Praktikum Day #6

Inheritance, Polimorfisma, Interface

Lab Assistant :
Fikri & Jo

Bahasan hari ini :

Inheritance?

Constructor (lagi) ?!

Polimorfisma ?!??!

Inheritance

(Pewarisan)

- Inheritance atau pewarisan adalah mekanisme dalam OOP di mana sebuah kelas (**subclass/child class**) dapat **mewarisi atribut dan method** dari kelas lain (**superclass/parent class**). Dengan ini, kita dapat membangun hierarki objek berdasarkan kesamaan karakteristik dan perilaku.

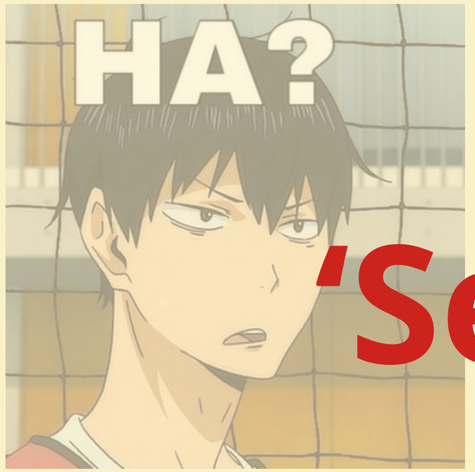
Konsep dasar Inheritance

Inheritance memungkinkan reuse kode. Parent class menyimpan fitur umum, sedangkan **subclass menambahkan atau memodifikasi perilaku sesuai kebutuhan**. Ini memperkuat prinsip DRY (Don't Repeat Yourself) dan modularitas dalam desain perangkat lunak.

Terminologi Penting !

- **Superclass (Parent Class)**: Kelas induk yang menyediakan atribut dan method dasar.
- **Subclass (Child Class)**: Kelas turunan yang mewarisi dan dapat menambah/mengubah method.
- Keyword **extends** (pd bhs Java): Digunakan untuk menyatakan **pewarisan** antar kelas.
- Overriding: Subclass dapat menimpa method dari superclass.

Contoh (w/ UML)



‘Setter’ dan ‘Getter’

- **Setter: Method untuk mengubah nilai atribut private.**
- **Getter:** Method untuk mengambil nilai atribut private.

Digunakan untuk menjaga **prinsip encapsulation** dalam OOP.

Kelebihan:

- Kontrol akses data
- Validasi data (bisa ditambahkan dalam setter)

Keuntungan Inheritance

- **Reusability**: Kode dapat digunakan ulang di subclass.
- **Maintainability:** Perubahan di superclass otomatis berlaku ke subclass.
- **Extensibility:** Mudah memperluas fungsionalitas tanpa mengubah struktur yang ada.
- **Reduksi Duplikasi:** Atribut dan method umum tidak perlu ditulis ulang.

CONSTRUCTOR PADA INHERITANCE

- Constructor tidak diwariskan ke subclass
- Saat membuat objek subclass, constructor parent class akan dipanggil terlebih dahulu
- Jika parent class tidak memiliki constructor default, subclass harus memanggil constructor parent secara eksplisit

```java
public class orang {
    public orang(){
        System.out.println("Ini constuctor ORANG");
    }
}
class mahasiswa extends orang{
    public mahasiswa(){
        System.out.println("Ini constructor MAHASISWA");
    }
}
class mahasiswaMatematika extends mahasiswa{
    public mahasiswaMatematika(){
        System.out.println("Ini constructor MAHASISWAMATEMATIKA");
    }
}
```

```java
public static void main(String[] args) {
        orang o1 = new orang();
        System.out.println("------------");
        mahasiswa m1 = new mahasiswa();
        System.out.println("------------");
        mahasiswaMatematika mt1 = new mahasiswaMatematika()
        System.out.println("------------");
}
```

Output :

```
Ini constuctor ORANG
------------
Ini constuctor ORANG
Ini constructor MAHASISWA
------------
Ini constuctor ORANG
Ini constructor MAHASISWA
Ini constructor MAHASISWAMATEMATIKA
```

Class

```
public class orang {
    public orang(){
        Systen.out.println("Ini constuctor ORANG");
    }
}
class mahasiswa extends orang{
    public mahasiswa(){
        Systen.out.println("Ini constructor MAHASISWA");
    }
}
class mahasiswaMatematika extends mahasiswa{
    public mahasiswaMatematika(){
        Systen.out.println("Ini constructor MAHASISWAMATEMATIKA");
    }
}
```

output

```
Ini constuctor ORANG
Ini constructor MAHASISWA
Ini constructor MAHASISWAMATEMATIKA
```

Main Program

```
public static void main(String[] args) {
    mahasiswaMatematika mt1 = new mahasiswaMatematika();
```

```java
class Parent {
    Parent() {
        System.out.println("Parent Constructor");
    }

    Parent(String message) {
        System.out.println("Parent Constructor: " + message);
    }
}

class Child extends Parent {
    Child() {
        // Secara implisit memanggil super()
        System.out.println("Child Constructor");
    }

    Child(String message) {
        super(message); // Memanggil constructor parent dengan parameter
        System.out.println("Child Constructor: " + message);
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Child c1 = new Child();
        // Output:
        // Parent Constructor
        // Child Constructor

        Child c2 = new Child("Hello");
        // Output:
        // Parent Constructor: Hello
        // Child Constructor: Hello
    }
}
```

ATURAN CONSTRUCTOR INHERITANCE

- super() harus menjadi statement pertama dalam constructor
- Jika tidak ada constructor dalam subclass, compiler akan membuat default constructor yang memanggil super()
- Jika parent class tidak memiliki constructor default, subclass harus memanggil constructor parent secara eksplisit

POLIMORFISME

Polimorfisme adalah kemampuan suatu objek untuk memiliki banyak bentuk. Dalam Java, polimorfisme terbagi menjadi 2:

- Compile-time polymorphism (method overloading)
- Runtime polymorphism (method overriding)

1. Method Overloading (Compile-time)

Terjadi ketika beberapa method memiliki nama sama tetapi parameter berbeda.

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
}
```

2. Method Overriding (Runtime)

Terjadi ketika subclass mengimplementasikan ulang method yang sudah ada di parent/super class.

```java
class Animal {
    void bersuara() {
        System.out.println("Hewan bersuara");
    }
}

class Kucing extends Animal {
    @Override
    void bersuara() {
        System.out.println("Meong");
    }
}

class Anjing extends Animal {
    @Override
    void bersuara() {
        System.out.println("Guk guk");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        Animal hewan1 = new Kucing(); // Upcasting
        Animal hewan2 = new Anjing(); // Upcasting

        hewan1.bersuara(); // Output: Meong
        hewan2.bersuara(); // Output: Guk guk
    }
}
```

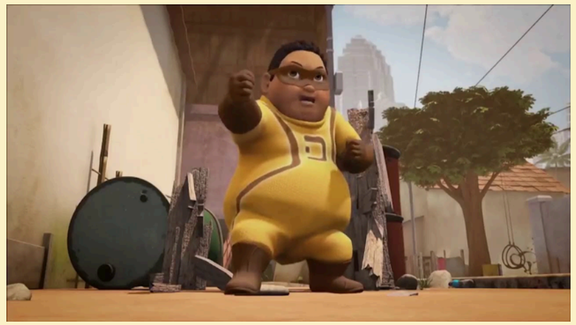
INTERFACE

adalah "kontrak" atau "janji" dalam Java yang berisi daftar method yang harus diimplementasikan oleh class yang menggunakannya.

- Bentuknya seperti template kosong (hanya deklarasi method, tanpa isi).
- Bisa berisi method wajib (tanpa implementasi) dan method opsional (dengan implementasi default).
- Class yang menggunakan interface harus menulis ulang semua method wajib tersebut.

Analogi

```java
interface Superhero {
    void selamatkanWarga();  // wajib
    void gunakanKekuatan();  // wajib
    default void pakaiJubah() {  // opsional
        System.out.println("Memakai jubah...");
    }
}
```



Bayangkan interface seperti "syarat menjadi superhero":

- **Jika suatu class ingin jadi Superhero, harus bisa selamatkanWarga() dan gunakanKekuatan().**
- **Method pakaiJubah() sudah ada implementasinya, jadi boleh dipakai/tidak.**

Kegunaan Interface

1.Memaksa Class untuk Memiliki Method Tertentu
Contoh: Semua class Hewan wajib punya method bersuara(), tapi cara setiap hewan bersuara berbeda.

2. Mencapai Multiple Inheritance
Java tidak bisa warisi banyak class, tapi bisa implement banyak interface.

```
class Robot extends Machine implements BisaJalan, BisaNgobrol { ... }
```

3. Menyembunyikan Detail
Hanya expose method yang penting, sembunyikan cara kerjanya (abstraksi).

Contoh

```java
interface AlatMusik {
    void mainkan();  // wajib
    String getJenis(); // wajib
}

class Gitar implements AlatMusik {
    public void mainkan() {
        System.out.println("Strummm...");
    }
    public String getJenis() {
        return "Gitar";
    }
}

class Piano implements AlatMusik {
    public void mainkan() {
        System.out.println("Ting ting...");
    }
    public String getJenis() {
        return "Piano";
    }
}
```

Kapan Pakai Interface?

1.Ketika beberapa class berbeda butuh method yang sama, tapi implementasinya beda.
(Contoh: Gitar dan Piano sama-sama butuh mainkan())
2. Ketika ingin memisahkan "apa yang harus dilakukan" dengan "bagaimana melakukannya".
(Contoh: Interface BisaDisimpan hanya bilang "harus bisa simpan", caranya terserah class).
3. Ketika butuh multiple inheritance (warisi banyak sifat).

```java
interface Bersuara {
    void suara();  // Semua class wajib implement ini
}
```

```java
class Hewan {
    void makan() {
        System.out.println("Sedang makan...");
    }
}
```

```java
class Kucing extends Hewan implements Bersuara {
    @Override
    public void suara() {  // Implementasi method interface
        System.out.println("Meong!");
    }
}

class Anjing extends Hewan implements Bersuara {
    @Override
    public void suara() {  // Implementasi method interface
        System.out.println("Guk guk!");
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        // Array dengan tipe Interface (Bersuara)
        Bersuara[] hewan = { new Kucing(), new Anjing() };

        // Loop: panggil method suara() dari masing-masing objek
        for (Bersuara h : hewan) {
            h.suara();  // Output beda tergantung objeknya
        }

        // Akses method dari superclass
        Kucing cat = new Kucing();
        cat.makan();  // Output: "Sedang makan..."
    }
}
```

Output :

```
Meong!
Guk guk!
Sedang makan...
```

Penjelasan

1. **Interface Bersuara:**
    - **"Kontrak" yang mewajibkan class punya method suara().**
2. **Inheritance (Hewan → Kucing/Anjing)**
    - **Subclass mewarisi method makan() dari Hewan.**
3. **Polimorfisme:**
    - **Objek Kucing dan Anjing disimpan dalam array Bersuara.**
    - **Saat panggil h.suara(), Java otomatis pilih implementasi yang benar.**

Bahasa Bayi

1. **Interface = "Semua hewan harus bisa bersuara, tapi caranya bebas".**
2. **Inheritance = "Kucing dan Anjing sama-sama butuh makan".**
3. **Polimorfisme = "Suruh semua hewan bersuara, tanpa peduli jenisnya".**

SELESAI