Object Oriented Programming

Jobsheet 12

Polimorfisme

Name: Azahra Salsabila

NIM: 2241720073

Class: 2I

No: 05

Major: Information Technology

Study Program: Informatic Engineering

Experiment

1. **Bentuk dasar polimorfisme**
    - Employee

```java
package semester3.jobsheet12;
public class Employee {
    protected String name;

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

    - Payable

```java
package semester3.jobsheet12;
public interface Payable {
    public int getPaymentAmount();
```

    - InternshipEmployee

```java
package semester3.jobsheet12;
public class InternshipEmployee extends  Employee {
    private int length;

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

    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 + " month/s\n";
        return info;
    }

}
```

- PermanentEmployee

```java
package semester3.jobsheet12;
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
package semester3.jobsheet12;
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";
    }
}
```

- Tester1

```java
package semester3.jobsheet12;
public class Tester1 {
    Run | Debug
    public static void main(String[] args) {
        PermanentEmployee pEmp = new PermanentEmployee(name:"Dedik", salary:500);
        InternshipEmployee iEmp = new InternshipEmployee(name:"Sunarto", length:5);
        ElectricityBill eBill = new ElectricityBill(kwh:5, category:"A-1");
        Employee e;
        Payable p;
        e = pEmp;
        e = iEmp;
        p = pEmp;
        p = eBill;

    }
}
```

- Hasil run

```
PS D:\terserah\terserah>  & 'C:\Program Files\Java\jdk1.8.0_121\bin\java.exe' '-cp' 'D:\terserah\terserah\bin' 'semester3.jobsheet12.Tester1'
PS D:\terserah\terserah>
```

Questions

1) Class apa sajakah yang merupakan turunan dari class Employee?
   - PermanentEmployee
     InternshipEmployee
2) Class apa sajakah yang implements ke interface Payable?
   - PermanentEmployee
     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)?
   - Pada baris 10 dan 11, e bisa diisi dengan objek pEmp dan iEmp karena keduanya merupakan turunan dari kelas Employee. Dalam konsep polimorfisme, objek dari kelas turunan dapat diassign ke variabel bertipe kelas induk.
4) Perhatikan class Tester1, baris ke-12 dan 13. Mengapa p, bisa diisi denganobjekpEmp (merupakan objek dari class PermanentEmployee) dan objek eBill (merupakan objek dari class ElectricityBill)?
   - Pada baris 12 dan 13, p bisa diisi dengan objek pEmp dan eBill karena keduanya mengimplementasikan interface Payable. Dalam konsep polimorfisme, objek yang mengimplementasikan interface dapat diassign ke variabel bertipe interface.
5) Coba tambahkan sintaks: p = iEmp; e = eBill; pada baris 14 dan 15 (baris terakhir dalam method main) ! Apa yang menyebabkan error?
   - Menambahkan sintaks p = iEmp; dan e = eBill; pada baris 14 dan 15 akan menyebabkan error karena kelas InternshipEmployee tidak mengimplementasikan interface Payable, dan kelas ElectricityBill bukanlah turunan dari kelas Employee.
6) Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme
   - Polimorfisme memungkinkan penggunaan objek dari kelas turunan sebagai objek dari kelas induk atau melalui antarmuka yang diimplementasikan.
   - Objek dari kelas turunan dapat diassign ke variabel bertipe kelas induk atau antarmuka yang diimplementasikan oleh kelas tersebut.
   - Polimorfisme memungkinkan pemanggilan metode yang sesuai dengan tipe objek pada saat runtime.
   - Ini mempermudah penggunaan objek secara umum tanpa perlu mengetahui detail implementasi kelas turunan atau antarmuka yang digunakan.

2. Virtual method invocation

- Tester2

```java
package semester3.jobsheet12;
public class Tester2 {
    Run | Debug
    public static void main(String[] args) {
        PermanentEmployee pEmp = new PermanentEmployee(name:"Dedik", salary:500);
        Employee e;
        e = pEmp;
        System.out.println("" + e.getEmployeeInfo());
        System.out.println(x:"------------------------");
        System.out.println("" + pEmp + "\n" + pEmp.getEmployeeInfo());
    }
}
```

- Hasil run

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

------------------------
semester3.jobsheet12.PermanentEmployee@15db9742
Name = Dedik
Registered as permanent employee with salary 500
```

Questions

1) Perhatikan class Tester2 di atas, mengapa pemanggilan e.getEmployeeInfo() pada baris 8 dan pEmp.getEmployeeInfo() pada baris 10 menghasilkan hasil sama?
    - Pada class Tester2, pemanggilan e.getEmployeeInfo() pada baris 8 dan pEmp.getEmployeeInfo() pada baris 10 menghasilkan hasil yang sama karena variabel e dideklarasikan dengan tipe kelas induk Employee, tetapi diinisialisasi dengan objek dari kelas turunan PermanentEmployee. Dalam konsep polimorfisme, pemanggilan metode akan merujuk pada implementasi yang sesuai dengan tipe aktual objek pada saat runtime. Meskipun variabelnya dideklarasikan sebagai tipe kelas induk, metode yang dipanggil adalah metode dari kelas turunan karena objek yang sebenarnya adalah objek dari kelas turunan.
2) Mengapa pemanggilan method e.getEmployeeInfo() disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan pEmp.getEmployeeInfo() tidak?
    - Pemanggilan e.getEmployeeInfo() disebut sebagai pemanggilan metode virtual karena metode tersebut ditentukan oleh objek aktual yang ada pada saat runtime. Di sisi lain, pemanggilan pEmp.getEmployeeInfo() tidak disebut sebagai pemanggilan metode virtual karena metode tersebut ditentukan oleh tipe variabel yang dideklarasikan (PermanentEmployee) dan bukan oleh objek aktual yang ada pada saat runtime.

3) Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?
    - Virtual method invocation merujuk pada pemanggilan metode yang ditentukan oleh objek aktual pada saat runtime. Metode virtual memungkinkan kelas turunan untuk menyediakan implementasi yang berbeda untuk metode yang sama yang dideklarasikan di kelas induk atau antarmuka. Kata "virtual" digunakan untuk menunjukkan bahwa metode yang dipanggil dapat bervariasi tergantung pada objek yang sebenarnya, yang mungkin adalah objek dari kelas turunan yang berbeda.

3. Heterogenous Collection

- Tester3

```
1   package semester3.jobsheet12;
2   public class Tester3 {
        Run | Debug
3       public static void main(String[] args) {
4           PermanentEmployee pEmp = new PermanentEmployee(name:"Dedik", salary:500);
5           InternshipEmployee iEmp = new InternshipEmployee(name:"Sunarto", length:5);
6           ElectricityBill eBill = new ElectricityBill(kwh:5, category:"A-1");
7
8           Employee e[] = {pEmp, iEmp};
9           Payable p[] = {pEmp, eBill};
10          Employee e2[] = {pEmp, iEmp, eBill};
11      }
12  }
```

- Hasil run

```
PS D:\terserah\terserah>  & 'C:\Program Files\Java\jdk1.8.0_121\bin\java.exe' '
-cp' 'D:\terserah\terserah\bin' 'semester3.jobsheet12.Tester3'
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Type mismatch: cannot convert from ElectricityBill to Employee

        at semester3.jobsheet12.Tester3.main(Tester3.java:10)
```

Questions

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)?
    - Pada baris ke-8, array e dapat diisi dengan objek-objek yang memiliki tipe yang berbeda, yaitu objek pEmp (objek dari PermanentEmployee) dan objek iEmp (objek dari InternshipEmployee). Hal ini mungkin karena kelas PermanentEmployee dan InternshipEmployee keduanya merupakan turunan dari kelas Employee. Dalam konteks polimorfisme, objek dari kelas turunan dapat dianggap sebagai objek dari kelas induknya.
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)?
    - Pada baris ke-9, array p dapat diisi dengan objek-objek yang memiliki tipe yang berbeda, yaitu objek pEmp (objek dari PermanentEmployee)

dan objek eBill (objek dari ElectricityBilling). Hal ini mungkin karena kelas PermanentEmployee mengimplementasikan interface Payable, dan ElectricityBilling juga mengimplementasikan interface yang sama. Dalam konsep polimorfisme, objek yang mengimplementasikan interface yang sama dapat dianggap sebagai objek dari tipe tersebut.

3) Perhatikan baris ke-10, mengapa terjadi error
   - Pada baris ke-10, terjadi error karena array `e2` dideklarasikan sebagai array yang hanya dapat menampung objek dari kelas `Employee`. Ketiga objek yang dicoba dimasukkan ke dalam array `e2` adalah objek `pEmp` (objek dari `PermanentEmployee`), `iEmp` (objek dari `InternshipEmployee`), dan `eBill` (objek dari `ElectricityBilling`). Karena `eBill` bukanlah turunan dari `Employee`, maka terjadi error pada saat kompilasi.
   - Di bawah ini adalah code yang benar
     Employee e2[] = {pEmp, iEmp};

4. Argumen polimorfisme, instanceof dan casting objek

- Owner

```java
package semester3.jobsheet12;
public class Owner {
    public void pay(Payable p) {
        System.out.println("Toatal 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(x:"You have to pay her/him monthly!!!");
        } else {
            System.out.println(x:"No need to pay him/her :)");
        }
    }
}
```

- Tester4

```java
package semester3.jobsheet12;
public class Tester4 {
    Run | Debug
    public static void main(String[] args) {
        Owner ow = new Owner();
        ElectricityBill eBill = new ElectricityBill(kwh:5, category:"R-1");
        ow.pay(eBill);
        System.out.println(x:"------------------------");

        PermanentEmployee pEmp = new PermanentEmployee(name:"Dedik", salary:500);
        ow.pay(pEmp);
        System.out.println(x:"------------------------");

        InternshipEmployee iEmp = new InternshipEmployee(name:"Sunarto", length:5);
        ow.showMyEmployee(pEmp);
        System.out.println(x:"------------------------");
        ow.showMyEmployee(iEmp);
    }

}
```

- Hasil run

```
Toatal payment= 500
kWH = 5
Category = R-1(100 per kWH)

------------------------
Toatal payment= 525
Name = Dedik
Registered as permanent employee with salary 500

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

You have to pay her/him monthly!!!
------------------------
Name = Sunarto
Registered as internship employee for 5 month/s

No need to pay him/her :)
```

Questions

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?
    - Pada baris ke-7 dan baris ke-11, pemanggilan ow.pay(eBill) dan ow.pay(pEmp) dapat dilakukan karena objek eBill dan pEmp keduanya mengimplementasikan interface Payable. Meskipun eBill adalah objek dari ElectricityBill dan pEmp adalah objek dari PermanentEmployee,

keduanya dapat dioperasikan dalam method pay() karena keduanya memiliki tipe yang sesuai dengan parameter Payable.

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() di dalam class Owner adalah untuk menerima objek dari kelas-kelas yang mengimplementasikan interface Payable. Dengan menggunakan tipe Payable, class Owner dapat bekerja dengan objek dari berbagai kelas yang memiliki kemampuan untuk melakukan pembayaran (Payable), tanpa perlu mengetahui detail implementasi kelas tersebut.
3) Coba pada baris terakhir method main() yang ada di dalam class Tester4 ditambahkan perintah ow.pay(iEmp); Mengapa terjadi error?
   - Pada baris terakhir method main() di dalam class Tester4, jika ditambahkan perintah ow.pay(iEmp); akan menghasilkan error karena objek iEmp merupakan objek dari kelas InternshipEmployee yang tidak mengimplementasikan interface Payable. Oleh karena itu, objek tersebut tidak dapat diterima sebagai argumen untuk method pay(Payable p).
4) Perhatikan class Owner, diperlukan untuk apakah sintaks p instanceof ElectricityBill pada baris ke-6?
   - Pada baris ke-6 di dalam class Owner, sintaks p instanceof ElectricityBill digunakan untuk mengecek apakah objek yang dioperasikan (p) merupakan instance dari kelas ElectricityBill. Ini dilakukan untuk menghindari kesalahan saat mencoba melakukan cast objek ke kelas yang tidak sesuai.
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?
   - Pada baris ke-7 di dalam class Owner, casting objek (ElectricityBill) p diperlukan karena pada baris ke-6 telah dilakukan pengecekan menggunakan instanceof untuk memastikan bahwa objek yang dioperasikan (p) adalah instance dari ElectricityBill. Sebagai hasilnya, kita dapat melakukan casting dengan aman karena kita sudah tahu bahwa objek tersebut adalah instance dari ElectricityBill. Casting dilakukan agar kita dapat mengakses metode atau properti khusus yang ada di kelas ElectricityBill.

Assignment

1. Screenshot program code

```java
package semester3.jobsheet12;
public abstract class Zombie {
    protected int health, level;
    public abstract void heal();
    public abstract void destroyed();
    public String getZombieInfo() {
        return "Health = " + health + "\nLevel = " + level;
    }
}
```

```java
package semester3.jobsheet12;
public class Plant {
    public void doDestroy(Destroyable d) {
        if (d instanceof WalkingZombie) {
            WalkingZombie wz = (WalkingZombie) d;
            wz.destroyed();
        } else if (d instanceof JumpingZombie) {
            JumpingZombie jz = (JumpingZombie) d;
            jz.destroyed();
        } else if (d instanceof Barrier) {
            Barrier b = (Barrier) d;
            b.destroyed();
        }
    }
}
```

```java
package semester3.jobsheet12;
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 this.strength;
    }

    @Override
    public void destroyed() {
        strength -= strength * 0.1;
    }

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

```java
package semester3.jobsheet12;
public class JumpingZombie extends Zombie implements Destroyable {

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

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

    @Override
    public void destroyed() {
        health -= (int) (health * 0.15);
    }

    @Override
    public String getZombieInfo() {
        System.out.println(x:"Jumping Zombie Data");
        return super.getZombieInfo();
    }
}
```

```java
package semester3.jobsheet12;
public class WalkingZombie extends Zombie implements Destroyable {

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

    @Override
    public void heal() {
        if (level == 1) {
            health += (int) (health * 0.2);
        } else if (level == 2) {
            health += (int) (health * 0.3);
        } else if (level == 3) {
            health += (int) (health * 0.4);
        }
    }

    @Override
    public void destroyed() {
        health -= health * 0.2;
    }

    @Override
    public String getZombieInfo() {
        System.out.println(x:"Walking Zombie Data");
        return super.getZombieInfo();
    }
}
```

```java
package semester3.jobsheet12;
public interface Destroyable {
    void destroyed();
}
```

```java
package semester3.jobsheet12;
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(x:"====================================");
        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++) {
            p.doDestroy(wz);
            p.doDestroy(jz);
            p.doDestroy(b);
        }
        System.out.println(x:"====================================");
        System.out.println("" + wz.getZombieInfo());
        System.out.println("" + jz.getZombieInfo());
        System.out.println("" + b.getBarrierInfo());
    }
}
```

Screenshot run java

```
====================================
Walking Zombie Data
Health = 100
Level = 1
Jumping Zombie Data
Health = 100
Level = 2
Barrier Strength = 100

 -
====================================
Walking Zombie Data
Health = 40
Level = 1
Jumping Zombie Data
Health = 54
Level = 2
Barrier Strength = 64
```