LAPORAN PRAKTIKUM

OOP

JOBSHEET 12

Disusun oleh:

Pascalis Dewangga S. L. 2241720140

TI - 2D

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2023

1. Percobaan 1
    a. Kode Program

Class Employee

```
public class Employee{
    protected String name;

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

}
```

Interface Payable

```
public interface Payable {
    public int getPaymentAmount();
}
```

Class InternshipEmployee

```
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;
    }

}
```

Class PermanentEmployee

```
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;
    }

}
```

Class ElectricityBill

```
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";
    }
}
```

Class Tester1

```
public class Tester1 {
    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;
        Payable p;
        e = pEmp;
        e = iEmp;
        p = pEmp;
        p = eBill;
    }
}
```

b. Hasil
Tidak memberikan output apapun

c. Pertanyaan
1. Class apa sajakah yang merupakan turunan dari class Employee?
Jawab: Class **InternshipEmployee** dan **PermanentEmployee**

2. Class apa sajakah yang implements ke interface Payable?
   Jawab: Class **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 **pEmp** dan **iEmp** merupakan object sub-class dari class **Employee**.
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) ?
   Jawab: Karena **pEmp**dan **eBill** merupakan object yang implement ke class **Payable**.
5. Coba tambahkan sintaks: p = iEmp; e = eBill; pada baris 14 dan 15 (baris terakhir dalam method main) ! Apa yang menyebabkan error?
   Jawab: class **iEmp** dan **eBill** masing-masing tidak berhubungan dengan class yang menjadi wadah penampung nilai objek.
6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!
   Jawab: polimorfisme bisa terjadi hanya jika kelas satu dengan yang lainnya saling berhubungan, yang dibentuk dari adanya hubungan sub-class atau pengimplementasian terhadap class lain.

2. Percobaan 2
   a. Kode Program

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

   b. Hasil

```
Name: dedik
Registered as permanent employee with salary 500

--------------------
Name: dedik
Registered as permanent employee with salary 500
```

   c. 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 dasarnya keduanya memanggil fungsi yang sama dimana fungsi tersebut telah dioverride pada class **PermanentEmployee**.

2. Mengapa pemanggilan method e.getEmployeeInfo() disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan pEmp.getEmployeeInfo() tidak?
   Jawab: Karena objek **e** memanggil method tersebut secara tidak langsung, alias memanggilnya dari class **PermanentEmployee**. Sedangkan objek **pEmp** adalah pemilik dari method itu sendiri sehingga bisa langsung memanggil method tersebut. Hal itu terjadi dikarenakan method **getEmployeeInfo()** sudah dioverride pada class **PermanentEmployee**
3. Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?
   Jawab: Merupakan pemanggilan method secara tidak langsung. Disebut virtual karena objek tersebut harus memanggilnya dari kelas yang berhubungan.

3. Percobaan 3
   a. Kode Program

```
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};
    }
}
```

   b. Hasil
      Tidak memberikan output apapun karena ada error.
   c. 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: Karena kedua objek yang berbeda tersebut masih berhubungan dengan class **Employee** sebagai parent class dari objek-objek tersebut.
      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 berimplementasi ke class **Payable** sehingga masih terdapat hubungan antara ketiganya.
      3. Perhatikan baris ke-10, mengapa terjadi error?
         Jawab: objek **eBill** tidak memiliki hubungan apapun dengan class **Employee** sehingga tidak dapat diisi dengan nilai yang ada pada objek **eBill**.
4. Percobaan 4
   a. Kode Program

```
public class Tester4 {
    public static void main(String[] args) {
        Owner ow = new Owner();
```

```
        ElectricityBill eBill = new ElectricityBill(5, "R-2");
        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);
    }
}
```

b. Hasil

```
Total payment = 1000
kWH = 5
Category = R-2(200 per kWH)


-----------------------------------------------
Total 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 months

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

c. 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 kedua objek tersebut berimplementasi ke class **Payable** sehingga masih memiliki hubungan satu sama lain.
2. Jadi apakah tujuan membuat argument bertipe Payable pada method pay() yang ada di dalam class Owner?
   Jawab: Supaya class **Owner** dapat memanggil method yang terdapat pada class **Payable**.
3. Coba pada baris terakhir method main() yang ada di dalam class Tester4 ditambahkan perintah ow.pay(iEmp); Mengapa terjadi error?
   Jawab: Karena objek **iEmp** tidak memiliki hubungan sama sekali dengan class **Payable** sehingga method tersebut tidak dapat diaplikasikan.
4. Perhatikan class Owner, diperlukan untuk apakah sintaks p instanceof ElectricityBill pada baris ke-6 ?

Jawab: Untuk menentukan method tersebut akan menjalankan method yang berhubungan dengan **Payable** dari objek yang mana. Pada kasus diatas maka method yang akan dijalankan adalah method 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: Casting ini memungkinkan program untuk memperlakukan **p** sebagai objek **ElectricityBill** sementara dalam lingkup blok if.

5. Tugas
   Kode Program:
   Class Zombie

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

    public void heal() {
    }

    public void destroyed() {
    }

    public String getZombieInfo() {
        return "Health= " + health + " Level= " + level;
    }
}
```

Class WalkingZombie

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

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

    public void destroyed() {
        health -= (health * 2/10);
    }
```

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

Class JumpingZombie

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

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

    public void destroyed() {
        health -= (health * 1/10);
    }

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

Interface Destroyable

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

Class Barrier

```
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 * 0.1);
    }

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

Class Plant

```
public class Plant {
    public void doDestroy(Destroyable d){
        if (d instanceof Zombie) {
            Zombie z = (Zombie) d;
            z.destroyed();
        } else if (d instanceof Barrier) {
            Barrier b = (Barrier) d;
            d.destroyed();
        }
    }
}
```

Class Tester

```
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());

    }
}
```

Hasil

```
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
```