Tugas Praktikum 10

Polimorfisme

Rafi Ody Prasetyo
(2341720180)

D-IV Teknik Informatika
Politeknik Negeri Malang
Semester 3
2024

Percobaan 1

Employee.java

```
public class Employee {

    protected String name;

    public String getEmployeeInfo() {
        return "Name = "+name;
    }

}
```

Payable.java

```
public interface Payable {

    public int getPaymentAmount();

}
```

InternshipEmployee.java

```
public class InternshipEmployee extends Employee {

    private int length;

    public InternshipEmployee(String name, int length) {
        this.length = length;
        this.name = name;
    }

    public int getLength() {
        return length;
    }

    public void setLength(int length) {
        this.length = length;
    }

    @Override
    public String getEmployeeInfo() {
        String info = super.getEmployeeInfo()+"\n";
        info += "Registered as internship employee for "+length+" months\n";
        return info;
```

PermanentEmployee.java

```java
public class PermanentEmployee extends Employee implements Payable{

    private int salary;

    public PermanentEmployee(String name, int salary) {
        this.name = name;
        this.salary = salary;
    }

    public int getSalary() {
        return salary;
    }

    public void setSalary(int salary) {
        this.salary = salary;
    }
    @Override
    public int getPaymentAmount() {
        return (int) (salary+0.05+salary);
    }
    @Override
    public String getEmployeeInfo() {
        String info = super.getEmployeeInfo()+"\n";
        info += "Registered as permanent employee with salary "+salary+"\n";
        return info;
    }

}
```

ElectricityBill.java

```java
public class ElectricityBill implements Payable{

    private int kwh;
    private String category;

    public ElectricityBill(int kwh, String category) {
        this.kwh = kwh;
        this.category = category;
    }

    public int getKwh() {
        return kwh;
    }

    public void setKwh(int kwh) {
        this.kwh = kwh;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }

    @Override
    public int getPaymentAmount() {
        return kwh*getBasePrice();
    }
    public int getBasePrice() {
        int bPrice = 0;
        switch (category) {
            case "R-1" : bPrice = 100;break;
            case "R-2" : bPrice = 200;break;
        }
        return bPrice;
    }

    public String getBillInfo() {
        return "KWH = "+kwh+"\n"+
               "Category = "+category+"("+getBasePrice()+"per KWH)\n";
    }


}
```

Pertanyaan:

1. Class apa sajakah yang merupakan turunan dari class Employee?
   Jawab: InternshipEmployee dan PermanentEmployee.
2. Class apa sajakah yang implements ke interface Payable?
   Jawab: ElectricityBill dan PermanentEmployee
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 subclass dari Employee
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 merupakan implements dari payable.
5. Coba tambahkan sintaks:

   ```
   p = iEmp;
   e = eBill;
   ```

   pada baris 14 dan 15 (baris terakhir dalam method main) ! Apa yang menyebabkan error?
   Jawab: Karena Payable pada class InternshipEmployee tidak ada keterikatannya, dan juga Employee tidak ada keterikatannya dengan class ElectricityBill.
6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!
   Jawab: Polimorfisme memungkinkan objek dari kelas yang berbeda untuk diperlakukan sebagai objek dari kelas yang sama.

Percobaan 2

Tester2.java

```
public class Tester2 {

    public static void main(String[] args) {
        PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
        Employee e;
        e = pEmp;
        System.out.println(""+e.getEmployeeInfo());
        System.out.println("------------------------");
        System.out.println(""+pEmp.getEmployeeInfo());
    }

}
```

Output:

```
Name = Dedik
Registered as permanent employee with salary 500

------------------------
Name = Dedik
Registered as permanent employee with salary 500
```

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 pada Tester2 terdapat deklarasi e = pEmp sehingga p dan pEmp akan memiliki value yang sama.
2. Mengapa pemanggilan method e.getEmployeeInfo() disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan pEmp.getEmployeeInfo() tidak?
   Jawab: e.getEmployeeInfo() disebut virtual karena pemilihan metode dilakukan saat runtime berdasarkan tipe objek aktual.
3. Jadi apakah yang dimaskud virtual method invocation? Mengapa disebut virtual?
   Jawab: Virtual Method Invocation adalah sebuah konsep dalam pemrograman berorientasi objek (OOP) di mana metode yang dipanggil pada suatu objek ditentukan pada saat runtime, bukan saat compile time. Konsep ini biasanya terjadi dalam kasus

polymorphism, di mana metode yang dipanggil bergantung pada tipe objek aktual (kelas turunan), meskipun objek tersebut diakses melalui referensi dari tipe induknya (kelas dasar).

Percobaan 3

Tester3.java

```
public class Tester3 {

    public static void main(String[] args) {
        PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
        InternshipEmployee iEmp = new InternshipEmployee("Sunarto", 5);
        ElectricityBill eBill = new ElectricityBill(5, "A-1");
        Employee e[] = {pEmp, iEmp};
        Payable p[] = {pEmp, eBill};
        Employee e2[] = {pEmp, iEmp, eBill};
    }

}
```

Pertanyaan

1. Perhatikan array e pada baris ke-8, mengapa ia bisa diisi dengan objek- objek dengan tipe yang berbeda, yaitu objek pEmp (objek dari PermanentEmployee) dan objek iEmp (objek dari InternshipEmployee) ?
   Jawab: e merupakan Employee sehingga jika diisi objek yang memiliki keterikatan dengan employee kode tidak akan error.
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: Sama seperti baris ke-8 objek pEmp dan eBill memiliki keterikatan dengan payable.
3. Perhatikan baris ke-10, mengapa terjadi error?
   Jawab: Karena eBill tidak memiliki keterikatan dengan e atau Employee.

Percobaan 4

Owner.java

```
public class Owner {

    public void pay(Payable p) {
        System.out.println("Total payment = "+p.getPaymentAmount());
        if (p instanceof ElectricityBill) {
            ElectricityBill eb = (ElectricityBill) p;
            System.out.println(""+eb.getBillInfo());
        } else if (p instanceof PermanentEmployee) {
            PermanentEmployee pe = (PermanentEmployee) p;
            pe.getEmployeeInfo();
            System.out.println(""+pe.getEmployeeInfo());
        }
    }

    public void showMyEmployee(Employee e) {
        System.out.println(""+e.getEmployeeInfo());
        if (e instanceof PermanentEmployee) {
            System.out.println("You have to pay her/him monthly!!");
        } else {
            System.out.println("No need to pay him/her");
        }
    }

}
```

Tester4.java

```
public class Tester4 {

    public static void main(String[] args) {
        Owner ow = new Owner();
        ElectricityBill eBill = new ElectricityBill(5, "R-1");
        ow.pay(eBill);
        System.out.println("-----------------------------------");
        PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
        ow.pay(pEmp);
        System.out.println("-----------------------------------");
        InternshipEmployee iEmp = new InternshipEmployee("Sunarto", 5);
        ow.showMyEmployee(pEmp);
        System.out.println("-----------------------------------");
        ow.showMyEmployee(iEmp);
    }
}
```

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 dalam class ElectricityBill dan PermanentEmployee mengimplementasikan Payable sehingga method pay() dapat dijalankan di eBill dan pEmp.
2. Jadi apakah tujuan membuat argument bertipe Payable pada method pay() yang ada di dalam class Owner?

   Jawab: Untuk memanggil method yang berkaitan dengan Payable tanpa harus dicasting Payable terlebih dahulu.
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
19          ow.pay(iEmp);
20      }
21  }
```

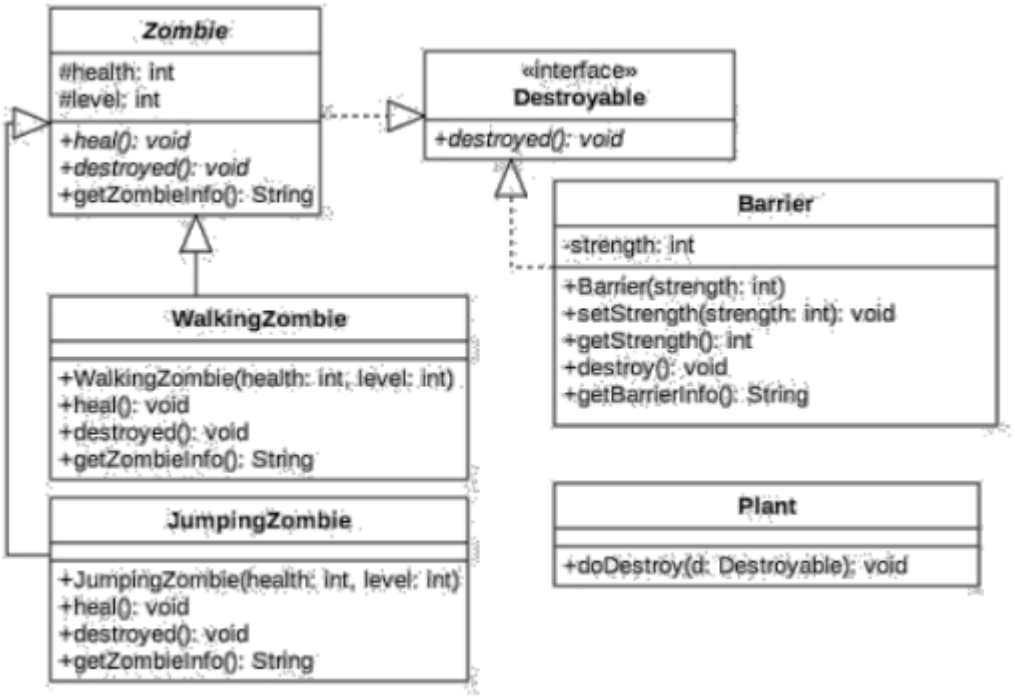
   Mengapa terjadi error?

   Jawab: Karena iEmp tidak berkaitan dengan Payable.
4. Perhatikan class Owner, diperlukan untuk apakah sintaks p instanceof ElectricityBill pada baris ke-6 ?

   Jawab: Untuk memastikan class ElectricityBill memiliki objek p.
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: Casting pada baris ElectricityBill eb = (ElectricityBill) p diperlukan karena objek p dideklarasikan sebagai tipe umum Payable, yang bisa merepresentasikan berbagai kelas (seperti ElectricityBill atau PermanentEmployee). Agar dapat mengakses metode atau atribut spesifik milik kelas ElectricityBill (misalnya getBillInfo()), objek tersebut harus di-cast terlebih dahulu ke tipe aslinya, yaitu ElectricityBill.

Tugas



Destroyable.java

```
package Tugas;

public interface Destroyable {

    void destroyed();

}
```

Zombie.java

```
package Tugas;

public class Zombie implements Destroyable{

    protected int health;
    protected int level;

    public void heal() {

    }

    @Override
    public void destroyed() {

    }

    public String getZombieInfo() {
        return ("health = "+health+"\n"+"level = "+level);
    }

}
```

Barrier.java

```
package Tugas;

public class Barrier implements Destroyable{

    private int strength;

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

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

    public int getStrength() {
        return strength;
    }

    @Override
    public void destroyed() {
```

```
        strength -= strength * (0.36);
        if (strength < 0) {
            strength = 0;
        }
    }

    public String getBarrierInfo() {
        return "Barrier Strength: "+ strength;
    }

}
```

WalkingZombie.java

```
package Tugas;
public class WalkingZombie extends Zombie {
    public WalkingZombie(int health, int level) {
        this.health = health;
        this.level = level;
    }

    @Override
    public void heal() {
        if (level == 1) {
            health += health * 0.20;
        } else if (level == 2) {
            health += health * 0.30;
        } else if (level == 3) {
            health += health * 0.40;
        }
    }

    @Override
    public void destroyed() {
        health -= health * 0.02;
        if (health < 0) {
            health = 0;
        }
    }

    @Override
    public String getZombieInfo() {
        return ("Walking Zombie Data = \nHealth = "+health+"\n"+"Level =
"+level);
    }
}
```

JumpingZombie.java

```
package Tugas;

public class JumpingZombie extends Zombie{

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

    @Override
    public void heal() {
        if (level == 1) {
            health += health * 0.30;
        } else if (level == 2) {
            health += health * 0.40;
        } else if (level == 3) {
            health += health * 0.50;
        }
    }

    @Override
    public void destroyed() {
        health -= health * 0.01;
        if (health < 0) {
            health = 0;
        }
    }

    @Override
    public String getZombieInfo() {
        return ("Jumping Zombie Data = \nHealth = "+health+"\n"+"Level =
"+level);
    }

}
```

Plant.java

```java
package Tugas;

public class Plant {

    public void doDestroy(Destroyable d) {
        if (d instanceof WalkingZombie) {
            WalkingZombie wb = (WalkingZombie) d;
            wb.destroyed();
        } else if (d instanceof JumpingZombie) {
            JumpingZombie jz = (JumpingZombie) d;
            jz.destroyed();
        } else if (d instanceof Barrier) {
            Barrier b = (Barrier) d;
            b.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:

```
Walking Zombie Data =
Health = 100
Level = 1
Jumping Zombie Data =
Health = 100
Level = 2
Barrier Strength: 100
------------------------------
Walking Zombie Data =
Health = 92
Level = 1
Jumping Zombie Data =
Health = 96
Level = 2
Barrier Strength: 16
```