Materi 6 : Abstract Class, Interface & Overloading

DAFTAR ISI

1.1 Abstract Class

Abstract Class merupakan class yang masih dalam bentuk abstrak. Seringkali juga disebut sebagai jenis kelas yang tidak dapat di instansiasi (tidak dapat membuat objek langsung dari kelas tersebut) dan seringkali berfungsi sebagai kerangka atau cetakan (blueprint) untuk kelas-kelas anak (subclass) yang akan mengimplementasikan metode-metode abstrak yang di deklarasikan dalam class abstract tersebut. Adapun beberapa karakteristik dari class abstract adalah sebagai berikut:

1. Tidak dapat di instansiasi

   Class abstract tidak dapat secara langsung dibuat sebuah objek dengan menggunakan keyword new. Contohnya `KelasAbstrak objek = new KelasAbstrak();`
2. Harus memiliki setidaknya satu method abstrak

   Class abstract dapat memiliki metode-metode yang di deklarasikan sebagai "abstract." Metode abstrak adalah metode yang tidak memiliki implementasi (tubuh) di dalam class abstract. Sebaliknya, metode abstrak ini harus diimplementasikan (disediakan tubuhnya) oleh kelas anak yang mewarisi class abstract tersebut
3. Dapat memiliki method biasa

   Selain metode abstrak, class abstract juga dapat memiliki metode biasa (konkrit) dengan implementasi yang lengkap
4. Dapat memiliki atribut

   Class abstract juga dapat memiliki atribut seperti kelas biasa.
5. Keyword abstract

   Untuk mendeklarasikan kelas sebagai abstrak, harus terdapat kata kunci `abstract` sebelum kata kunci `class` saat mendefinisikan kelas tersebut

   Berikut adalah contoh sederhana dari abstract class

```java
public abstract class Character {

    // Metode abstrak yang akan diimplementasikan oleh kelas anak
    public abstract void attack();
    // Metode biasa
    public void survive() {
        System.out.println("Method bertahan.");
    }
}
```

Dalam contoh di atas, `Character` adalah class abstract dengan satu metode abstrak `attack()`, yang harus di implementasikan oleh kelas anak yang mewarisi kelas `Character`.

1.1.1 Cara Membuat Abstract Class

Untuk lebih memahami konsep dari class abstract, mari kita buat sebuah program sederhana. Yaitu membuat sebuah program dengan konsep permainan menyerang antar karakter.

Pertama, buat sebuah program java dengan nama **`CharacterGame`**. Setelah itu buat beberapa file dengan nama **`Character.java`**, **`Hero.java`** dan **`Enemy.java`** dalam package **`src`**.

Selanjutnya, jadikan class **`Character.java`** sebagai class abstract dengan menambahkan kode sebagai berikut.

```
public abstract class Character {
    // Atribut atau properti
    private String name;
    // Constructor
    public Character(String name) {
        this.name = name;
    }
    // Method abstrak
    public abstract void attack();
    // Method biasa
    public void survive(){
        System.out.println("Karakter bertahan");
    }
    // Method Getter
    public String getName() {
        return name;
    }
    // Method Setter
    public void setName(String name) {
        this.name = name;
    }
}
```

Kemudian untuk class **`Hero.java`** dan **`Enemy.java`** buat jadi subclass dengan meng-extends class abstract **`Character.java`**. Untuk kode program nya seperti dibawah ini:

`Hero.java` :

```
public class Hero extends Character{
    public Hero(String name) {
        super(name);
    }
```

```
    @Override
    public void attack() {
        System.out.println("Pahlawan menyerang musuh !!!");
    }
}
```

`Enemy.java` :

```
public class Enemy extends Character{

    public Enemy(String name) {
        super(name);
    }
    @Override
    public void attack() {
        System.out.println("Musuh menyerang pahlawan !!!");
    }
}
```

Sekarang kita sudah membuat semua entitas, menjadi sebuah class. Sekarang ki

Selanjutnya untuk class utama atau `App.java` , kode program nya sebagai berikut :

```
public class App {
    public static void main(String[] args) throws Exception {
        // Instansiasi objek
        Hero hero = new Hero("Anas");
        Enemy enemy = new Enemy("Jokowow");
        // Memanggil method attack() dari kedua kubu
        hero.attack();
        enemy.attack();
    }
}
```

1.2 Interface

Untuk lebih memahami konsep dari interface, mari kita buat sebuah program sederhana. Yaitu membuat sebuah program hewan.

Pertama, buat sebuah program java dengan nama Animal. Setelah itu buat beberapa file dengan nama `Animal.java`, `Cat.java` dan `Crocodile.java` dalam package `src`.

Selanjutnya, jadikan class `Animal.java` sebagai interface dengan menambahkan kode sebagai berikut.

```
```

```
public interface Animal {
    // Method abstract
    String eat();
    String sleep();
}
```

Kemudian untuk class `Cat.java` dan `Crocodile.java` buat jadi subclass dengan meng-implements interface `Character.java`. Untuk kode program nya seperti dibawah ini:

`Cat.java`

```
public class Cat implements Animal{
    @Override
    public String eat() {
        return "Kucing makan ikan bakar";
    }
    @Override
    public String sleep() {
        return "Kucing tidur di kamar tidur";
    }
}
```

Crocodile.java

```
public class Crocodile implements Animal{
    @Override
    public String eat() {
        return "Buaya darat makan ayam bakar";
    }
    @Override
    public String sleep() {
        return "Buaya darat tidur di asrama";
    }
}
```

Selanjutnya untuk class utama atau `App.java`, kode program nya sebagai berikut:

```
public class App {
    public static void main(String[] args) throws Exception {
        // Instansiasi objek dari class Cat dan Crocodile
        Cat kucing = new Cat();
        Crocodile buaya = new Crocodile();
        // Memanggil method yang ada di setiap class
        System.out.println(kucing.eat());
        System.out.println(kucing.sleep());
```

```
        System.out.println(buaya.eat());
        System.out.println(buaya.sleep());
    }
}
```

Nah sekarang kan tadi class bisa multiple interface, adapun buat file baru lagi yaaitu **`Attack.java`** dengan tipe interface. Untuk kode nya sebagai berikut:

```
public interface Attack {
    String attack();
}
```

Kemudian di class utama atau **`App.java`** ada juga tambahan kode, yaitu:

```
System.out.println(buaya.attack());
```

1.3 Perbedaan Abstract Class & Interface

Interface	Abstract Class
Interface tidak dapat memiliki access specifier untuk fungsinya karena secara default bersifat public.	Abstract dapat memiliki access specifier.
Hanya dapat berisi signature, dan tidak dapat berisi implementasinya.	Dapat berisi implementasi secara lengkap.
Hanya dapat berisi signature, dan tidak dapat berisi implementasinya.	Kecepatan proses relative lebih cepat.
Tidak dapat berisi field.	Dapat berisi field dan konstanta.
Interface hanya bisa menampung method abstract.	Dapat menampung method non-abstract.

1.4 Overloading

Overloading adalah sebuah konsep dalam pemrograman Java yang memungkinkan untuk mendefinisikan beberapa metode dengan nama yang sama dalam satu kelas, asalkan parameter-parameter yang diterima oleh metode-metode tersebut berbeda. Overloading memungkinkan

membuat beberapa versi dari metode dengan nama yang sama, tetapi dengan tipe, jumlah, atau urutan parameter yang berbeda.

Dalam overloading, metode-metode yang memiliki nama yang sama harus memiliki parameter yang berbeda, yaitu kombinasi yang unik dari jenis, jumlah, dan urutan parameter. Berikut adalah contoh sederhana overloading dalam Java:

```
public class Result {
    // Method pertama dengan 1 parameter bertipe int
    void printData(int nim){
        System.out.println("NIM : " + nim);
    }
    // Method kedua dengan 2 parameter bertipe int dan string
    void printData( int nim, String name){
        System.out.println("NIM : " + nim);
        System.out.println("Nama  : " + name);
    }
    // Method ketiga dengan 3 parameter bertipe int, string dan string
    void printData(int nim, String name, String major){
        System.out.println("NIM : " + nim);
        System.out.println("Nama  : " + name);
        System.out.println("Prodi : " + major);
    }
    public static void main(String[] args) {
        Result result = new Result();
        result.printData(123456789);
        result.printData(123456789, "Aisyah Romaito Siregar");
        result.printData(123456789, "Aisyah Romaito Siregar",
                         "D3 Manajemen Informatika");
    }
}
```

Overloading memungkinkan membuat metode yang lebih fleksibel dan mudah digunakan dalam berbagai konteks. Ini mempermudah pemrograman karena dapat menggunakan nama metode yang sama untuk tugas yang berbeda, selama parameter berbeda.