LAPORAN HASIL PRAKTIKUM 10

PEMROMGRAMAN BERBASIS OBJEK

ATHAULLA HAFIZH

244107020030

TI 2A

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2025

4. Percobaan 1 – Bentuk dasar polimorfisme

Class Employee

```
public class Employee {
    protected String name;

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

Class Payable

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

Class InternshipEmployee

```
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 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);
    }
```

```
        ani.ajakPeliharaanJalanJalan();
        budi.ajakPeliharaanJalanJalan();
    }
}
```

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);
        // Menggunakan "R-1" agar sesuai logika getBasePrice()
        ElectricityBill eBill = new ElectricityBill(5, "R-1");
        Employee e;
        Payable p;
```

```
        e = pEmp;
        e = iEmp;

        p = pEmp;
        p = eBill;
    }
}
```

Pertanyaan

1. **Class turunan Employee:** InternshipEmployee dan PermanentEmployee.
2. **Class implements Payable:** PermanentEmployee dan ElectricityBill.
3. **Mengapa e bisa diisi pEmp dan iEmp?** Karena e adalah variabel referensi bertipe Employee (sebuah *super class*), dan pEmp (objek PermanentEmployee) serta iEmp (objek InternshipEmployee) adalah objek dari *sub class* (turunan) Employee.
4. **Mengapa p bisa diisi pEmp dan eBill?** Karena p adalah variabel referensi bertipe Payable (sebuah *interface*), dan pEmp (objek PermanentEmployee) serta eBill (objek ElectricityBill) adalah objek dari class yang *implements* (menerapkan) interface Payable.
5. **Error saat menambah p = iEmp; dan e = eBill;?**
   p = iEmp; **error** karena iEmp adalah objek InternshipEmployee, dan class InternshipEmployee **tidak implements** interface Payable.
   e = eBill; **error** karena eBill adalah objek ElectricityBill, dan class ElectricityBill **bukan turunan** (sub class) dari class Employee.
6. **Kesimpulan konsep polimorfisme:** Polimorfisme adalah kemampuan suatu variabel referensi untuk merujuk ke objek yang memiliki berbagai bentuk (berasal dari *sub class* yang berbeda atau *class implementasi* yang berbeda). Sebuah variabel *super class* bisa merujuk ke objek *sub class*-nya, dan sebuah variabel *interface* bisa merujuk ke objek dari class manapun yang meng-implementasikannya.

5. Percobaan 2 – Virtual method invocation

Class Tester2

```
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. Mengapa e.getEmployeeInfo() dan pEmp.getEmployeeInfo() hasilnya sama?
   Keduanya menghasilkan output yang sama karena meskipun variabel e bertipe Employee, ia sedang merujuk ke objek pEmp (yang bertipe PermanentEmployee). Saat *run time*, Java menjalankan method getEmployeeInfo() yang ada di objek aslinya (PermanentEmployee), bukan yang ada di tipe referensinya (Employee). pEmp.getEmployeeInfo() juga jelas memanggil method dari PermanentEmployee.

2. Mengapa e.getEmployeeInfo() disebut virtual, tapi pEmp.getEmployeeInfo() tidak?
   - e.getEmployeeInfo() disebut *virtual* karena ada perbedaan antara apa yang *compiler* lihat dan apa yang *JVM* (Java Virtual Machine) jalankan. Saat *compile*

time, compiler hanya tahu e adalah Employee. Saat *run time*, JVM mengecek objek aslinya (yaitu PermanentEmployee) dan "secara virtual" memanggil method getEmployeeInfo() milik PermanentEmployee.

- pEmp.getEmployeeInfo() tidak virtual karena tipe referensi (PermanentEmployee) dan tipe objeknya (PermanentEmployee) sama. Compiler dan JVM sama-sama tahu method yang dipanggil adalah milik PermanentEmployee.

3. Apa itu *virtual method invocation* dan mengapa disebut virtual? *Virtual method invocation* adalah mekanisme pemanggilan *overriding method* (method yang di-override di sub class) melalui referensi *super class*. Disebut "virtual" karena method yang *sebenarnya* akan dijalankan tidak ditentukan saat *compile time* (oleh compiler), melainkan baru ditentukan saat *run time* (oleh JVM) berdasarkan objek aslinya.

6. Percobaan 3 – Heterogenous Collection

Class Tester3

```
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, "R-1");

        // Array heterogen berbasis Super Class
        Employee e[] = { pEmp, iEmp };

        // Array heterogen berbasis Interface
        Payable p[] = { pEmp, eBill };

        // Baris ini akan error jika diaktifkan
        // Employee e2[] = { pEmp, iEmp, eBill };
    }
}
```

Pertanyaan

1. Mengapa array e bisa diisi pEmp dan iEmp? Karena array e dideklarasikan bertipe Employee[]. Ini adalah *collection heterogen* yang bisa menampung objek apapun yang merupakan turunan (sub class) dari Employee. pEmp (PermanentEmployee) dan iEmp (InternshipEmployee) keduanya adalah turunan Employee.

2. Mengapa array p bisa diisi pEmp dan eBill? Karena array p dideklarasikan bertipe Payable[]. Ini adalah *collection heterogen* yang bisa menampung objek apapun dari class yang meng-implementasikan interface Payable. pEmp (PermanentEmployee) dan eBill (ElectricityBill) keduanya meng-implementasikan Payable.

3. Mengapa baris 10 (Employee e2[] = {pEmp, iEmp, eBill};) error? Error terjadi karena eBill (objek ElectricityBill) bukan turunan (sub class) dari Employee. Array Employee[] hanya bisa menampung objek yang "IS-A" (adalah seorang) Employee.

7. Percobaan 4 – Argumen polimorfisme, instanceof, dan casting

Class Owner

```
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;
            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 :)");
    }
}
```

Class Tester4

```
public class Tester4 {
    public static void main(String[] args) {
        Owner ow = new Owner();
        ElectricityBill eBill = new ElectricityBill(5, "R-1");
        ow.pay(eBill);// bayar tagihan listrik
        System.out.println("-------------------------------");

        PermanentEmployee pEmp = new PermanentEmployee("Dedik",
500);
        ow.pay(pEmp); // bayar gaji pegawai tetap
        System.out.println("-------------------------------");

        InternshipEmployee iEmp = new InternshipEmployee("Sunarto",
5);
        ow.showMyEmployee(pEmp);// tampilkan info pegawai tetap
        System.out.println("-------------------------------");

        ow.showMyEmployee(iEmp);// tampilkan info pegawai magang

        // Baris ini akan error jika diaktifkan
        // ow.pay(iEmp);
    }
}
```

Pertanyaan

1. Mengapa ow.pay(eBill) dan ow.pay(pEmp) bisa dilakukan padahal parameter pay() adalah Payable? Ini adalah konsep argumen polimorfisme. Method pay() didefinisikan untuk menerima parameter apapun yang bertipe Payable. Karena eBill (objek ElectricityBill) dan pEmp (objek PermanentEmployee) keduanya berasal dari class yang meng-implementasikan Payable, keduanya valid untuk dijadikan argumen.

2. Tujuan argumen Payable di method pay()? Tujuannya adalah untuk membuat method pay() fleksibel. Owner tidak perlu tahu detail spesifik dari apa yang dia bayar (apakah itu tagihan listrik, gaji, atau tagihan lainnya). Selama objek tersebut "bisa dibayar" (meng-implementasikan Payable dan punya method getPaymentAmount()), method pay() bisa memprosesnya.

3. Mengapa ow.pay(iEmp); error? Error terjadi karena iEmp adalah objek InternshipEmployee, dan class InternshipEmployee tidak meng-implementasikan interface Payable.

4. Kegunaan p instanceof ElectricityBill? Operator instanceof digunakan untuk mengecek tipe objek asli saat *run time*. Perlu dicek apakah variabel p (yang tipenya Payable) pada kenyataannya adalah sebuah objek ElectricityBill sebelum mencoba memanggil method spesifik ElectricityBill (seperti getBillInfo()). Ini untuk mencegah ClassCastException (error casting).

5. Kegunaan *casting* (ElectricityBill) p? *Casting* (secara spesifik *downcasting*) diperlukan untuk mengubah tipe referensi variabel p dari Payable kembali menjadi ElectricityBill. Tujuannya adalah agar kita bisa mengakses method yang hanya ada di ElectricityBill (yaitu getBillInfo()), yang tidak dikenal di dalam interface Payable.

Tugas

Class Interface Destroyable

```
package Tugas;


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

Class Zombie

```
package Tugas;


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

    public abstract void heal();

    @Override
    public abstract void destroyed();

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

Class WalkingZombie

```
    }

    @Override
    public void heal() {
        switch (level) {
            case 1:
                this.health += (this.health * 0.1); // 10%
                break;
            case 2:
                this.health += (this.health * 0.3); // 30%
                break;
            case 3:
                this.health += (this.health * 0.4); // 40%
                break;
        }
    }

    @Override
    public void destroyed() {
        this.health -= (this.health * 0.20);
    }

    @Override
    public String getZombieInfo() {
        return "Walking Zombie Data =" + super.getZombieInfo();
    }
}
```

Class JumpingZombie

```
        this.health = health;
        this.level = level;
    }

    @Override
    public void heal() {
        switch (level) {
            case 1:
                this.health += (this.health * 0.3); // 30%
                break;
            case 2:
                this.health += (this.health * 0.4); // 40%
                break;
            case 3:
                this.health += (this.health * 0.5); // 50%
                break;
        }
    }

    @Override
    public void destroyed() {
        this.health -= (this.health * 0.10);
    }

    @Override
    public String getZombieInfo() {
        return "Jumping Zombie Data =" + super.getZombieInfo();
    }
}
```

Class Barrier

```
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() {
        this.strength -= (this.strength * 0.10);
    }

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

Class Plant

```
package Tugas;


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

Class Tester

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