Praktikum 8

Polymorphism

Tujuan

Memahami dan menerapkan konsep polimorfisme, overloading, overriding method, penggunaan instance of dan casting object dalam pemrograman berorientasi objek.

Dasar Teori

Polymorphism (polimorfisme) adalah kemampuan untuk mempunyai beberapa bentuk class yang berbeda. Polimorfisme ini terjadi pada saat suatu obyek bertipe parent class, akan tetapi pemanggilan constructornya melalui subclass. Misalnya deklarasi pernyataan berikut ini:

```
Employee employee=new Manager();
```

dimana Manager() adalah kontruktor pada class Manager yang merupakan subclass dari class Employee.

Overloading adalah suatu keadaan dimana beberapa method sekaligus dapat mempunyai nama yang sama, akan tetapi mempunyai fungsionalitas yang berbeda. Contoh penggunaan overloading dilihat dibawah ini:

- `Gambar(int t1)`
  - Terdapat 1 parameter titik, untuk menggambar titik
- `Gambar(int t1,int t2)`
  - Terdapat 2 parameter titik, untuk menggambar garis
- `Gambar(int t1,int t2,int t3)`
  - Terdapat 3 parameter titik, untuk menggambar segitiga
- `Gambar(int t1,int t2,int t3,int t4`
  - Terdapat 4 parameter titik, untuk menggambar persegi empat

Overloading ini dapat terjadi pada class yang sama atau pada suatu parent class dan subclass-nya. Overloading mempunyai ciri-ciri sebagai berikut:

1. Nama method harus sama,
2. Daftar parameter harus berbeda,
3. Return type boleh sama atau berbeda.

Overriding adalah suatu keadaan dimana method pada subclass menolak method pada parent class-nya. Overriding mempunyai ciri-ciri sebagai berikut :

1. Nama method harus sama.

2. Daftar parameter harus sama.
3. Return type harus sama.

Berikut ini contoh terjadinya overriding dimana method Info() pada class Child meng-override method Info() pada class parent:

```
class Parent {
    public void Info() {
        System.out.println("Ini class Parent");
    }
}

class Child extends Parent {
    public void Info() {
        System.out.println("Ini class Child");
    }
}
```

Method yang terkena override (overridden method) diharuskan tidak boleh mempunyai modifier yang lebih luas aksesnya dari method yang meng-override (overriding method).

Virtual Method Invocation (VMI) bisa terjadi jika terjadi polimorfisme dan overriding. Pada saat obyek yang sudah dibuat tersebut memanggil overridden method pada parent class, kompiler Java akan melakukan invocation (pemanggilan) terhadap overriding method pada subclass, dimana yang seharusnya dipanggil adalah overridden method. Berikut contoh terjadinya VMI:

```
class Parent {
     int x = 5;
     public void Info() {
     System.out.println("Ini class Parent");
     }
}

class Child extends Parent {
     int x = 10;
     public void Info() {
     System.out.println("Ini class Child");
     }
}

public class Tes {
     public static void main(String args[]) {
     Parent tes=new Child();
     System.out.println("Nilai x = " + tes.x);
     tes.Info();
     }
}
```

Hasil dari running program diatas adalah sebagai berikut:

```
Nilai x = 5

Ini class Child
```

Polymorphic arguments adalah tipe suatu parameter yang menerima suatu nilai yang bertipe subclass-nya. Berikut contoh dari polymorphics arguments:

```
class Pegawai {
     // …
}

class Manajer extends Pegawai {
     // …
}

public class Tes {
      public static void Proses(Pegawai peg) {
           // …
      }

      public static void main(String args[]) {
          Manajer man = new Manajer();
          Proses(man);
      }
}
```

Pernyataan instance of sangat berguna untuk mengetahui tipe asal dari suatu polymorphic arguments. Untuk lebih jelasnya, misalnya dari contoh program sebelumnya, kita sedikit membuat modifikasi pada class Tes dan ditambah sebuah class baru Kurir, seperti yang tampak dibawah ini:

```
…
class Kurir extends Pegawai {
     …
}

public class Tes {
      public static void Proses(Pegawai peg) {
           if (peg instanceof Manajer) {
               …lakukan tugas-tugas manajer…
           } else if (peg instanceof Kurir) {
               …lakukan tugas-tugas kurir…
           } else {
               …lakukan tugas-tugas lainnya…
           }
      }

      public static void main(String args[]) {
          Manajer man = new Manajer();
          Kurir kur = new Kurir();
          Proses(man);
          Proses(kur);
      }
}
```

Seringkali pemakaian instanceof diikuti dengan casting object dari tipe parameter ke tipe asal. Misalkan saja program kita sebelumnya. Pada saat kita sudah melakukan instanceof dari tipe Manajer, kita dapat melakukan casting object ke tipe asalnya, yaitu Manajer. Caranya adalah seperti berikut:

```
…
if (peg instanceof Manajer) {
   Manajer man = (Manajer) peg;
   // … lakukan tugas-tugas manajer …
}
…
```

Percobaan 1

Program berikut ini menerapkan konsep Overriding Method. Pertama kali buatlah dalam satu package class Karyawan seperti pada program dibawah ini.

```
class Karyawan {

       private String nama;
       private double gaji;
       private static double persennaikgaji=0.10;

       public void Karyawan(String nm,double gj ){
           this.setNama(nm);
           this.setGaji(gj);
       }

       void setNama(String nm){
           nama=nm;
       }

       void setGaji(double gj){
           gaji=gj;
       }

       static void setPresentase(double persen){
          persennaikgaji=persen;
       }

       String getNama(){
           return nama;
       }

       double getGaji(){
           return gaji;
       }

       static double getPersentase(){
           return persennaikgaji;
       }

       void naikkanGaji(){
           gaji+=(gaji*persennaikgaji);
       }

}
```

Kemudian buatlah class Manager yang extends dari class Karyawan. Fungsi getGaji pada kelas Manager akan meng-*override* method "getGaji" yang dimiliki oleh class parent yaitu class Karyawan.

```
public class Manager extends Karyawan {

    private static double bonus=500;

    Manager(String nm, double gj){
        super.Karyawan(nm, gj);
    }

    double getBonus(){
        return bonus;
    }
```

```
    void setBonus(double bns){
        bonus=bns;
    }

    double getGaji(){
        double gajidasar=super.getGaji();
        return(gajidasar+bonus);
    }

}
```

Kemudian jalankan program TestManajer.java berikut ini. Amati dan jelaskan penggunaan overriding method pada program tersebut.

```
public class TestManager {

    public static void main(String args[])
    {
        Manager mng=new Manager("Andi",300.0);
        System.out.println(mng.getNama());
        System.out.println(mng.getBonus());
        System.out.println(mng.getGaji());
    }
}
```

Percobaan 2

Program berikut ini menerapkan konsep constructor overriding, dengan melakukan pemanggilan parent constructor. Pertama kali buatlah dalam satu package class Employee seperti pada program dibawah ini.

```
class Employee {

    String nama;

    Employee(){
        System.out.println("Konstruktor Employee() dijalankan");
    }

    void Employee(String n){
        this.nama=n;
        System.out.println("Konstruktor Employee(String n) dijalankan");
    }

    void getDetails(){
        System.out.println(nama);
    }

}
```

Buatlah dan jalankan class Manager2 yang extends dari class Employee. Amati dan jelaskan pemanggilan parent constructor pada program tersebut.

```
class Manager2 extends Employee{
```

```
    String departement;

    Manager2(){
        this.Employee("sales");
        System.out.println("Konstruktor manager() dijalankan");
    }

    Manager2(String dept){
        departement=dept;
        System.out.println("Konstruktor Manager(String dept)
                           dijalankan");
    }

    void getDetails(){
        System.out.println(departement);
    }

    public static void main(String args[]){
        Manager2 e=new Manager2();
        e.getDetails();
    }

}
```

Percobaan 3

Program berikut ini menerapkan konsep Virtual Method Invocation (VMI) dan Polymorphic Arguments. Pertama kali buatlah dalam satu package class Employee seperti pada program dibawah ini.

```
public class Employee {

       String nama;
       int Salary;

       void Employee(){
       }

       void Employee(String nm){
           this.nama=nm;
           System.out.println("Employee");
       }

       public int salary(){
           return 0;
       }
}
```

Buatlah class Programmer yang extends dari class Employee.

```
public class Programmer extends Employee {

    private static final int prgSal=50000;
    private static final int prgBonus=10000;

    public int Salary(){
        return prgSal;
    }
```

```
    public int bonus(){
        return prgBonus;
    }
}
```

Buatlah class Manager yang extends dari class Employee.

```
class Manager extends Employee{

     private static final int mgrSal=40000;
     private static final int tunjungan=40000;

     public int salary(){
         return mgrSal;
     }

     public int tunjungan(){
         return tunjungan;
     }

}
```

Buatlah dan jalankan class Payroll yang digunakan untuk menghitung gaji dari Programer dan Manager.

```
class Payroll {

    public int calcPayroll(Employee2 emp)
    {
       int money=emp.salary();
       if(emp instanceof Manager)
       money+=((Manager) emp).tunjungan();
       return money;
       if(emp instanceof Programmer)
       money+=((Programmer)emp).bonus();
       return money;
   }

   public static void main(String[] args)
   {
       Payroll pr=new Payroll();
       Programmer prg=new Programmer();
       Manager mgr=new Manager();
       System.out.println("Payroll untuk Programmer: "+pr.)
       System.out.println("Payroll untuk Manager:" +pr.calcPayroll(mgr))
   }

}
```

Jelaskan penggunaan Virtual Method Invocation (VMI) dan Polymorphic Arguments pada program di atas.

Percobaan 4

Program berikut ini menerapkan konsep Overloading Method. Amati dan jelaskan penggunaannya pada program tersebut.

```
import java.awt.Point;

class MyRect {

    int x1=0;
    int y1=0;
    int x2=0;
    int y2=0;

    MyRect buildRect(int x1,int y1,int x2, int y2) {
        this.x1=x1;
        this.x2=y1;
        this.x2=x2;
        this.y2=y2;
        return this;
    }

    MyRect buildRect(Point topleft,Point bottomRight) {
        x1=topleft.x;
        x2=topleft.y;
        x2=bottomRight.x;
        y2=bottomRight.y;
        return this;
    }

    MyRect buildRect(Point topleft, int w, int h) {
        x1=topleft.x;
        y1=topleft.y;
        x2=(x1+w);
        y2=(y1+h);
        return this;
    }

    void printRect() {
        System.out.println("MyRect :<" +x1+","+y1);
        System.out.println(", "+x2+","+y2+">");
    }

    public static void main(String[] args) {
       MyRect rect=new MyRect();
       System.out.println("Calling buildRect with coordinates 25,25,
50,50:");
       rect.buildRect(25,25,50,50);
       rect.printRect();
       System.out.println("****");
       System.out.println("Calling buildRect with points(10,10),
(20,20):");
       System.out.println("width(50) and height(50): ");
       rect.printRect();
       System.out.println("***");
    }

}
```

Percobaan 5

a) Program berikut ini menerapkan konsep upcasting pada konversi suatu kelas turunan pada kelas parent pada class Music.

```
class Note {
       private int value;

       private Note(int val){
            value=val;
       }

       public static final Note{
           middleC=new Note(0);
           cSharp= new Note(1);
           cFlat=new Note(2);
       }
}

class Instrument {

       public void play(Note n){
               System.out.println("Instrument.play()");

       }
}

class Wind extends Instrument {

     public void play(Note n){
         System.out.println("Wind.play()");
     }
}

public class Music {

     public static void tune(Instrument i){
         i.play(Note.MIDDLE_C);
     }

     public static void main(String[]args){
         Wind flute=new Wind();
         tune(flute); // UPCASTING
     }
}
```

Jalankan program tersebut, amati dan jelaskan penggunaan Upcasting pada program tersebut.

b) Kemudian tambahkan class Stringed yang extends class Instrument.

```
class Stringed extends Instrument {

    public void play(Note n)
    {
        System.out.println("Stringed.play()");
    }
}
```

Kemudian modifikasi class Music seperti pada baris berikut ini:

```
public class Music {

     public static void tune(Instrument i){
         i.play(Note.MIDDLE_C);
     }

     public static void main(String[]args){
         Wind flute=new Wind();
         Stringed violin=new Stringed();
         tune(flute); // UPCASTING
         tune(violin); // UPCASTING
     }
}
```

Jalankan program tersebut, amati dan jelaskan penggunaan Upcasting pada program tersebut.

c) Tambahkan class Brass yang extends class Instrument.

```
class Brass extends Instrument {

    public void play(Note n){
        System.out.println("Brass.play()");
    }
}
```

Kemudian modifikasi class Music seperti pada baris berikut ini:

```
public class Music {

     public static void tune(Wind i){
         i.play(Note.MIDDLE_C);
     }

     public static void tune(Stringed i){
         i.play(Note.MIDDLE_C);
     }

     public static void tune(Brass i){
         i.play(Note.MIDDLE_C);
     }

     public static void main(String[]args){
         Wind flute=new Wind(); // NO UPCASTING
         Stringed violin=new Stringed();
         Brass frenchHorn=new Brass();
         tune(flute);
         tune(violin);
         tune(frenchHorn);
     }
 }
```

Program diatas menggunakan metode overloading sebagai pembanding upcasting Jalankan program tersebut, amati dan jelaskan penggunaannya pada program tersebut.

Percobaan 6

Program berikut ini menerapkan metode Upcasting, simpan ke dalam Shapes.java. Amati dan jelaskan penggunaan Upcasting pada program tersebut.

```
class Shapes {
    void draw(){}
    void erase(){}
}

class Circle extends Shapes {
    void draw(){
        System.out.println("Circle.draw()");
    }

    void erase(){
        System.out.println("Circle.erase()");
    }
}

class Square extends Shapes {
    void draw(){
        System.out.println("Shape.draw()");
    }

    void erase(){
        System.out.println("Square.erase()");
    }
}

class Triangle extends Shapes {
    void draw(){
        System.out.println("Triangle.draw()");
    }
    void erase(){
        System.out.println("Triangle.erase");
    }
}

public class Shape {
    public static Shape randShape() {
        switch((int)(Math.random()*3)) {
            default:
            case 0:return new Shape();
            case 1:return new Square();
            case 2:return new Triangle();
        }
    }
}
```

Percobaan 7

Program berikut ini menerapkan penggunaan urutan constructor pada polymorphism. Amati dan jelaskan penggunaannya pada program tersebut.

```
class Meal{
    Meal(){
        System.out.println("Meal()");
    }
}

class Bread{
    Bread(){
        System.out.println("Bread();");
    }
}

class Cheese{
     Cheese(){
         System.out.println("Cheese");
     }
}

class Lettuce{
    Lettuce(){
        System.out.println("Lettuce()");
    }
}

class Lunch extends Meal{
    Lunch(){
        System.out.println("Lunch()");
    }
}

class PortableLunch extends Lunch{
    PortableLunch(){
        System.out.println("PortableLunch()");
    }
}

public class Sandwich extends PortableLunch{
    Bread b=new Bread();
    Cheese c=new Cheese();
    Lettuce l=new Lettuce();
    Sandwich()
    {
        System.out.println("Sandwich()");
    }
    public static void main(String[]args)
    {
        new Sandwich();
    }
}
```

Percobaan 8

Program berikut ini menerapkan konsep downcasting and Runtime Type Identification (RTI). Amati dan jelaskan penggunaannya pada program tersebut.

```
import java.util.*

class Useful{
    public void f(){}
    public void g(){}
}

class Moreuseful extends Useful{
    public void f(){}
    public void g(){}
    public void u(){}
    public void v(){}
    public void w(){}
}

public class RTI {
    public static void main(String[] args) {
        Useful[]x={
            new Useful(),
            new Moreuseful()
        };
        x[0].g();
        x[1].f();
        ((Moreuseful)x[1]).u();
        ((Moreuseful)x[0]).u();
    }
}
```