PRAKTIKUM PEMROGRAMAN BERBASIS OBJEK
"Polimorfisme"

DOSEN PEMBIMBING

Septian Enggar Sukmana, S.Pd., M.T.

Disusun oleh :

Muhammad Reza Khatami

(2041720076)

PROGRAM STUDI D-IV TEKNIK INFORMATIKA
JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG 2021

1. **Percobaan 1 – Bentuk dasar polimorfisme**
   - **Pertanyaan**
     1. Class apa sajakah yang merupakan turunan dari class Employee?
        Jawab : Class PermanentEmployee dan Class InternshipEmployee
     2. Class apa sajakah yang implements ke interface Payable?
        Jawab : Class PermanentEmployee dan Class ElectricityBill
     3. Perhatikan class **Tester1**, baris ke-10 dan 11. Mengapa **e**, bisa diisi dengan objek **pEmp** (merupakan objek dari class **PermanentEmployee**) dan objek **iEmp** (merupakan objek dari class **InternshipEmploye**) ?
        Jawab : Karena kedua class tersebut merupakan turunan dari class Employee yg diinisialkan dengan huruf e
     4. Perhatikan class **Tester1**, baris ke-12 dan 13. Mengapa **p**, bisa diisi dengan objek **pEmp** (merupakan objek dari class **PermanentEmployee**) dan objek **eBill** (merupakan objek dari class **ElectricityBill**) ?
        Jawab : Karena kedua class tersebut mengimplementasikan Class Interface Payable yang diinisialkan dengan huruf p
     5. Coba tambahkan sintaks:
        p = iEmp;
        e = eBill;
        pada baris 14 dan 15 (baris terakhir dalam method **main**) ! Apa yang menyebabkan error?
        Jawab : Error tersebut dapat terjadi karena class InterenshipEmployee tidak mengimplementasikan Class Interface Payable
     6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!
        Jawab : Polimorfisme adalah kemampuan suatu objek untuk memiliki banyak Bentuk dan polimorphisme dapat diterapkan dalam class-class yang memiliki relasi inheritance dan interface.

2. **Percobaan 2 – Virtual method invocation**
   - **Pertanyaan**
     1. Perhatikan class **Tester2** di atas, mengapa pemanggilan **e.getEmployeeInfo()** pada baris 8 dan **pEmp.getEmployeeInfo()** pada baris 10 menghasilkan hasil sama?
        Jawab : karena e dan pEmp adalah objek yang sama yaitu e merupakan inisialilasi dari Class Employee dan e sama dengan pEmp.
     2. Mengapa pemanggilan method **e.getEmployeeInfo()** disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan **pEmp.getEmployeeInfo()** tidak?
        Jawab : karena objek ‘e’ menggunakan method dari class turunan atau child-nya sedangkan ‘pEmp’ dari class-nya sendiri.
     3. Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?

Jawab : virtual method invocation adalah penggunaan method yang dilakukan superclass kepada subclass ketika adanya override method. Disebut virtual dikarenakan method tidak diakses langsung oleh class-nya sendiri, melainkan harus membuat objek yang mengarah pada method di luar class.

3. **Percobaan 3 – Heterogenous Collection**
   - **Pertanyaan**
     1. Perhatikan array **e** pada baris ke-8, mengapa ia bisa diisi dengan objekobjek dengan tipe yang berbeda, yaitu objek **pEmp** (objek dari **PermanentEmployee**) dan objek **iEmp** (objek dari **InternshipEmployee**) ?
        Jawab : karena kedua objek tersebut sama sama mengextends Class Employee sehingga keduanya bisa diisikan dalam satu array yang sama.
     2. Perhatikan juga baris ke-9, mengapa array **p** juga biisi dengan objek-objek dengan tipe yang berbeda, yaitu objek **pEmp** (objek dari **PermanentEmployee**) dan objek **eBill** (objek dari **ElectricityBilling**) ?
        Jawab : karena kedua objek tersebut sama sama mengimplementasikan Class Interface Playable sehingga keduanya bisa diisikan dalam satu array yang sama.
     3. Perhatikan baris ke-10, mengapa terjadi error?
        Jawab : Terjadi error karena Class ElectricityBill tidak mengextends Class Employee.

4. **Percobaan 4 – Argumen polimorfisme, instanceof dan casting objek**
   - **Pertanyaan**
     1. Perhatikan class **Tester4** baris ke-7 dan baris ke-11, mengapa pemanggilan **ow.pay(eBill)** dan **ow.pay(pEmp)** bisa dilakukan, padahal jika diperhatikan method **pay()** yang ada di dalam class **Owner** memiliki argument/parameter bertipe **Payable**?
        Jika diperhatikan lebih detil eBill merupakan objek dari ElectricityBill dan pEmp merupakan objek dari PermanentEmployee?
        Jawab : Karena ‘pEmp’ dan ‘eBill’ mengimplementasi Payable, oleh karena itu semua container yang menerima objek dari Payable dapat dimasukkan oleh objek yang mengimplementasikannya
     2. Jadi apakah tujuan membuat argument bertipe **Payable** pada method **pay()** yang ada di dalam class **Owner**?
        Jawab : agar dapat dimasuki objek yang akan dibayar atau yang mengimplementasikan Payable
     3. Coba pada baris terakhir method **main()** yang ada di dalam class **Tester4** ditambahkan perintah **ow.pay(iEmp);**

```
3   public class Tester4 {
4       public static void main(String[] args) {
5           Owner ow = new Owner();
6           ElectricityBill eBill = new ElectricityBill(5, "R-1");
7           ow.pay(eBill);//pay for electricity bill
8           System.out.println("------------------------------");
9
10          PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
11          ow.pay(pEmp);//pay for permanent employee
12          System.out.println("------------------------------");
13
14          InternshipEmployee iEmp = new InternshipEmployee("Sunarto", 5);
15          ow.showMyEmployee(pEmp);//show permanent employee info
16          System.out.println("------------------------------");
17          ow.showMyEmployee(iEmp);//show internship employee info
18
            ow.pay(iEmp);
20      }
21  }
```

Mengapa terjadi error?
Jawab : objek ‘iEmp’ tidak terhubung dengan Payable atau tidak mengimplementasikan Payable

4. Perhatikan class **Owner**, diperlukan untuk apakah sintaks **p instanceof ElectricityBill** pada baris ke-6 ?
   Jawab : untuk mengecek apakah objek ‘p’(objek masukan) merupakan objek yang sama dengan class ElectricityBill
5. Perhatikan kembali class Owner baris ke-7, untuk apakah casting objek disana (**ElectricityBill eb = (ElectricityBill) p**) diperlukan ? Mengapa objek **p** yang bertipe **Payable** harus di-casting ke dalam objek **eb** yang bertipe **ElectricityBill ?**
   Jawab : untuk mencegah keambiguan objek maka ketika objek ‘p’ di-casting maka yang masuk merupakan objek dari class ElectricityBill dan bukan dari class interface Payable

5. TUGAS

- IDestroyable.java

```
1   package Tugas;
2
    public interface IDestroyable {
        public abstract void destroyed();
5   }
```

- Zombie.java

```java
package Tugas;

public abstract class Zombie implements IDestroyable{
    protected int health,level;
    public abstract void heal();
    @Override
    public abstract void destroyed();
    public String getZombieInfo(){
        return "Health = "+health+"\nLevel = "+level+"\n";
    }
}
```

- WalkingZombie.java

```java
package Tugas;

public class WalkingZombie extends Zombie{

    public WalkingZombie(int health,int level) {
        this.health=health;
        this.level=level;
    }

    @Override
    public void heal() {
        switch(level){
            case 1:health*=1.1;break;
            case 2:health*=1.3;break;
            case 3:health*=1.4;break;
        }
    }

    @Override
    public void destroyed() {
        health-=health*20/100;
    }

    public String getZombieInfo(){
        return "Walking Zombie Data =\n"+super.getZombieInfo();
    }

}
```

- JumpingZombie.java

```java
package Tugas;

public class JumpingZombie extends Zombie{

    public JumpingZombie(int health,int level) {
        this.health=health;
        this.level=level;
    }

    @Override
    public void heal() {
        switch(level){
            case 1:health*=1.3;break;
            case 2:health*=1.4;break;
            case 3:health*=1.5;break;
        }
    }

    @Override
    public void destroyed() {
        health-=health*10/100;
    }

    @Override
    public String getZombieInfo(){
        return "Jumping Zombie Data =\n"+super.getZombieInfo();
    }
}
```

- Barrier.java

```java
package Tugas;

public class Barrier implements IDestroyable{
    private int strength;

    public Barrier(int strength) {
        this.strength = strength;
    }

    public int getStrength() {
        return strength;
    }

    public void setStrength(int strength) {
        this.strength = strength;
    }

    @Override
    public void destroyed() {
        strength*=0.9;
    }

    public String getBarrierInfo(){
        return "Barrier Strength = "+strength+"\n";
    }

}
```

- Plant.java

```java
package Tugas;

public class Plant {
    public void doDestroy(IDestroyable d){
        if(d instanceof WalkingZombie){
            ((WalkingZombie) d).destroyed();
        }else if(d instanceof JumpingZombie){
            ((JumpingZombie) d).destroyed();
        }else if(d instanceof Barrier){
            ((Barrier)d).destroyed();
        }
    }
}
```

- Tester.java

```java
package Tugas;

public class Tester {
    public static void main(String[] args) {
        WalkingZombie wz=new WalkingZombie(100,1);
        JumpingZombie jz=new JumpingZombie(100,2);
        Barrier b=new Barrier(100);
        Plant p=new Plant();
        System.out.println(""+wz.getZombieInfo());
        System.out.println(""+jz.getZombieInfo());
        System.out.println(""+b.getBarrierInfo());
        System.out.println("----------------------");
        for(int i=0;i<4;i++){
            p.doDestroy(wz);
            p.doDestroy(jz);
            p.doDestroy(b);
        }
        System.out.println(""+wz.getZombieInfo());
        System.out.println(""+jz.getZombieInfo());
        System.out.println(""+b.getBarrierInfo());
    }

}
```

- Output

```
run:
Walking Zombie Data =
Health = 100
Level = 1

Jumping Zombie Data =
Health = 100
Level = 2

Barrier Strength = 100

----------------------
Walking Zombie Data =
Health = 42
Level = 1

Jumping Zombie Data =
Health = 66
Level = 2

Barrier Strength = 64

BUILD SUCCESSFUL (total time: 0 seconds)
```