MODUL PRAKTIKUM

Pewarisan, Polymorphism Dan Interface

Versi 1.4

JAVA EDUCATION NETWORK INDONESIA

1. Tujuan

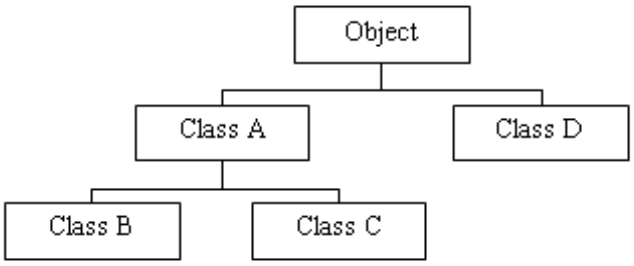
- Mendefinisikan *superclasses* dan *subclasses*
- *Override* method dari *superclasses*
- Membuat method final dan class final

2. Latar Belakang

Dalam bagian ini, kita akan membicarakan bagaimana suatu class dapat mewariskan sifat dari class yang sudah ada. Class ini dinamakan *subclass* dan induk class dinamakan *superclass*. Kita juga akan membicarakan sifat khusus dari Java dimana kita dapat secara otomatis memakai method yand tepat untuk setiap object tanpa memperhatikan asal dari s*ubclass* object. Sifat ini dinamakan polimorfisme. Pada akhirnya, kita akan mendiskusikan tentang interface yang membantu mengurangi penulisan program.

Dalam Java, semua class, termasuk class yang membangun Java API, adalah subclasses dari superclass Object. Contoh hirarki class diperlihatkan di bawah ini.

Beberapa class di atas class utama dalam hirarki class dikenal sebagai superclass. Sementara beberapa class di bawah class pokok dalam hirarki class dikenal sebagai subclass dari class tersebut.



Class hierarchy in Java.

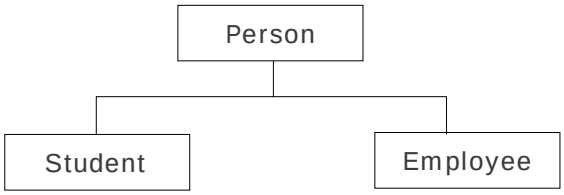
Pewarisan adalah keuntungan besar dalam pemrograman berbasis object karena suatu sifat atau method didefinisikan dalam *superclass*, sifat ini secara otomatis diwariskan dari semua *subclasses*. Jadi, Anda dapat menuliskan kode method hanya sekali dan mereka dapat digunakan oleh semua subclass. Subclass hanya butuh mengimplementasikan perbedaannya sendiri dan induknya.

Interface adalah jenis khusus dari blok yang hanya berisi method signature(atau constant). Interface mendefinisikan sebuah(signature) dari sebuah kumpulan method tanpa tubuh.

Interface mendefinisikan sebuah cara standar dan umum dalam menetapkan sifat-sifat dari class-class. Mereka menyediakan class-class, tanpa memperhatikan lokasinya dalam

hirarki class, untuk mengimplementasikan sifat-sifat yang umum. Dengan catatan bahwa interface-interface juga menunjukkan polimorfisme, dikarenakan program dapat memanggil method interface dan versi yang tepat dari method yang akan dieksekusi tergantung dari tipe object yang melewati pemanggil method interface.

Sekarang, class induk Person dan subclass Student dari contoh sebelumnya, kita tambahkan subclass lain dari Person yaitu Employee. Di bawah ini adalah hierarkinya,



Dalam Java, kita dapat membuat referensi yang merupakan tipe dari superclass ke sebuah object dari subclass tersebut.
Kemampuan dari referensi untuk mengubah sifat menurut object apa yang dijadikan acuan dinamakan polimorfisme. Polimorfisme menyediakan multiobject dari subclasses yang berbeda untuk diperlakukan sebagai object dari superclass tunggal, secara otomatis menunjuk method yang tepat untuk menggunakannya ke *particular* object berdasar subclass yang termasuk di dalamnya.

Contoh lain yang menunjukkan properti polimorfisme adalah ketika kita mencoba melalui referensi ke method. Misalkan kita punya method statis **printInformation** yang mengakibatkan object Person sebagai referensi, kita dapat me-referensi dari tipe Employee dan tipe Student ke method ini selama itu masih subclass dari class Person.

3. Percobaan

Percobaan 1 Mendefinisikan Subclass dan Superclass :

```
public class Person {
    protected String name;
    protected String address;
    /**
    * Default constructor
    */
    public Person(){
        System.out.println("Inside Person:Constructor");
        name = "";
        address = "";
    }
    /**
    * Constructor dengan dua parameter
    */
    public Person( String name, String address) {
        this.name = name;
        this.address = address;
    }
    /**
    * Method accessor
    */
    public String getName() {
        return name;
    }
    public String getAddress() {
        return address;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setAddress(String add) {
        this.address = add;
    }
}
```

```
public class Student extends Person{
    public Student()
    {
        //super("SomeName", "SomeAddress");
        //super();
        //super.name = "name";
        System.out.println("Inside Student:Constructor");
    }
    public static void main( String[] args) {
            Student anna = new Student();
    }
}
```

Percobaan 2 Superclass dan Subclass :

```
public class Pakaian {
    private int ID = 0; // Default ID untuk semua Pakaian
    private String keterangan = "-keterangan diperlukan-"; // default
    private double harga = 0.0; // Harga default untuk semua Pakaian
    private int jmlStok = 0; // jumlah default untuk semua Pakaian
    private static int UNIQUE_ID=0; // Static member ditambahkan dalam //constructor
                                    //untuk menghasilkan ID yang unik
    public Pakaian() {
        ID=UNIQUE_ID++;
    }
    public int getID() {
        return ID;
    }
    public void setKeterangan (String d){
        keterangan=d;
    }
    public String getKeterangan(){
        return keterangan;
    }
    public void setHarga(double p) {
        harga = p;
    }
    public double getHarga() {
        return harga;
    }
    public void setJmlStok (int q){
        jmlStok=q;
    }
    public int getJmlStok(){
        return jmlStok;
    }
}
```

```
public class Baju extends Pakaian {
    //Kode Warna R=Merah, B=Biru, G=Hijau, U=Belum Ditentukan
    public char kodeWarna = 'U';
    public Baju(){
        super.setHarga(1500.0);
        super.setJmlStok(5);
        super.setKeterangan("biru");
    }
    //Method ini menampilkan nilai untuk suatu item
    public void tampilInformasiBaju(){
        System.out.println("ID Baju: " + getID());
        System.out.println("Keterangan: " + getKeterangan());
        System.out.println("Kode Warna: " + kodeWarna);
        System.out.println("Harga baju: " + getHarga());
        System.out.println("Jumlah stok: " + getJmlStok());
    }//akhir method display
    public static void main(String args[]){
        Baju b=new Baju();
        b.tampilInformasiBaju();
    }
}//akhir kelas
```

Percobaan 3 Polimorphisme :

```
Public class Person {
    protected String name;
    protected String address;
    /**
    * Default constructor
    */
    public Person(){
        System.out.println("Inside Person:Constructor");
        name = "";
        address = "";
    }
    /**
    * Constructor dengan dua parameter
    */
    public Person( String name, String address) {
        this.name = name;
        this.address = address;
    }
    /**
    * Method accessor
    */
    public String getName() {
        System.out.println("Person Name : " +name);
                return name;
    }
    public String getAddress() {
        return address;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setAddress(String add) {
        this.address = add;
    }
}
```

```
public class Student extends Person{
        public Student() {
            //super("SomeName", "SomeAddress");
            //super();
            //super.name = "name";
            System.out.println("Inside Student:Constructor");
        }
        public String getName() {
            System.out.println("Student Name : " +name);
            return name;
        }
        public static void main( String[] args) {
            Student anna = new Student();
        }
}
```

```
public class Employee extends Person{
    public String getName() {
        System.out.println("Employee Name:" +name);
        return name;
    }
    public static void main(String[] args)
    {
        Person ref;
        Student studentObject = new Student();
        Employee employeeObject = new Employee();

        ref = studentObject; //Person menunjuk kepada object Student

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

        ref = employeeObject; //Person menunjuk kepada object Employee

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

Percobaan 4 Enkapsulasi:

```
public class PrivateElevator2 {
    private boolean bukaPintu = false;
    private int lantaiSkrg = 1;
    private int berat = 0;
    private final int KAPASITAS = 1000;
    private final int LANTAI_ATAS = 5;
    private final int LANTAI_BAWAH = 1;
    public void buka() {
        bukaPintu = true;
    }
    public void tutup() {
        hitungKapasitas();
        if (berat <= KAPASITAS) {
            bukaPintu = false;
        } else {
          System.out.println("Elevator kelebihan beban");
          System.out.println("Pintu akan tetap terbuka sampai seseorang keluar");
        }
    }
//pada dunia nyata, elevator menggunakan sensor berat untuk memeriksa beban, tetapi agar lebih sederhana,
//kami menggunakan bilangan acak untuk berat
    private void hitungKapasitas() {
        berat = (int)(Math.random()*1500);
        System.out.println("Berat: " + berat);
    }
    public void naik() {
        if (!bukaPintu) {
            if (lantaiSkrg < LANTAI_ATAS) {
                lantaiSkrg++;
                System.out.println(lantaiSkrg);
            } else {
                System.out.println("Sudah mencapai lantai atas");
            }
        } else {
            System.out.println("Pintu masih terbuka");
        }
    }
```

```
    public void turun() {
        if (!bukaPintu) {
            if (lantaiSkrg > LANTAI_BAWAH) {
                lantaiSkrg--;
                System.out.println(lantaiSkrg);
            } else {
                System.out.println("Sudah mencapai lantai bawah");
            }
        } else {
            System.out.println("Pintu masih terbuka");
        }
    }

    public void setLantai(int tujuan) {
        if ((tujuan >= LANTAI_BAWAH)&&(tujuan <= LANTAI_ATAS)) {
            while (lantaiSkrg != tujuan){
                if (lantaiSkrg < tujuan) {
                    naik();
                } else {
                    turun();
                }
            }
        } else {
            System.out.println("Lantai Salah");
        }
    }
    public int getLantai() {
        return lantaiSkrg;
    }
    public boolean getStatusPintu() {
        return bukaPintu;
    }
}
```

```
public class PrivateElevator2Test {
    public static void main(String args[]){
        PrivateElevator2 privElevator = new PrivateElevator2();
        privElevator.buka();
        privElevator.tutup();
        privElevator.turun();
        privElevator.naik();
        privElevator.naik();
        privElevator.buka();
        privElevator.tutup();
        privElevator.turun();
        privElevator.buka();
        privElevator.turun();
        privElevator.tutup();
        privElevator.turun();
        privElevator.turun();
        int lantai = privElevator.getLantai();
        if (lantai != 5 && !privElevator.getStatusPintu()) {
            privElevator.setLantai(5);
        }
        privElevator.setLantai(10);
        privElevator.buka();
    }
}
```

Percobaan 5 Menampilkan 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-overridden oleh subclasses
    */
    public abstract void walk();
}
```

```
public class Human extends LivingThing {
     public void walk(){
        System.out.println("Human walks...");
    }
}
```

Percobaan 6 Interface:

```
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;
    }
    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);
    }
}
```

Relation.java ➔ bertindak sebagai interface

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

4. Latihan

4.1 Extend StudentRecord

Dalam latihan ini, kita ingin untuk membuat catatan siswa yang lebih khusus yang berisi informasi tambahan tentang pengetahuan komputer siswa. Tugasnya adalah meng-extend class StudentRecord yang mengimplementasikan pelajaran sebelumnya. Cobalah untuk meng-override beberapa method yang ada dalam superclass StudentRecord, jika Anda benar-benar membutuhkannya.

4.2 Bentuk Abstract Class

Cobalah untuk membuat class abstract yang dinamai Shape dengan method abstract getArea() dan getName(). Tulis dua subclasses-nya yaitu Circle dan Square. Anda dapat menambahkan method tambahan ke dalam subclasses jika diinginkan.