LAPORAN PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK

JOBSHEET 12: POLIMORFISME

oleh :
Halim Teguh Saputro
2E
2141762122

PROGRAM STUDI D-IV SISTEM INFORMASI BISNIS
JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG
Jl. Soekarno Hatta No .9, Jatimulyo, Kec. Lowokwaru, Kota Malang,
JawaTimur 65141

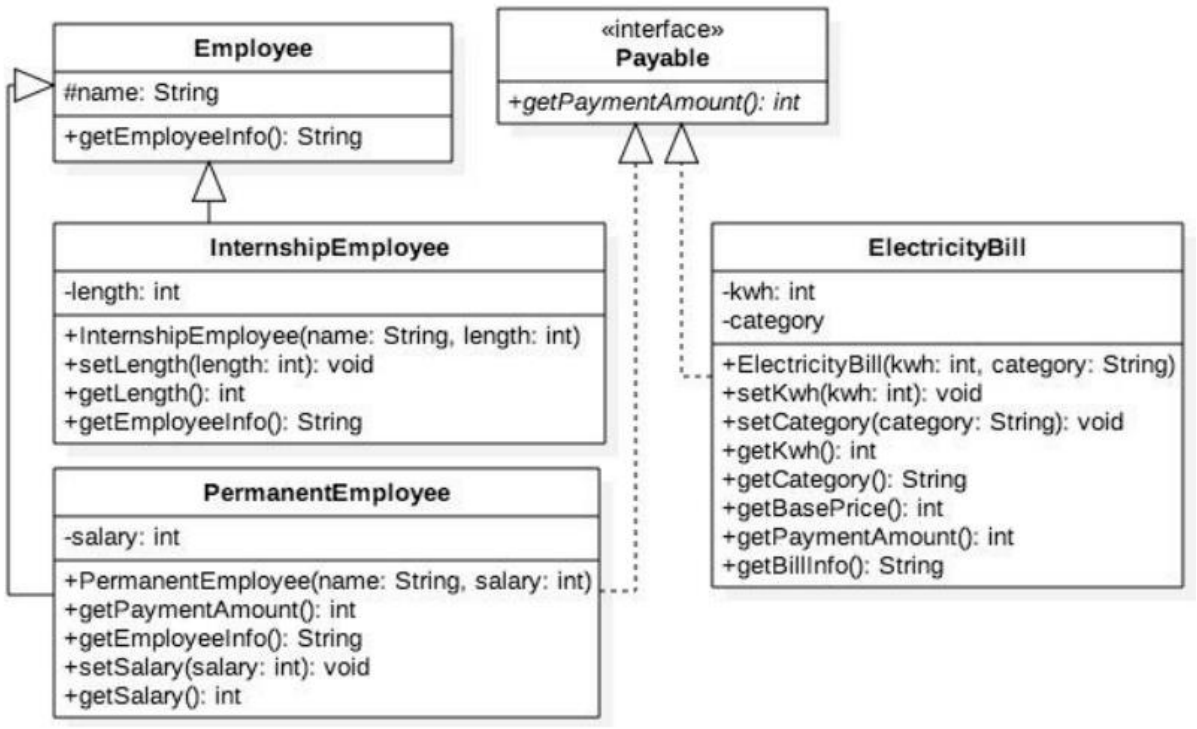
KOMPETENSI

Setelah melakukan percobaan pada modul ini, mahasiswa memahami konsep:

1. Memahami konsep dan bentuk dasar polimorfisme
2. Memahami konsep virtual method invication
3. Menerapkan polimorfisme pada pembuatan heterogeneous collection
4. Menerapkan polimorfisme pada parameter/argument method
5. Menerapkan object casting untuk mengubah bentuk objek

STUDI KASUS

Untuk percobaan pada jobsheet ini akan digunakan class diagram di bawah ini:



Dalam suatu perusahaan, pemilik pada tiap bulannya harus membayar gaji pegawai tetap dan rekening listrik. Selain pegawai tetap perusahaan juga memiliki pegawai magang, dimana pegawai ini tidak mendapatkan gaji.

PRAKTIKUM 1. BENTUK DASAR POLIMORFISME

A. Langkah Percobaan

1. Buat class Employee

```
package Praktikum;

public class Employee {
    protected String name;

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

2. Buat interface Payable

```
package Praktikum;

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

3. Buat class InternshipEmployee, subclass dari Employee

```
package Praktikum;

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() {
        this.length = length;
    }

    public String getEmployeeInfo() {
        String info = super.getEmployeeInfo() + "\n";
        info += "Registered as internship employee for " + length + " month/s\n";
```

```
        return info;
    }
}
```

4. Buat class PermanentEmployee, subclass dari Employee dan implements ke Payable

```
package Praktikum;

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() {
        this.salary = salary;
    }

    @Override
    public int getPaymentAmount() {
        // TODO Auto-generated method stub
        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;
    }
}
```

5. Buat class ElectricityBill yang implements ke interface Payable

```
package Praktikum;

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() {
        // TODO Auto-generated method stub
        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";
    }
}
```

6. Buat class Tester1

```java
import Praktikum.ElectricityBill;
import Praktikum.Employee;
import Praktikum.InternshipEmployee;
import Praktikum.Payable;
import Praktikum.PermanentEmployee;

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. Pertanyaan

1. Class apa sajakah yang merupakan turunan dari class Employee?
   Jawab: class turunan (subclass) dari class Employee adalah Class InternshipEmployee dan Class PermanentEmployee.

2. Class apa sajakah yang implements ke interface Payable?
   Jawab: Class yang mengimplement interface Payable adalah Class PermanentEmployee dan Class ElectricityBill.

3. Perhatikan class Tester1, baris ke- 10 dan 11. Mengapa e, bisa diisi dengan objek pEmp (merupakan objek dari class PermanentEmployee) dan ojek iEmp (merupakan objek dari class InternshipEmployee)?
   Jawab: e bisa diisi dengan objek pEmp yang merupakan objek dari class PermanentEmployee dan iEmp yang merupakan objek dari class InternshipEmployee karena e bertipe data Employee yang merupakan super class dari class PermanentEmployee dan InternshipEmployee sehingga e bisa diisi dengan objek pEmp dan iEmp.

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: p bisa diisi dengan objek pEmp yang merupakan PermanentEmployee dan objek eBill yang merupakan objek dari class ElectricityBill, yang mana class PermanentEmployee dan ElectricityBill ini mengimplemets class Payable. Jadi, p bisa diisi dengan objeck pEmp dan eBill bisa

5. Coba tambahkan sintaks:
   p = iEmp;
   e = eBill;
   Pada Baris 14 dan 15 (baris terakhir dalam method main)! Apa yang menyebabkan error?
   Jawab:

```
p = iEmp;
e = eBill;
```

   Keterangan: karena iEmp itu tidak mengimplement class Payable sehingga p yang bertipe data Payable. Jadi, p tidak di isi dengan objek iEmp. eBill juga tidak meng-extend Employee sehingga tidak bisa dimasukkan ke e yang bertipe data Employee.

6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!
   Jawab: polimorfisme merupakan sebuah konsep yang mana sebuah class itu bisa memiliki berbagai bentuk, namun untuk mengubah itu harus menggunakan super classnya. Bila suatu class yang diharapkan untuk berubah bentuk tapi tidak terintegrasi dengan super class nya atau dasar class yang ingin di ubah bentuknya itu tidak akan bisa.

PRAKTIKUM 2. VIRTUAL METHOD INVOCATION

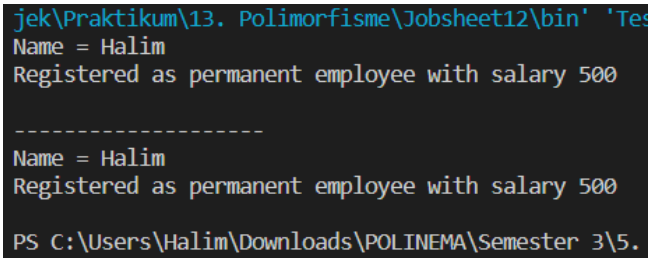
A. Langkah Percobaan
   1. Pada Percobaan ini masih akan digunakan class-class dan interface yang digunakan pada percobaan sebelumnya.
   2. Buat class baru dengan nama Tester2

```
import Praktikum.Employee;
import Praktikum.PermanentEmployee;

public class Tester2 {
    public static void main(String[] args) {
        PermanentEmployee pEmp = new PermanentEmployee("Halim", 500);
        Employee e;
```

```
        e = pEmp;
        System.out.println("" + e.getEmployeeInfo());
        System.out.println("--------------------");
        System.out.println("" + pEmp.getEmployeeInfo());
    }
}
```

3. Jalankan class Tester2, dan akan didapatkan hasil sebagai berikut:



B. Pertanyaan

1. Perhatikan class Tester2 di atas, mengapa pemanggilan e.getEmployee() pada baris 8 dan pEmp.getEmployeeInfo() pada baris 10 menghasilkan hasil sama?
   Jawab:
   Sintaks-sintaks tersebut memiliki output yang sama karena e itu mengambil nilai dari pEmp dan pEmp yang objek dari PermanentEmployee merupakan subclass dari Employee sehingga e bisa diisi dengan objek pEmp;

2. Mengapa pemanggilan method e.getEmployeeInfo() disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan pEmp.getEmployee() tidak?
   Jawab:
   pEmp.getEmployee() tidak termasuk virtual method invocation karena method getEmployee() yang dipanggil berasal dari class PermanentEmployee yang mana itu hanya pendeklarasian dan pengisian objek biasa. Sedangkan, e.getEmployeeinfo() termasuk ke virtual method invocation karena e diisi oleh pEmp sehingga method getEmployeeInfo() yang dipanngil oleh e itu berasal dari objek pEmp class PermanentEmployee, bukan dari e class Employee.

3. Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?
   Jawab:
   Virtual method invocation adalah method yang dipanggil dari class lain yang mana class-class tersebut saling terintegrasi sehingga saat dipanggil terjadi suatu overriding method yang membuat isi atau rule dari method yang dipanggil berubah menjadi rule dari class yang terintegrasi tersebut.

PRAKTIKUM 3. HETEROGENOUS COLLECTION

A. Langkah Percobaan

1. Pada percobaan ke-3 ini, masih akan digunakan class-class dan interface pada percobaan sebelumnya.
2. Buat class baru Tester3.

```
public class Tester3 {
    public static void main(String[] args) {
        PermanentEmployee pEmp = new PermanentEmployee("Halim", 500);
        InternshipEmployee iEmp = new InternshipEmployee("Teguh", 5);
        ElectricityBill eBill = new ElectricityBill(5, "A-1");
        Employee e[] = { pEmp, iEmp };
        Payable p[] = { pEmp, eBill };
        Employee e2[] = { pEmp, iEmp, eBill };
    }
}
```

B. Pertanyaan

1. Perhatikan array e pada baris ke-8, mengapa ia bisa diisi dengan objek-objek dengan tipa yang berbeda, yaitu objek pEmp (objek dari PermanentEmployee) dan objek iEmp (objek dari InternshipEmployee)?
   Jawab:
   e bisa diisi dengan objek dengan tipe yang berbeda selama class dari objek tersebut meng-extends ke classnya e yaitu Employee. Jadi, pEmp yang merupakan objek dari class Permanent Employee itu meng-extends class Employee begitupun objek iEmp yang merupakan class InternshipEmployee.

2. Perhatikan juga baris ke-9, mengapa array p juga diisi dengan objek-objek dengan tipa yang berbeda, yaitu objek pEmp (ojek dari PermanentEmployee) dan objek eBill (objek dari ElectricityBilling)?
   Jawab:
   p bisa diisi dengan objek objek tersebut karena objek objek tersebut memiliki keterkaitan dengan class nya yang merupakan tipe data P yaitu Payable. pEmp yang merupakan objek dari class PermanentEmployee dan eBill yang merupakan objek dari class ElectricityBill itu mengimplements interfave Payable. Sehingga p bisa diisikan objek tersebut.

3. Perhatikan baris ke-10, mengapa terjadi error?
   Jawab:
   Terjadi error karena e yang berasal bertipe data Employee tidak dapat diisi dengan objek eBill karena Employee dan ElectricityBilling tidak ada keterkaitan

PRAKTIKUM 4. ARGUMEN POLIMORFISME, INSTANCEOF, DAN CASTING OBJEK

A. Langkah Percobaan

1. Percobaan 4 ini juga masih menggunakan class-class dan interface yang digunakan pada percobaan sebelumnya.
2. Buat class baru dengan nama Owner. Owner bisa melakukan permbayaran baik kepada pegawai permanen maupun rekening listrik melalui method Pay(). Selain itu juga bisa menampilkan info pegawai permanen maupun pegawai magan melalui method showMyEmployee().

```
package Praktikum;

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("Yout have to pay her.him mounthly!!");
        } else {
            System.out.println("No need to pay him/her :");
        }
    }
}
```

3. Buat class baru Tester4

```
import Praktikum.ElectricityBill;
import Praktikum.InternshipEmployee;
import Praktikum.Owner;
import Praktikum.PermanentEmployee;

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("----------------------");

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

        InternshipEmployee iEmp = new InternshipEmployee("Teguh", 5);
        ow.showMyEmployee(pEmp);
        System.out.println("----------------------");
        ow.showMyEmployee(iEmp);
    }
}
```

4. Jalankan class Tester4, dan akan didapatkan hasil sebagai berikut

```
ester4'
Total payment = 500
kwh = 5
Category = R-1(100 per kwh)

----------------------
Total payment = 525
Name = Halim
Registered as permanent employee with salary 500

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

Yout have to pay her.him mounthly!!
----------------------
Name = Teguh
Registered as internship employee for 5 month/s
```

B. 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:
   Method pay() bisa dijalankan karena yang pertama method tersebut memiliki parameter dengan tipe data Payable. Yang kedua, eBill dan pEmp merupakan objek dari class yang mengimplement class Payable sehingga perubahan bentuknya (polimorfisme) terjadi pada parameter method pay(). Jadi intinya, class asal objek eBill dan pEmp yaitu ElectricityBill dan PermanentEmployee itu mengimplements Payable sehingga eBill dan pEmp bisa dibilang memiliki karakteristik yang sama dengan class Payable namun dengan rule yang berbeda.

2. Jadi apakah tujuan membuat argument bertipe Payable pada method pay() yang ada didalam class Owner?
   Jawab:
   Tujuannya agar objek dari PermanentEmployee dan ElectricityBill bisa menggunakan method pay() tersebut. Atau class class lain yang juga mengimplements Payable juga bisa menggunakan method pay tersebut.

3. Coba pada baris terakhir method main() yang ada didalam class Tester4 ditambahkan perintah ow.pay(iEmp);

```
ow.pay(iEmp);
```

   Mengapa terjadi Error?
   Jawab:
   Terjadi error karena objek iEmp merupakan objek dari InternshipEmployee yang mana class tersebut tidak berkaitan atau tidak mengimplement class Payable.

4. Perhatikan class Owner, diperlukan untuk apakah sintaks p instanceof ElectricityBill pada baris ke-6?
   Jawab:
   Sintak tersebut berguna untuk menyeleksi parameter yang masuk apakah p tersebut dari class ElectricityBill atau bukan, jika iya maka akan menjalankan suatu rule tertentu.

5. Perhatikan Kembali class Owner baris ke-7, untuk apakah castion objek disan (ElectricityBill eb = (ElectricityBill) p) diperlukan? Mengapa objek p yang bertipe Payable harus di-casting ke dalam objek eb yang bertipa ElectricityBill?
   Jawab:
   p perlu dicasting karena sebelumnya bagian parameter p itu di deklarasikan menggunakan tipe data Payable. Sehingga p perlu di ubah menjadi bertipe ElectricityBill agar datanya bisa masuk ke objek eb.

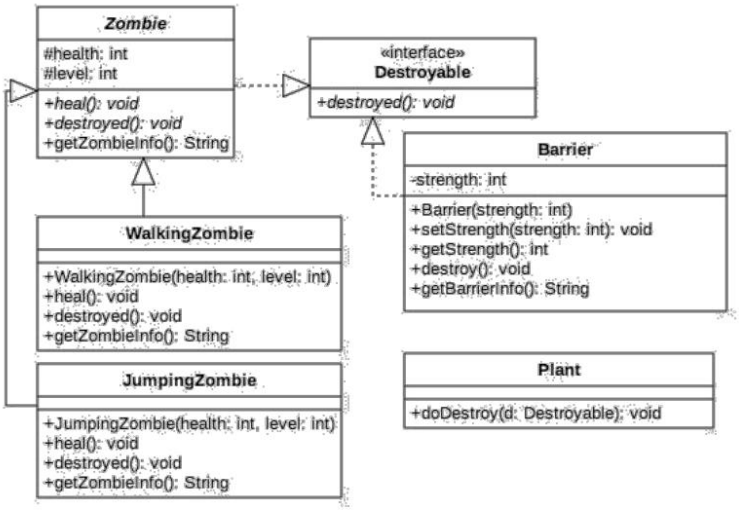
TUGAS

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%
    - Penghacuran: setiap kali penghancura, health akan berkuran 2%
- Pada Jumping Zombie
    - Penyembuhan: Penyembuhan ditentukan 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!



Contoh: jika 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  }
```

Akan menghasilkan output:

JAWAB

A. Source Code

1. Destroyable.java

```
package Tugas;

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

2. Barrier.java

```
package Tugas;

public class Barrier implements Destroyable {
    private int strenght;

    public Barrier(int strenght) {
        this.strenght = strenght;
    }

    public void setStrength(int strenght) {
        this.strenght = strenght;
    }

    public int getStrength() {
        return strenght;
    }

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

```
    @Override
    public void destroyable() {
        strenght = strenght - (strenght * 10 / 100);
    }

}
```

3. Zombie.java

```
package Tugas;

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

    public void heal() {
        if (level == 1) {
            health = health;
        } else if (level == 2) {
            health = health;
        } else if (level == 3) {
            health = health;
        }
    }

    public void destroyable() {
        health = health - (health * 5 / 100);
    }

    public String getZombieInfo() {
        String info = "Zombie Data :\n"
                + "Health\t= " + (int) health + "\n"
                + "Level\t= " + level + "\n";
        return info;

    }
}
```

4. WalkingZombie.java

```
package Tugas;

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 + (health * 10 / 100);
        } else if (level == 2) {
            health = health + (health * 30 / 100);
        } else if (level == 3) {
            health = health + (health * 40 / 100);
        }
    }

    public String getZombieInfo() {
        String info = "Walking Zombie Data :\n"
                + "Health\t= " + (int) health + "\n"
                + "Level\t= " + level + "\n";
        return info;
    }

    @Override
    public void destroyable() {
        // TODO Auto-generated method stub
        health = health - (health * 2 / 100);
    }

}
```

5. JumpingZombie.java

```
package Tugas;

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 + (health * 30 / 100);
        } else if (level == 2) {
            health = health + (health * 40 / 100);
        } else if (level == 3) {
            health = health + (health * 50 / 100);
        }
```

```
    }

    public String getZombieInfo() {
        String info = "Jumping Zombie Data :\n"
                + "Health\t= " + (int) health + "\n"
                + "Level\t= " + level + "\n";
        return info;
    }

    @Override
    public void destroyable() {
        // TODO Auto-generated method stub
        health = health - (health * 1 / 100);
    }

}
```

6. Plant.java

```
package Tugas;

public class Plant {

    public void doDestroyable(Destroyable d) {
        if (d instanceof WalkingZombie) {
            WalkingZombie wz = (WalkingZombie) d;
            wz.destroyable();
        } else if (d instanceof JumpingZombie) {
            JumpingZombie jz = (JumpingZombie) d;
            jz.destroyable();
        } else if (d instanceof Barrier) {
            Barrier b = (Barrier) d;
            b.destroyable();
        }
    }
}
```

7. Tester.java

```
package Tugas;

public class Tester {
    public static void main(String[] args) {
        WalkingZombie wz = new WalkingZombie(100, 1);
        JumpingZombie jz = new JumpingZombie(100, 1);

        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.doDestroyable(wz);
            p.doDestroyable(jz);
            p.doDestroyable(b);
        }

        System.out.println("" + wz.getZombieInfo());
        System.out.println("" + jz.getZombieInfo());
        System.out.println("" + b.getBarrierInfo());
        System.out.println("==========================");

    }
}
```

B. Output

```
graman Berbasis Objek\Praktik

Walking Zombie Data :
Health  = 100
Level   = 1

Jumping Zombie Data :
Health  = 100
Level   = 1

Barrier Strength = 100
==========================
Walking Zombie Data :
Health  = 92
Level   = 1

Jumping Zombie Data :
Health  = 96
Level   = 1

Barrier Strength = 66
==========================
PS C:\Users\Halim\Downloads\P
```

LINK GITHUB: https://github.com/HalimTeguh/Praktikum/tree/master/13.%20Polimorfisme