JOBSHEET 10

POLIMORFISME

Hawa Esanda

2241720079_11_TI-2I

Percobaan 1 – Bentuk dasar polimorfisme

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 : PermanentEmployee dan 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 **PermanentEmployee** dan **IntershipEmployee** adalah turunan dari class **Employee**, dan objek dari class turunan dapat di assign ke variabel dengan tipe class induknya.

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 **PermanentEmployee** dan **ElectricityBill** implements interface **Payable**, objek dari class yang implements interface dapat diassign ke variabel dengan tipe interface tersebut.

5. Coba tambahkan sintaks:
   p = iEmp;
   e = eBill;
   pada baris 14 dan 15 (baris terakhir dalam method **main**) ! Apa yang menyebabkan error?
   Jawab : Karena **IntershipEmployee** tidak implements interface **Payable**, sehingga tidak dapat diassign ke variabel dengan tipe **Payable**.

6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!
   Jawab : Polimorfisme memungkinkan akses objek melalui referensi class induk atau interface yang sama. Contohnya, objek pEmp dari class PermanentEmployee dan iEmp dari class IntershipEmployee bisa diakses melalui variabel e dengan tipe Employee. Objek pEmp dan eBill juga bisa diakses melalui variabel p dengan tipe Payable.

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 variabel **e** diassign dengan objek **pEmp** yang merupakan instance dari class **PermanentEmployee**. Oleh karena itu, pemanggilan method **getEmployeeInfo()** pada variabel **e** akan merujuk pada implementasi method yang ada di class **PermanentEmployee**.

2. Mengapa pemanggilan method **e.getEmployeeInfo()** disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan **pEmp.getEmployeeInfo()** tidak?
   Jawab : Pemanggilan **e.getEmployeeInfo()** disebut virtual karena JVM menentukan implementasi method saat runtime berdasarkan objek sesungguhnya, meskipun tipe variabelnya **Employee**. Sebaliknya, pada **pEmp.getEmployeeInfo()**, JVM tidak perlu penentuan karena tipe variabel dan objek sesungguhnya sama, sehingga tidak disebut virtual.

3. Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?
   Jawab : Virtual method invocation adalah konsep di mana pemanggilan method terkait dengan objek aktual pada saat runtime, bukan hanya tipe variabelnya. Disebut virtual karena JVM, pada saat kompilasi, tidak mengetahui objek mana yang akan dioperasikan dengan pasti, dan penentuan implementasi method terjadi pada runtime berdasarkan objek yang sesungguhnya.

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 keduanya merupakan turunan dari class **Employee**. Dalam konsep polimorfisme, objek-objek tersebut dapat dianggap sebagai objek dari tipe **Employee**, yang merupakan tipe common dari keduanya.

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 keduanya mengimplementasikan interface **Payable**. Dalam hal ini, keduanya dapat dianggap sebagai objek dari tipe **Payable**, yang merupakan tipe common dari keduanya.

3. Perhatikan baris ke-10, mengapa terjadi error?
   Jawab : Karena array **e2** mencoba menggabungkan objek dari tipe yang berbeda, seperti
   pEmp dari class **PermanentEmployee**, **iEmp** dari class **IntershipEmployee**, dan **eBill** dari class **ElectricityBill**. Meskipun semuanya turunan dari class **Employee**, tidak semua objek dalam array **e2** mengimplementasikan interface **Payable**.

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 : Pemanggilan **ow.pay(eBill)** dan **ow.pay(pEmp)** dapat dilakukan karena objek **eBill** dari class **ElectricityBill** dan objek **pEmp** dari class **PermanentEmployee** keduanya mengimplementasikan interface **Payable**. Meskipun tipe deklarasi parameter pada method **pay()** adalah **Payable**, namun pada saat runtime, objek yang sesungguhnya akan menentukan implementasi method yang akan dipanggil.

2. Jadi apakah tujuan membuat argument bertipe **Payable** pada method **pay()** yang ada di dalam class **Owner**?
   Jawab : Dengan menggunakan tipe interface sebagai parameter, method **pay()** dapat menerima objek dari class-class yang mengimplementasikan interface **Payable**, memungkinkan pemanggilan dengan objek berbagai tipe yang sesuai.

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 : Jika ditambahkan perintah **ow.pay(iEmp);** akan terjadi error karena objek **iEmp** dari class **IntershipEmployee** tidak mengimplementasikan interface **Payable**. Sehingga, objek yang tidak sesuai dengan tipe parameter **Payable** tidak dapat digunakan pada method tersebut.

4. Perhatikan class **Owner**, diperlukan untuk apakah sintaks **p instanceof ElectricityBill** pada baris ke-6 ?
   Jawab : Digunakan untuk memeriksa apakah objek yang diterima oleh method **Pay** adalah instance dari 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 : Diperlukan karena pada saat kompilasi, tipe variabel **p** diketahui hanya sebagai **Payable**.

Karena jika ingin mengakses method atau atribut yang khusus ada pada class **ElectricityBill**, kita perlu melakukan casting untuk memberitahu kompiler bahwa objek tersebut sebenarnya adalah instance dari class **ElectricityBill**. Agar kita dapat mengakses method **getBillInfo()** yang spesifik hanya untuk objek dari class **ElectricityBill**.

Tugas

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

```
public class Zombie implements Destroyable {
    private int health;
    private int level;

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

    void heal() {
        int healingPercentage = (level == 1) ? 20 : (level
 == 2) ? 30 : 40;
        health += (health * healingPercentage) / 100;

        // Ensure health does not exceed 100
        health = Math.min(health, 100);
    }

    public void destroyed() {
        health -= (level == 1) ? 2 : (level == 2) ? 1 : 0;

        // Ensure health does not go below 0
        health = Math.max(health, 0);
    }

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

```
public class WalkingZombie extends Zombie {
    WalkingZombie(int health, int level) {
        super(health, level);
    }

    void heal() {
        super.heal();
    }

    public void destroyed() {
        super.destroyed();
    }

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

```
public class JumpingZombie extends Zombie{
    JumpingZombie(int health, int level) {
        super(health, level);
    }

    void heal() {
        super.heal();
    }

    public void destroyed() {
        super.destroyed();
    }

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

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

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

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

    int getStrength() {
        return strength;
    }

    public void destroyed() {
        strength -= 10;
    }

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

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

```java
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());
    }
}
```

```
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 = 60
```