2020

Pemrograman Berorientasi Objek

MODUL 5 POLYMORPHISM, ABSTRACT DAN INTERFACE

ASISTEN LABORATORIUM

Tujuan Pembelajaran :

1. Mengetahui apa itu polymorphism
2. Mengetahui apa itu abstract
3. Mengetahui apa itu interface

Polymorphism

Kata polymorphism adalah memiliki banyak bentuk. Jadi, kita dapat mengartkan bahwa polymorphism adalah suatu konsep untuk membuat sebuah objek memliki banyak bentuk. Polymorphism adalah suatu konsep pada OOP yang memungkinkan kita untuk melakukan sebuah tindakan memiliki cara yang berbeda-beda, contohnya kita memiliki satu kelas dan kelas tersebut diimplementasikan secara berbeda-beda.

Pada java polymorphism dibagi menjadi 2, yaitu:

a. Compile time Polymorphism

   Dapat juga disebut sebagai static polymorphism. Metode ini dapat digunakan dengan method overloading.

b. Runtime Polymorphism

   Dapat juga disebut dynamic polymorphism. Ini adalah proses dimana fungsi yang di overridden kan. Metode ini menggunakan metode override.

Contoh :

Compile time polymorphism :

```
class MultiplyFun {
      static int Multiply(int a, int b)
    {
        return a * b;
    }

    static double Multiply(double a, double b)
    {
        return a * b;
    }
}
```

```
class Main {
    public static void main(String[] args)
    {

        System.out.println(MultiplyFun.Multiply(2, 4));

        System.out.println(MultiplyFun.Multiply(5.5, 6.3));
    }
}
```

Runtime polymorphism

```
class Parent {

    void Print()
    {
        System.out.println("parent class");
    }
}

class subclass1 extends Parent {

    void Print()
    {
        System.out.println("subclass1");
    }
}

class subclass2 extends Parent {

    void Print()
    {
        System.out.println("subclass2");
    }
}
```

```
class TestPolymorphism3 {
    public static void main(String[] args)
    {

        Parent a;

        a = new subclass1();
        a.Print();

        a = new subclass2();
        a.Print();
    }
}
```

Abstract

Kelas abstract dan method adalah dimana sebuah parent class mempunyai nama method tetapi membutuhkan child class untuk mengisi isi/proses dalam method tersebut. Kelas abstract tidak dapat di buat karena itu harus memlikik child class.

Abstract class setidaknya berisi 1 abstract method seperti berikut

```
abstract class Shape {
    int color;

    // An abstract function (like a pure virtual function in C++)
    abstract void draw();
}
```

Saat menurunkan dari abstract class, child class method harus dideklarasikan dengan nama yang sama seperti method abstract class dan juga jumlah argument harus sama.

Jadi, ketika child class mewarisi dari kelas abstrak memiliki aturan berikut:

1. Metode child class harus didefinisikan dengan nama yang sama dan itu me-redecode method abstract induknya
2. Metode child class harus didefinisikan dengan pengubah akses yang sama atau kurang dibatasi
3. Jumlah argumen yang diperlukan harus sama. Namun, child class mungkin memiliki argumen opsional sebagai tambahan

Contoh

```
abstract class Base {
    abstract void fun();
}
class Derived extends Base {
    void fun() { System.out.println("Derived fun() called"); }
}
class Main {
    public static void main(String args[]) {

        // Uncommenting the following line will cause compiler error as the
        // line tries to create an instance of abstract class.
        // Base b = new Base();

        // We can have references of Base type.
        Base b = new Derived();
        b.fun();
    }
}
```

Interface

Cara lain untuk membuat abstraction class adalah menggunakan interface. Dimana interface class betul-betul sebuah class yang abstract yang digunakan untuk mengelompokkan method yang kosong.

```
interface Animal {
    public void animalSound(); // interface method (does not have a body)
    public void run(); // interface method (does not have a body)
  }
```

Untuk mengakses interface method, interface harus di "implemented" oleh kelas lain dengan keyword implements. Badan dari method akan diberikan oleh kelas yang mengimplementasi.

```
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
```

```
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

class MyMainClass {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```