# Object Oriented Programming (OOP) --Polimorphisme--

Saniati

saniati@teknokrat.ac.id

STMIK Teknokrat, Bandar Lampung

# Karakteristik OOP

- Encapsulation
- Inheritance
- **Polimorphisme**

# 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

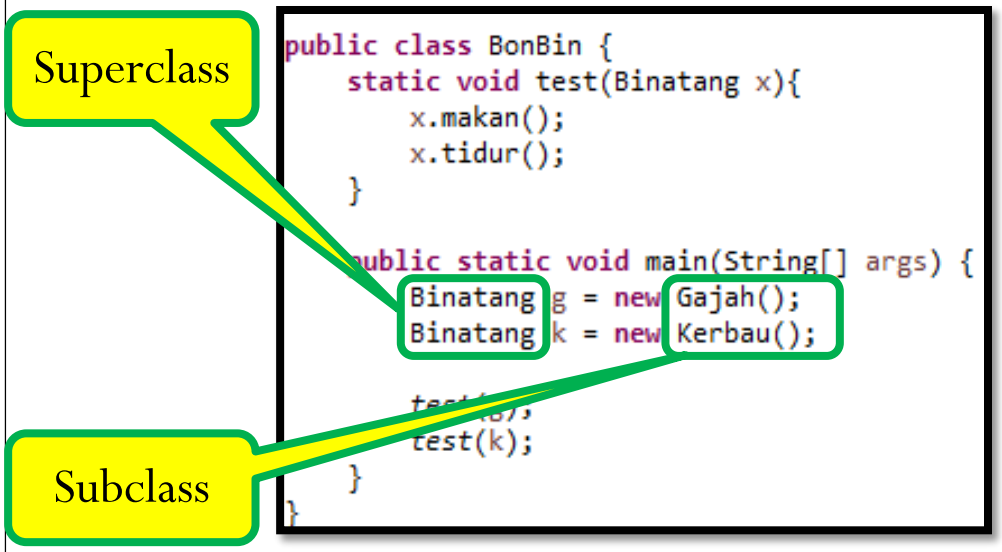
- POLY = banyak
- MORP = bentuk

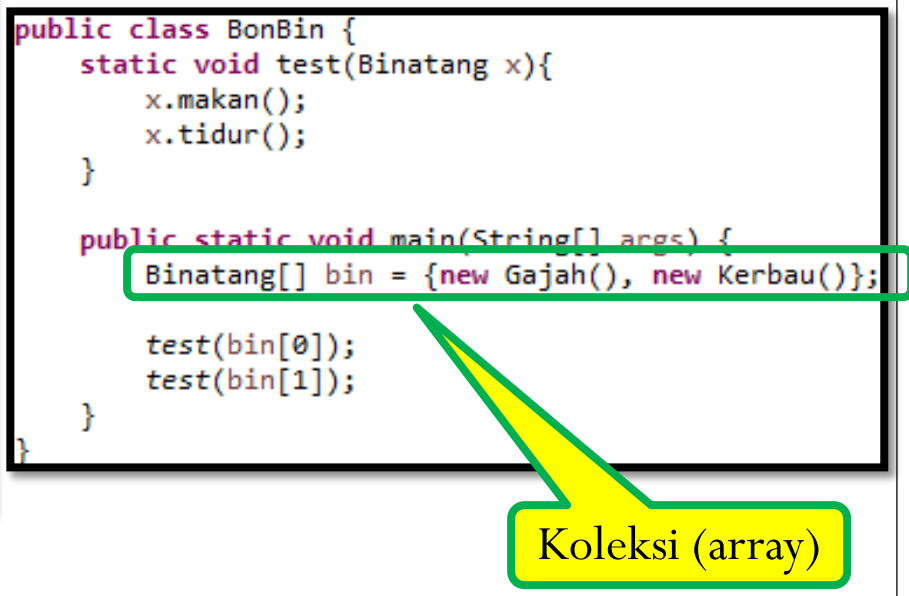
```
//early binding
//compile time binding
static void test(Kerbau x){
    x.makan();
    x.tidur();
}

//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



# 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);
    }
}
```

# Extends + Implements

- Sebuah class HANYA bisa meng-extends SATU class, namun bisa mengimplements BANYAK interface

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

```
public interface Dosen {
    void mengajar();
}
```

```
public interface Sniper {
    void menembak();
}
```

```
public class ManusiaSakti extends Manusia implements Dosen, Sniper{

    @Override
    public void menembak() {
        System.out.println("menembak..");
    }

    @Override
    public void mengajar() {
        System.out.println("mengajar..");
    }

}
```

Class ManusiaSakti meng-"extends" class Manusia dan meng-"implements" interface Dosen & Sniper

```
public class Main {
    static void testDosen(Dosen d){
        d.mengajar();
    }

    static void testSniper(Sniper d){
        d.menembak();
    }

    static void testManusia(Manusia d){
        d.bernafas();
    }
    public static void main(String[] args) {
        ManusiaSakti budi = new ManusiaSakti();
        testDosen(budi);
        testSniper(budi);
        testManusia(budi);
```

budi sebagai object dari Class ManusiaSakti, dapat mewakili Dosen, Sniper dan Manusia

# Program to Interface

- *Program to interface* merupakan pendekatan dimana program dengan konsep OOP lebih fokus pada *interface* bukan pada implementasi (*hidden implementation*).

Mysql implement Database

```
public interface Database {
    abstract void connect();
    abstract void disconnect();
}
```

IbmDB2 implement Database

```
public class Mysql implements Database{
    @Override
    public void connect() {
        System.out.println("mysql connect...");
    }

    @Override
    public void disconnect() {
        System.out.println("mysql disconnect...");
    }
}
```

```
public class IbmDB2 implements Database{
    @Override
    public void connect() {
        System.out.println("ibm db2 connect...");
    }

    @Override
    public void disconnect() {
        System.out.println("ibm db2 disconnect...");
    }
}
```

```
public class AksesDB {
    public static void main(String[] args) {
        Database d = new Mysql();
        d.connect();
        d.disconnect();
    }
}
```

Mengubah dari Mysql ke IbmDB2

```
public class AksesDB {
    public static void main(String[] args) {
        Database d = new IbmDB2();
        d.connect();
        d.disconnect();
    }
}
```

# 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