

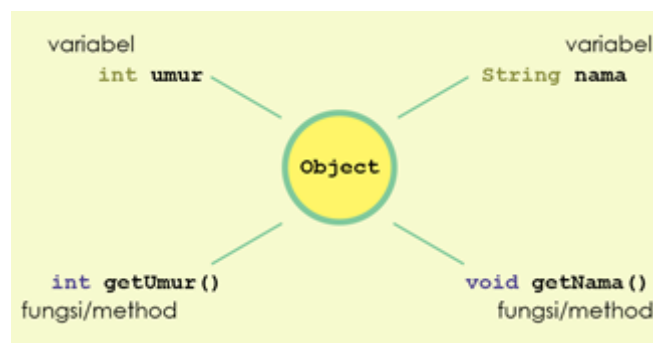
# PEMROGRAMAN BERBASIS OBJEK

## OBJEKTIF :

1. Mahasiswa Mampu Memahami Karakteristik dari Pemrograman Berbasis Objek.
2. Mahasiswa Mampu Menggunakan *Software* IntelliJ IDEA dalam Pembuatan Program yang memiliki Karakteristik Pemrograman Berbasis Objek dengan Bahasa Pemrograman Java.

## PENDAHULUAN

Pemrograman Berbasis Objek (PBO) atau biasa dikenal dengan *Object Oriented Programming*(OOP) merupakan sebuah teknik pemrograman yang berorientasikan objek. OOP merupakan teknik pemrograman modern yang efisien dan paling banyak digunakan. Semua data maupun fungsi dibungkus dalam beberapa objek atau *class* yang dapat saling berinteraksi, sehingga terbentuk sebuah program dengan tujuan untuk memecahkan suatu masalah.



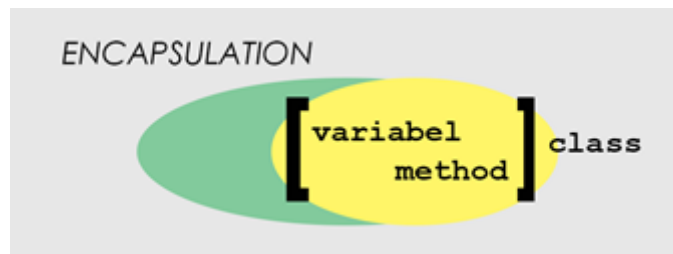
Terdapat beberapa karakteristik dan konsep dalam pemrograman berbasis objek, yaitu:

1. **Encapsulation** (Pengapsulan)
2. **Polymorphism**
3. **Inheritance** (Pewarisan)

## 9.1 ENCAPSULATION

*Encapsulation* atau dalam bahasa Indonesia nya adalah pengapsulan, merupakan suatu metode pada PBO dalam menyembunyikan atau memproteksi program dan data yang sedang diolah. Metode tersebut digunakan untuk memproteksi program dari kemungkinan penyalahgunaan dari luar sistem dan berfungsi juga sebagai penyederhaan implementasi sistem. Rincian-rincian implementasi dari internal akan disembunyikan. Pada encapsulation, paket data akan diproses bersama dengan metode-metodenya.

Bagian luar sistem yang dimaksud adalah antarmuka objek, yang berlaku sebagai antarmuka objek terhadap objek lain. Dengan encapsulation, sistem internal tidak dapat diakses oleh sistem luar dan perubahan yang terjadi pada internal, tidak akan mempengaruhi bagian-bagian program lain.



Pada gambar di atas, variabel dan method yang terdapat dalam class merupakan encapsulation yang membungkus class dan menjaga isi yang terdapat dalam class tersebut.

Pengapsulan menyediakan dua manfaat utama bagi pemrogram, yaitu:

1. Penyembunyian informasi

Penyembunyian implementasi (*implementation hiding*) merupakan perlindungan implementasi internal objek. Objek disusun dari antarmuka *public* dan bagian *private* merupakan bagian dari data internal.

2. Modularitas

Modularitas (*modularity*) merupakan objek yang dapat dikelola secara independen. Kode sumber bagian internal objek dikelola secara terpisah dari antarmuka, maka modifikasi dapat dilakukan secara bebas tanpa menyebabkan masalah pada bagian-bagian lain dari sistem.

Pengapsulan pada Java dilakukan dengan pembentukan kelas-kelas, yaitu *keyword* 'class'. Sedangkan penyembunyian informasi dilakukan dengan memberikan hak akses pembentuk kelas atau yang biasa disebut dengan *modifier*, berikut merupakan *keyword* untuk hak akses kelas:

- Default

*Modifier* default, berarti kelas tersebut dapat diakses oleh semua kelas. Tetapi harus berada di dalam folder/package yang sama. Untuk mendeklarasikan *modifier* default, tidak perlu menuliskan apapun (kosong).

Contoh:

```
class kursus
int nilai;
void getNilai()
```

`class kursus` merupakan contoh kelas dengan *modifier* default. `int nilai` merupakan contoh atribut nilai dengan *modifier* default. `void getNilai()` merupakan contoh method dengan *modifier* default.

- Private

*Modifier* private memberikan hak akses hanya pada kelas tersebut. Data yang terdapat pada kelas tersebut, hanya dapat diakses oleh kelas itu saja dan kelas lain tidak dapat mengaksesnya.

Contoh:

```
private class kursus
private int nilai;
private void getNilai()
```

`class kursus` merupakan contoh kelas dengan *modifier* private. `int nilai` merupakan contoh atribut nilai dengan *modifier* private. `void getNilai()` merupakan contoh method dengan *modifier* private.

- Protected

*Modifier* protected memberikan hak akses kepada kelas itu sendiri dan subkelas hasil turunannya (inheritance). Data yang terdapat disebuah kelas tersebut, dapat diakses oleh kelas itu sendiri dan kelas lain yang melakukan 'extends' terhadap kelas tersebut. Kelas yang berada di luar package atau folder harus melakukan 'extends' untuk dapat mengaksesnya.

Contoh:

```
protected class kursus
protected int nilai;
protected void getNilai()
```

`class kursus` merupakan contoh kelas dengan *modifier* protected. `int nilai` merupakan contoh atribut nilai dengan *modifier* protected. `void getNilai()` merupakan contoh method dengan *modifier* protected.

- Public

*Modifier* public akan memberikan hak akses yang dapat diakses oleh kelas lain atau property manapun diluar kelas yang bersangkutan.

Contoh:

```
public class kursus
public int nilai;
public void getNilai()
```

`class kursus` merupakan contoh kelas dengan *modifier* public. `int nilai` merupakan contoh atribut nilai dengan *modifier* public. `void getNilai()` merupakan contoh method dengan *modifier* public.

Modifier	Class	Package	Subclass	World
Default	✓	✓		
Private	✓			
Protected	✓	✓	✓	
Public	✓	✓	✓	✓

Berikut ini merupakan contoh implementasi `encapsulation` pada program:

Membuat `class Nilai` terlebih dahulu untuk menyimpan variabel dan method yang akan diakses oleh class utama.

```
package com.integratedlaboratory.program;

public class Nilai {
    private int nilaiSaya;

    public String nama;
    public String NPM;
```

```

    public void setNilai(int nilai){
        nilaiSaya=nilai;
        if (nilai >= 100){
            System.out.println("Nilai tidak boleh lebih dari 100");
        }
    }

    public int getNilai(){
        return nilaiSaya;
    }
}

```

Setelah itu membuat class utama yaitu `TestNilai`.

```

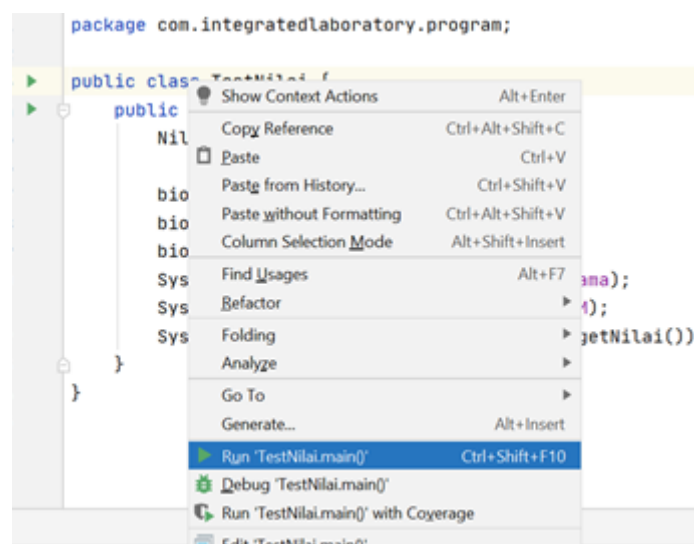
package com.integratedlaboratory.program;

public class TestNilai {
    public static void main(String[] args) {
        Nilai biodata = new Nilai();

        biodata.nama = "Putri";
        biodata.NPM = "123456";
        biodata.setNilai(90);
        System.out.println("Nama: " + biodata.nama);
        System.out.println("NPM: " + biodata.NPM);
        System.out.println("Nilai: " + biodata.getNilai());
    }
}

```

Tekan tombol Ctrl+Shift+F10 untuk melakukan Run pada IntelliJ IDEA atau dengan melakukan klik kanan pada file java seperti berikut:



Hasil program:

```
Run: TestNilai x
D:\Downloads\Compressed\jdk-14.0.2\
Nama: Putri
NPM: 123456
Nilai: 90

Process finished with exit code 0
```

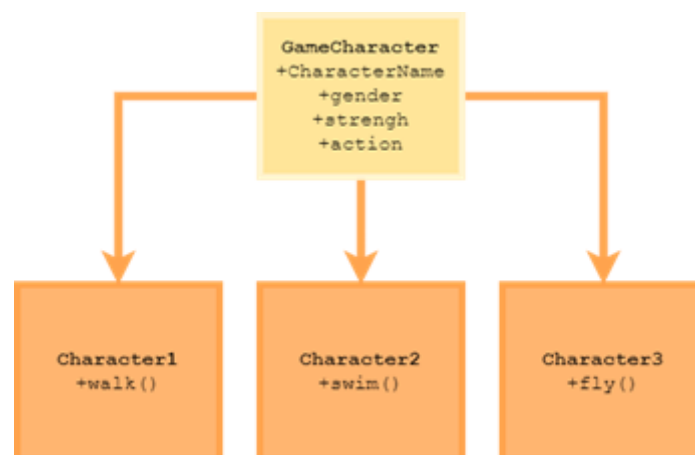
Pada contoh di atas, variabel dibungkus ke dalam sebuah file bernama Nilai.java dan variabel atau method yang terdapat pada kelas ini hanya dapat diakses oleh file utama yaitu TestNilai.java. Pada contoh di atas, dalam Nilai.java digunakan 'getter' untuk mengambil nilai dari atribut. Method dalam Nilai.java menggunakan modifier *public* agar dapat diakses oleh kelas utama.

## 9.2 INHERITANCE

*Inheritance* atau turunan (pewarisan) merupakan salah satu konsep penting dalam PBO. Sebuah class pada Java, dapat memiliki satu atau lebih keturunan atau class anak. Inheritance adalah proses penciptaan kelas baru dengan mewarisi kelas yang sudah ada, ditambah dengan karakteristik unik pada kelas baru. Kelas baru akan memiliki property dan method dari kelas yang sudah ada.

Inheritance dapat dilakukan untuk menciptakan kelas umum yang mendefinisikan perilaku umum dari kelas-kelas yang lain. Pada kelas lain tersebut, dideklarasikan lebih spesifik hal-hal mengenai karakteristik unik dari kelas lain tersebut.

Pada Java, kelas yang akan mewarisi properti nya disebut dengan **superclass**, sedangkan kelas yang diwarisi disebut **subclass**.



Pada contoh di atas, class `GameCharacter` adalah kelas parent atau induk. Class `Character1`, `Character2`, dan `Character3` merupakan kelas anak yang memiliki properti sama dengan kelas induk, namun ditambahkan dengan keunikan masing-masing karakter.

Untuk mengimplementasikan kelas induk pada kelas lain, maka digunakan kata kunci `extends` yang dideklarasikan pada kelas turunannya.

```
public class nama-subclass extends nama-superclass {
    //isi kelas
}
```

Berikut ini adalah contoh implementasi *inheritance* dalam sebuah program yang memiliki *parent class* Bicycle.java dan mewariskan sifat yang terdapat dalam class tersebut ke *subclass* RoadBike.java dan MountainBike.java dengan menggunakan kata kunci `extends`.

```
package com.integratedlaboratory.program;

class Bicycle {
    public Bicycle(int kecepatan, int gigi) {
        this.kecepatan = kecepatan;
        this.gigi = gigi;
    }
    int kecepatan = 0;
    int gigi = 1;

    public void printDescription(){
        System.out.println("\nSepeda ini " + "pada gigi " + this.kecepatan +
            " dan berjalan dengan kecepatan " + this.kecepatan + ". ");
    }
}
```

```
package com.integratedlaboratory.program;

public class RoadBike extends Bicycle{
    private int lebarRoda;

    public RoadBike(int mulaiKecepatan,
        int mulaiGigi,
        int lebarRodaBaru){
        super(mulaiKecepatan,
            mulaiGigi);
        this.setLebarRoda(lebarRodaBaru);
    }

    public int getLebarRoda(){
        return this.lebarRoda;
    }

    public void setLebarRoda(int lebarRodaBaru){
        this.lebarRoda = lebarRodaBaru;
    }

    public void printDescription(){
        super.printDescription();
        System.out.println("RoadBike ini" + " memiliki " + getLebarRoda() +
            " MM roda.");
    }
}
```

```
package com.integratedlaboratory.program;

public class MountainBike extends Bicycle{
```

```

private String suspension;

public MountainBike(
    int mulaiKecepatan,
    int mulaiGigi,
    String tipeSuspension){
    super(mulaiKecepatan,
        mulaiGigi);
    this.setSuspension(tipeSuspension);
}

public String getSuspension(){
    return this.suspension;
}

public void setSuspension(String suspensionType) {
    this.suspension = suspensionType;
}

public void printDescription() {
    super.printDescription();
    System.out.println("MountainBike ini memiliki" +
        getSuspension() + " suspension.");
}
}

```

```

package com.integratedlaboratory.program;

public class TestBike {
    public static void main(String[] args){
        Bicycle bike01, bike02, bike03;

        bike01 = new Bicycle(20, 1);
        bike02 = new MountainBike(20, 2,"Dual");
        bike03 = new RoadBike(40, 3, 23);

        bike01.printDescription();
        bike02.printDescription();
        bike03.printDescription();
    }
}

```

Jalankan TestBike.java dengan menekan Ctrl+Shift+F10 atau dengan menekan tombol run. Hasil program sebagai berikut:

```
Run: TestBike x
D:\Downloads\Compressed\jdk-14.0.2\bin\java.exe "-javaagen

Sepeda ini pada gigi 20 dan berjalan dengan kecepatan 20.

Sepeda ini pada gigi 20 dan berjalan dengan kecepatan 20.
MountainBike ini memilikiDual suspension.

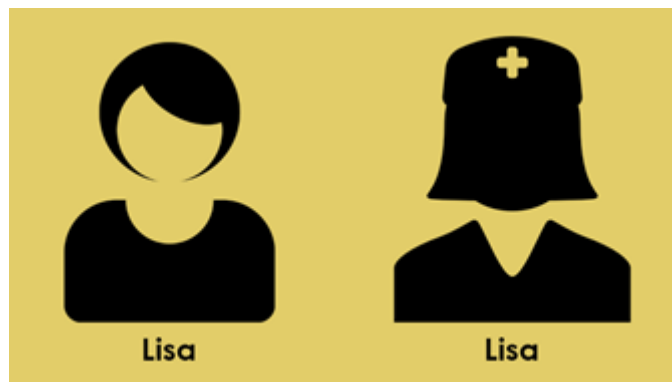
Sepeda ini pada gigi 40 dan berjalan dengan kecepatan 40.
RoadBike ini memiliki 23 MM roda.

Process finished with exit code 0
```

### 9.3 POLYMORPHISM

*Polymorphism* dalam PBO, merupakan prinsip di mana sebuah class dapat memiliki banyak bentuk method yang berbeda-beda, dengan nama yang sama. Polymorphism merupakan konsep pokok dari pemrograman berbasis objek. Polymorphism merupakan pembeda antara pemrograman berbasis objek dan bahasa tradisional yang hanya sampai memilik tipe data abstrak. Polymorphism ada karena interaksi antara konsep pewarisan dan *dynamic binding* (overriding).

Dalam *polymorphism*, sebuah method dapat memiliki nama yang sama dengan isi yang berbeda, parameter berbeda, dan tipe data yang berbeda.



Gambar di atas merupakan contoh dari penggambaran *polymorphism*. Pada gambar di atas, terdapat dua orang yang memiliki nama yang sama. Namun terlihat pada gambar bahwa dua orang tersebut memiliki atribut dan sifat yang berbeda. Pekerjaan dari dua orang di atas pun berbeda.

Pada Java, terdapat dua macam polymorphism dengan perbedaan cara pembuatannya, yaitu:

#### 1. Static Polymorphism

Pada polymorphism statis, menggunakan method overloading yang terjadi pada sebuah class dengan nama method yang sama tetapi memiliki parameter dan tipe data yang berbeda. Contoh:



```

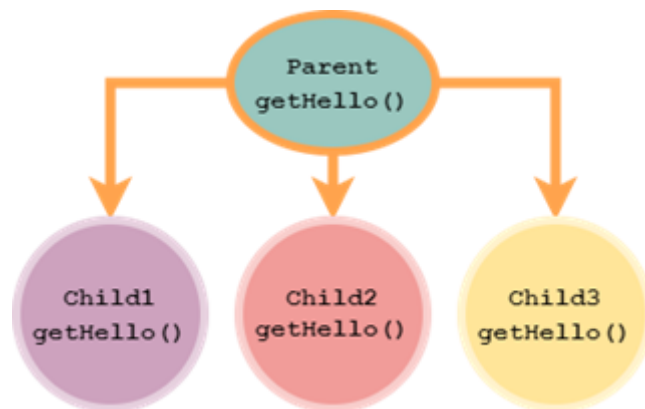
class Lingkaran {
    float luas(float r){
        return (float) (Math.PI * r * r);
    }
    double luas(double d){
        return (double) (1/4 * Math.PI * d);
    }
}

```

Pada kode di atas, terdapat dua method dengan nama yang sama yaitu method luas. Namun kedua method tersebut memiliki parameter dan tipe data yang berbeda. Rumus yang terdapat di dalamnya pun berbeda.

## 2. Dynamic Polymorphism

Polymorphism dinamis, menggunakan method overriding yang terjadi saat menggunakan *inheritance* (pewarisan) dan implementasi *interface*. *Interface* adalah class kosong yang berisi nama-nama method yang harus diimplementasikan pada class lain. Dalam pengimplementasiannya tiap-tiap class akan mengimplementasikan secara berbeda dengan nama method yang sama. Pada *inheritance*, atribut dan method dari class induk ke class anak dapat diwariskan dengan class anak memiliki nama method yang sama dengan class induk dan class anak yang lainnya.



Pada gambar di atas, class anak memiliki nama method yang sama. Namun, nanti nya parameter dalam class anak akan berbeda dari class induk karena class anak melakukan method 'overriding' dimana akan melakukan tindi dari method yang diwariskannya.

Dapat diambil kesimpulan bahwa pada polymorphism statis hanya terjadi dalam sebuah class saja, namun polymorphism dinamis akan terjadi pada hubungan dengan class lain seperti *inheritance*.

Berikut ini merupakan contoh program dengan polymorphism:

```

package com.integratedlaboratory.program;

public class MainBangunDatar {
    public static void main(String[] args) {

        BangunDatar bangunDatar = new BangunDatar();
        Persegi persegi = new Persegi(4);
        Segitiga segitiga = new Segitiga(6, 3);

        // memanggil method luas dan keliling
        bangunDatar.luas();
        bangunDatar.keliling();
    }
}

```

```

        System.out.println("Luas persegi: " + persegi.luas());
        System.out.println("keliling persegi: " + persegi.keliling());
        System.out.println("Luas segitiga: " + segitiga.luas());
    }
}

```

```

package com.integratedlaboratory.program;

public class BangunDatar {
    float luas(){
        System.out.println("Menghitung luas bangun datar");
        return 0;
    }

    float keliling(){
        System.out.println("Menghitung keliling bangun datar");
        return 0;
    }
}

```

```

package com.integratedlaboratory.program;

public class Persegi extends BangunDatar{
    int sisi;

    public Persegi(int sisi) {
        this.sisi = sisi;
    }

    @Override
    public float luas() {
        return this.sisi * this.sisi;
    }

    @Override
    public float keliling(){
        return this.sisi * 4;
    }
}

```

```

package com.integratedlaboratory.program;

public class Segitiga extends BangunDatar{
    int alas;
    int tinggi;

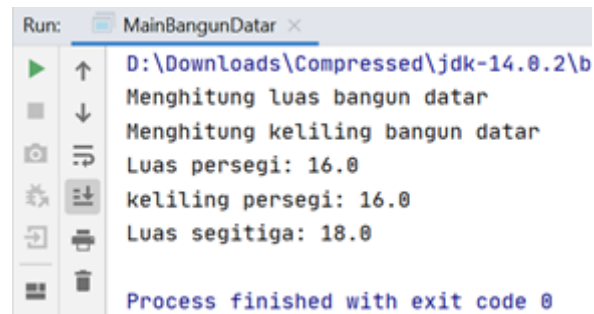
    public Segitiga(int alas, int tinggi) {
        this.alas = alas;
        this.tinggi = tinggi;
    }

    @Override
    public float luas(){
        return this.alas * this.tinggi;
    }
}

```

```
}  
}
```

Setelah membuat keempat class di atas, maka lakukan run pada MainBangunDatar.java dan hasil program sebagai berikut:



```
Run: MainBangunDatar x  
D:\Downloads\Compressed\jdk-14.0.2\bin  
Menghitung luas bangun datar  
Menghitung keliling bangun datar  
Luas persegi: 16.0  
keliling persegi: 16.0  
Luas segitiga: 18.0  
Process finished with exit code 0
```

Class yang dapat dijalankan hanya class MainBangunDatar.java yang memiliki **method main**.

**REFERENSI:**

[1] Hariyanto, Bambang. 2017. *Esensi-Esensi Bahasa Pemrograman Java Revisi Kelima*. Bandung: Informatika.

[2] Muhardian, Ahmad. 2017. "Belajar Java OOP: Memahami Inheritance dan Method Overriding", <https://www.petanikode.com/java-oop-inheritance/>. ( 23 Juli 2020)