BAB VI
ABSTRACT CLASS DAN INTERFACE

Salah satu pilar dalam Pemrograman Berorientasi Objek (OOP) adalah *abstraction*. Abstraction adalah sebuah proses yang dilakukan untuk menyembunyikan detail implementasi dengan hanya menampilkan fungsi atau penggunaannya kepada user. Sebagai contoh dari abstraction ini adalah fitur SMS pada telepon genggam. Kita mengetahui bahwa fitur tersebut dapat kita gunakan untuk mengirim pesan kepada pengguna lain. Namun implementasi dari fitur tersebut disembunyikan dari kita, sehingga kita hanya mengetahui fungsi dari fitur SMS namun kita tidak mengetahui bagaimana fitur tersebut dapat bekerja. Konsep abstraction ini membantu kita untuk dapat berfokus tentang apa yang dapat dilakukan oleh sebuah object dibandingkan berfokus pada bagaimana proses tersebut dilakukan.

Dalam OOP, kita dapat menggunakan abstraction dengan beberapa cara. Pada bahasa pemrograman Java, kita dapat mengimplementasikan konsep abstraction, dengan menggunakan abstract class dan interface.

6.1. Abstract Class

Abstract class merupakan sebuah class yang ditandai dengan keyword `abstract`. Abstract class class dapat memiliki abstract method di dalam class tersebut. Abstract method adalah method yang tidak memiliki implementasi body dari method tersebut.

```
// Class abstract yang ditandai dengan keyword abstract
abstract class Plane{

    // Abstract method
    abstract String getPlaneName();

}
```

Pada potongan kode diatas, kita dapat melihat contoh dari implementasi abstract class pada bahasa pemrograman Java. Di dalam class tersebut juga terdapat sebuah abstract method yang hanya memiliki nama serta return type dari method tersebut. Karena tidak memiliki body, maka setiap class yang mewarisi/memperluas class tersebut harus membuat implementasi dari method tersebut.

Abstract class tidak dapat diinstansiasi menjadi sebuah objek. Oleh karena itu, untuk dapat menggunakan class tersebut maka kita perlu memperluas class tersebut pada kelas turunannya dan mengimplementasikan semua abstract method pada abstract class tersebut.

```
abstract class Plane{

    // Abstract method
    abstract String getPlaneName();

    int getDimesion(){
       return 2
    }

}

class Rectangle extends Plane{
    @Override
    String getPlaneName(){
        return “Rectangle”;
    }
}
```

Pada contoh diatas, kita memiliki sebuah class `Rectangle` yang memperluas kelas `Plane`. Pada kelas `Rectangle`, kita membuat implementasi dari method `getPlaneName()` karena method tersebut pada kelas `Plane` merupakan method abstract. Perhatikan bahwa kita tidak wajib untuk membuat implementasi dari method `getDimension()` karena method tersebut pada dasarnya bukanlah sebuah abstract method.

6.2. Interface

Interface dapat dianalogikan sebagai sebuah blueprint untuk sebuah class. Melalui interface, kita dapat mendefinisikan property dan behaviour yang harus dimiliki oleh sebuah class. Sama seperti abstract class, interface juga merupakan sebuah mekanisme yang memungkinkan kita untuk menerapkan konsep abstraction.

Dalam sebuah interface, kita hanya dapat memiliki method abstract yakni method yang tidak memiliki body. Selain itu, property dalam sebuah interface secara default akan memiliki modifier `public, static, dan final`. Oleh karena itu, sama seperti abstract class, kita tidak dapat membuat sebuah instance dari interface.

Untuk membuat interface, kita perlu menambahkan keyword `interface` di depan nama dari interface tersebut

```
interface Describable{
    void describe();
}
```

Pada contoh diatas, kita membuat sebuah interface Describable yang didalamnya memiliki method `describe()`. Secara default, seluruh method yang didefenisikan dalam sebuah interface akan memiliki modifier `public` dan `abstract`. Oleh karena itu, kita tidak wajib untuk menambahkan kedua modifier tersebut.

Karena sebuah interface tidak dapat kita instansiasi, maka kita perlu membuat sebuah kelas yang mengimplementasikan interface tersebut.

```
interface Describable{
    void describe();
}

class Human implements Describable{
    String name;
    int age;

    @Override
    public void describe(){
        System.out.println(“This class covers basic
```

```
information about human”);
    }

    private void walk(){
        System.out.println(“Human is currently walking”);
    }
}
```

Kita membuat sebuah class bernama Human yang menerapkan interface Describable. Untuk melakukannya kita perlu menambahkan keyword `implements` setelah nama class diikuti oleh sejumlah interface yang akan diimplementasikan. Setelah itu, kita perlu menuliskan body dari setiap method yang kita buat pada interface pada class yang mengimplementasikannya. Pada contoh diatas, kita menuliskan body dari method `describe()`.

Pada java, kita dapat mengimplementasikan beberapa interface berbeda yang dipisahkan oleh tanda koma. Hal ini berbeda saat kita ingin memperluas sebuah kelas dimana kita hanya dibatasi untuk dapat memperluas satu kelas saja pada java.

6.2.1. Default method

Sebelum diperkenalkannya Java 8, interface hanya dapat menampung abstract method. Apabila kita ingin menambahkan method baru, maka kita perlu mengimplementasikannya kembali pada seluruh class yang mengimplementasikan interface tersebut. Dengan hadirnya Java 8, kita dapat membuah sebuah default method yang memiliki implementasi body sehingga hal tersebut tidak akan mempengaruhi setiap class yang mengimplementasikan interface tersebut.

```
interface Describable{
    void describe();
    default void show(){
        System.out.println(“Default show
method”);
    }
```

```
}

class Human implements Describable{
    String name;
    int age;

    @Override
    public void describe(){
        System.out.println(“This class covers
basic information about human”);
    }

    private void walk(){
        System.out.println(“Human is currently
walking”);
    }

    public static void main(String args[]){
        Human human = new Human();
        human.show() // Default show method
    }

}
```

6.2.2. Static method

Sejak Java 8, kita juga dapat membuat method static pada interface

```
interface Describable{
    void describe();
    static void show(){
        System.out.println(“Default show
method”);
    }
}

class Human implements Describable{
    String name;
    int age;
```

```
    @Override
    public void describe(){
        System.out.println(“This class covers
basic information about human”);
    }

    private void walk(){
        System.out.println(“Human is currently
walking”);
    }

    public static void main(String args[]){
        Describable.show() // Default show
method
    }

}
```

Untuk memanggil sebuah method static, kita perlu memanggil interface yang memiliki method tersebut.

6.2.3. Private method

Sejak rilisnya Java 9, kita dapat menggunakan access modifier private pada method dalam interface. Dengan menggunakan modifier private, kita dapat menerapkan konsep enkapsulasi ke dalam interface dan memungkinkan kita untuk berbagi kode antara method non-abstract dalam interface. Perlu diperhatikan bahwa private method tidak dapat diakses dari kelas yang mengimplementasikan interface tersebut.

```
interface Describable{
    void describe();
    default void show(){
        showMessage();
    }
    private void showMessage(){
        System.out.println(“Default show
method”);
```

```
    }
}

class Human implements Describable{
    String name;
    int age;

    @Override
    public void describe(){
        System.out.println(“This class covers
basic information about human”);
    }

    private void walk(){
        System.out.println(“Human is currently
walking”);
    }

    public static void main(String args[]){
        Human human = new Human();
        human.show() // Default show method
    }

}
```