Object Oriented Programming (OOP) --Inheritance-

Saniati

saniati@teknokrat.ac.id

STMIK Teknokrat, Bandar Lampung

Karakteristik OOP

- Encapsulation
- **Inheritance**
- Polimorphisme

Inheritance

- Membuat class baru yang merupakan turunan dari existing class
- Class baru tersebut akan mewarisi semua field dan method yang ada di kelasnya.
- Untuk membuat sebuah kelas (subclass/child class) menjadi turunan kelas lain(super class/parent class), digunakan kata kunci "extends"

```
public static SubClass extends ParentClass{...}
```

PesawatTempur extends Pesawat

```
public class Pesawat {
    int sayap=2;

    void terbang(){
        System.out.println("terbang...");
    }

    void mendarat(){
        System.out.println("mendarat...");
    }
}
```

```
public class PesawatTempur extends Pesawat {
    int rudal = 4;

    void manuver(){
        System.out.println("manuver...");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Pesawat boeing = new Pesawat();
        PesawatTempur f16 = new PesawatTempur();

        boeing.terbang();
        f16.terbang();

        boeing.mendarat();
        f16.mendarat();

        //boeing.manuver();      ERROR
        f16.manuver();
    }
}
```

F16 merupakan object subclass(PesawatTempur), sehingga memiliki semua method parentclass(Pesawat) ditambah milik sendiri.

Overriding

- Mengijinkan subclass mendefinisikan ulang method yang dimiliki parentclassnya.

```
public class Pesawat {
    int sayap=2;

    void terbang(){
        System.out.println("terbang...");
    }

    void mendarat(){
        System.out.println("mendarat...");
    }
}
```

```
public class PesawatTempur extends Pesawat {
    int rudal = 4;

    void manuver(){
        System.out.println("manuver...");
    }

    void terbang(){
        System.out.println("terbang ala tempur...");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Pesawat boeing = new Pesawat();
        PesawatTempur f16 = new PesawatTempur();

        f16.terbang();
    }
}
```

Method terbang() didefinisikan ulang oleh subclass (PesawatTempur)

Method terbang() yang di eksekusi adalah milik sendiri (subclass)

```
terbang ala tempur...
```

- Jika ingin membuat object dari sebuah subclass, yang terjadi yaitu:
  - Jvm akan membuat object dari parent class terlebih dahulu
  - Setelah itu jvm baru akan membuat object dari subclass

```
public class Pesawat {
    int sayap=2;
    Pesawat(){
        System.out.println("object pesawat dibuat...");
    }
```

```
public class PesawatTempur extends Pesawat {
    int rudal = 4;
    PesawatTempur(){
        System.out.println("object pesawat TEMPUR dibuat...");
    }
```

```
public class Main {
    public static void main(String[] args) {
        PesawatTempur f16 = new PesawatTempur();
    }
}
```

Object superclass/parentclass akan buat terlebih dahulu, lalu object subclass/childclass

```
object pesawat dibuat...
object pesawat TEMPUR dibuat...
```

Super();

- Pada kasus sebelumnya:
  - object subclass dibuat oleh programmer,
  - sedangkan object superclass dibuatkan oleh JVM dengan syarat constructor tanpa parameter
  - jika ingin menggunakan parameter, maka pada constructor subclass didefinisikan pemanggilan constructor superclass dengan keyword "super"
- Super untuk memanggil constructor parent class, syarat:
  - Harus dibaris paling atas
  - Hanya boleh satu kali
  - Hanya boleh dilakukan dari constructor subclass

Inheritance

```java
public class Lima {
    Lima(){
        System.out.println("Lima...");
    }

}
```

```java
public class Empat extends Dua{
    Empat(){
        System.out.println("Empat..");
    }
}
```

```java
public class Dua extends Lima{

    Dua(){
        System.out.println("Dua...");
    }
}
```

```java
public class Satu extends Empat {
    Satu(){
        System.out.println("Satu...");
    }
}
```

```java
public class Tiga {

    public static void main(String[] args) {
        Satu sa = new Satu();

    }

}
```

Object Parameter

```
public class Gajah {
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

```
public class Kerbau {
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

Parameter bukan variabel primitif, melainkan user defined (object)

```
public class BonBin {
    static void test(Gajah x){
        x.makan();
        x.tidur();
    }

    static void test(Kerbau x){
        x.makan();
        x.tidur();
    }

    public static void main(String[] args) {
        Gajah g = new Gajah();
        Kerbau k = new Kerbau();

        test(g);
        test(k);
    }
}
```

Javap Namaclass

- Jika ingin mengetahui, sebuah class memiliki method apa saja, dapat digunakan perintah "javap namaclass"

```
E:\pelatihan>javap Mobil
Compiled from "Mobil.java"
public class Mobil {
  int mesin;
  int roda;
  int body;
  Mobil();
  Mobil(int, int, int);
  Mobil(int, int, int, java.lang.String);
  void maju();
  void mundur();
  void belok();
}
```

Kasus

Jika ingin memiliki fungsi test () untuk banyak binatang, maka harus didefinisikan banyak fungsi test. Bagaimana agar cukup satu fungsi untuk semua nya?

```
public class Gajah {
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

```
public class Kerbau {
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

```
public class BonBin {
    static void test(Gajah x){
        x.makan();
        x.tidur();
    }

    static void test(Kerbau x){
        x.makan();
        x.tidur();
    }

    public static void main(String[] args) {
        Gajah g = new Gajah();
        Kerbau k = new Kerbau();

        test(g);
        test(k);
    }
}
```

IS-A

- Buatlah kelas yang memiliki sifat umum (generik) yang mewakili kelas-kelas lainnya, lalu jadikan superclass bagi sub-subclass.
- Karena tiap subclass IS-A superclass

Polymorphism

Parameter merujuk pada Object yang lebih generik, sehingga semua turunan Binatang (Gajah, Kerbau) dapat dimasukkan ke dalam parameter

```
public class BonBin {
    static void test(Binatang x){
        x.makan();
        x.tidur();
    }

    public static void main(String[] args) {
        Gajah g = new Gajah();
        Kerbau k = new Kerbau();

        test(g);
        test(k);
    }
}
```

```
public class Binatang {
    void makan(){
        System.out.println("Makan...");
    }

    void tidur(){
        System.out.println("Tidur...");
    }
}
```

Superclass

```
public class Gajah extends Binatang{
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

Subclass

```
public class Kerbau extends Binatang{
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

Subclass

Polymorphisme

- POLY = banyak
- MORP = bentuk

```
//early binding
//compile time binding
static void test(Kerbau x){
    x.makan();
    x.tidur();
}

//late binding
//run time binding
static void test(Binatang x){
    x.makan();
    x.tidur();
}
```

- Object bisa berubah-ubah bentuk
- RTTI = Run Time Type Identification = tipe data diidentifikasi ketika program running
- LATE BINDING = sebuah object akan di bind ke sebuah method ketika runtime (bukan ketika compile)

Polymorphisme

- Deklarasi **SuperClass namaobject = new SubClass()** lebih sering diimplementasikan.
- Hal tersebut memudahkan dalam melakukan koleksi object

Superclass

Subclass

```
public class BonBin {
    static void test(Binatang x){
        x.makan();
        x.tidur();
    }

    public static void main(String[] args) {
        Binatang g = new Gajah();
        Binatang k = new Kerbau();

        test(g);
        test(k);
    }
}
```

```
public class BonBin {
    static void test(Binatang x){
        x.makan();
        x.tidur();
    }

    public static void main(String[] args) {
        Binatang[] bin = {new Gajah(), new Kerbau()};

        test(bin[0]);
        test(bin[1]);
    }
}
```

Koleksi (array)

Casting Object

```
public class Gajah extends Binatang{
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }

    void duduk(){
        System.out.println("Gajah Duduk...");
    }
}
```

```
public class Binatang {
    void makan(){
        System.out.println("Makan...");
    }

    void tidur(){
        System.out.println("Tidur...");
    }
}
```

```
 9      public static void main(String[] args) {
10          Binatang g = new Gajah();
11          g.makan();
12          g.tidur();
13          g.duduk();
14      }
```

Error, karena object g hanya dapat mengakses atribut dan method dari Binatang

```
public static void main(String[] args) {
    Binatang g = new Gajah();
    g.makan();
    g.tidur();
    ((Gajah)g).duduk();
}
```

Object g di**casting** sehingga dapat mengakses semua atribut dan method dari Gajah

Abstract

- Abstract class merupakan class yang hanya mendeklarasikan methodnya, tanpa implementasi isi.
- Abstract class minimal memiliki satu abstract method
- Baik class maupun method harus ditambahkan keyword “abstract”
- Konsekuensi abstract class, tidak dapat dicreate objectnya, tapi yang mengimplementasikannya bisa.
- Keuntungannya, lebih simpel dan hemat memori(tidak dibuatkan objectnya oleh jvm).

Abstract Class (2)

```
abstract class Binatang {
    abstract void makan();
    abstract void tidur();
}
```

```
public class Gajah extends Binatang{
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

```
public class Kerbau extends Binatang{
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

```
public class BonBin {
    static void test(Binatang x){
        x.makan();
        x.tidur();
    }

    public static void main(String[] args) {
        Binatang g = new Gajah();
        Binatang k = new Kerbau();
        test(g);
        test(k);
    }
}
```

Abstract Class (3)

```
abstract class Binatang {
    abstract void makan();
    abstract void tidur();

    void bernafas(){
        System.out.println("bernafas");
    }
}
```

Boleh mengandung method yang tidak abstract

```
public class Gajah extends Binatang{
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

```
public class Kerbau extends Binatang{
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

```
public class BonBin {
    static void test(Binatang x){
        x.makan();
        x.tidur();
    }

    public static void main(String[] args) {
        Binatang g = new Gajah();
        Binatang k = new Kerbau();
        test(g);
        test(k);
    }
}
```

Abstract Class (4)

- Jika subclass dari class abstract tidak mengimplementasikan method abstract, maka subclass wajib menjadi kelas abstract juga, sehingga tidak bisa dibuat objeknya

```
abstract class Binatang {
    abstract void makan();
    abstract void tidur();
}
```

Tidak implement method makan(), maka wajib menjadi abstract class

```
abstract public class Kerbau extends Binatang{
    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

```
public class Gajah extends Binatang{
    void makan(){
        System.out.println("Gajah Makan...");
    }

    void tidur(){
        System.out.println("Gajah Tidur...");
    }
}
```

```
public class BonBin {
    static void test(Binatang x){
        x.makan();
        x.tidur();
    }

    public static void main(String[] args) {
        Binatang g = new Gajah();
        //Binatang k = new Kerbau();     TIDAK BISA
        test(g);
        //test(k);                       TIDAK BISA
    }
}
```

Abstract class Kerbau tidak dapat dibuat objectnya

Interface

- Cara menghasilkan efek polimorphisme
  - Extends dari class biasa
  - Extend dari abstract class
  - **Implement sebuah interface**
- **Interface:**
  - Mirip seperti abstract class, tapi semua method HARUS abstract
  - Class yang mengimplement menggunakan keyword "implement" bukan "extends
  - Class yang mengimplement HARUS mengimplement SEMUA method yang dideklarasikan dalam interface dan HARUS diberi label "public"
  - Sebuah class BISA mengimplement lebih dari satu interface

Interface (2)

Interface, seluruh method akan bersifat abstract, baik ditulis maupun tidak

Class yang ingin memiliki sifat interface, harus "implements" dan semua method yang method harus "public"

```
interface Binatang {
    abstract void makan();
    abstract void tidur();
}
```

```
class Kerbau implements Binatang{
    public void makan(){
        System.out.println("Gajah Makan...");
    }
    public void tidur(){
        System.out.println("Gajah Tidur
    }
}
```

```
public class Gajah implements Binatang{
    public void makan(){
        System.out.println("Gajah Makan...");
    }

    public void tidur(){
        System.out.println("Gajah Tidur...");
    }

    void duduk(){
        System.out.println("Gajah Duduk...");
    }
}
```

```
public class BonBin {
    static void test(Binatang x){
        x.makan();
        x.tidur();
    }

    public static void main(String[] args) {
        Binatang g = new Gajah();
        Binatang k = new Kerbau();
        test(g);
        test(k);
    }
}
```

Reference

- Pemaparan materi TOT Java Fundamental oleh bapak Tri Haryoko (7-11 Nopember 2016, Bandar Lampung)
- https://docs.oracle.com/javase/tutorial/java
- "Thinking in Java", Bruce Eckel