Encapsulation

Encapsulation yang terkadang disebut juga dengan information hiding pada dasarnya adalah kombinasi antara data dan method pada sebuah class yang ditujukan untuk menyembunyikan detail dari user (pengakses class) terhadap suatu object. Jika kita ambil contoh, maaf, yang agak sedikit vulgar tentang bagaimana orang lain "mengakses data" milik kita. Anggaplah orang ingin tahu apakah kita mempunyai penyakit panu yang merupakan suatu "data" pada diri kita. Sudah barang tentu, secara umum, kita tidak akan mengijinkan orang lain melihat langsung kulit kita karena itu adalah bagian "pribadi". Akan tetapi orang lain dapat mengetahuinya menggunakan "metode" penyampaian yang kita berikan: "Saya tidak panuan". Jadi ada bagian "publik" dan juga ada bagian "pribadi".

Di dalam OOP, encapsulation dapat dilakukan dengan terlebih dahulu memahami **access modifier** yang mendefiniskan bagaimana suatu data atau method dapat diakses**.** Ada empat macam access modifier pada OOP, yaitu:

- Private : hanya diakses class itu sendiri
- Public : dapat diakses dari manapun
- Protected : hanya dapat diakses dari package (satu folder) dan subclass
- Default : tanpa modifier, hanya bisa diakses dari package dan class itu sendiri.

Dengan menggunakan encapsulation kita dapat membatasi akses langsung suatu class atau program kecuali melalui suatu method yang sudah diberikan.

Perhatikan class Mahasiswa dan class MahasiswaTest berikut ini. Perhatikan pula cara deklarasi atribut dan method-method yang terdapat dalam class Mahasiswa.

```
Public class Mahasiswa{
//deklarasi private attribute
    private String nama;
    private String nim;
    private String alamat;

public Mahasiswa() {

    }
//Constructor dengan dua input parameter
    public Mahasiswa(String nama, String nim) {
        this.nama = nama;
        this.nim = nim;
    }
//Constructor dengan tiga input parameter
    public Mahasiswa(String nama, String nim, String alamat) {
        this.nama = nama;
        this.nim = nim;
        this.alamat = alamat;
    }
//mengakses atribut alamat
    public String getAlamat() {
        return alamat;
    }
//mengakses atribut nama
    public String getNama() {
        return nama;
    }
//mengakses attribut nim
    public String getNim() {
```

```
        return nim;
    }
//mengisi atribut alamat
    public void setAlamat(String alamat) {
        this.alamat = alamat;
    }
}
```

Class MahasiswaTest

```
class MahasiswaTest{
    public static void main(String[] args){
        Mahasiswa objMhs;
        String localNama, localNim;
        objMhs=new Mahasiswa("Andriani","123456");
        objMhs.setAlamat("Watugong 212 Malang");
        localNama=objMhs.getNama();
        localNim=objMhs.getNim();
    }
}
```

Bukankah lebih sederhana jika pada class Mahasiswa kita deklarasikan data nama, nim, dan alamat dengan acces modifier public daripada harus membuat method-method khusus untuk mengaksesnya. Demikian pula cara penggunaannya class MahasiswaTest, mengambil dan mengubah isi data akan lebih sederhana.

```
class Mahasiswa{
//deklarasi private attribute
    public String nama;
    public String nim;
    public String alamat;
}

class MahasiswaTest{
    public static void main(String[] args){
        Mahasiswa objMhs;
        String localNama, localNim;
        objMhs=new Mahasiswa("Andriani","123456");

        objMhs.alamat="Watugong 212 Malang";
        localNama=objMhs.nama;
        localNim=objMhs.nim;
    }
}
```

Saat membuat object objMhs yang bisa dianggap sebagai proses "membuat seorang mahasiswa", maka data nama dan nim tidak boleh diubah selama mahasiswa tersebut ada sehingga kita harus membuat data nama dan nim sebagai data *read only*. Jika data-data ini dibuat tidak readonly yang dapat memberikan peluang akses penuh, maka kemungkinan akan terdapat masalah tersendiri. Dengan demikian kita perlu melindungi data nama dan nim dari perubahan-perubahan dari luar class, sehingga perlu diberikan modifier **private.** Kita akan buktikan pada kasus-kasus berikutnya.

Method getNama, getNim, dan getAlamat disebut sebagi accesor method karena method tersebut memberikan kembalian nilai data yang sederhana. Sedangkan setAlamat disebut sebagai mutator method karena digunakan untuk memberikan suatu nilai yang sederhana.

Langkah-langkah di atas selanjutnya dapat kita gunakan untuk menerapkan sifat encapsulation atau information hiding. Jadi pada dasarnya encapsulation dapat kita lakukan dengan :

- Mendeklarasikan data atau attribut dengan modifier **private**
- Memberikan accessor method (getter) untuk mengakses data
- Memberikan mutator method (setter) untuk memberikan nilai ke suatu data.

Panduan Mendesain Class

1. **Selalu gunakan modifier private untuk data atau attributh**
   Selain menerapkan sifat encapsulation, langkah ini akan memberikan keuntungan antara lain:
   - Desain yang lebih modular
   - Perubahan representasi class tidak terlalu berpengaruh pada class tersebut
   - Lebih mudah mendeteksi kesalahan
2. **Selalu buat nilai inisial (nilai awal) pada data**
   Inisialisasi data dapat dilakukan dengan menggunakan constructor atau dengan memberikan nilai default.
3. **Hindari penggunaan banyak tipe data pada sebuah class**
   Kumpulkan multiple tipe data menjadi sebuah kelas. Misalkan jika pada class Mahasiswa terdapat
   `private String jalan; private string kota; private string propinsi; private string kodePos;`
   dapat diganti dengan sebuah `class Alamat` yang didalamnya terdapat data-data di atas.
4. **Tidak semua data memerlukan setter dan getter**
   Untuk data yang hanya perlu di set satu kali, cukup manfaatkan constructor.
5. **Biasakan menggunakan standar penulisan yang sama pada setiap class**
   Manfaatkan penulisan indent untuk menjaga kerapian code sehingga mempermudah trace kesalahan.
6. **Buatlah class sesederhana mungkin, hindari membuat class yang kompleks**
   Ubah class Mahasiswa berikut

```
class Mahasiswa{
    private String nama;
    private String nim;
    private String alamat;
    private String[] namaMK;
    private String[] nilaiMK;

    public Mahasiswa(String nama, String nim) { ... }
    public String getAlamat() {...}
    public String getNama() { ... }
    public String getNim() { ... }
    public void setNama(String nama) {...}
    public String getNilaiMK(int i ){ ... }
    public String namaMK(int i) {...}
}
```

Menjadi dua buah kelas berikut ini (contoh ini mungkin masih belum sederhana, harus disesuaikan dengan model permasalahan yang ada):

```
class Mahasiswa{
    private String nama;
    private String nim;
    private String alamat;
    private MataKuliah[] mk;
```

```
    public Mahasiswa(String nama, String nim) { ... }
    public String getAlamat() {...}
    public String getNama() { ... }
    public String getNim() { ... }
    public void setNama(String nama) {...}
    public MataKuliah getMK(int i ){ ... }
}
    class MataKuliah{
        private nama;
        private int sks;
        private String nilai;
        private double nilaiAngka;
        .... constructor;
        .... setter;
        .... getter;
    }
```

7. **Gunakan nama class dan method sesuai fungsinya.**
   Ini akan mempermudah kita (dan tim) dalam menggunakan class dan trace error. Ada saran untuk penulisan class dan method:
   - Gunakan kata benda (Noun) untuk sebuah nama class : Mahasiswa, MataKuliah, JadwalKuliah
   - Gunakan kata kerja (Verb) untuk method : getNama, hitungIPK, setNilai.

Inherintance

Pada materi ini kita akan mengenal: constructor, override, overload, keyword extends, this, super.
Pada materi sebelumnya telah dijelaskan tentang konsep inheritance. Istilah-istilah yang terkait dengan inheritance:

- Superclass atau parentClass : Class yang mewariskan data dan methodnya kepada class lain.
- Subclass atau childClass : Class yang mewarisi data dan method dari Superclass

Sifat hierarki suatu inheritance pada class diagram adalah semakin keatas semakin generic.

Pada bahasa pemrograman Java, inheritance ditandai dengan keyword **extends** pada saat deklarasi **child class.**

Keyword *super* dan *this*

Keyword super digunakan untuk merefer ke class parent, sedangkan this digunakan untuk ke kelas itu sendiri.
Contoh:

super.getNama() ➔ mengakses method getNama() yang dimiliki oleh parent class
super.gajiPokok ➔ mengakses data publik gajiPokok yang dimiliki oleh parent class
this.getNama() ➔ mengakses method getNama() yang dimiliki kelas ini sendiri.
this.gajiPokok ➔ mengakses data member (public ataupun privat) yang dimiliki kelas.

Jika terdapat suatu method sbb:

```
void gantiNilai(String nilai){
      this.nilai=nilai;
}
```

Perhatikan pada `this.nilai=nilai;`

Variable **nilai** di kanan tanda "=" adalah variabel lokal yang dimiliki oleh method gantiNilai() sedangkan variable **nilai** pada **this.nilai** mengacu pada data member yang dimiliki oleh suatu class.

Pada saat melakukan inheritance, semua method dan data member yang tidak privat akan diwariskan kepada child class. Akan tetapi hal tersebut **tidak berlaku pada method constructor.** Method constructor tidak dapat diwariskan, sehingga child class harus membuat constructor sendiri. Lalu bagaimana menjaga agar contstructor dapat identik dengan constructor pada parent class? Caranya adalah dengan memberikan akses ke constructor parent class pada constructor child class dengan menggunakan method **super()**

super() ➔ mengakses constructor parent class

Overload dan override

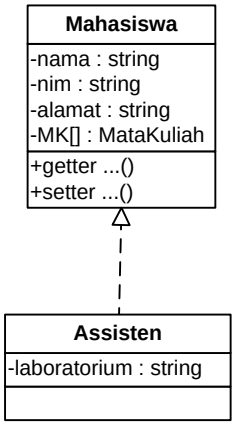
Overload adalah membuat lebih dari satu method dengan nama sama pada suatu class (Lihat contoh pada materi sebelumnya)

Override adalah proses implementasi yang memodifikasi method yang lebih spesifik pada child class, method hasil modifikasi tetap menggunakan nama dan parameter input yang sama seperti yang digunakan oleh parent class.

Pada kasus inheritance child class sering memerlukan method yang lebih spesifik dibanding method dengan nama sama pada parent class.

Contoh:

Perhatikan suatu contoh **inheritance** dan penggunaan **super** berikut:
Pada suatu perkuliahan, Assisten laboratorium adalah seorang Mahasiswa. Maka semua sifat mahasiswa akan diwarisi oleh asisten. Akan tetapi Assisten akan mempunyai sifat yang lebih khusus yaitu asisten memiliki laboratorium (sebagai tempat memberi asistensi).



Implementasi dengan java

Class Mahasiswa sebagai parent class sudah dibuat pada contoh sebelumnya

```
//Class assisten mewarisi sifat Mahasiswa
class Assisten extends Mahasiswa{
    private String laboratorium; //Attribut khusus yang dimiliki Assisten

//Cosntructor
    public Assisten(){
        super(); // super(), memanggil contrucor milik parent class
(Mahasiswa).
      String nama=”Tidak ada nama”;
      String nim=”Tida ada nim”;
      super(nama,nim);
    }
//Contructor dua input
    public Assisten(String nama, String nim) {
        super(nama, nim);
    }
//constructor 3 input, dengan input ketiga dikhususkan untuk data milik Child
    public Assisten(String nama, String nim, String laboratorium) {
        super(nama, nim);
        this.laboratorium = laboratorium;
    }
}
```

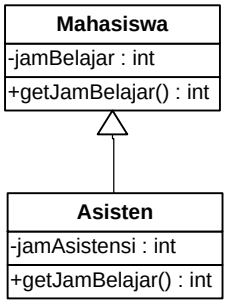
Contoh lain ?

Jumlah waktu maksimum perminggu yang diperlukan seorang mahasiswa dan asisten berbeda. Mahasiswa biasa hanya perlu kira-kira 40 jam hanya terkait dengan tugas kuliahnya. Asisten adalah seoarang mahasiswa, oleh karena itu asisten juga memiliki 40 jam per minggu terkait kuliahnya. Akan tetapi asisten mempunyai tanggungan tambahan terkait jam belajar untuk asistensi, asisten akan memerlukan waktu 15 jam lebih banyak dari mahasiswa biasa sebagai waktu asistensi. Sehingga total jam terkait tugas kuliah seorang asisten adalah jam belajar mahasiswa ditambah jam belajar laboratorium. Masing-masing akan memberitahukan jam belajar.

Bagaimana mewujudkannya dalam OOP?

Obyek: Mahasiswa, Asisten
Proses:
Mahasiswa: Menghitung jam belajar.
Asisten: Menghitung jam belajar
Relasi antar object/ class:
Mahasiswa memiliki jam belajar
Asisten memiliki jam Asistensi
Asisten adalah Mahasiswa.

Class diagram (hanya digambarkan bagian penting)




```
class Mahasiswa{
    private String nama;
    private String nim;
    private int jamBelajar=40;

    Mahasiswa(){

    }

    public Mahasiswa(String nama, String nim) {
        this.nama = nama;
        this.nim = nim;
    }

    public int getJamBelajar() {
        return jamBelajar;
    }

    public String getNama() {
        return nama;
    }

    public String getNim() {
        return nim;
    }

}
```

```
class Assisten extends Mahasiswa{
    private int JamAsistensi=15;

    public Assisten(String nama, String nim) {
```

```
        super(nama, nim);
    }

    public Assisten() {
    }

    @Override
    public int getJamBelajar() {  //mengubah method getJamBelajar
        return super.getJamBelajar()+this.JamAsistensi;
    }
}
```

Mencegah Inheritance Dari Sebuah Class (final Class)

Ada kalanya diperlukan suatu class yang tidak boleh diwariskan lagi menjadi class baru. Class yang seperti ini Jika misalnya seorang eksekutif manajer merupakan ujung dari organisasi maka di organisasi tersebut sudah tidak ada lagi variasi atau kekhasan suatu jabatan yang merupakan turunan dari eksekutif manajer. Dengan demikian perlu dilakukan pencegahan pewarisan dari class eksekutif manajer.

Untuk mencegah suatu kelas diwariskan digunakan keyword **final** pada saat deklarasi kelas.

Contoh:

```
final class EksekutifManajer extend Manajer {
      . . . .
}
```

Contoh lagi:

```
final class ClassTerakhir {
      . . . .
}
```

Mencegah Override oleh Child Class (final method)

Keyword **final** juga dapat diterapkan pada method. Jika keyword **final** diterapkan pada method maka method tersebut tidak dapat di*override* oleh child class. Child class masih mewarisi method tersebut hanya saja tidak dapat melakukan override.

Contoh deklarasi pada parent class

```
public final String getNama(){
      . . . .
}
```

Mencegah Pembuatan Object dari Sebuah Class (Abstract Class)

Pada suatu hierarki inheritance, grand parent tertinggi selalu mempunyai sifat yang lebih umum dibandingkan dengan sub class nya. Misalnya, class Civitas adalah (dapat dianggap) class tertinggi pada model struktur kampus. Yang termasuk civitas antara lain Mahasiswa, Dosen, Karyawan, StakeHolder. Jika dinyatakan bahwa setiap orang yang merupakan bagian dari civitas harus berada pada kelompok tersebut, maka tidak boleh ada orang yang hanya menyebut diri sebagai civitas tetapi tidak bisa menyebutkan kelompoknya. Disini terlihat bahwa, tidak boleh membuat object dari class Civitas.

Class yang dicegah untuk diwujudkan menjadi sebuah object disebut sebagai abstract Class. Untuk membuatnya, cukup diberikan keyword **abstract** pada saat deklarasi class.
Contoh:

```
abstract class Civitas{
      private String nama;
      private String jenisKelamin;

      public getNama(){
            return nama
      }
}
```

Java akan menganggap error jika kita instance class tersebut, misal:

```
      Civitas myCiv;
      myCiv= new Civitas();   //error
```

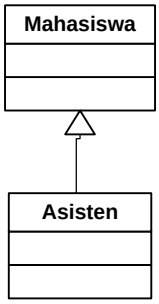
Q: Kapan memerlukan inheritance?
A: Inherintance diperlukan ketika kita ingin membuat class baru yang merupakan jenis dari class yang sudah ada. Jika dibuat kalimatnya akan berbunyi "Class baru adalah class lama" "Mahasiswa adalah civitas" "Asisten termasuk Mahasiswa"

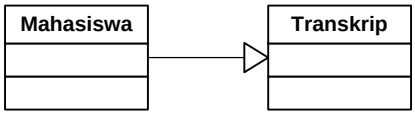
Hubungan antar Class

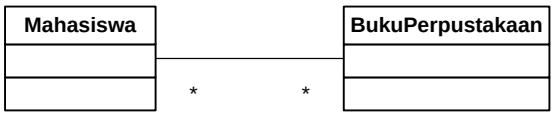
Secara umum relasi antar class adalah

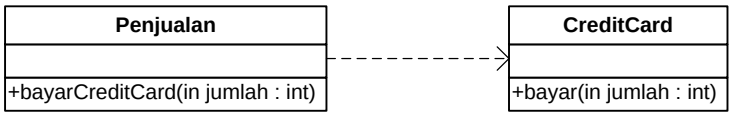
- Inheritance: is-a (adalah)
- Dependensi : uses-a (menggunakan)
- Agregasi, Assosiasi : has-a , part-of (mempunyai)

Model	Kalimat	Java

	Asisten adalah mahasiswa
	
	Jenis: Inheritance
		class Mahasiswa{
		
		}
		
		class Asisten extend Mahasiswa(){
		
		}

	Mahasiswa mempunyai transkrip.
	
	Jenis: Asosisasi satu arah
		class Transkrip {
		
		}
		
		class Mahasiswa{
		private Transkrip myTranskrip;
		}

	Mahasiswa memiliki pinjaman buku perpustakaan, buku perpustakaan mempunyai mahasiswa yang meminjam buku.
	Jenis: Asosiasi dua arah
		class BukuPerpus {
		private Mahasiswa myMhs;
		}
		
		class Mahasiswa{
		private BukuPerpus myBuku;
		}

	Penjualan memerlukan proses pembayaran yang dilakukan oleh CreditCard
	Jenis: Dependesi
		class Penjualan{
		public void bayarCC(int Jumlah){
		CreditCard.bayar(Jumlah);
		}
		}
		
		class CreditCard{
		public void bayar(int Jumlah) {
		
		}
		}

TUGAS:

Pada sebuah organisasi terdapat beberapa jenis pegawai pada suatu perusahaan yaitu Pegawai biasa, Manajer, Eksekutif Manajer, Sekretaris, dan Programmer. Seorang eksekutif manajer adalah seorang manajer yang memilki sekretaris. Semua mempunyai base salary (gaji pokok) yang sama. Akan tetapi masing-masing mempunyai perhitungan gaji yang berbeda. Masing-masing dalam organisasi tersebut harus dapat menginformasikan nama, jabatan, dan total gaji.

- Pegawai biasa hanya dibayar dengan gaji pokok
- Manajer mendapatkan dua kali gaji pokok plus bonus
- Eksekutif manajer mendapatkan satu setengah kali total gaji Manajer
- Sekretaris mendapatkan gaji pokok plus upah lembur.

Tugas:

- Definisikan obyek-obyek (kata benda) yang terdapat dalam permasalahan
- Definiskan proses/tindakan (kata kerja) yang terdapat dalam permasalahan dan tetapkan (apa atau siapa/ object) pelakunya
- Definiskan relasi antar classnya (menggunakan kalimat)
- Gambarkan class diagramnya
- Implementasikan dalam Java
- Simulasikan untuk menampilkan nama, jabatan, dan total gaji masing-masing pegawai.