OBJECT ORIENTED PROGRAMMING

By :

Sri Kresna Maha Dewa 2241720244

STUDY PROGRAM D-IV INFORMATIC ENGINEERING

INFORMATION TECHNOLOGY DEPARTMENT

MALANG STATE POLYTECHNIC

Soekarno Hatta Street No.9, Jatimulyo, Lowokwaru District, Malang City, East Java 65141

4.2. Pertanyaan

1. Class apa sajakah yang merupakan turunan dari class Employee?
   kelas PermanentEmployee dan kelas InternshipEmployee
2. Class apa sajakah yang implements ke interface Payable?
   kelas PermanentEmployee dan kelas 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) ?
   karena kelas PermanentEmployee dan kelas InternshipEmployee merupakan turunan dari kelas Employee dan kelas Employee implements ke interface Payable sehingga objek pEmp dan objek iEmp dapat diisi ke dalam variabel e yang bertipe Employee dan Payable
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) ?
   karena kelas PermanentEmployee dan kelas ElectricityBill implements ke interface Payable sehingga objek pEmp dan objek eBill dapat diisi ke dalam variabel p yang bertipe Payable
5. Coba tambahkan sintaks:
   p = iEmp;
   e = eBill;
   pada baris 14 dan 15 (baris terakhir dalam method main) ! Apa yang menyebabkan error?
   karena objek iEmp merupakan objek dari kelas InternshipEmployee yang tidak implements ke interface Payable sehingga objek iEmp tidak dapat diisi ke dalam variabel p yang bertipe Payable dan objek eBill merupakan objek dari kelas ElectricityBill yang tidak merupakan turunan dari kelas Employee sehingga objek eBill tidak dapat diisi ke dalam variabel e yang bertipe Employee
6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!
   Polimorfisme merupakan konsep dimana suatu objek dapat memiliki banyak bentuk, contohnya pada kelas Tester1, objek pEmp dan objek iEmp dapat diisi ke dalam variabel e yang bertipe Employee dan Payable, objek pEmp dan objek eBill dapat diisi ke dalam variabel p yang bertipe Payable, dan objek eBill tidak dapat diisi ke dalam variabel e yang bertipe Employee dan objek iEmp tidak dapat diisi ke dalam variabel p yang bertipe Payable

5.2. Pertanyaan

1. Perhatikan class Tester2 di atas, mengapa pemanggilan e.getEmployeeInfo() pada baris 8 dan pEmp.getEmployeeInfo() pada baris 10 menghasilkan hasil sama?
   karena variabel e bertipe Employee dan variabel pEmp bertipe PermanentEmployee, variabel e dapat diisi dengan objek pEmp sehingga variabel e dapat mengakses method getEmployeeInfo() yang terdapat pada kelas PermanentEmployee yang merupakan

turunan dari kelas Employee sehingga hasil pemanggilan method e.getEmployeeInfo() dan pEmp.getEmployeeInfo() sama

2. Mengapa pemanggilan method e.getEmployeeInfo() disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan pEmp.getEmployeeInfo() tidak?
   Pemanggilan method e.getEmployeeInfo() disebut sebagai pemanggilan metode virtual karena jenis objek yang sebenarnya yang akan menjalankan implementasi metode tersebut ditentukan secara dinamis pada waktu runtime. Pada saat kompilasi, tipe variabel e dideklarasikan sebagai tipe Employee, tetapi objek yang sebenarnya yang dikaitkan dengan variabel tersebut adalah objek dari kelas turunan, yaitu PermanentEmployee.
3. Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?
   Virtual method invocation adalah pemanggilan overriding method dari suatu objek polimorfisme. Disebut virtual karena antara method yang dikenali oleh compiler dan method yang dijalankan oleh JVM berbeda.

6.2. 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) ?
   karena kelas PermanentEmployee dan kelas InternshipEmployee merupakan turunan dari kelas Employee sehingga objek pEmp dan objek iEmp dapat diisi ke dalam array e yang bertipe Employee
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) ?
   karena kelas PermanentEmployee dan kelas ElectricityBill implements ke interface Payable sehingga objek pEmp dan objek eBill dapat diisi ke dalam array p yang bertipe Payable
3. Perhatikan baris ke-10, mengapa terjadi error?
   karena objek eBill merupakan objek dari kelas ElectricityBill yang tidak merupakan turunan dari kelas Employee sehingga objek eBill tidak dapat diisi ke dalam array e yang bertipe Employee

7.2. 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?
   karena kelas ElectricityBill dan kelas PermanentEmployee implements ke interface Payable sehingga objek eBill dan objek pEmp dapat diisi ke dalam variabel bertipe Payable sehingga dapat digunakan sebagai parameter pada method pay() yang ada di dalam class Owner

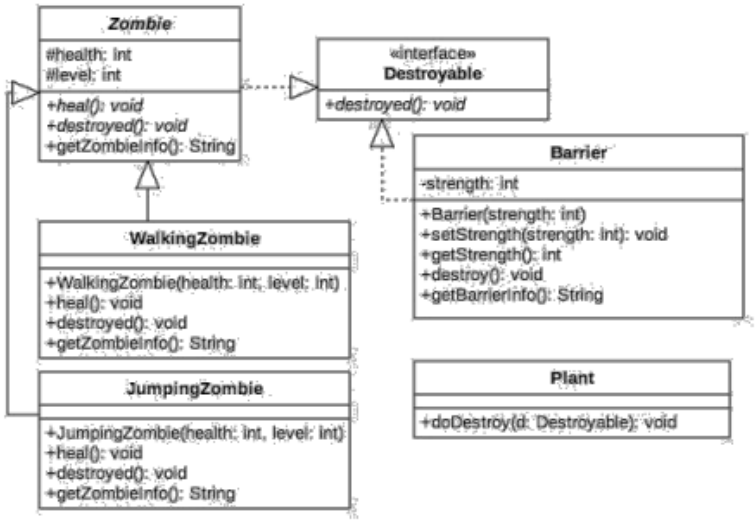
2. Jadi apakah tujuan membuat argument bertipe Payable pada method pay() yang ada di dalam class Owner?
   Tujuan membuat argument bertipe Payable pada method pay() yang ada di dalam class Owner adalah agar method pay() dapat menerima objek dari kelas ElectricityBill dan kelas PermanentEmployee yang merupakan turunan dari kelas Payable
3. Coba pada baris terakhir method main() yang ada di dalam class Tester4 ditambahkan perintah ow.pay(iEmp); Mengapa terjadi error?
   karena objek iEmp merupakan objek dari kelas InternshipEmployee yang tidak implements ke interface Payable sehingga objek iEmp tidak dapat diisi ke dalam variabel bertipe Payable sehingga tidak dapat digunakan sebagai parameter pada method pay() yang ada di dalam class Owner
4. Perhatikan class Owner, diperlukan untuk apakah sintaks p instanceof ElectricityBill pada baris ke-6 ?
   Sintaks p instanceof ElectricityBill pada baris ke-6 diperlukan untuk mengecek apakah objek yang diisi ke dalam variabel p merupakan objek dari kelas ElectricityBill atau bukan
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?
   Casting objek disana diperlukan agar variabel eb dapat mengakses method

TUGAS

Dalam suatu permainan, Zombie dan Barrier bisa dihancurkan oleh Plant dan bisa menyembuhkan diri. Terdapat dua jenis Zombie, yaitu Walking Zombie dan Jumping Zombie. Kedua Zombie tersebut memiliki cara penyembuhan yang berbeda, demikian juga cara penghancurannya, yaitu ditentukan oleh aturan berikut ini:

- Pada WalkingZombie
  - Penyembuhan : Penyembuhan ditentukan berdasar level zombie yang bersangkutan
    - Jika zombie level 1, maka setiap kali penyembuhan, health akan bertambah 10%
    - Jika zombie level 2, maka setiap kali penyembuhan, health akan bertambah 30%
    - Jika zombie level 3, maka setiap kali penyembuhan, health akan bertambah 40%
  - Penghancuran : setiap kali penghancuran, health akan berkurang 2%
- Pada Jumping Zombie
  - Penyembuhan : Penyembuhan ditentukan berdasar level zombie yang bersangkutan
    - Jika zombie level 1, maka setiap kali penyembuhan, health akan bertambah 30%
    - Jika zombie level 2, maka setiap kali penyembuhan, health akan bertambah 40%
    - Jika zombie level 3, maka setiap kali penyembuhan, health akan bertambah 50%
  - Penghancuran : setiap kali penghancuran, health akan berkurang 1%

Buat program dari class diagram di bawah ini!



CLASS :

```java
public class Zombie implements Destroyable{
    protected int health;
    protected int level;

    public void heal() {
        if (level == 1) {
            health += health * 0.1;
        } else if (level == 2) {
            health += health * 0.3;
        } else if (level == 3) {
            health += health * 0.4;
        }
    }

    public void destroyed() {
        health -= health * 0.02;
    }

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

```java
public class Barrier implements Destroyable{
    private int health;

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

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

    public int getStrength() {
        return health;
    }

    public void destroyed() {
        health -= health * 0.1;
    }

    public String getBarrierInfo() {
        return "\nBarrier Strength = " + health + "\n";
    }
}
```

```java
public interface Destroyable {
    public void destroyed();
}
```

```java
public class Plant {
    public void doDestroy(Destroyable d) {
        d.destroyed();
    }
}
```

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

    public void heal() {
        if (level == 1) {
            health += health * 0.1;
        } else if (level == 2) {
            health += health * 0.3;
        } else if (level == 3) {
            health += health * 0.4;
        }
    }

    public void destroyed() {
        health -= health * 0.19;
    }

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

```java
public class JumpingZombie extends Zombie {
    public JumpingZombie(int health, int level) {
        this.health = health;
        this.level = level;
    }

    public void heal() {
        if (level == 1) {
            health += health * 0.3;
        } else if (level == 2) {
            health += health * 0.4;
        } else if (level == 3) {
            health += health * 0.5;
        }
    }

    public void destroyed() {
        health -= health * 0.095;
    }

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

MAIN :

```java
public class Tester {
    Run | Debug
    public static void main(String[] args) {
        WalkingZombie wz = new WalkingZombie(health:100, level:1);
        JumpingZombie jz = new JumpingZombie(health:100, level:2);
        Barrier b = new Barrier(strength:100);
        Plant p = new Plant();
        System.out.println("" + wz.getZombieInfo());
        System.out.println("" + jz.getZombieInfo());
        System.out.println("" + b.getBarrierInfo());
        System.out.println(x:"--------------------------");
        for (int i = 0; i < 4; i++) {// destroying the enemies 4 times
            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 = 42
Level = 1


Jumping Zombie Data =
Health = 66
Level = 2


Barrier Strength = 64

PS D:\remote\code-for-edu>
```