# OBJECT ORIENTED PROGRAM PRACTICUM

## JOBSHEET 12 : Polimorfisme


BY :

**ATHRIYA GENFERIN**

**D4 INFORMATICS ENGINEERING (2I)**

**2241720075**

**(03)**

## State Polytechnic of Malang

**Soekarno Hatta street No.9, Malang, East Java 65141**

**2022/2023**

ATHRIYA GENFERIN
2I – 03 - 2241720075
D-IV INFORMATICS ENGINEERING

# Praktikum 1

```
package jobsheet12;

public class Employee {
    protected String name;

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

package 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";
    }
}

package jobsheet12;

public class IntershipEmployee extends Employee {
    private int length;

    public IntershipEmployee(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 intership employee for " + length + "
month/s\n";
        return info;
    }

}

package jobsheet12;
```

ATHRIYA GENFERIN
2I – 03 - 2241720075
D-IV INFORMATICS ENGINEERING

```
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;
            pe.getEmployeeInfo();
            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 :)");
        }
    }

}
```

```
package jobsheet12;

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

```
package 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;
    }
}

package jobsheet12;

public class Tester1 {
    public static void main(String[] args) {
        PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
        IntershipEmployee iEmp = new IntershipEmployee("Sunarto", 5);
        ElectricityBill eBill = new ElectricityBill(5, "A-1");
        Employee e;
        Payable p;
        e = pEmp;
        e = iEmp;
        p = pEmp;
        p = eBill;

    }
}
```

**Pertanyaan**

1. **Class apa sajakah yang merupakan turunan dari class Employee?**
   ElectricityBill
   IntershipEmployee
   PermanentEmployee
2. **Class apa sajakah yang implements ke interface Payable?**
   ElectricityBill
   PermanentEmployee
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 keduanya adalah turunan dari class Employee. Dalam konsep polimorfisme, objek dari subclass dapat diassign ke variabel superclass

ATHRIYA GENFERIN
2I – 03 - 2241720075
D-IV INFORMATICS ENGINEERING

4. **Perhatikan class Tester1, baris ke-12 dan 13. Mengapa p, bisa diisi dengan objekpEmp (merupakan objek dari class PermanentEmployee) dan objek eBill (merupakan objek dari class ElectricityBill) ?**
   karena keduanya implementasi dari interface Payable. Dalam konsep polimorfisme, objek yang mengimplementasikan suatu 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?**
   akan terjadi error karena IntershipEmployee tidak mengimplementasikan interface Payable. Begitu juga dengan e = eBill; pada baris 15, karena ElectricityBill bukan merupakan turunan dari class Employee.
6. **Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!**
   Polimorfisme memungkinkan objek untuk mengambil banyak bentuk, terutama ketika menggunakan referensi superclass untuk merujuk objek subclass. Hal ini terutama berlaku pada class-class dengan hubungan warisan (inheritance) atau IS-A. Selain itu, polimorfisme juga berlaku pada interface, di mana objek yang dideklarasikan sebagai suatu interface dapat merujuk ke objek dari class yang mengimplementasikan interface tersebut.

# Praktikum 2

```
package jobsheet12;

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 + "\n" + pEmp.getEmployeeInfo());
    }
}
```

ATHRIYA GENFERIN
2I – 03 - 2241720075
D-IV INFORMATICS ENGINEERING

```
PROBLEMS 6   TERMINAL   ...   Run: Tester2

Roaming\Code\User\workspaceStorage\22ef0bb066b2e04bb7720e
106b736e70\redhat.java\jdt_ws\OOP_71cf1347\bin' 'jobsheet
12.Tester2'
Name = Dedik
Registered as permanent employee with salary 500

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

PS E:\SEMESTER 3\OOP\OOP>
```

**Pertanyaan**

1. **Perhatikan class Tester2 di atas, mengapa pemanggilane.getEmployeeInfo() pada baris 8 dan pEmp.getEmployeeInfo() pada baris 10 menghasilkan hasil sama?**
   karena pada baris 8, pemanggilan e.getEmployeeInfo() menghasilkan hasil yang sama dengan pEmp.getEmployeeInfo() pada baris 10 karena e adalah referensi dari objek pEmp yang merupakan instance dari class PermanentEmployee. Ini adalah contoh polimorfisme, di mana pemanggilan metode superclass (Employee) merujuk pada metode subclass (PermanentEmployee).

2. **Mengapa pemanggilan method e.getEmployeeInfo() disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan pEmp.getEmployeeInfo() tidak?**
   Karena pemanggilan metode menjadi "virtual" ketika kita menggunakan referensi superclass untuk merujuk ke objek subclass. Pemanggilan e.getEmployeeInfo() disebut sebagai pemanggilan method virtual karena e adalah referensi superclass (Employee) yang diarahkan pada objek subclass (PermanentEmployee).

   pEmp.getEmployeeInfo() pada dasarnya memiliki sifat virtual yang sama, tetapi "virtual" sering kali tidak diakui karena referensi pEmp langsung menunjuk ke objek PermanentEmployee.

3. **Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?**
   Virtual method invocation terjadi ketika ada pemanggilan overriding method dari suatu objek polimorfisme. Disebut virtual karena antara method yang dikenali oleh compiler dan method yang dijalankan oleh JVM berbeda.

# Praktikum 3

```java
package jobsheet12;

public class Tester3 {
    public static void main(String[] args) {
```

```
        PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
        IntershipEmployee iEmp = new IntershipEmployee("Sunarto", 5);
        ElectricityBill eBill = new ElectricityBill(5, "A-1");
        Employee e[] = { pEmp, iEmp };
        Payable p[] = { pEmp, eBill };
        Employee e2[] = { pEmp, iEmp, eBill };
    }
}
```

**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 keduanya adalah turunan dari class Employee. Dalam konsep polimorfisme, array atau koleksi dapat menyimpan objek dari subclass menggunakan referensi superclass.

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 keduanya mengimplementasikan interface Payable. Dalam konsep polimorfisme, array atau koleksi dapat menyimpan objek yang mengimplementasikan suatu interface menggunakan referensi interface tersebut.

3. **Perhatikan baris ke-10, mengapa terjadi error?**
   Terjadi error pada baris ke-10 karena array e2 mencoba menyimpan objek dengan tipe yang berbeda-beda, yaitu pEmp (objek dari PermanentEmployee), iEmp (objek dari IntershipEmployee), dan eBill (objek dari ElectricityBill).

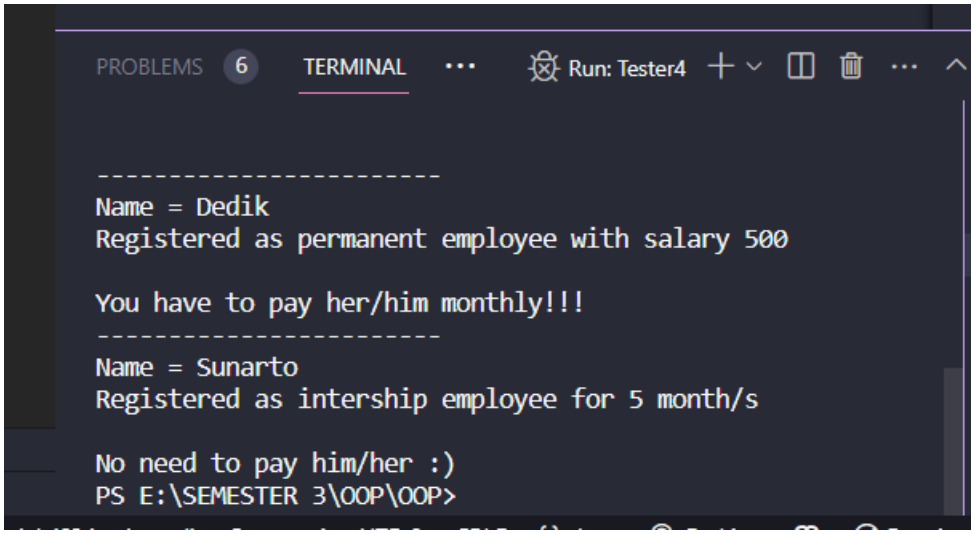
   Meskipun semuanya adalah turunan dari Employee, tetapi pada saat compile time, array tersebut membutuhkan satu tipe data yang spesifik. Jadi, kita tidak dapat menyimpan objek dengan tipe yang berbeda di dalam array dengan cara tersebut

# Praktikum 4

```
package jobsheet12;

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

ATHRIYA GENFERIN
2I – 03 - 2241720075
D-IV INFORMATICS ENGINEERING

```
        PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
        ow.pay(pEmp);
        System.out.println("------------------------");

        IntershipEmployee iEmp = new IntershipEmployee("Sunarto", 5);
        ow.showMyEmployee(pEmp);
        System.out.println("------------------------");
        ow.showMyEmployee(iEmp);
    }

}
```




**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?**
   Keduanya dapat dilakukan karena eBill dan pEmp mengimplementasikan interface Payable, memungkinkan pemrosesan objek dengan tipe yang berbeda tetapi mengimplementasikan interface yang sama.

2. **Jika diperhatikan lebih detil eBill merupakan objek dari ElectricityBill dan pEmp merupakan objek dari PermanentEmployee?**
   **Jadi apakah tujuan membuat argument bertipe Payable pada method pay() yang ada di dalam class Owner?**
   Menerima objek dengan tipe yang berbeda tetapi mengimplementasikan Payable, memberikan fleksibilitas pada Owner untuk memproses pembayaran untuk berbagai jenis objek.

ATHRIYA GENFERIN
2I – 03 - 2241720075
D-IV INFORMATICS ENGINEERING

3. **Coba pada baris terakhir method main() yang ada di dalam class Tester4 ditambahkan perintah ow.pay(iEmp); Mengapa terjadi error?**
   Terjadi error karena IntershipEmployee tidak mengimplementasikan Payable

4. **Perhatikan class Owner, diperlukan untuk apakah sintaks p instanceof ElectricityBill pada baris ke-6 ?**
   Memeriksa apakah objek di p adalah instance dari ElectricityBill, memastikan objek sebelum casting.

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 ?**
   Diperlukan untuk memberi tahu compiler bahwa objek p (yang bertipe Payable) sebenarnya adalah objek dari class ElectricityBill.

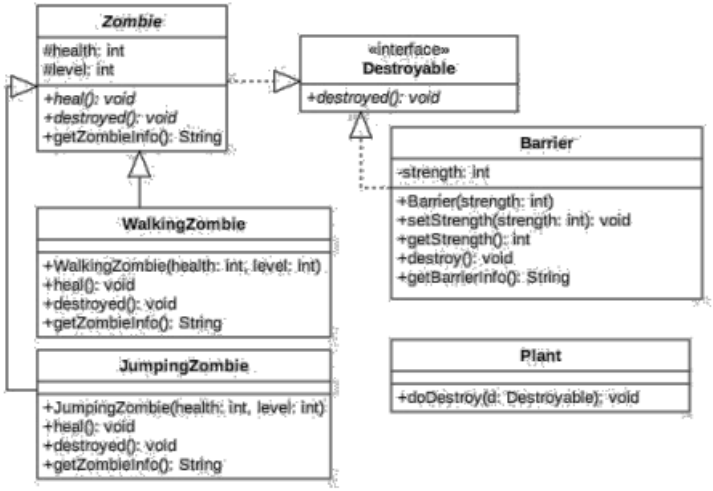
# tugas

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!

ATHRIYA GENFERIN
2I – 03 - 2241720075
D-IV INFORMATICS ENGINEERING




**Contoh**: jika class Tester seperti di bawah ini:

```
3   public class Tester {
4       public static void main(String[] args) {
5           WalkingZombie wz = new WalkingZombie(100, 1);
6           JumpingZombie jz = new JumpingZombie(100, 2);
7           Barrier b = new Barrier(100);
8           Plant p = new Plant();
9           System.out.println(""+wz.getZombieInfo());
10          System.out.println(""+jz.getZombieInfo());
11          System.out.println(""+b.getBarrierInfo());
12          System.out.println("----------------------");
13          for(int i=0;i<4;i++){//Destroy the enemies 4 times
14              p.doDestroy(wz);
15              p.doDestroy(jz);
16              p.doDestroy(b);
17          }
18          System.out.println(""+wz.getZombieInfo());
19          System.out.println(""+jz.getZombieInfo());
20          System.out.println(""+b.getBarrierInfo());
21      }
22  }
```

```java
package jobsheet12.assigment;

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";
    }
}
```

```
package jobsheet12.assigment;

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

```
package jobsheet12.assigment;

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

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

    @Override
```

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

```
package jobsheet12.assigment;

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();
        }
    }
}
```

```
package jobsheet12.assigment;

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 += (health * 0.1);
        } else if (level == 2) {
            health += health * 0.3;
        } else if (level == 3) {
            health += health * 0.4;
```

```
        }
    }

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

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

```
package jobsheet12.assigment;

public abstract class Zombie {
    protected int health, level;

    public abstract void heal();

    public abstract void destroyed();

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

```
package jobsheet12.assigment;

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("====================================");
        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++) {// Destroy the enemies 4 times
            p.doDestroy(wz);
```

ATHRIYA GENFERIN
2I – 03 - 2241720075
D-IV INFORMATICS ENGINEERING

```
            p.doDestroy(jz);
            p.doDestroy(b);
        }
        System.out.println("====================================");
        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 = 40
Level = 1
Jumping Zombie Data
 Health = 51
Level = 2
Barrier Strength = 64

PS E:\SEMESTER 3\OOP\OOP>
```