

Pertemuan 5. Karakteristik OOP

Constructor, Inheritance, Polymorphism dan Encapsulation

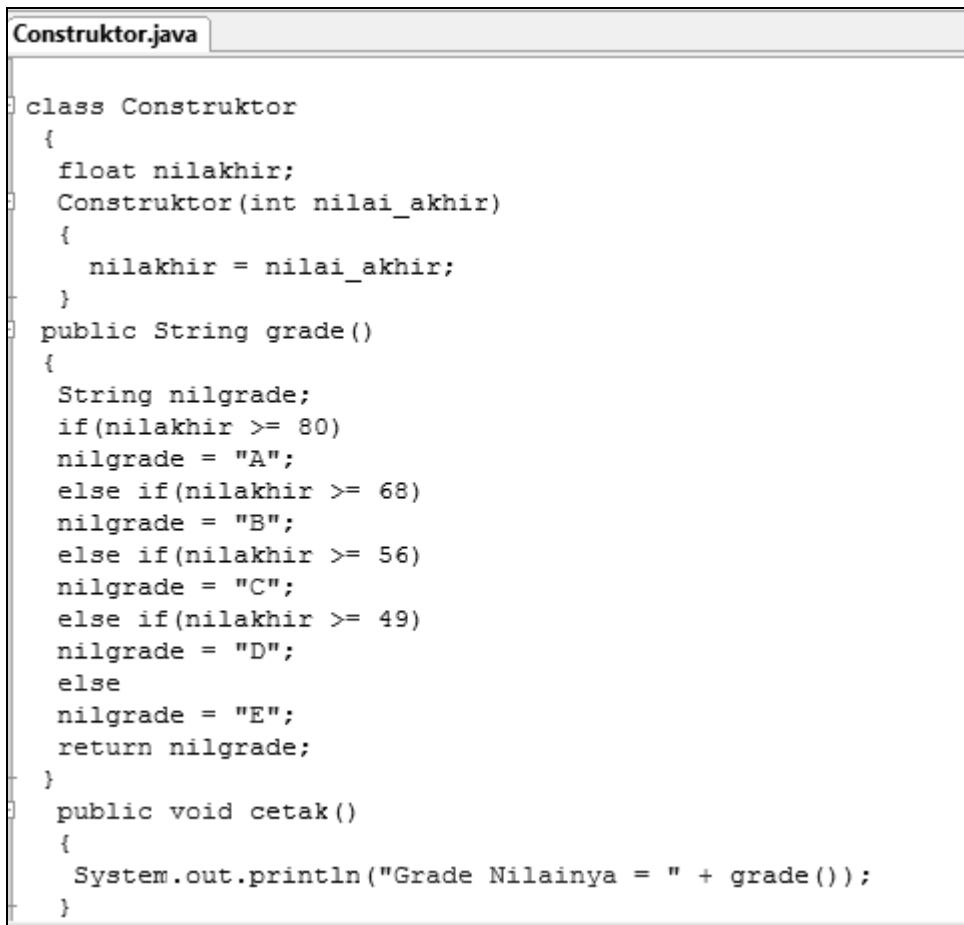
A. Constructor

Constructor merupakan suatu method yang akan memberikan nilai awal pada saat suatu objek dibuat. Pada saat program dijalankan, constructor akan langsung memberikan nilai awal pada saat perintah new, membuat suatu objek.

Pada saat kita bekerja dengan constructor, hal mendasar yang perlu diperhatikan, yaitu :

- Nama Constructor sama dengan nama Class.
- Tidak ada *return type* yang diberikan kedalam *Constructor Signature*.
- Tidak ada *return statement*, didalam tubuh *constructor*.

Contoh constructor :



```

Konstruktor.java
class Konstruktor
{
    float nilakhir;
    Konstruktor(int nilai_akhir)
    {
        nilakhir = nilai_akhir;
    }
    public String grade()
    {
        String nilgrade;
        if(nilakhir >= 80)
            nilgrade = "A";
        else if(nilakhir >= 68)
            nilgrade = "B";
        else if(nilakhir >= 56)
            nilgrade = "C";
        else if(nilakhir >= 49)
            nilgrade = "D";
        else
            nilgrade = "E";
        return nilgrade;
    }
    public void cetak()
    {
        System.out.println("Grade Nilainya = " + grade());
    }
}

```

```

public static void main(String[] args)
{
    Konstruktor hasil = new Konstruktor(67);
    hasil.cetak();
}
}

```

Hasilnya :

```

General Output
-----
Grade Nilainya = C

Process completed.

```

Constructor Overloading

Overloading adalah suatu cara membuat lebih dari *constructor* pada suatu *class*. Supaya pengaksesan *constructor* tersebut lancar, maka sebagai pembedanya adalah tipe parameter dan atau jumlah parameternya. Untuk lebih jelasnya anda bisa lihat program dibawah ini :

```

ConstrukOver.java
class ConstrukOver
{
    float nilakhir, a;
    ConstrukOver(int nilai_akhir){
        nilakhir = nilai_akhir;
    }
    ConstrukOver(int nil1, int nil2){
        nilakhir = nil1 + nil2;
    }
    public String grade()
    {
        String nilgrade;
        if(nilakhir >= 80)
            nilgrade = "A";
        else if(nilakhir >= 68)
            nilgrade = "B";
        else if(nilakhir >= 56)
            nilgrade = "C";
        else if(nilakhir >= 49)
            nilgrade = "D";
        else
            nilgrade = "E";
        return nilgrade;
    }
}

```

```

public void cetak()
{
    System.out.println("Grade Nilainya = " + grade());
}
public static void main(String[] args)
{
    ConstrukOver hasil = new ConstrukOver(67);
    hasil.cetak();
    ConstrukOver hasilnya = new ConstrukOver(45, 35);
    hasilnya.cetak();
}
}

```

Hasilnya :

```

General Output
-----
Grade Nilainya = C
Grade Nilainya = A

Process completed.

```

B. Inheritance

Pewarisan, bahasa kerennya *Inheritance*. Dalam pemrograman berbasis objek, Inheritance memungkinkan suatu *Class* bisa mewariskan atribut dan *Method* kepada *Class* yang lainnya atau *subClass*, sehingga membentuk *Class* hirarki.

Inheritance merupakan suatu aspek atau pengarah pokok kecerdasan manusia untuk mencari, mengenali, dan menciptakan hubungan antar konsep. Kita membangun hirarki, matriks, jaringan, dan hubungan timbal balik lain untuk menjelaskan dan memahami tata cara di mana hal-hal saling berhubungan.

Tanpa Inheritance, kelas-kelas merupakan sebuah unit berdiri sendiri. Inheritance, akan membentuk suatu konsep dimana jika konsep yang diatas berubah, maka perubahan akan berlaku dibawahnya.

Inheritance sangat mirip dengan hubungan orang tua dengan anak. Manakala suatu kelas menerima warisan dari semua anggota data dan fungsi menerima warisan, walaupun tidak semua di antara mereka akan dapat diakses oleh anggota fungsi dari kelas

Penentuan akses pada Inheritance ada tiga macam, yaitu :

a. Public

Penentuan akses berbasis Public, menyebabkan anggota dari public dari sebuah class utama akan menjadi anggota public class turunan dan menyebabkan juga anggota protect class utama menjadi anggota protect class turunan, tetapi untuk anggota class private tetap pada private class utama.

b. Private

Penentu akses berbasis Private, menyebabkan anggota dari anggota public dari class utama akan menjadi anggota protect class turunan, dan menyebabkan anggota dari class utama menjadi anggota private dari class turunan. Anggota Private dari class utama tidak dapat diakses kecuali oleh class utama.

c. Protected

Penentu akses berbasis Protect, menyebabkan anggota dari anggota protect dan public dari class utama akan menjadi anggota private dari class turunan. Anggota Private dari class utama selalu menjadi anggota private class utama.

Mulai Melakukan Pewarisan

Untuk melakukan pewarisan suatu class, yang perlu diperhatikan, yaitu kita harus menggunakan *keyword extends*. *Extends* digunakan untuk mendeklarasikan suatu class adalah turunan dari class lainnya. Terpenting yang perlu diingat, sebuah class tidak diperbolehkan mempunyai lebih dari satu class induk atau superclass. Supaya lebih jelas bagaimana melakukan pewarisan suatu class, sekarang perhatikan beberapa program dibawah ini :

Matematika.java

```
class Matematika
{
    private int a, b;
    public Matematika()
    {
        a = 1;
        b = 2;
    }
    public int tambah()
    {
        return a + b;
    }
    public int kali()
    {
        return b * 3;
    }
}
```

Hitungan.java

```
class Hitungan extends Matematika
{
    private int x, y;
    public Hitungan()
    {
        x = 1;
        y = 2;
    }
    public Hitungan(int i, int j)
    {
        x = i;
        y = j;
    }
    public int tambah()
    {
        return x + y;
    }
    public int kali()
    {
        return y * 3;
    }
}
```

LatHitungMatematika.java

```
class LatHitungMatematika
{
    public static void main(String[] args)
    {
        Hitungan ngitung = new Hitungan(5, 6);
        System.out.println(" Hasil Pertambahannya = " + ngitung.tambah());
        System.out.println(" Hasil Perkaliannya = " + ngitung.kali());
    }
}
```

C. Polymorphism

Karakteristik dari polymorphism yaitu memungkinkan suatu objek dapat memiliki berbagai bentuk atau banyak bentuk. Bentuk dari objek ini bisa sebagai *Object* dari *Class*nya sendiri atau *Object* dari *superClass*nya.

Pada polymorphism kita akan sering menjumpai 2 (dua) istilah yang sering digunakan dalam pemrograman berbasis objek, istilah tersebut yaitu :

a. *Overloading*.

Overloading yaitu menggunakan 1 (satu) nama objek untuk beberapa *Method* yang berbeda ataupun bisa juga beda parameternya.

b. *Overriding*

Overriding akan terjadi apabila ketika pendeklarasian suatu *Method* subClass dengan nama objek dan parameter yang sama dengan *Method* dari superClassnya.

Contohnya sebagai berikut :

Binatang.java

```
class Binatang
{
    String namaBinatang;
    Binatang()
    {
    }

    Binatang(String namaBinatang)
    {
        this.namaBinatang = namaBinatang;
    }
    public void cetakjenis()
    {
        System.out.println("Nama Binatang : "+ namaBinatang);
    }
}
```

suara.java

```
class suara extends Binatang
{
    String suara;
    suara()
    {
        super();
    }
    public void cetakjenis()
    {
        suara="Mengaum";
        System.out.println("Suara : " + suara);
    }
}
```

jenisBinatang.java

```
class jenisBinatang extends Binatang
{
    String jenisBinatang;
    jenisBinatang(String jenisBinatang)
    {
        super(jenisBinatang);
    }
    public void cetakjenis()
    {
        super.cetakjenis();
        jenisBinatang="Predator";
        System.out.println("Jenis Binatang : " + jenisBinatang);
    }
}
```

Harimau.java

```
class Harimau
{
    public static void main(String[] args)
    {
        suara binatang = new suara();
        jenisBinatang suara = new jenisBinatang("Harimau");
        suara.cetakjenis();
        binatang.cetakjenis();
    }
}
```

Hasilnya :

```
General Output
-----Configuration:
Nama Binatang : Harimau
Jenis Binatang : Predator
Suara : Mengaum

Process completed.
```

D. Encapsulation (Enkapsulasi)

Merupakan suatu cara bagaimana menyembunyikan sedemikian rupa suatu proses kedalam sistem, hal ini berguna untuk menghindari interferensi dari luar sistem dan juga lebih untuk menyederhakan sistem itu sendiri.

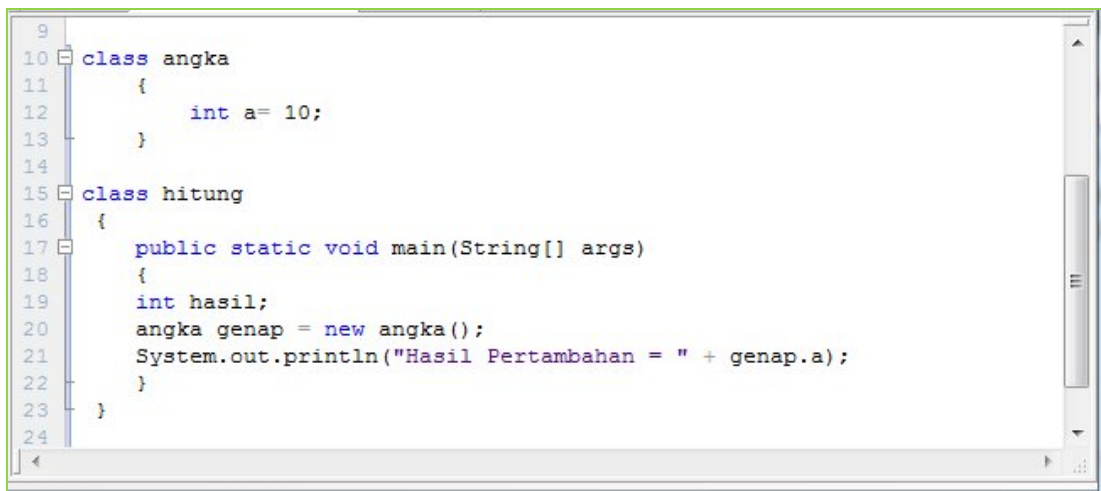
Fungsinya :

- Menyembunyikan rincian-rincian implementasi dari pemakai.
- Manfaat untuk pemrogram : Implementation hiding (kendali pengasesan private atau protected pada elemen data) dan modularity

Kontrol Akses Pada Enkapsulasi

1. Akses Default

Variabel atau metode bisa dideklarasikan tanpa kontrol akses modifier yang tersedia untuk setiap kelas lainnya dalam satu paket. Setiap variabel dideklarasikan tanpa modifier dapat dibaca atau diubah oleh yang lain di kelas satu paket. Metode apapun dapat dinyatakan dengan cara yang sama dan dapat dipanggil oleh kelas lain dalam satu paket.



```
9
10 class angka
11 {
12     int a = 10;
13 }
14
15 class hitung
16 {
17     public static void main(String[] args)
18     {
19         int hasil;
20         angka genap = new angka();
21         System.out.println("Hasil Pertambahan = " + genap.a);
22     }
23 }
24
```

2. Akses Private

Secara utuh menyembunyikan variabel atau metoda yang akan digunakan oleh kelas-kelas lain, dengan menggunakan modifier private. Satu-satunya tempat dimana variabel-variabel atau metoda-metoda ini dapat diakses yaitu dari dalam kelasnya sendiri. Method Private dapat dipanggil oleh metoda-metoda lain di dalam kelas mereka sendiri tetapi tidak bisa dipanggil oleh lainnya.

Variabel-variabel private akan bermanfaat pada 2 (dua) keadaan, yaitu apabila kelas-kelas lain tidak punya alasan untuk menggunakan variabelnya dan apabila kelas lain bisa menghadirkan nilai dengan mengubah variabel pada satu cara yang tidak sesuai.

```
9
10 class angka
11 {
12     private int a= 10;
13     public int getVAR ()
14     {
15         return this.a;
16     }
17 }
18
19 class hitung
20 {
21     public static void main(String[] args)
22     {
23         int hasil;
24         angka genap = new angka();
25         System.out.println("Ini angka = " + genap.getVAR());
26     }
27 }
28
29
```

Menggunakan modifier private adalah cara utama suatu object mengencapsulasi dirinya.

3. Akses Public

Modifier public membuat suatu variabel atau metoda dengan sepenuhnya dapat diakses oleh semua kelas.

```
9
10 class saya {
11
12     public static void main(String[] args)
13     {
14         System.out.println("Nama saya : Fakhrina Amalia");
15     }
16
17
18 }
```

Method main(), haruslah bersifat public, karena tidak akan bisa dipanggil oleh Java Interpreter untuk menjalankan class.

4. Akses Protect

Protected digunakan untuk membatasi variabel dan metoda untuk digunakan oleh Sub Class dari satu Class dan Class-class lain di dalam paket yang sama