Polymorphism (Polimorfisme)

Pemrograman Berorientasi Objek
Minggu 7

Alfa Faridh Suni

Polymorphisme

* Poly + morph = banyak + bentuk
* Polymorphism yang berarti satu objek dapat memiliki banyak bentuk yang berbeda.
* Polymorphism adalah konsep sederhana dalam pemrograman berorientasi objek yang berarti kemampuan suatu variabel referensi objek untuk memiliki aksi yang berbeda bila method yang sama dipanggil, dimana aksi method tergantung dari tipe objeknya.
* Keywords: *abstract, interface, implements*

Polymorphism

Polymorphism

Kemampuan sebuah variabel reference untuk merubah behavior sesuai dengan apa yang dipunyai object.

polymorphism membuat objek-objek yang berasal dari subclass yang berbeda, diperlakukan sebagai objek-objek dari satu superclass. Hal ini terjadi ketika memilih method yang sesuai untuk diimplementasikan ke objek tertentu berdasarkan pada subclass yang memiliki method bersangkutan.

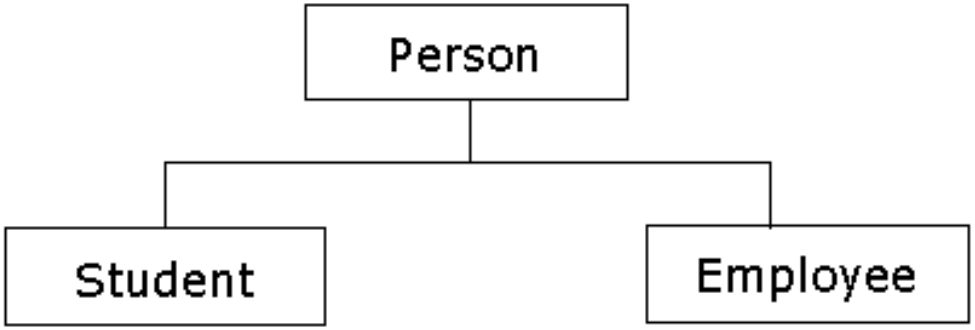
Polymorphism

Kondisi yang harus dipenuhi supaya ***polimorfisme*** dapat diimplementasikan adalah :

* Method yang dipanggil harus melalui variabel dari basis class atau superclass.
* Method yang dipanggil harus juga menjadi method dari basis class.
* Signature (argumen/parameter) method harus sama baik pada superclass maupun subclass.
* Method access attribute pada subclass tidak boleh lebih terbatas dari basis class. (ingat tentang encapsulasi)

Polymorphism

- Pada contoh sebelumnya, kita diberikan parent class yaitu Person dan subclassnya adalah Student, sekarang kita tambahkan subclass lainnya dari Person yaitu Employee
- Berikut adalah hirarki class nya.



Polymorphism

Dalam Java, kita dapat membuat referensi dari suatu superclass ke object dari subclassnya. Sebagai contoh,

```
public static main( String[] args ) {

    Person  ref;
    Student studentObject = new Student();
    Employee employeeObject = new Employee();

    ref = studentObject; //titik referensi Person kepada
                  // sebuah object Student
}
```

Polymorphism

misalnya, kita memiliki sebuah method getName dalam superclass Person. Dan kita meng-override method ini di kedua subclass yaitu Student dan Employee

```
public class Student {
    public String getName(){
        System.out.println("Student Name:" + name);
        return name;
    }
}

public class Employee {
    public String getName(){
        System.out.println("Employee Name:" + name);
        return name;
    }
}
```

Polymorphism

- Kembali ke method utama kita, ketika kita mencoba memanggil method getName dari referensi Person ref, method getName dari object Student akan dipanggil.

- Sekarang, jika kita memberi ref kepada object Employee, maka method getName juga akan dipanggil

Polymorphism

```
public static main( String[] args ) {
        Person  ref;
        Student studentObject = new Student();
        Employee employeeObject = new Employee();

        ref = studentObject; //titik referensi Person kepada object Student

        //getName dari class Student dipanggil
        String temp=ref.getName();
        System.out.println( temp );

        ref = employeeObject; //titik referensi Person kepada  object Employee

        //getName dari class Employee dipanggil
        String temp = ref.getName();
        System.out.println( temp );
}
```

Polymorphism

- Contoh lain yang menggambarkan polymorphism adalah ketika kita mencoba untuk passing reference kepada method

- jika kita memiliki sebuah method static *printInformation* yang menerima referensi Person sebagai parameter

```
public static printInformation( Person p ){
                        . . . .
}
```

Polymorphism

Sebenarnya kita dapat passing reference dari Employee dan Student kepada method *printInformation* selama kedua class tersebut merupakan subclass dari Person

```
public static main( String[] args )
{
        Student   studentObject = new Student();
        Employee  employeeObject = new Employee();

        printInformation( studentObject );

        printInformation( employeeObject );
}
```

Casting Object

Instance dari class juga dapat di-casting menjadi instance dari class lain, dengan satu batasan ::

- Class asal dan tujuan harus direlasikan dalam inheritance, salah satu class harus berupa subclass dari class lain.

- Casting object digunakan untuk mengkonversi nilai primitif menuju tipe yang lebih besar, beberapa object tidak perlu mengalami casting secara explisit.

Casting Object

Gunakan sintaks berikut untuk casting object,

```
(classname)object
```

classname - nama class tujuan
object - reference untuk object asal

Contoh Casting Object

Berikut ini merupakan contoh proses casting sebuah instance dari class VicePresident ke instance dari class Employee.

VicePresident merupakan subclass dari class Employee dengan beberapa informasi tambahan.

```
Employee emp = new Employee();
VicePresident veep = new VicePresident();

emp = veep;

// casting explisit
veep = (VicePresident)emp;
```

Abstract Class

Abstract class

- Sebuah class yang tidak bisa diinstansiasi
- Sering muncul pada puncak hierarki class object-oriented programming(OOP), mendefinisikan segala type action/tindakan yang mungkin dengan object semua subclass dari class

Abstract Class

- Method abstract
  - Method di dalam abstract class tidak mempunyai implementasi
  - Untuk membuat abstract method, tulis saja deklarasi method tanpa body dan gunakan keyword abstract

- Sebagai contoh,

```
public abstract void someMethod();
```

Contoh Abstract Class

```
public abstract class LivingThing {
    public void breath(){
        System.out.println("Living Thing breathing...");
    }

    public void eat(){
        System.out.println("Living Thing eating...");
    }

    /**
     * abstract method walk
     * Kita ingin method ini di-overrride oleh subclass dari
     * LivingThing
     */
    public abstract void walk();
}
```

Abstract class

- Ketika sebuah class meng-extends abstract class LivingThing, diwajibkan meng-override abstract method walk(), jika tidak, subclass tersebut juga akan menjadi abstract class, dan oleh karena itu tidak bisa diinstansiasi

- Sebagai contoh,

```
public class Human extends LivingThing {

    public void walk(){
        System.out.println("Human walks...");
    }

}
```

Petunjuk penulisan program

- Menggunakan abstract class untuk mendefinisi jenis-jenis yang luas dari behavior yang ada di puncak hirarki class object-oriented programming, dan menggunakan subclassnya untuk menyediakan detail implementasi dari abstract class.

interface

interface

- Interface
  - Adalah semacam blok spesial yang hanya berisi tanda tangan method (dan mungkin konstan).
  - Menggambarkan tandatangan dari seperangkat method, tanpa body
  - Menggambarkan cara standard dan publik penetapan behavior class
  - Mengijinkan class,dengan mengabaikan lokasi mereka di dalam hirarki class untuk menerapkan behavior.
  - CATATAN: Interface memperlihatkan polymorphism juga, sejak program diperbolehkan memanggil sebuah method interface, dan versi yang sesuai dari method tersebut akan dieksekusi tergantung pada jenis object yang dilewatkan kepada pemanggilan method interface

Mengapa menggunakan interface?

- Untuk memiliki class yang tidak berhubungan yang mengimplementasikan method yang sama.
  - Contoh:
    - Class Line dan MyInteger
      - Tidak berhubungan
      - Keduanya mengimplementasikan method perbandingan
        - isGreater
        - isLess
        - isEqual

Mengapa menggunakan interface?

- Untuk mengungkapkan sebuah interface pemrograman object tanpa pernyataan classnya
- Untuk model multi inheritance yang mengijinkan sebuah class untuk memiliki lebih dari satu superclass.

Membuat interface

- Untuk membuat sebuah interface, kita tulis:

```
public interface [InterfaceName] {
      //beberapa method tanpa body
}
```

Membuat interface

- Sebagai contoh, mari buat sebuah interface yang mendefinisikan hubungan antara dua object sesuai dengan pesanan dari object.

```
public interface Relation
{
    public boolean isGreater( Object a, Object b);
    public boolean isLess( Object a, Object b);
    public boolean isEqual( Object a, Object b);
}
```

Membuat interface

- Untuk menggunakan interface, kita gunakan keyword implements
- Sebagai contoh,

```
/**
 * Class ini menjelaskan segment garis
 */
public class Line implements Relation {
   private double x1;
   private double x2;
   private double y1;
   private double y2;
public Line(double x1, double x2, double y1, double y2){
        this.x1 = x1;
        this.x2 = x2;
        this.y1 = y1;
        this.y2 = y2;
   }
```

Membuat interface

```
public double getLength(){
          double length = Math.sqrt((x2-x1)*(x2-x1) +
                              (y2-y1)* (y2-y1));
          return length;
     }

     public boolean isGreater( Object a, Object b){
          double aLen = ((Line)a).getLength();
          double bLen = ((Line)b).getLength();
          return (aLen > bLen);
     }

     public boolean isLess( Object a, Object b){
          double aLen = ((Line)a).getLength();
          double bLen = ((Line)b).getLength();
          return (aLen < bLen);

     }

     public boolean isEqual( Object a, Object b){
          double aLen = ((Line)a).getLength();
          double bLen = ((Line)b).getLength();
          return (aLen == bLen);
     }
}
```

Membuat interface

- Ketika class Anda mencoba untuk mengimplementasikan sebuah interface, pastikan selalu bahwa Anda mengimplementasikan semua method dari interface tersebut, jika tidak, Anda akan mendapatkan kesalahan ini.

```
Line.java:4: Line is not abstract and does not override
  abstract method
  isGreater(java.lang.Object,java.lang.Object) in Relation
public class Line implements Relation
       ^
1 error
```

Interface vs. Abstract Class

- Semua interface method tidak memiliki body
- Beberapa Abstract class memiliki method dengan implementasi
- Sebuah interface hanya dapat didefinisikan constant
- Sebuah abstract class tampak seperti class biasa yang dapat mendeklarasikan variabel.
- Interface tidak memiliki hubungan inheritance secara langsung dengan setiap class tertentu, mereka didefinisikan secara independen.
- Abstract class dapat di-subclass-kan

Interface vs. Class

- Persamaan:
  - Interface dan class adalah sama-sama sebuah type
  - Hal ini berarti bahwa interface dapat digunakan di tempat dimana sebuah class dapat digunakan
  - Sebagai contoh:

```
PersonInterface  pi = new Person();
Person     pc = new Person();
```

- Perbedaan:
  - Anda tidak dapat membuat instance dari sebuah interface
  - Sebagai contoh:

```
PersonInterface  pi = new PersonInterface(); //ERROR!
```

Interface vs. Class

- Persamaan:
  - Interface dan class, keduanya dapat mendefinisikan method
- Perbedaan:
  - Interface tidak memiliki segala implementasi dari method

Meng-extends Class vs. Implementasi interface

- Sebuah class hanya bisa meng-EXTENDS SATU superclass, tetapi juga bisa meng-IMPLEMENTASIKAN BANYAK interface
- Sebagai contoh:

```
public class Person implements PersonInterface,
                    LivingThing,
                    WhateverInterface {

    //beberapa kode disini
}
```

Meng-extends Class vs. Implementasi interface

- Contoh lain:

```
public class ComputerScienceStudent extends Student
                    implements PersonInterface,
                               LivingThing {

    //Beberapa kode disini
}
```

Inheritance antar interface

- Interface bukanlah bagian dari hirarki class, bagaimanapun juga interface dapat memiliki hubungan inheritance antar mereka sendiri
- Sebagai contoh:

```
public interface PersonInterface {
    . . .
}

public interface StudentInterface extends PersonInterface {
    . . .
}
```

Kesimpulan

- Inheritance (superclass, subclass)
- Menggunakan keyword super untuk mengakses field dan constructor dari superclass
- Meng-override Method
- Final Method dan Final Class
- Polymorphism (Abstract Class, interface)