Minggu 01
Hari 2: Pola Pikir Obyek, Tooling, Typing Discipline dan Penanganan Error

Oleh : Dedi Gunawan
Full Stack Enterprise Java - Praxis Academy

Konsep Dasar OOP

Pemrograman berorientasi objek (OOP) merupakan paradigma pemrograman berdasarkan konsep "**objek**", yang dapat berisi :

- **data**, dalam bentuk field atau dikenal juga sebagai **attribute**;
- **kode**, dalam bentuk fungsi/prosedur atau dikenal juga sebagai **method**.

Semua attribute dan method di dalam paradigma ini dibungkus dalam **kelas-kelas** atau **objek-objek**.

Java OOP – Class, Object, Attribute, Method

Class

```
class Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
         cadence = newValue;
    }

    void changeGear(int newValue) {
         gear = newValue;
    }

    void speedUp(int increment) {
         speed = speed + increment;
    }

    void applyBrakes(int decrement) {
         speed = speed - decrement;
    }

    void printStates() {
         System.out.println("cadence:" +
             cadence + " speed:" +
             speed + " gear:" + gear);
    }
}
```

Class

Attribute

Method

Class : suatu “Blueprint” atau “Cetakan” dari object.

Object : instance dari class., Sebuah instance adalah representasi nyata dari class itu sendiri.

Attribute : identitas atau informasi objek, atau biasa disebut variables

Method : tingkah laku atau apa yang dapat dilakukan oleh object

Java OOP – Object, Attribute, Method, Enkapsulasi

Pengimplementasian

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

        // Create two different
        // Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on
        // those objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike1.printStates();

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
        bike2.printStates();
    }
}
```

Object/
Class Instantiation

Cara mengakses method suatu object

Hasil Run program :

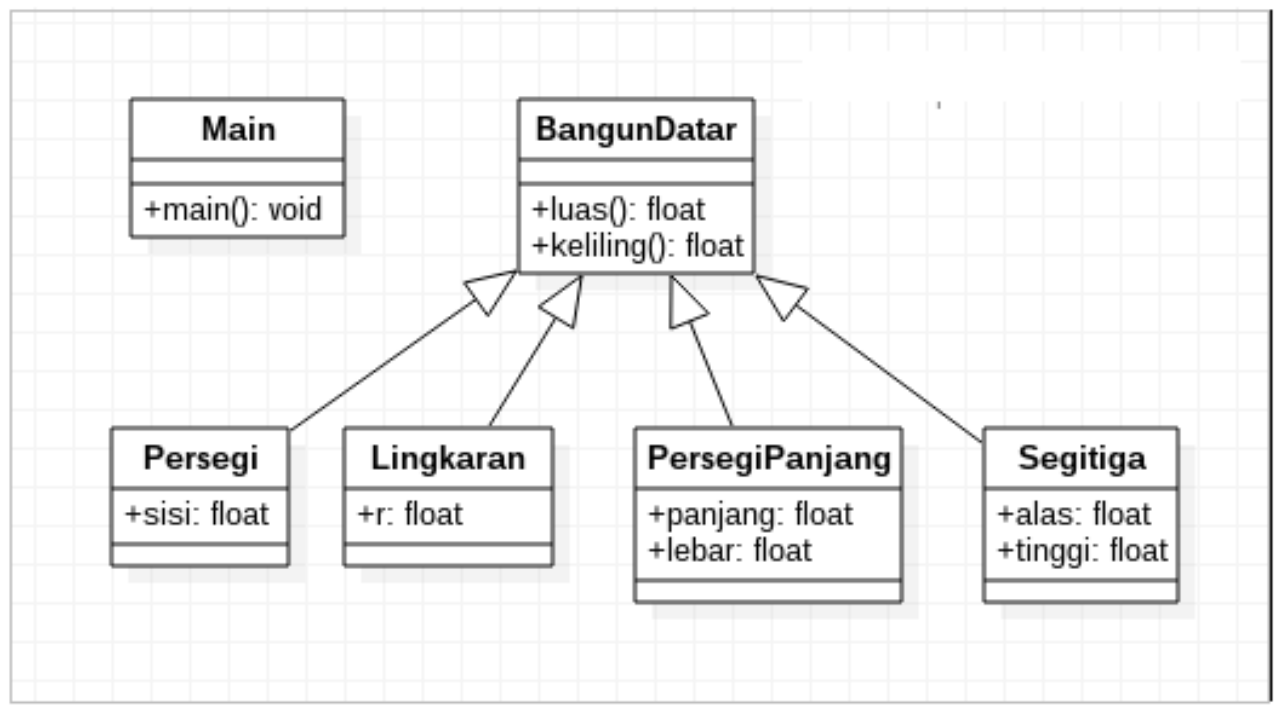
```
cadence:50 speed:10 gear:2
cadence:40 speed:20 gear:3
```

Sebaiknya attribute tidak diakses langsung dari program (implementasi dari **Enkapsulasi**), tetapi melalui method.
Method untuk mengakses attribute :
- **Set** method : untuk merubah nilai attribute
- **Get** method : untuk mendapatkan nilai attribute

Constructor method : method yang pertama kali dijalankan pada saat object dibuat. Biasanya digunakan untuk menginisialisasi nilai dari attribute.

Enkapsulasi : public, protected, private

Java OOP – Inheritance, Override



Persegi :
luas = sisi X sisi
keliling = (4 X sisi)

Lingkaran :
luas = 3.14 X jari
keliling = (2 X 3.14 X jari)

Persegi Panjang :
luas = panjang X lebar
keliling = (2 X (panjang + lebar))

Segitiga :
luas = sisi X sisi

Meskipun untuk menghitung luas dan keliling tiap bangun datar, diambil dari variable / attribute yg berlainan, tapi cukup melakukan **inheritance,** untuk menyediakan attribute tsb**.**

Karena tiap bangun datar memiliki rumus luas dan keliling yang berlainan, tidak perlu membuat method yang baru, tapi tinggal meng-**override** dari parent class.

Implementasi Inheritance dan Overriding pada Java

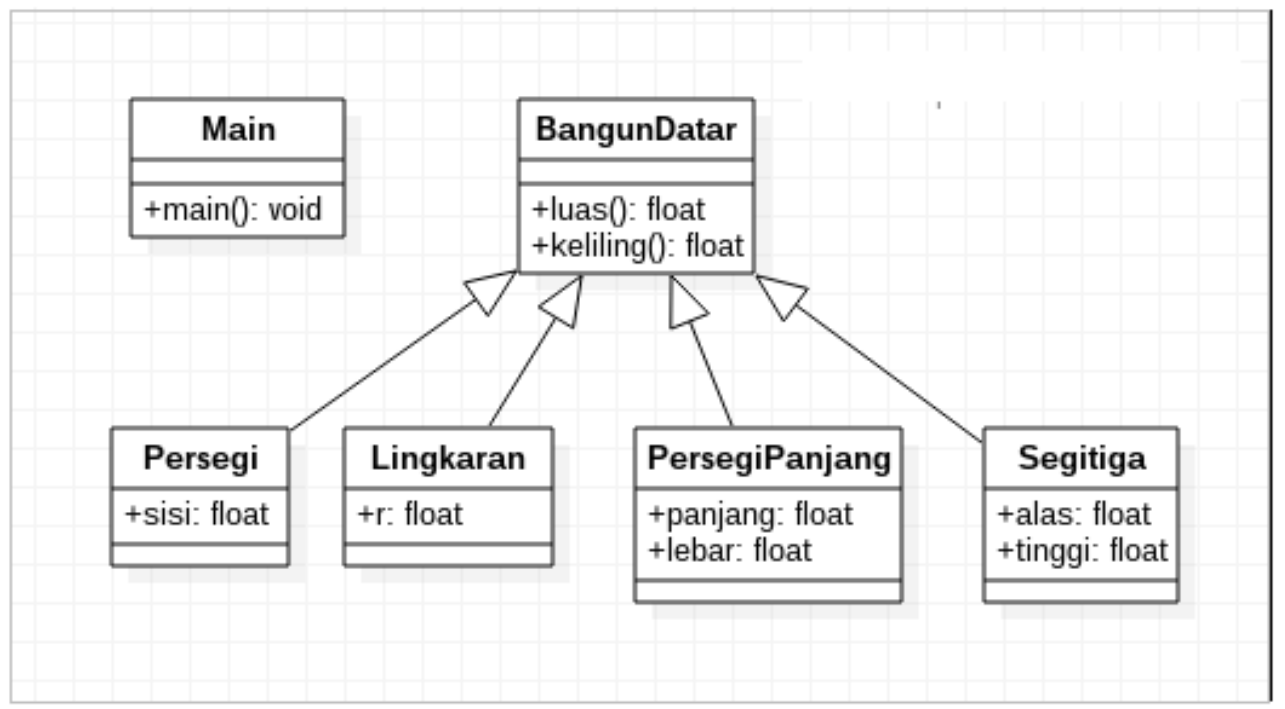
Child Class

```
// Ini adalah implementasi inheritance
public class PersegiPanjang extends BangunDatar
{
    float panjang;
    float lebar;

    // Ini adalah implementasi override
    @Override
    float luas(){
        float luas = panjang * lebar;
        System.out.println
        ("Luas Persegi Panjang:" + luas);
        return luas;
    }

    // Ini adalah implementasi override
    @Override
    float keliling(){
        float kll = 2*panjang + 2*lebar;
        System.out.println
        ("Keliling Persegi Panjang: " + kll);
        return kll;
    }
}
```

Parent Class

```
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;
    }
}
```

Java OOP – Polymorphism



Persegi :
luas = sisi X sisi
keliling = (4 X sisi)

Lingkaran :
luas = 3.14 X jari
keliling = (2 X 3.14 X jari)

Persegi Panjang :
luas = panjang X lebar
keliling = (2 X (panjang + lebar))

Segitiga :
luas = sisi X sisi

Polymorphism adalah perubahan bentuk, dimana method (pada parent class, yaitu method luas dan/atau keliling) yang sama diimplementasikan secara berlainan pada child-child class-nya. Dengan hal ini, dapat menghilangkan kerumitan.

Parent class (class BangunDatar) harus merupakan **abstract** class.

Implementasi Polymorphism pada Java

Parent Class

```
abstract class BangunDatar {
    abstract public float luas();
    abstract public float keliling();
}
```

Child Class 1

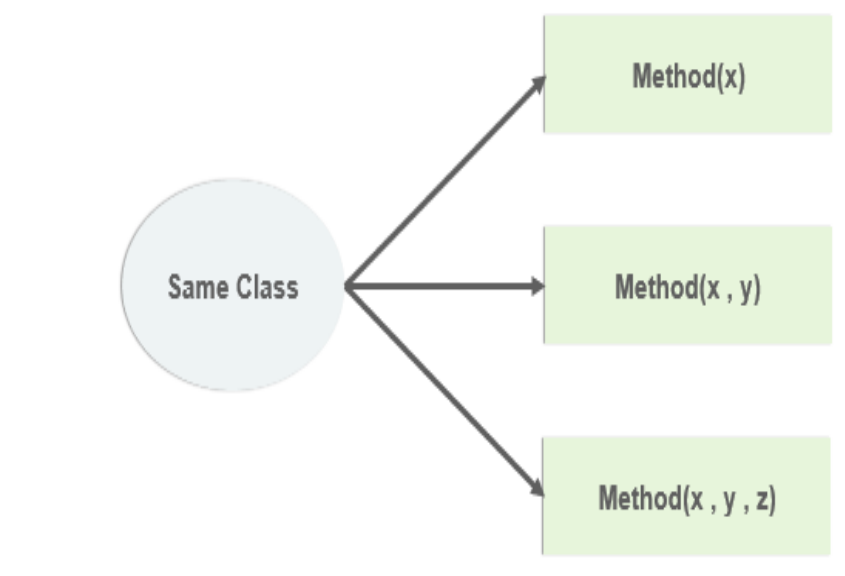
```
public class PersegiPanjang extends BangunDatar{
   float panjang;
   float lebar;

   @Override
   public float luas(){
      float luas = panjang * lebar;
      System.out.println("Luas Persegi Panjang:" + luas);
      return luas;
   }

   @Override
   public float keliling(){
      float kll = 2*panjang + 2*lebar;
      System.out.println("Keliling Persegi Panjang: " + kll);
      return kll;
   }
}
```

Diimplementasi berbeda, Polymorphism

Child Class 2

```
public class Lingkaran extends BangunDatar{

   float r;

   @Override
   public float luas(){
      float luas = (float) (Math.PI * r * r);
      System.out.println("Luas lingkaran: " + luas);
      return luas;
   }

   @Override
   public float keliling(){
      float keliling = (float) (2 * Math.PI * r);
      System.out.println("Keliling Lingkaran: " + keliling);
      return keliling;
   }

}
```

Java OOP - Overloading



Overloading adalah sebuah kemampuan yang membolehkan sebuah class mempunyai 2 atau lebih method dengan nama yang sama, yang membedakan adalah parameternya.

```
class DisplayOverloading
{
   public void disp(char c)
   {
       System.out.println(c);
   }
   public void disp(char c, int num)
   {
       System.out.println(c + " "+num);
   }
}
class Sample
{
  public static void main(String args[])
  {
      DisplayOverloading obj = new DisplayOverloading();
      obj.disp('a');
      obj.disp('a',10);
  }
}
```

Overloading

Output:

```
a
a 10
```

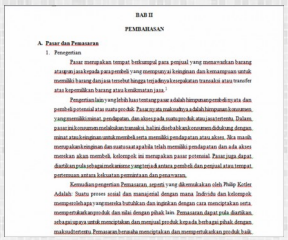
Java OOP - Abstraction

Class **Abstract** : class dimana sebagian method nya diimplementasi.
Interface : class Abstract dimana **semua** method nya masih belum diimplementasi.
Attribute pada Interface adalah semua harus berupa konstanta.

```
// contoh implementasi dari interface
class ACMEBicycle implements Bicycle {

   int speed = 0;
   int gear = 1;

  // baru diimplementasikannya disini
   void changeGear(int newValue) {
        gear = newValue;
   }

   void speedUp(int increment) {
        speed = speed + increment;
   }

   void printStates() {
        System.out.println("cadence:" +
            cadence + " speed:" +
            speed + " gear:" + gear);
   }
}
```

Java syntax

```
interface Bicycle {

    //  wheel revolutions per minute
    void changeGear(int newValue);

    void speedUp(int increment);

}
```



Java Packages

```
package backend.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import org.hibernate.annotations.GenericGenerator;

public class AccessLevel {

    private Integer id;
    private String access_level;

    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getAccess_level() {
        return access_level;
    }
    public void setAccess_level(String access_level) {
        this.access_level = access_level;
    }
}
```

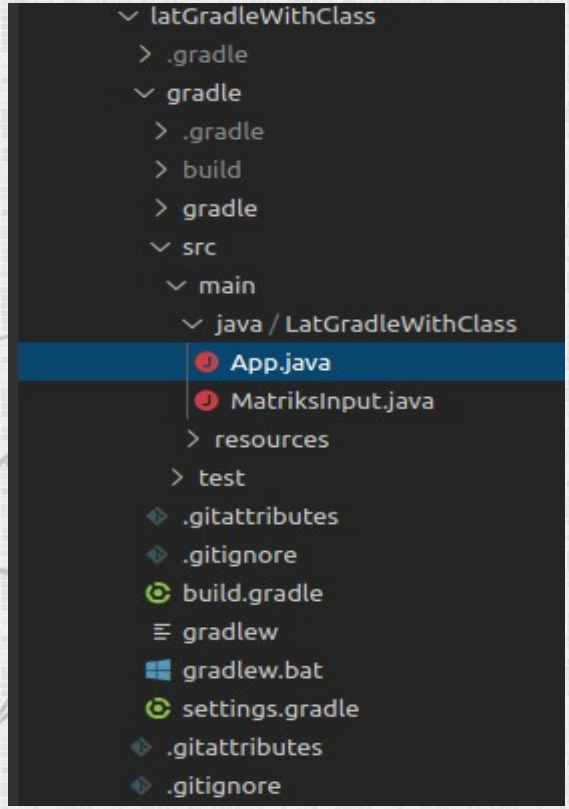
Cara create package

Cara menggunakan class yang ada dalam package

Packages : suatu unit tunggal untuk mengelompokkan kelas-kelas dan interfaces

Java Build Tool - Gradle

Build tool : suatu tool untuk melakukan build, terutama yaitu aktivitas melakukan *compile* dan *packaging.*
Pada tutorial ini, akan menggunakan Gradle sebagai Build Tool



Cara install gradle melalui apt :
sudo apt install gradle

Cara cek apakah gradle sudah terinstall :
gradle -version

Cara membuat project java dengan Gradle :
mkdir nama_folder
cd nama_folder
gradle init

1. Jenis Proyek, pilih `2: Application`
2. Bahasa, pilih `3: Java`
3. DSL pilih `1: Groovy`
4. Test Framework pilih `1: JUnit 4`

Maka Gradle akan membuatkan struktur projects

Cara build aplikasi
gradlew build

Cara run aplikasi
gradlew run

Dynamic – Static Typing

Perbedaan dynamic dan static typing :
1. **Dynamic** typing akan melakukan type checking pada saat **runtime**, sedangkan **static** typing akan melakukan pengecekan pada saat **compilation**
2. Pada static-typing, programmer harus mendeklarasikan **tipe data dari variable** sebelum digunakan, sedangkan dynamic-typing tidak memerlukan

```
// Java example
int num;
num = 5;
```

```
// Groovy example
Num = 5
```

Kelebihan dynamic-typing adalah lebih flexible dan cepat pada saat menulis program, tapi kelebihan static-typing adalah **memperkecil kemungkinan error**.

Java adalah bahasa pemrograman yang menggunakan **static-typing**

Java Exception Handling

```
public void writeList() {
    PrintWriter out = null;

    try {
        System.out.println("Entering" + " try statement");

        out = new PrintWriter(new FileWriter("OutFile.txt"));
        for (int i = 0; i < SIZE; i++) {
            out.println("Value at: " + i + " = " + list.get(i));
        }
    } catch (IndexOutOfBoundsException e) {
        System.err.println("Caught IndexOutOfBoundsException: "
                           +  e.getMessage());

    } catch (IOException e) {
        System.err.println("Caught IOException: " +  e.getMessage());

    } finally {
        if (out != null) {
            System.out.println("Closing PrintWriter");
            out.close();
        }
        else {
            System.out.println("PrintWriter not open");
        }
    }
}
```

Exception : kondisi abnormal yang muncul selama waktu berjalan (execusi), seperti koneksi putus, memory habis, dll, jadi berlainan dengan bug.

Java menangani exception dengan exception handling melalui block **try and catch**.

try : blok untuk melakukan monitoring program

catch : untuk melakukan penangkapan exception

finally : blok yang akan selalu dijalankan baik ada maupun tidak ada exception

Terima Kasih

Selamat mengerjakan
latihan dan kasus