang berbeda-beda
private	method visible hanya didalam class itu sendiri
protected	method visible didalam package atau subclassnya
static	mendefinisikan class method
final	method tidak dapat dirubah / dioverride pada subclassnya
abstract	method harus dioverride / didefinisikan pada subclassnya
sychronized	method hanya dapat dieksekusi oleh sebuah thread dalam waktu

	bersamaan

Data Modifier
Data modifier diletakkan pada posisi :

```
modifier datatype datafieldName = inisialisasi;
```

contoh :

```
private String vehicleName;
public int vehicleGear;
```

Modifier	Explanation
(default)	data field visible atau dapat digunakan pada package yang sama
public	data field visible terhadap semua package yang berbeda-beda
private	data field visible hanya didalam class itu sendiri
protected	data field visible didalam package atau subclassnya
static	mendefinisikan class variable
final	mendefinisikan constant

"this" keyword

Dalam setiap *method* non-static (bukan *class method*) yang dipanggil melalui *object* / *instance* dari *class*. terdapat secara implisit suatu variable reference yang bernama **"this"**. "this" menunjuk ke *object* yang memanggil *method* itu.

contoh *Constructor* yang memanggil "this" :

```
public class Lingkaran {
    public double x,y,r;   //instance variable
    public Lingkaran (double x, double y, double r) {
        this.x = x;    //instance variabel x diisi x dari argumen
        this.y = y;    //instance variabel y diisi y dari argumen
        this.r =  r;  //instance variabel r diisi r dari argumen
    }

    public Lingkaran () {
       this(0.0, 0.0, 0.1);    // memanggil constructor Lingkaran
    }
}
```

new Operator

new Operator digunakan untuk membuat suatu *instance* dari *class* menjadi *object* yang dapat digunakan.

Syntaks :

```
ClassName objectName = new  ClassConstructor(arg1,arg2...);
```

Contoh :
pada Vehicle.java

```
public class Vehicle {
     public static final boolean RUN = true;
```

```
     public static final boolean STOP = false;
     private String vehicleName;
     private String vehicleColor;
     private int numberOfTire;
     private boolean engineState;
     static String vehicleMaker;

   Vehicle(String pVehicleName, String pVehicleColor,int
pNumberOfTire) {
         vehicleName = pVehicleName;
         vehicleColor = pVehicleColor;
         numberOfTire = pNumberOfTire;
         engineState = this.STOP;
     }

     Vehicle() {
         this(“”,””,0);
         engineState = this.STOP;
     }


    public void setVehicleName(String pVehicleName) {
      vehicleName=pVehicleName;
     }

     public String getVehicleName() {
      return vehicleName;
     }

    public static void vehicleVersion {
         System.out.println(“Vehicle Version is 0.1”);
    }

   public void startEngine(boolean pEngineState) {
              engineState=pEngineState;
         if(engineState == RUN)
            System.out.println(“The engine is run”);
         else
            System.out.println(“The engine is stop”);
     }

    public boolean engineIsStart() {
        return engineState;
    }

   public static void main(String [] args) {
       Vehicle.VehicleVersion();

       Vehicle MyCar = new Vehicle(“Suzuki”,”blue”,4);
       System.out.println(“My car is “+MyCar.getVehicleName());
       MyCar.startEngine(Vehicle.RUN);

   }
}
```

Import Dan Package

Package

Package merupakan cara untuk membuat grup dari suatu class.

Package berguna untuk :

- melakukan organisasi class-class menjadi unit –unit
- menghindari konflik nama pada saat membuat dua nama class yang sama
- melakukan proteksi terhadap class, variables dan method dalam skala lebih besar
- Memberikan identitas terhadap class-class

cara membuat Package adalah dengan menggunakan keyword package
contoh :

```
package MyClass;

class ParseCombiner {
...
}

package MyClass.util;

class ParseCombiner {
...
}
```

aturan yang perlu diperhatikan adalah nama package merupakan nama directory dari tempat class itu berada:
contoh :

```
package MyClass;

maka directory :
  +MyClass : ParseCombiner.java

package MyClass.util;
  +MyClass +
                |
              + util : ParseCombiner.java
```

import

import digunakan untuk menggunakan class-class dari suatu package.
Syntaks : import PackageName;
contoh :

```
import MyClass.ParseCombiner;  //mengimport sebuah class
atau
import MyClass.util.*;  //mengimport semua class pada package Myclass.util
```