

PRAKTIKUM PBO JOBSHEET 12

POLIMORFISME DALAM OOP

JAVA

PERTEMUAN 13



MARGARETHA VIOLINA PUTRI P.
2141762056 / 11
SIB-2F

POLITEKNIK NEGERI MALANG
2022

Daftar Isi

DAFTAR ISI	2
1. KOMPETENSI	3
2. LINK GITHUB.....	3
3. STUDI KASUS	3
4. PRAKTIKUM.....	3
PERCOBAAN 1 – BENTUK DASAR POLIMORFISME	3
<i>Langkah Percobaan.....</i>	3
<i>Pertanyaan Percobaan 1.....</i>	6
PERCOBAAN 2 – VIRTUAL METHOD INVOCATION	7
<i>Langkah Percobaan.....</i>	7
<i>Pertanyaan Percobaan 2.....</i>	8
PERCOBAAN 3 – HETEROGENOUS COLLECTION	8
<i>Langkah Percobaan.....</i>	8
<i>Pertanyaan Percobaan 3.....</i>	9
PERCOBAAN 4 – ARGUMEN POLIMORFISME, INSTANCEOF DAS CASTING OBJEK.....	9
<i>Langkah Percobaan.....</i>	9
<i>Pertanyaan Percobaan 4.....</i>	11
5. TUGAS	12

1. Kompetensi

Setelah melakukan percobaan pada jobsheet ini, diharapkan mahasiswa mampu:

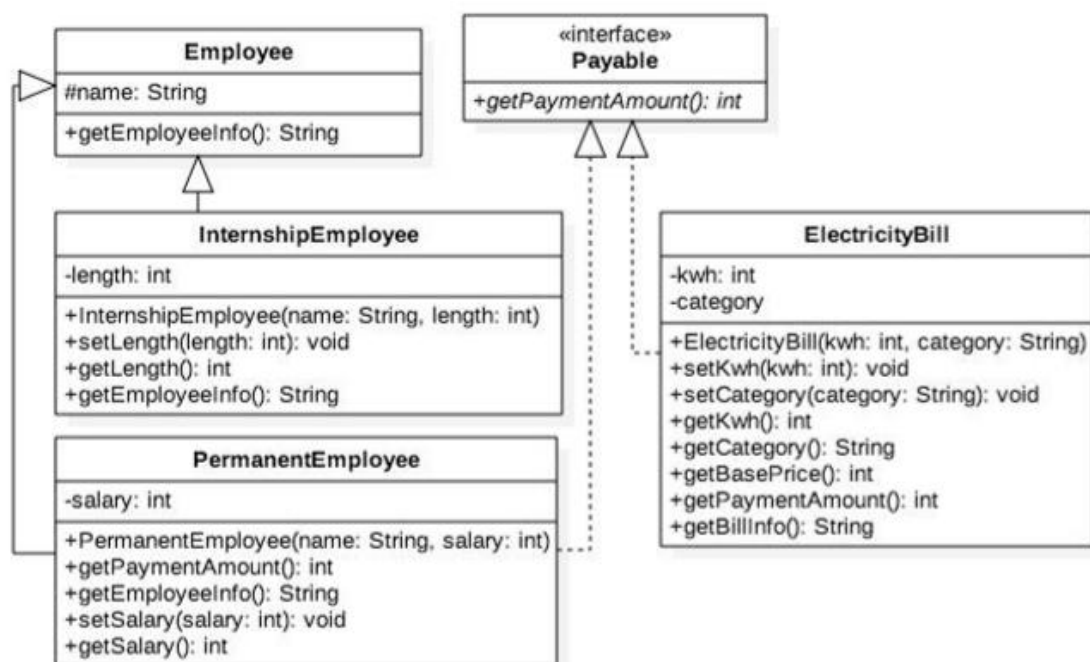
- Memahami konsep dan bentuk dasar polimorfisme
- Memahami konsep virtual method invocation
- Menerapkan polimorfisme pada pembuatan heterogeneous collection
- Menerapkan polimorfisme pada parameter/argument method
- Menerapkan object casting untuk meng-ubah bentuk objek

2. Link Github

<https://github.com/MargarethaViolinaPutri/PBO->

3. 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.

4. Praktikum

Percobaan 1 – Bentuk Dasar Polimorfisme

Langkah Percobaan

1. Buat class Employee

```

10  L  */
11  public class Employee {
12      protected String name;
13
14      public String getEmployeeInfo() {
15          return "Name = "+name;
16      }
17  }

```

2. Buat interface Payable

```

10  L  */
11  public interface Payable {
12      public int getPaymentAmount();
13  }

```

3. Buat class InternshipEmployee, subclass dari Employee

```

11  public class Employee {
12      protected String name;
13
14      public String getEmployeeInfo() {
15          return "Name = "+name;
16      }
17  }

```

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

```

11  public class PermanentEmployee extends Employee implements Payable{
12      private int salary;
13
14      public PermanentEmployee(String name, int salary){
15          this.name=name;
16          this.salary=salary;
17      }
18      public int getSalary(){
19          return salary;
20      }
21      public void setSalary(int salary){
22          this.salary=salary;
23      }
24      @Override
25      public int getPaymentAmount() {
26          return(int) (salary+0.05*salary);
27      }
28      @Override
29      public String getEmployeeInfo() {
30          String info = super.getEmployeeInfo()+"\n";
31          info += "Registered as permanent employee with salary"+salary+"\n";
32          return info;
33      }
34  }

```

5. Buat class ElectricityBill yang implements ke interface Payable



```

11 public class ElectricityBill implements Payable{
12     private int kwh;
13     private String category;
14
15     public ElectricityBill(int kwh, String category){
16         this.kwh=kwh;
17         this.category=category;
18     }
19     public int getKwh(){
20         return kwh;
21     }
22     public void setKwh(int kwh){
23         this.kwh=kwh;
24     }
25     public String getCategory(){
26         return category;
27     }
28     public void setCategory(String category){
29         this.category=category;
30     }
31     @Override
32     public int getPaymentAmount() {
33         return kwh*getBasePrice();
34     }
35     public int getBasePrice(){
36         int bPrice = 0;
37         switch(category){
38             case "R-1" : bPrice = 100; break;
39             case "R-2" : bPrice = 200; break;
40         }return bPrice;
41     }
42     public String getBillInfo(){
43         return "kWH = "+kwh+"\n"+
44             "Category =" +category+" (" +getBasePrice()+ " per kWH)\n";
45     }
46 }

```

6. Buat class Tester1

```

public class Tester1 {
    public 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;
    }
}

```

Output :

```
|
| --- exec-maven-plugin:3.0.0:exec (default-cli) @ Pr
|-----
| BUILD SUCCESS
|-----
| Total time: 2.307 s
| Finished at: 2022-12-10T13:42:10+07:00
|-----
```

Pertanyaan Percobaan 1

1. Class apa sajakah yang merupakan turunan dari class Employee?

Jawab : Class PermanentEmployee dan Class InternshipEmployee karena mereka mengextends class Employee di dalam kode programnya

2. Class apa sajakah yang implements ke interface Payable?

Jawab : 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 objek iEmp (merupakan objek dari class InternshipEmployee) ?

Jawab : Karena e merupakan objek dari Class Employee yang merupakan superclass dan memiliki class turunan/subclass berupa PermanentEmployee dan InternshipEmployee.

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 : Selain pada class-class yang memiliki relasi inheritance, polimorfisme juga bisa diterapkan pada interface. Ketika ada objek yang dideklarasikan dari suatu interface, maka ia bisa digunakan untuk mereferensi ke objek dari class-class yang implements ke interface tersebut.

Karena p merupakan objek dari class Payable yang merupakan sebuah class interface yang memiliki turunan class PermanentEmployee dan class InternshipEmployee

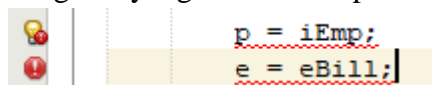
5. Coba tambahkan sintaks:

a) p = iEmp;

b) e = eBill;

c) pada baris 14 dan 15 (baris terakhir dalam method main) ! Apa yang menyebabkan error?

Jawab : Terjadi error karena p bukanlah super class dari InternshipEmployee. Begitu juga dengan e yang bukan merupakan super class dari ElectricityBill.



6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!

Jawab : Polimorfisme merupakan sebuah objek yang memiliki class dengan banyak bentuk yang berbeda. Penggunaan polimorfisme terjadi ketika ada referensi super class yang digunakan untuk merujuk ke objek dari sub class. Artinya, ketika ada suatu objek yang dideklarasikan dari super class, maka objek tersebut bisa diinstansiasi sebagai objek dari sub class. Dari uraian tersebut bisa dilihat bahwa konsep polimorfisme bisa diterapkan pada class-class yang memiliki relasi inheritance

Percobaan 2 – Virtual Method Invocation

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.

```
public class Tester2 {
    public void main(String[] args){
        PermanentEmployee pEmp = new PermanentEmployee( name: "Dedik", salary: 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:

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

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

Output:

Output sama

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ Prak_PBO ---
Name = Dedik
Registered as permanent employee with salary 500

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

-----
BUILD SUCCESS
```

Pertanyaan Percobaan 2

1. Perhatikan class Tester2 di atas, mengapa pemanggilan `e.getEmployeeInfo()` pada baris 8 dan `pEmp.getEmployeeInfo()` pada baris 10 menghasilkan hasil sama?

Jawab : Karena baris tersebut merupakan pemanggilan method virtual yaitu `e = pEmp`. Sehingga saat di run time akan sama-sama mengenali method `getEmployee` dari class `PermanentEmployee` karena objek `e` dideklarasikan di kelas tersebut.

2. Mengapa pemanggilan method `e.getEmployeeInfo()` disebut sebagai pemanggilan method virtual (virtual method invocation), sedangkan `pEmp.getEmployeeInfo()` tidak?

Jawab : Karena `e = pEmp` saat di compiler runtime di `e.getEmployeeInfo()` merupakan nama objek dari class `Employee` dengan objek `e`. Namun method yang dipanggil adalah objek dari `pEmp` dari class `Permanent Employee` sehingga hal tersebut disebut virtual method.

3. Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?

Jawab : 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. Jadi, suatu objek yang telah dibuat untuk memanggil suatu method pada super class polimorfisme. Disebut virtual pada pengenalan method tersebut, pemanggilan dan saat mengompilasi berbeda.

Percobaan 3 – Heterogenous Collection

Langkah Percobaan

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

```

11 public class Tester3 {
12     public static void main(String[] args){
13         PermanentEmployee pEmp = new PermanentEmployee( name: "Dedik", salary:500);
14         InternshipEmployee iEmp = new InternshipEmployee( name: "Sunarto", length:5);
15         ElectricityBill eBill = new ElectricityBill( kwh:5, category: "A-1");
16         Employee e[] = {pEmp, iEmp};
17         Payable p[] = {pEmp, eBill};
18         Employee e2[] = {pEmp, iEmp, eBill};
19     }

```

Output :


```

-----< com.mycompany:Prak_PBO >-----
[+] Building Prak_PBO 1.0-SNAPSHOT
-----[ jar ]-----

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Prak_PBO ---
[+] Exception in thread "main" java.lang.RuntimeException: Uncompilable code - '}' ,
    at Job12_Retha.Tester3.main(Tester3.java:1)
    Command execution failed.
[+] org.apache.commons.exec.ExecuteException: Process exited with an error: 1 (Exit
    at org.apache.commons.exec.DefaultExecutor.executeInternal (DefaultExecutor

```

Pertanyaan Percobaan 3

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 pada objek e dapat ditambahkan objek dengan tipe data class berbeda, pada class tersebut sama-sama merupakan subclass dari parent yang sama yaitu class employee

2. Perhatikan juga baris ke-9, mengapa array p juga diisi dengan objek-objek dengan tipe yang berbeda, yaitu objek pEmp (objek dari PermanentEmployee) dan objek eBill (objek dari ElectricityBilling)?

Jawab : Karena pada array p merupakan deklarasi dari class Payable yang dimana class PermanentEmployee dan ElectricityBill merupakan class yang mengimplements class interface Payable,.

3. Perhatikan baris ke-10, mengapa terjadi error?

Jawab : Karena Employee e2 [] tersebut berisi objek eBill yang dimana eBill ini bukan merupakan subclass dari class Employee.

Percobaan 4 – Argumen Polimorfisme, Instanceof dan Casting Objek

Langkah Percobaan

1. Percobaan 4 ini juga masih menggunakan class-class dan interface yang digunakan pada percobaan sebelumnya.

Buat class baru dengan nama Owner. Owner bisa melakukan pembayaran baik kepada pegawai permanen maupun rekening listrik melalui method pay(). Selain itu juga bisa menampilkan info pegawai permanen maupun pegawai magang melalui method showMyEmployee()

2. Buat class baru Tester4.

```

L */
public class Tester4 {
    public static void main(String[] args){
        Owner ow = new Owner();
        ElectricityBill eBill = new ElectricityBill( kwh:5, category: "R-1");
        ow.pay( p:eBill); //pay for electricity bill
        System.out.println( x: "-----");

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

        InternshipEmployee iEmp = new InternshipEmployee( name: "Sunarto", length:5);
        ow.showMyEmployee( e:pEmp); //show permanent employee info
        System.out.println( x: "-----");
        ow.showMyEmployee( e:iEmp); //show internship employee info
    }
}

```

3. Jalankan class Tester4, dan akan didapatkan hasil sebagai berikut:

```

Total payment = 1000
kWh = 5
Category = R-1(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 month/s
No need to pay him/her :)

```

Output :

POLIMORFISME

```
-----[ jar ]-----

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Prak
Total payment = 500
kWH = 5
Category = R-1(100 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 him/his monthly!!
-----

Name = Sunarto
Registered as internship employee for 5 month/s

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

BUILD SUCCESS
```

Pertanyaan Percobaan 4

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 keduanya merupakan implements dari class interface sehingga dapat melakukan pemanggilan

2. Jadi apakah tujuan membuat argument bertipe `Payable` pada method `pay()` yang ada di dalam class Owner?

Jawab : Supaya class yang implement dari interface `Payable` dapat dipanggil

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



Mengapa terjadi error?

Jawab : Karena pada method `pay` di objek `ow` tidak terdapat implement oleh class `InternshipEmployee`

4. Perhatikan class Owner, diperlukan untuk apakah sintaks `p instanceof ElectricityBill` pada baris ke-6 ?

Jawab : Perlu untuk melakukan pengecekan apakah p merupakan objek 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?

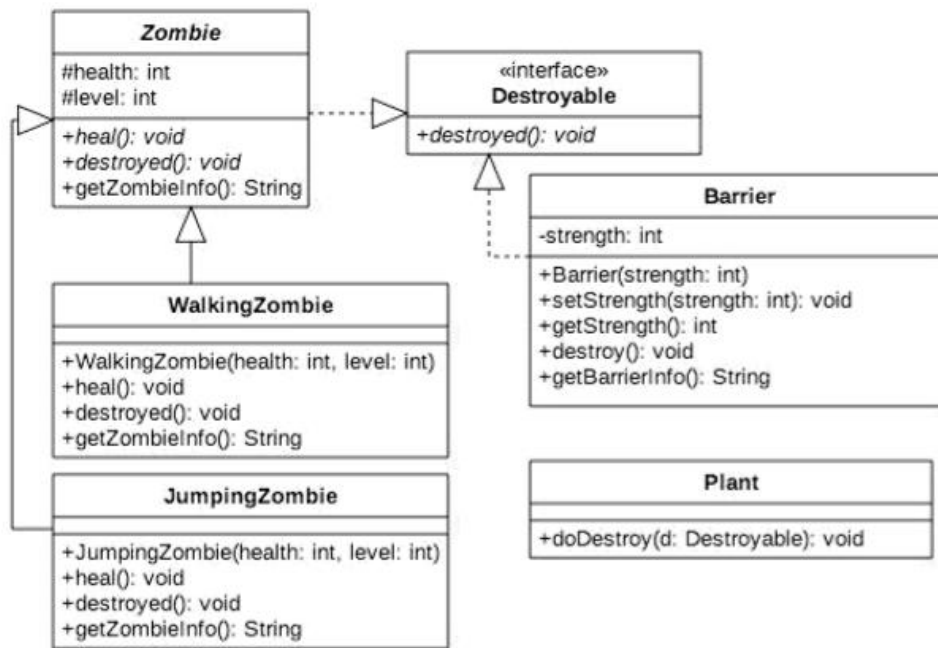
Jawab : Untuk mengembalikan referensi dari Payable p menjadi bentuk yang awal

5. 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
 - o 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%
 - o Penghancuran : setiap kali penghancuran, health akan berkurang 2%
- Pada Jumping Zombie
 - o 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%
 - o Penghancuran : setiap kali penghancuran, health akan berkurang 1%

Buat program dari class diagram di bawah ini!



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 }
  
```

Akan menghasilkan output:

```

run:
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

BUILD SUCCESSFUL (total time: 2 seconds)

```

Jawab :

Class IDestroyable

```

10  /
11  public interface IDestroyable {
12      public abstract void destroyed();
13  }
14  |

```

Class Abstract Zombie

```

10  /**
11  *
12  * @author
13  */
14  abstract class Zombie implements IDestroyable{
15      protected int health;
16      protected int level;
17      protected String ZombieInfo;
18
19      public abstract void heal();
20      public abstract void destroyed();
21      public String getZombieInfo() {
22          return ZombieInfo;
23      }
24  }

```

Class Walking Zombie

```

11 public class WalkingZombie extends Zombie{
12     public WalkingZombie(int health, int level) {
13         this.health = health;
14         this.level = level;
15     }
16
17     @Override
18     public void heal() {
19         if(level == 1){
20             health += (health * 2/100);
21         }else if(level == 2){
22             health += (health * 3/100);
23         }else{
24             health += (health * 4/100);
25         }
26     }
27
28     @Override
29     public void destroyed() {
30         health -= (health * 20/100);
31     }
32
33     public String getZombieInfo() {
34         return "Walking Zombie Data = \nHealth = " + this.health
35             + " \nLevel = " + this.level + "\n";
36     }
37 }

```

Class Jumping Zombie

```

11 public class JumpingZombie extends Zombie{
12     public JumpingZombie(int health, int level) {
13         this.health = health;
14         this.level = level;
15     }
16
17     @Override
18     public void heal() {
19         if(level == 1){
20             health += (health * 3/100);
21         }else if(level == 2){
22             health += (health * 4/100);
23         }else{
24             health += (health * 5/100);
25         }
26     }
27
28     @Override
29     public void destroyed() {
30         health -= (health * 10/100);
31     }
32
33     public String getZombieInfo() {
34         return "Jumping Zombie Data = \nHealth = " + this.health
35             + " \nLevel = " + this.level + "\n";
36     }
37 }

```

Class Barrier

```

11 public class Barrier implements IDestroyable{
12     private int strength;
13
14     public Barrier(int strength){
15         this.strength = strength;
16     }
17     public void setStrength(int strength){
18         this.strength=strength;
19     }
20     public int getStrength(){
21         return strength;
22     }
23     public String getBarrierInfo(){
24         return "Barrier Strength =" + strength;
25     }
26     @Override
27     public void destroyed() {
28         strength -= (strength*0.1);
29     }
30 }
31

```

Class Plant

```

11 public class Plant {
12     public void doDestroy(IDestroyable d) {
13         if (d instanceof WalkingZombie) {
14             ((WalkingZombie) d).destroyed();
15         } else if (d instanceof JumpingZombie) {
16             ((JumpingZombie) d).destroyed();
17         } else if (d instanceof Barrier) {
18             ((Barrier) d).destroyed();
19         }
20     }
21 }

```

Class Tester

```

public class Tester {
    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(""+wz.getZombieInfo());
        System.out.println(""+jz.getZombieInfo());
        System.out.println(""+b.getBarrierInfo());
        System.out.println("x: " + "-----");
        for(int i=0; i<4;i++){ //Destroy the enemies 4 times
            p.doDestroy( d:wz);
            p.doDestroy( d:jz);
            p.doDestroy( d:b);
        }
        System.out.println(""+wz.getZombieInfo());
        System.out.println(""+jz.getZombieInfo());
        System.out.println(""+b.getBarrierInfo());
    }
}

```


Output :

```
--- exec-maven-plugin:3.0.0:exec
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
-----
BUILD SUCCESS
```