CLASS, METHOD DAN MODIFIER

CLASS (KELAS)

- Kelas mendefinisikan sekumpulan objek yang memiliki kesamaan sifat dan perilaku
- Ada dua kelompok kelas :
  - kelas standard
  - kelas yang didefinisikan sendiri
- Kumpulan dari method/kelas standard dalam java dikenal dengan API (Aplication Programming Interface)

Pengertian Kelas

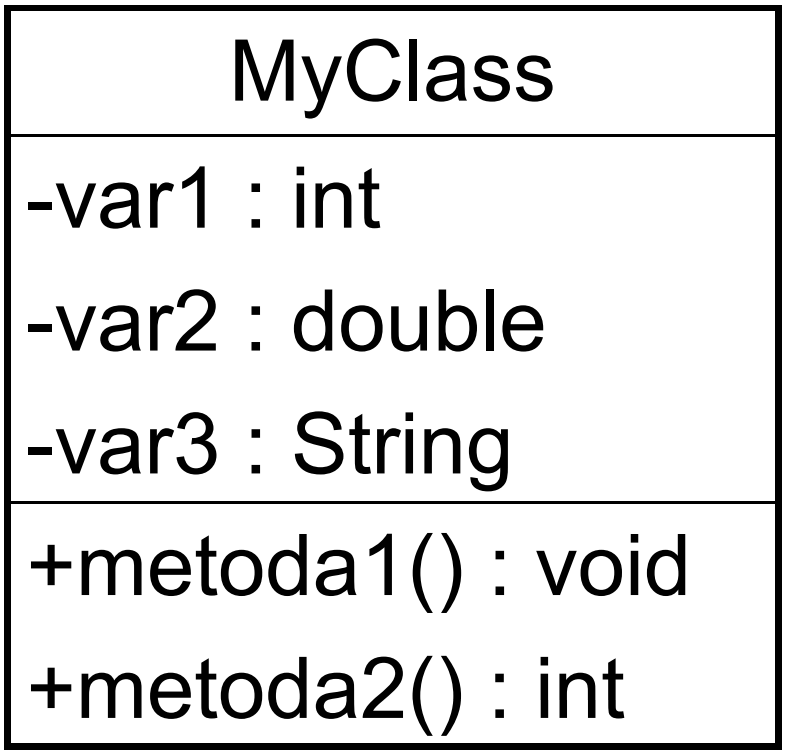
- untuk membuat objek, dan berperan sebagai tipe data dari objek
- sarana pengkapsulan kumpulan data/properti dan kumpulan method

Membuat Class dan Obyek dalam JAVA

- Dibentuk sebuah Class yang merupakan interface (bentuk) dari obyek yang akan dibuat

- Setelah Class didefinisikan, maka Obyek dapat dideklarasikan sebagai *bentuk* dari Class tsb (*instance* of a class)

Misal Diagram Class :



Deklarasi Class :

```
public class MyClass {
  //Deklarasi Anggota Kelas(member)
 int       var1;
 double    var2;
 String    var3;

 void metoda1 (parameter…) {
     …
 }
 int metoda2 (parameter…) {
     …
 }
}
```

Anatomi Kelas

```
[modifier1] class NamaKelas [modifier2] {
    classbody
}
```

- *Classbody* : properti/attribute, constructor dan method
- Modifier sifatnya **optional**, menunjukkan sifat-sifat tertentu dari kelasnya, methodnya, atau attributenya

- 10 keyword modifier1 :
  1. akses (*public*, *protected*, *default*, *private*)
  2. *final*
  3. *static*
  4. *abstract*
  5. *synchronized*
  6. *native*
  7. *storage* (*transient*, *volatile*)
- 2 keyword modifier2, yaitu *extends* dan *implements*.

DEKLARASI ATTRIBUTE (VARIABEL ANGGOTA DATA)

- Letak di dalam *classbody* (di luar method).
- Bentuk umum :

```
[modifier] tipedata namavariabel;
[public] [static] final tipedata NAMA_KONSTANTA = nilai;
```

- Contoh :

```
public class CircleClass {
    public static final double PI = 3.14159265358979323846;
    public double x, y, r;
   // dan seterusnya
}
```

METHOD

- Tingkah laku dari suatu objek
- Letak di dalam *classbody*
- Bentuk umum :

```
[modifier] tipe_return_value namaMethod(tipe parameter) {
  methodbody;
}
```

- Modifier boleh lebih dari satu (dipisah oleh spasi).
- Method dpt tdk mempunyai nilai balik, disebut sebagai **void**, atau mempunyai nilai balik berupa tipe primitive Java atau class.
- Pasangan tipe dan parameter dapat lebih dari satu (dipisah oleh koma).

Contoh Method :

```
public void f1() {
   System.out.println(”Fungsi f1”);
}
public int f2() {
   return 300;
}
public String f3() {
   return ”Hallo Apa Khabar”;
}
```

f1 menjalankan routine untuk mencetak string

f2 memberikan nilai balik berupa tipe integer

f3 memberikan nilai balik berupa String

Method

- Bentuk umum method **main()** sebagai berikut :

```
[modifier] tipe_return_value main(String args[]) {
    methodbody
}
```

- Ada dua sintaks pemanggilan suatu method :

```
namaObjek.namaMethod([nilaiParamater]);
namaKelas.namaMethod([nilaiParamater]);
```

Parameter

- Parameter (argumen) : daftar variabel yg disertakan pada sebuah method
- Variabel ini akan diolah sebagai input untuk method tersebut

Ruang Lingkup Variabel

- Disebut juga sbg Variable Scope : jangkauan nilai yg berlaku untuk sebuah variabel.
- Variable dapat diakses secara global, lokal atau hanya dalam satu blok saja.

Program

```
public class MyClass {
     int v = 100;  //var global
     int x = 0;    //var global

   public void mySubroutine1() {
     int v = 300;  // var lokal
     System.out.println("var= " + v);
   }
   public void mySubroutine2() {
     for (int i=0; i<v; i+=100)
          System.out.println("i= " + i);
     i=v; //Error, i tidak dapat diakses
          //i hanya di blok for-loop
   }
   public static void main (String args[]) {
     int x=100;  //var lokal
     int y;
     int z;

     z=x;
   }
}
```

1

2

3

4

no 3 pada program diubah:

```
public class MyClass {
(1)     int v = 100;  //var global
        int x = 0;    //var global

    public void mySubroutine1() {
(2)     int v = 300;  // var lokal
        System.out.println("var= " + v);
    }
    public void mySubroutine2() {
(3)     int i;
        for (i=0; i<v; i+=100)
              System.out.println("i= " + i);
        v=i; //i lokal, dapat diakses
             //v global, nilai diganti dengan i
    }
(4) public static void main (String args[]) {
        int x=100;  //var lokal
        int y;
        int z;

        z=x;
    }
}
```

Tidak semua member (class, attribute, dan method) dapat diakses method, berikut tabel aksesnya :

method	member (class, attribute, method)
static	static boleh lewat objek ataupun class, boleh langsung kalau dalam kelas sendiri
static	non static boleh lewat objek, langsung tidak boleh, lewat class tidak boleh
non static	static boleh lewat objek ataupun class, boleh langsung kalau dalam kelas sendiri
non static	non static boleh hanya lewat objek, langsung tidak boleh, lewat class tidak boleh

Method *static*

- Keyword ***static*** di depan nama method berarti sebagai penempatan memori (alokasi memori)
- Java Run Time akan segera menyiapkan memori untuk menempatkan **code** yang berkunci kata **static** (*static loading),* sedangkan yang tidak menggunakan kata static *(non static),* akan diaktifkan pada saat dibutuhkan *(dynamic loading)*

Contoh :

```
public class MyClassXYZ {
    public static void printX() {
      …
    }
    public void printY() {
      static int i;
      …
    }
    public void printZ() {
      int z;
    }
    public static void main(String arg[]) {
      …
    }
}
```

- Proses Loading (eksekusi awal) oleh Java Run Time dilakukan untuk method main(), printX() dan printY().
- Method printZ() tidak dilakukan Loading sebelum ada program code yang memanggilnya.

Method *static*

- Method static tidak dapat mengakses variabel di luar method tersebut bila variabel tersebut bukan static
- Method static tidak dapat mengeksekusi method lain yang bukan static

Contoh :

```
public class MyStatic {
     int saldo = 400;

     public void cetakSaldo() {
       System.out.println("Saldo = " + saldo);
     }

     public static void main(String args[]) {
       saldo = 300;
       cetakSaldo();
     }
}
```

- Program main akan mengakses variabel saldo dan menjalankan method cetak.
- Kompilasi memberikan pesan error.
- Error → variabel saldo bukan tipe static (belum ada di memori), dan class belum dialokasikan ke memori, sedangkan main() adalah method static yg lebih dulu pada awal aplikasi diinisialisasi, code main() diaktifkan ke memori.

Maka variabel saldo diberi kata static :

```
public class MyStatic {
    static int saldo = 400;

    public void cetakSaldo() {
      System.out.println("Saldo = " + saldo);
    }

    public static void main(String args[]) {
      saldo = 300;
      cetakSaldo();
    }
}
```

- Masih memberikan pesan error.
- Error → method cetakSaldo() membutuhkan kunci kata untuk alokasi memori **static**

Maka method cetakSaldo diberi kata static :

```
public class MyStatic {
    static int saldo = 400;

    public static void cetakSaldo() {
      System.out.println("Saldo = " + saldo);
    }

    public static void main(String args[]) {
      saldo = 300;
      cetakSaldo();
    }
}
```

Method getter dan setter

- Method dasar ada dua jenis yaitu **getter()** dan **setter().**
- Method jenis getter() merupakan method-method yang berfungsi untuk mendapatkan informasi dari class,
- sedangkan jenis setter() berfungsi untuk menentukan isi atribut (variabel) dalam class.

Contoh Method getter dan setter

```
public class Dog {
   private int weight; // information hiding
   public int getWeight() {   //getter
      return weight;
   }
   Public void setWeight(int newWeight) {   //setter
      weight = newWeight;
   }
}
public class TesDog {
  public static void main(String[] args) {
      Dog d = new Dog();
      d.setWeight(42);
      System.out.println(”Dog d’s weight is ”+d.getWeight())
   }
}
```

Overloading terhadap Method

- mendefinisikan dua atau lebih method di dalam kelas sama, nama sama, deklarasi parameter berbeda.
- Tugas dari method-method yang dioverloading tersebut berbeda.

Contoh Overloading terhadap Method

```
import java.lang.*;
public class Perkalian {
   private double pangkat(int a, int b) {
      double hasil = 1.0;
      //kode program
      return hasil;
   }
   private double pangkat(double a, int b) {
      double hasil = 1.0;
      //kode program
      return hasil;
   }
public static void main(String[] args) {
       Perkalian kali = new Perkalian();
       System.out.println(kali.pangkat(10,308));
       System.out.println(kali.pangkat(0.5,2));
       // dst untuk data yg lain
   }
}
```

Keyword this

- 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.

Keyword this

Contoh penggunaan yang eksplisit :

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

Contoh pemanggilan implisit :

```
class RectangleToy {
   private double width, height;
    public void setRectangle(double newwidth, double newheight)
{
      width = newwidth;
      height = newheight;
   }
}
```

CONSTRUCTOR

- method yang tidak memiliki return value (secara implisit adalah instant dari kelasnya),
- nama sama dengan nama kelas,
- dapat diberi modifier akses (public, protected, default, private).
- Bentuk umum :

```
[modifier] NamaConstructor(tipe namaparameter) {
   constructorBody;
}
```

- Tujuan constructor dibuat adalah untuk melakukan **inisialisasi** yang diperlukan objek baru.

Program MyClass :

```
public class MyClass1 {
   /*Creates new MyClass1 */
   public MyClass1() {
        f1();
        int x = f2();
        System.out.println(" x = " + x);
        String s;
        s = f3();
        System.out.println(s);
    }
  public void f1() {
        System.out.println("Fungsi f1");
  }
  public int f2() {
        return 300;
  }
  public String f3() {
        return "Hallo Apa Khabar";
  }
  public static void main (String args[]) {
        new MyClass1();
  }
}
```

Contoh constructor dan overloadingnya :

```
class PersonToy {
    String name;
    String addressLine;
    int age;

    public PersonToy() {
        name = ” ”;
        addressLine = ” ”;
        age = 0;
    }

    public PersonToy (String newName, String newAddress, int newAge) {
      name = newName;
      addressLine = newAddressLine;
      age = newAge;
    }
}
```

MODIFIER

- untuk menentukan hubungan suatu unsur kelas dengan unsur kelas lainnya, contohnya hubungan kepemilikan antara kelas dan objek.
- Modifier Akses :
  1. *public*,
  2. *protected*,
  3. *default*,
  4. *private*

Tabel Wilayah Modifier **Akses**

Wilayah Akses	*public*	*protected*	*default*	*private*
Di kelas yg sama	√	√	√	√
Beda kelas, di package yg sama	√	√	√	x
Beda kelas, beda package, di kelas turunan	√	√	x	x
Beda kelas, beda package, tidak di kelas turunan	√	x	x	x

Modifier **Final**

- **no extended** class
- **no overrided** method
- membentuk suatu attribute menjadi konstanta.

Modifier **Static**

- **no need instanciation.**

  method dan attribute milik kelas, menjadi sifat bersama dari semua objek dalam kelas tersebut (tidak memerlukan instansiasi objek untuk menjalankannya).
- **no overrided** method

  method main() harus memiliki modifier static.

Modifier **Abstract**

- **no instanciation**

  Abstract class adalah kelas murni (tanpa objek) dan tidak boleh memiliki objek (tidak boleh ada instansiasi)
- **should be overrided**

  method-method yang abstract harus disempurnakan oleh kelas turunannya melalui override.
- Konsekuensi penggunaan sifat abstract :
  1. Tidak dapat dibuat constructor yang abstract.
  2. Tidak dapat dibuat method yang static dan abstract (kedua sifat sali ng kontradiktif).
  3. Tidak diijinkan membuat method yang private dan abstract (kedua sifat ini juga saling kontradiktif).

Modifier **khusus modifier method**

- **Synchronized**

  Pada lingkungan multithread, dimungkinkan lebih dari satu jalur eksekusi yang berjalan di kode yang sama, kondisi tersebut dapat diatur sehingga pada selang waktu tertentu hanya ada satu jalur eksekusi yang diijinkan di method yang synchronized (eksekusi dilakukan secara mutual exclusive).

- **Native**

  Modifier ini digunakan untuk memanggil/mengakses method yang ditulis dalam bahasa C/C++. Seperti method yang abstract, method yang native hanya berupa prototype, implementasi method ini berada di file external (dalam folder yang sama).

Modifier **khusus modifier attribute**

Modifier **Extends**

- Bila terjadi pewarisan, kelas yang mewariskan method dan attributenya disebut kelas super, sedangkan yang diwariskan disebut subkelas.
- Kelas yang memiliki modifier ini berarti merupakan subkelas dari suatu kelas super.
- Caranya :

```
[modifier1] class NamaSubKelas extends NamaKelasSuper {
   classBody
}
```

Modifier **Implements**

- Kelas yang memilki modifier implements artinya kelas tersebut mengimplementasikan satu atau lebih interface. Bila terdapat lebih dari satu interface, gunakan tanda koma di antara interface-interface tersebut.
- Caranya :

```
[modifer] class NamaKelas implements NamaInterface1, NamaInterface2 {
   classBody
}
```