JOBSHEET
POLYMORPHISM

1. Objective

In this jobsheet, the students will be able to:

a. Understand about concept and basic of polymorphism
b. Understand about virtual method invocation concept
c. Apply polymorphism concept on heterogeneous collection
d. Apply polymorphism concept on methods arguments
e. Apply object casting concept to cast objects

2. Pendahuluan

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object. A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

Look at this example. There is an interface class **`Vegetarian`** and super class **`Animal`**. Class is a subclass from **`Animal`** and implements class **`Vegetarian`**. Lion class is a subclass from **`Animal`**, but not implementing **`Vegetarian`**.

```
public interface Vegetarian{}
public class Animal{}
public class Deer extends Animal implements Vegetarian{}
public class Lion extends Animal {}
```

The example below show the valid and invalid object instantiation of polymorphism concept.

```
Deer d = new Deer();
Lion l = new Lion();
```

```
Animal a = d;
Animal a2 = l;
Vegetarian v = d;
```

valid

```
Vegetarian v2 = l;
```

invalid

Class **a** can be a reference to object **d** (which is an object from **Deer**), because class **Deer** is inherited from **Animal**. Class **a2** also can be a reference to object **l** (object from **Lion**), because **Lion** is inherited from **Animal**.
Object **v** can be a reference to object **d**, because class **Deer** implements **Vegetarian**, which is an interface class. Object **v2** can't be a reference to object **l**, because class **Lion** didn't implement **Vegetarian**.
This illustration above is a concept and basic form of polymorphism.

Virtual method Invocation

Virtual method invocation happened when overridden method called from polymorphism object. It called virtual method because the recognized method on compiler and run method on JVM is different.

```
public class Animal{
        public void walk(){
                System.out.println("The animal is walking around
                the jungle");
        }
}

public class Deer extends Animal {
        @override
        public void walk(){
                System.out.println("The deer is walking around
                the jungle");
}
```

Assume there's an polymorphism object called **a**.

```
        Deer d = new Deer();
        Animal a = d;
```

Then we call the overridden method, it will call the virtual method like this one:

```
        a.walk();
```

If it in the compile time, it will call method **`walk()`** from class **`Animal`**, but when the run time, it will call method **`walk()`** from class **`Deer`**.

If method **`walk()`** called from object **`d`** (which is not a polymorphism object) like this one:

```
        d.walk();
```

then method **`walk()`** that run in JVM is method **`walk()`** from class **`Deer`**.

Heterogeneous Collection

Within polymorphism concept, thus an array objects can be instantiated with different class type. For example:

```
        Animal animalArray[] = new Animal[2];
        animalArray[0] = new Deer();
        animalArray[1] = new Lion();
```

First array **`animalArray`** is **`Deer`**, dan second array of **`animalArray`** is **`Lion`**. This can be happened because **`animalArray`** declared as **`Animal`** (superclass from **`Deer`** and **`Lion`**).

Polymorphic Argument

Polymorphism can be implemented on an argument of method, so the method can get the argument value from many objects form. For example:

```
        public class Human{
                public void drive(Animal anim){
                        anim.walk();
                }
        }
```

See at method **`drive()`**. It has an argument which is **`Animal`**. Because of **`Animal`** has subclasses **`Lion`** and **`Deer`**, so **`drive()`** can get the argument value from **`Lion`** and **`Deer`**.

```
Deer d = new Deer();
Lion l = new Lion();
Human hum = new Human();
hum.drive(d);
hum.drive(l);
```

Instanceof Operator

Instanceof Operator used to check an object as a result from a class or not. The result is in Boolean. For example:

```
Deer d = new Deer();
Lion l = new Lion();
Animal a1 = d;
Animal a2 = l;
```

If we use instanceof operator, it will be like this:

```
a1 instanceof Deer  →→  result: true
a2 instanceof Lion →→  result: false
```

Object Casting

Object casting is to cast an object into another object. There is two terms about this, upcast and downcast. Upcast is an object casting from a subclass into superclass like this one:

```
Deer d = new Deer();
Animal a1 = d; // upcasting
```

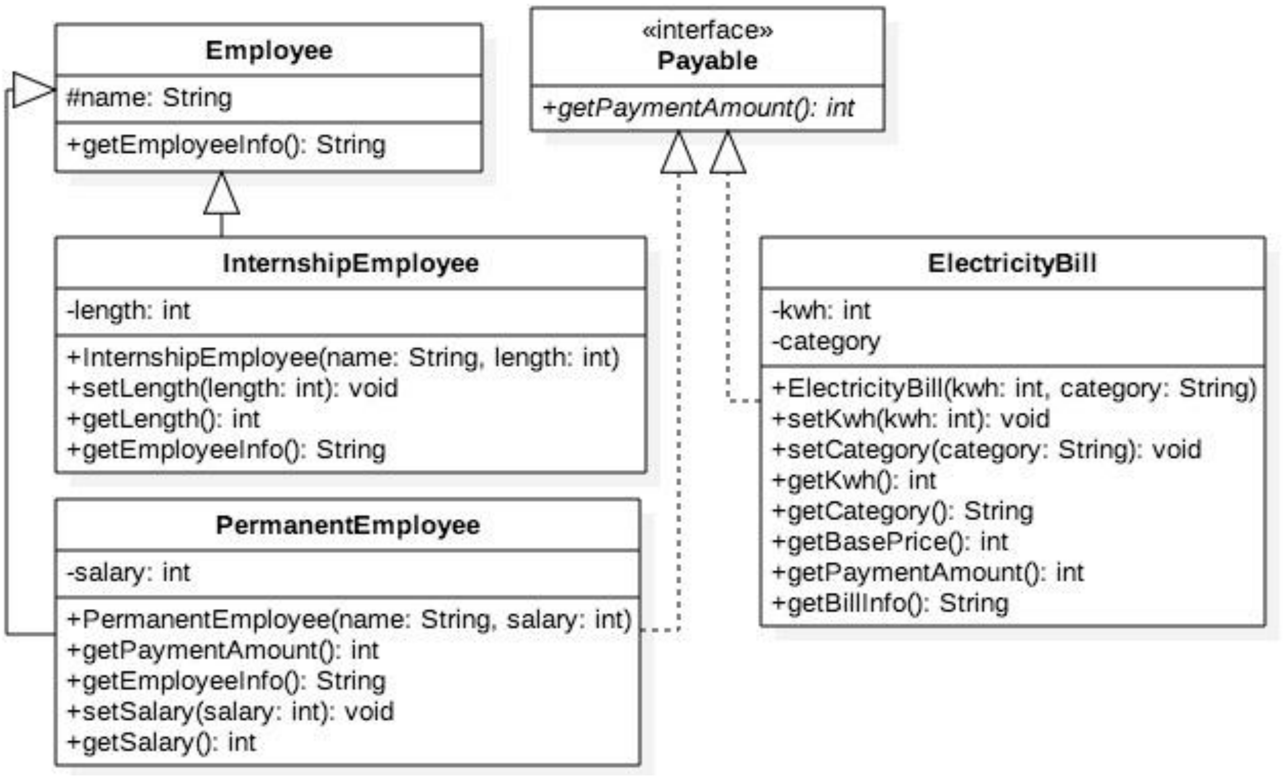
Downcast is an object casting from superclass into subclass like this one:

```
Deer  d = new Deer();
 Animal a1 = d; //            upcasting
Deer  d = (Deer) a1; //        downcasting
```

Downcasting also similarly called as explicit casting.

3. Case Study

This class diagram below will be used on our practice

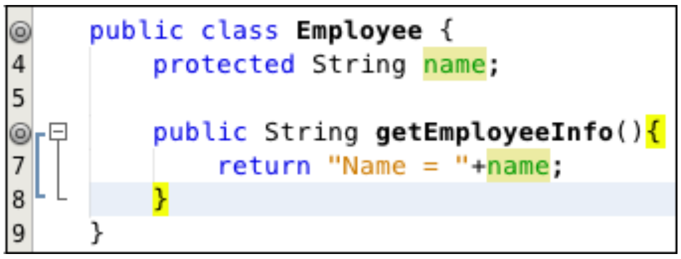


A company must pay their employee and electrical bill every single month. Beside the permanent employee, they have internship employee too, but they didn't have to pay them, because their status is an intern.

4. Experiment 1 – Base form of Polymorphism

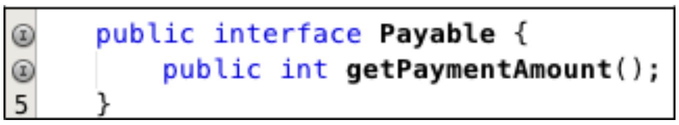
4.1. Steps

1. Make class **`Employee`**



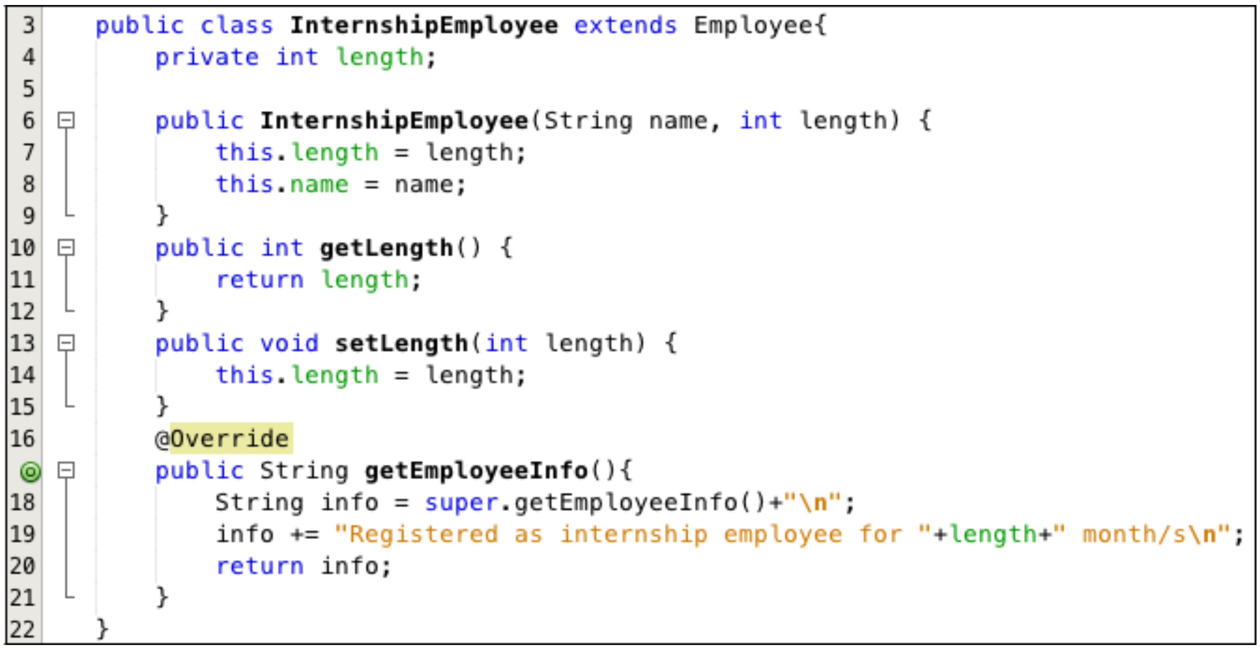
```java
public class Employee {
    protected String name;

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

2. Make interface **Payable**



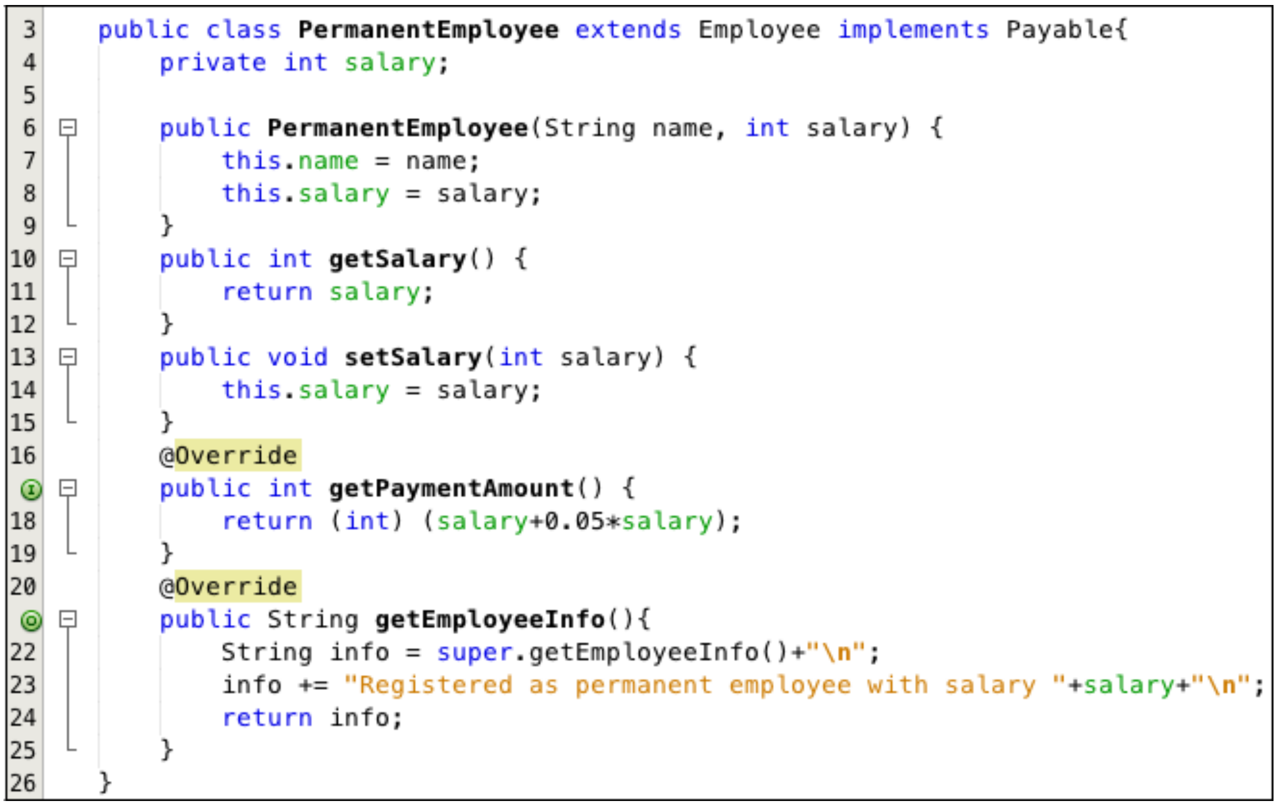
```
public interface Payable {
    public int getPaymentAmount();
}
```

3. Make class **InternshipEmployee**, subclass of **Employee**



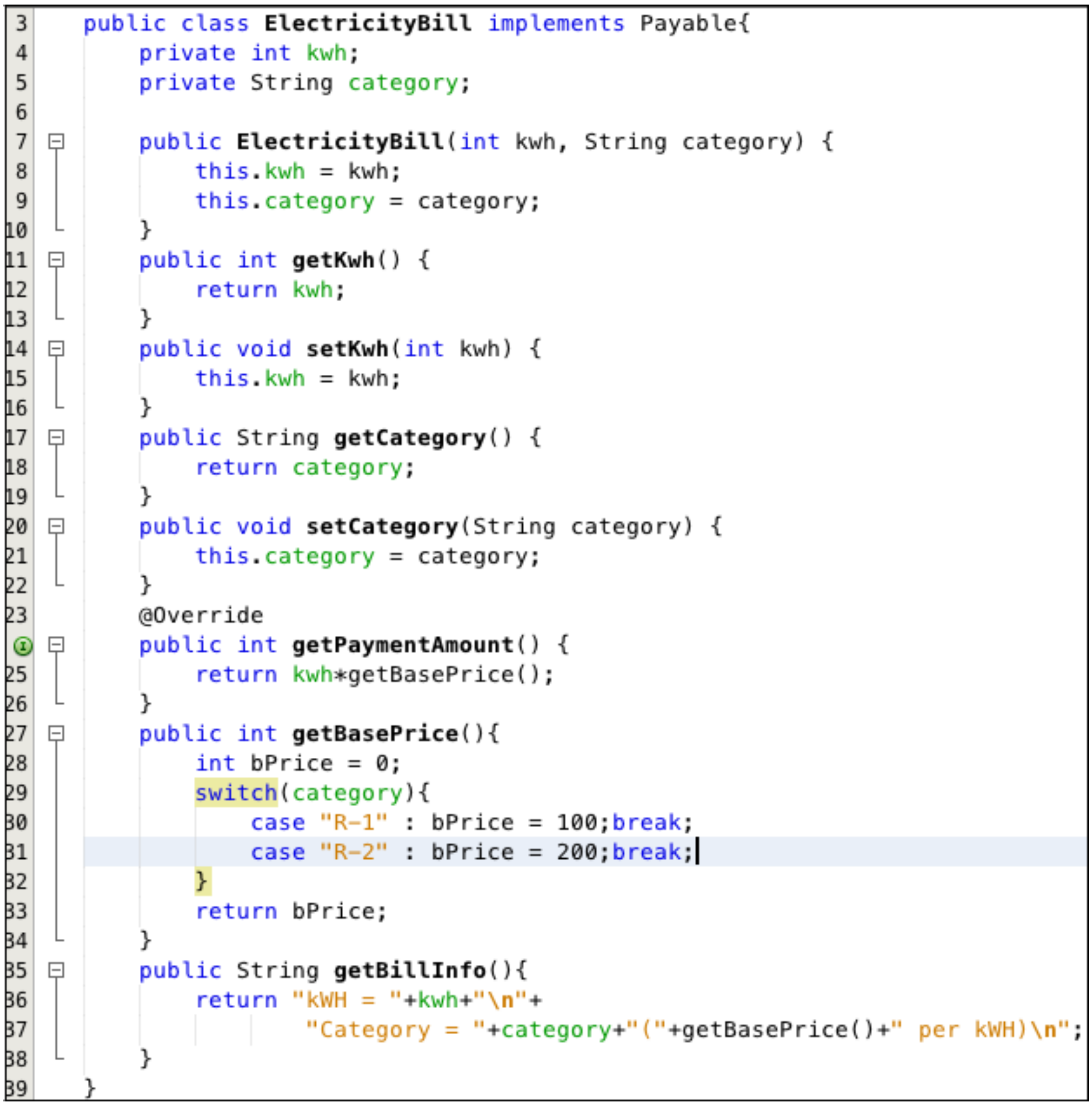
```
public class InternshipEmployee extends Employee{
    private int length;

    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+" month/s\n";
        return info;
    }
}
```

4. Make class **`PermanentEmployee`**, subclass of **`Employee`** and implements to **`Payable`**



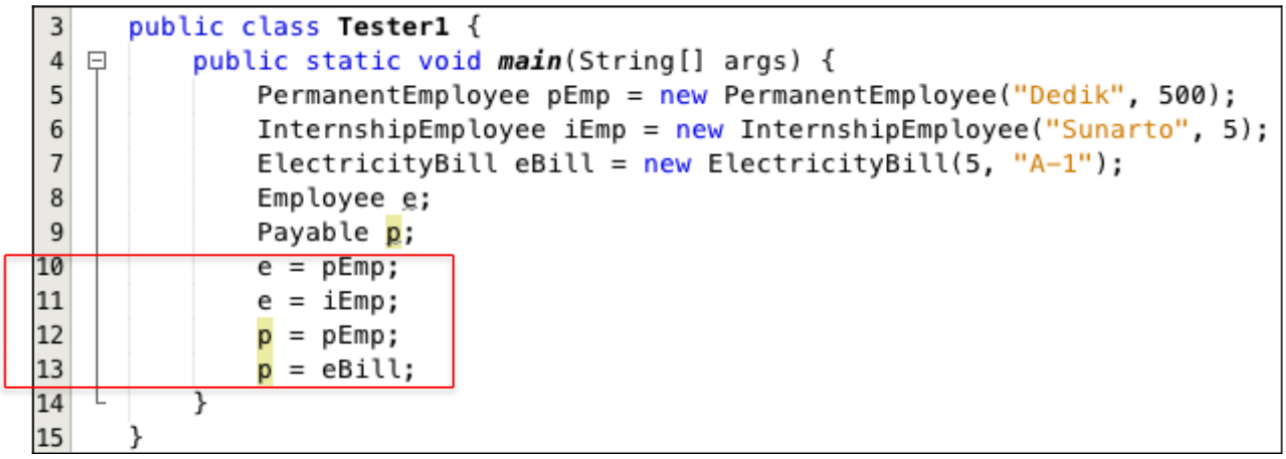
```java
public class PermanentEmployee extends Employee implements Payable{
    private int salary;

    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;
    }
}
```

5. Make class **`ElectricityBill`** and implements to **`Payable`**



```java
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";
    }
}
```

6. Make class **`Tester1`**



```java
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;
    }
}
```

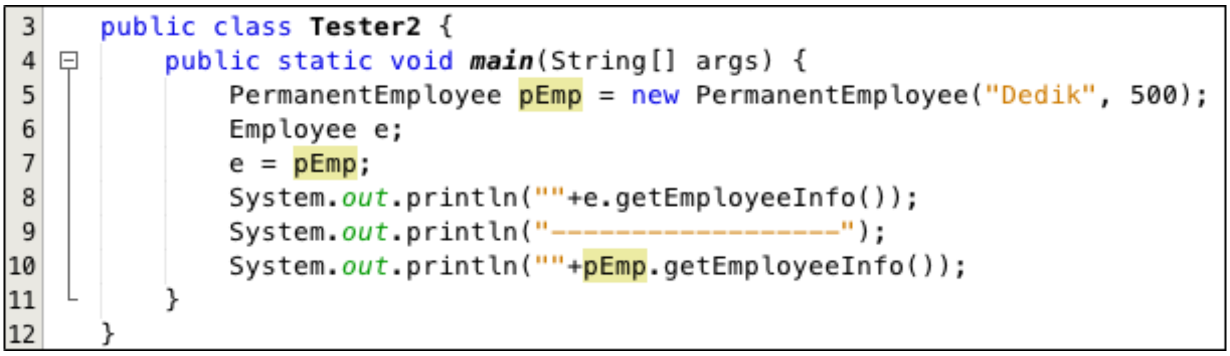
4.2. Question

1. Which classes are derived from **`Employee`**?
2. Which classes are implemented interface **`Payable`**?
3. In class **`Tester1`**, look at line-10 and 11. Why we can assign **`e`** with object **`pEmp`** (an object from class **`PermanentEmployee`**) and object **`iEmp`** (an object from class **`InternshipEmploye`**) ?
4. In class **`Tester1`**, look at line 12 and 13. Why we can assign **`p`**, with object **`pEmp`** (an object from class **`PermanentEmployee`**) and object **`eBill`** (an object from class **`ElectricityBill`**) ?
5. Add this code to line 14 and 15:
   ```
   p = iEmp;
   e = eBill;
   ```
   Why it resulted error?
6. Make a conclusion about polymorphism concept!

5. Experiment 2 – Virtual method invocation

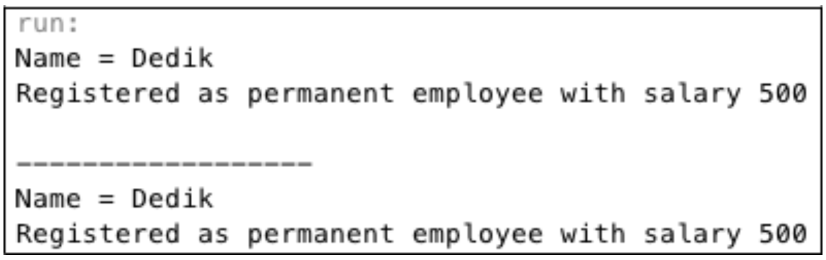
5.1. Steps

1. In this part we still use same class from the experiment before.
2. Create a new class name **`Tester2`**.



```
3   public class Tester2 {
4       public static void main(String[] args) {
5           PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
6           Employee e;
7           e = pEmp;
8           System.out.println(""+e.getEmployeeInfo());
9           System.out.println("-------------------");
10          System.out.println(""+pEmp.getEmployeeInfo());
11      }
12  }
```

3. Run class **`Tester2`**, then the result like this one:



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

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

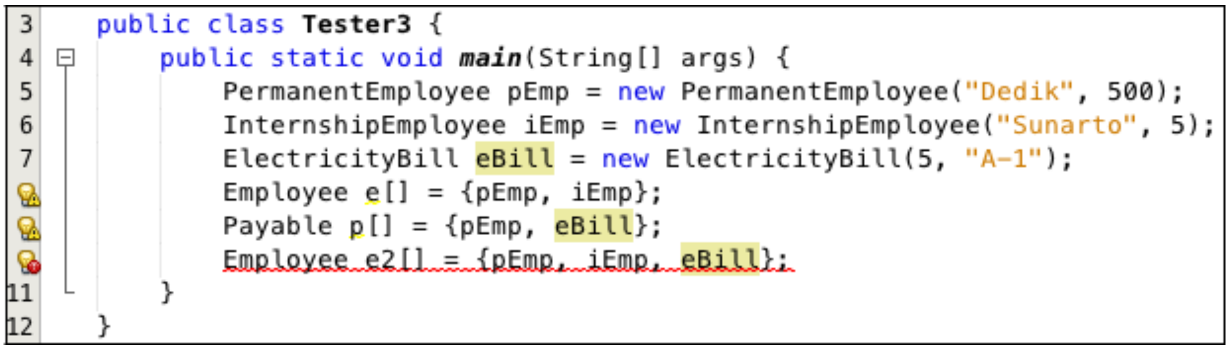
5.2. Question

1. Look at class **`Tester2`**, why method calling **`e.getEmployeeInfo()`** at line 8 and **`pEmp.getEmployeeInfo()`** at line 10 has the same result?
2. Why method calling **`e.getEmployeeInfo()`** called virtual method invocation, but not **`pEmp.getEmployeeInfo()`**?
3. So, whats the meaning about virtual method invocation? Why it called 'virtual'?

6. Experiment 3 – Heterogenous Collection

6.1. Steps

1. Use the remaining project before.
2. Make a new class **`Tester3`**.



```
 3  public class Tester3 {
 4      public static void main(String[] args) {
 5          PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
 6          InternshipEmployee iEmp = new InternshipEmployee("Sunarto", 5);
 7          ElectricityBill eBill = new ElectricityBill(5, "A-1");
            Employee e[] = {pEmp, iEmp};
            Payable p[] = {pEmp, eBill};
            Employee e2[] = {pEmp, iEmp, eBill};
11      }
12  }
```

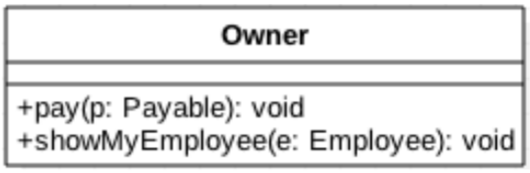
6.2. Question

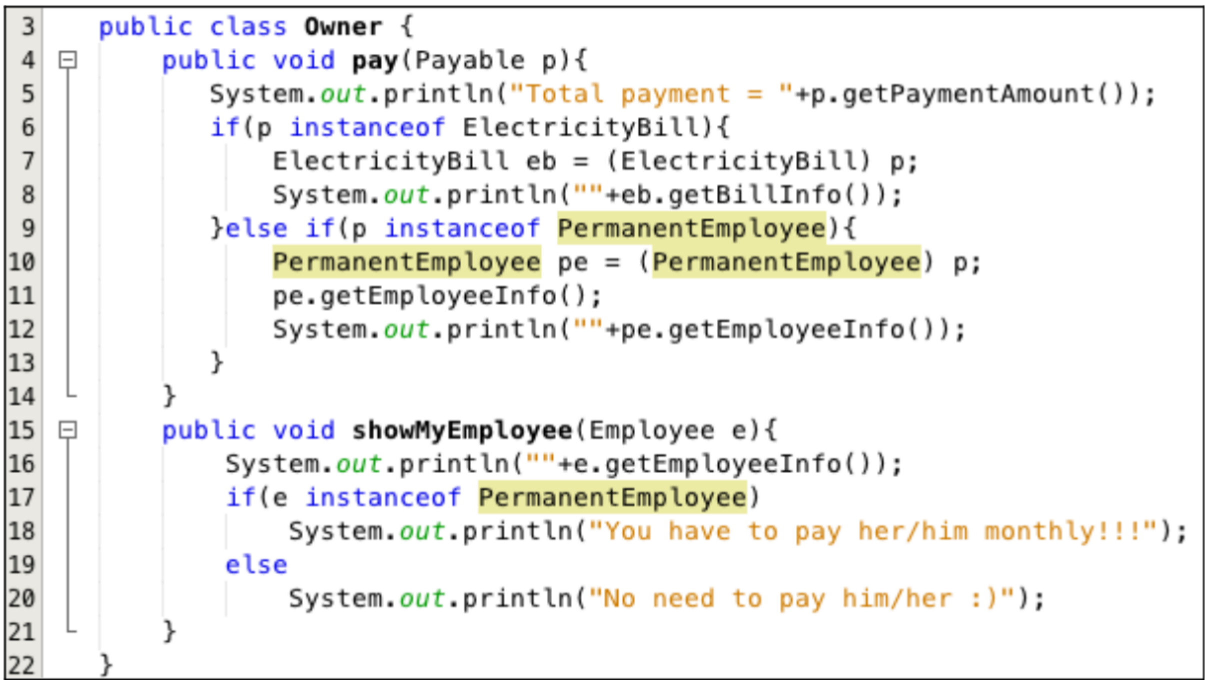
1. Why array **`e`** at line 8, can be filled with different object type?
2. Why array **`p`** at line 9, can be filled with different object type?
3. Look at line 10, why it returned an error?

7. Experiment 4 – Polymorphism argument, instanceof and object casting

7.1. Steps

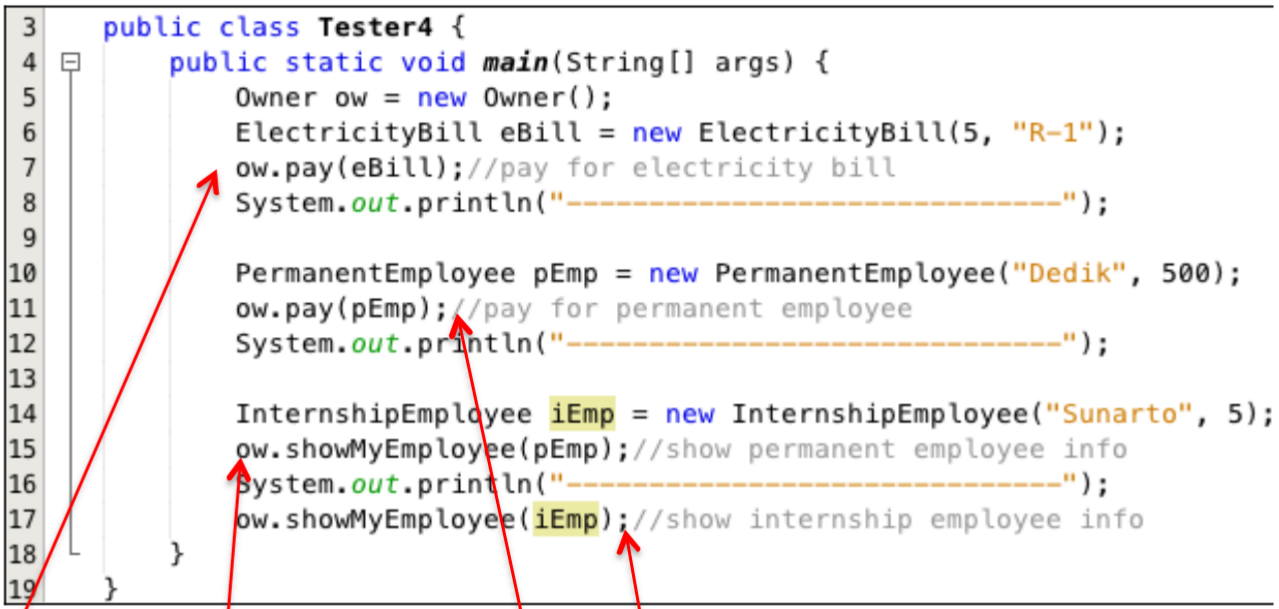
1. In this section we still use the remaining project.
2. Make a new class called **`Owner`**. **`Owner`** can pay an electrical bill and their employee. It represented in method **`pay()`**. Otherwise, it can show the employee data at method **`showMyEmployee()`**.





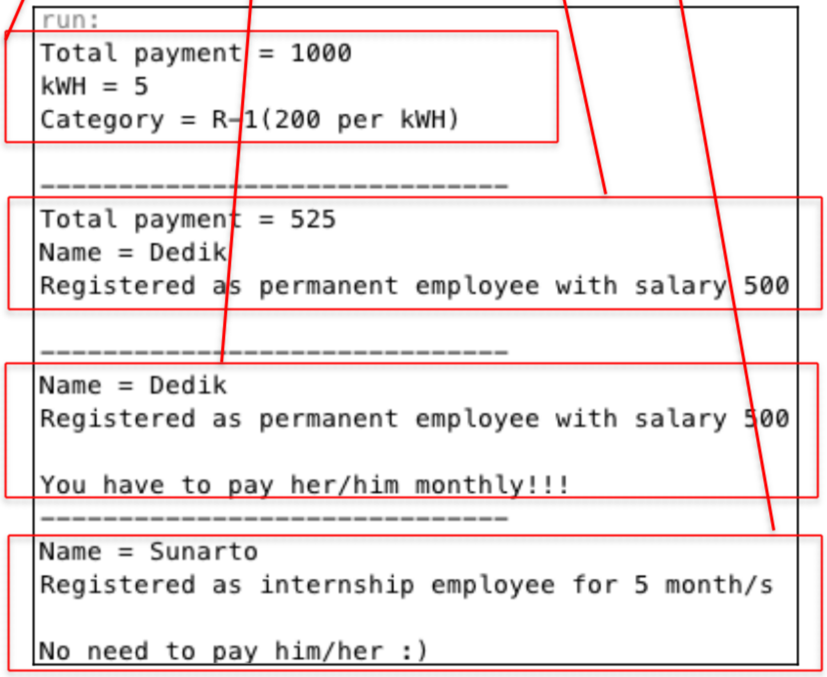
```java
public class Owner {
    public void pay(Payable p){
        System.out.println("Total payment = "+p.getPaymentAmount());
        if(p instanceof ElectricityBill){
            ElectricityBill eb = (ElectricityBill) p;
            System.out.println(""+eb.getBillInfo());
        }else if(p instanceof 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 :)");
    }
}
```

2. Create a new class **`Tester4`**.



```
 3  public class Tester4 {
 4      public static void main(String[] args) {
 5          Owner ow = new Owner();
 6          ElectricityBill eBill = new ElectricityBill(5, "R-1");
 7          ow.pay(eBill);//pay for electricity bill
 8          System.out.println("-------------------------------");
 9  
10          PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
11          ow.pay(pEmp);//pay for permanent employee
12          System.out.println("-------------------------------");
13  
14          InternshipEmployee iEmp = new InternshipEmployee("Sunarto", 5);
15          ow.showMyEmployee(pEmp);//show permanent employee info
16          System.out.println("-------------------------------");
17          ow.showMyEmployee(iEmp);//show internship employee info
18      }
19  }
```

3. Run class **`Tester4`**, it will be resulted like this below:

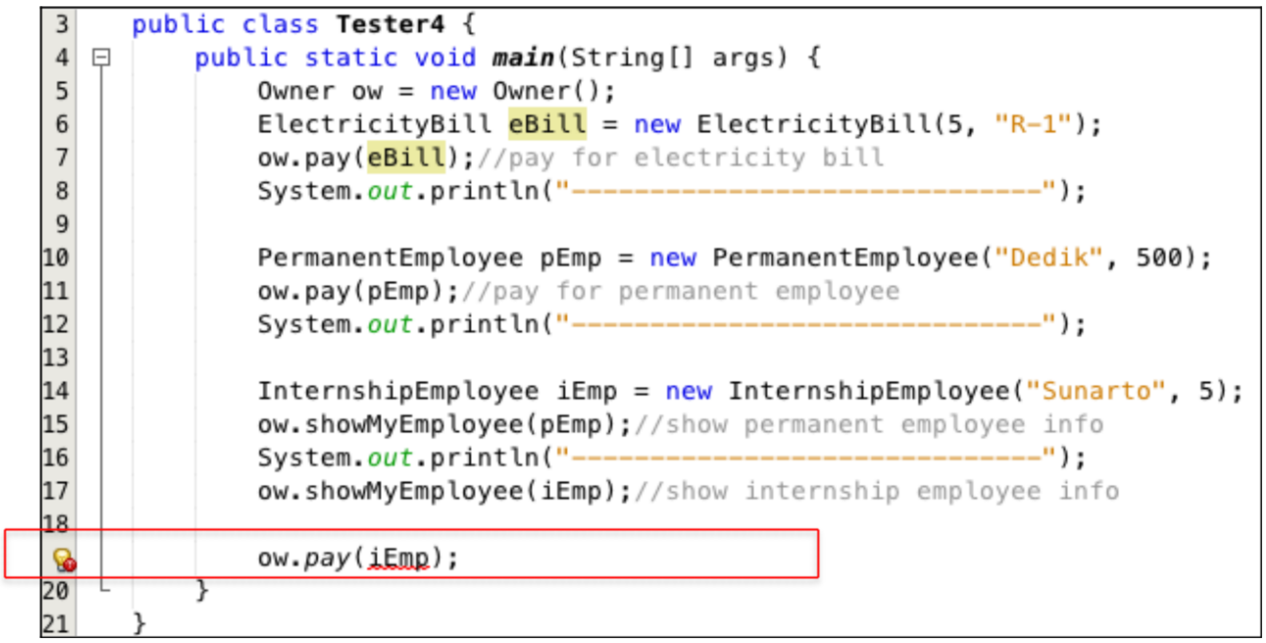


```
run:
Total payment = 1000
kWH = 5
Category = R-1(200 per kWH)

------------------------------
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 :)
```

7.2. Question

1. At class **`Tester4`**, in line 7 and 11, why we can call method **`ow.pay(eBill)`** and **`ow.pay(pEmp)`**, eventhough **`pay()`** at class **`Owner`** has an argument type **`Payable`**?

2. What is the purpose we make an argument typed **`Payable`** at method **`pay()`** on class **`Owner`**?
3. Add this line of code:



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

        PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
        ow.pay(pEmp);//pay for permanent employee
        System.out.println("------------------------------");

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

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

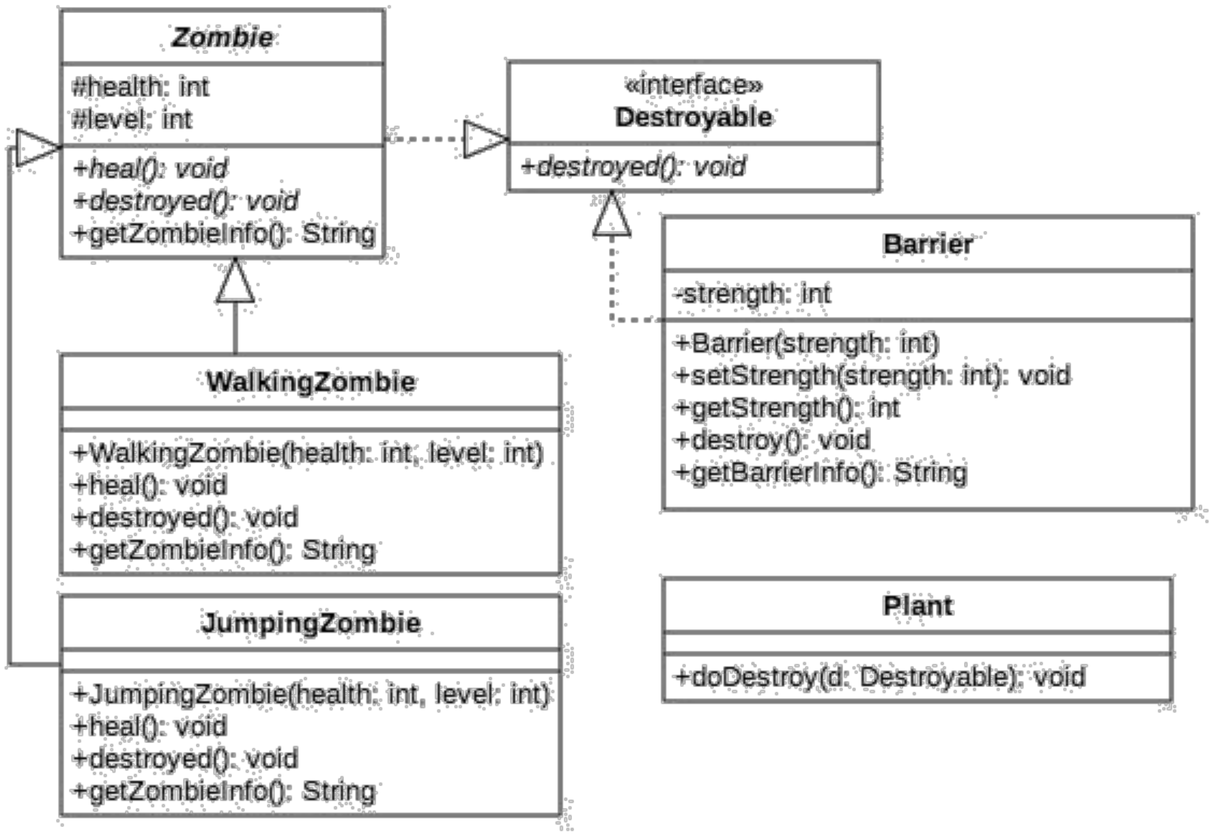
   Why error happened?
4. Look at class **`Owner`**, is it important we add **`p instanceof ElectricityBill`** at line 6 ?
5. Is still needed to cast this object (**`ElectricityBill eb = (ElectricityBill) p`**)? Why object **`p`** typed **`Payable`** must be casted into object **`eb?`**

8. Assignment

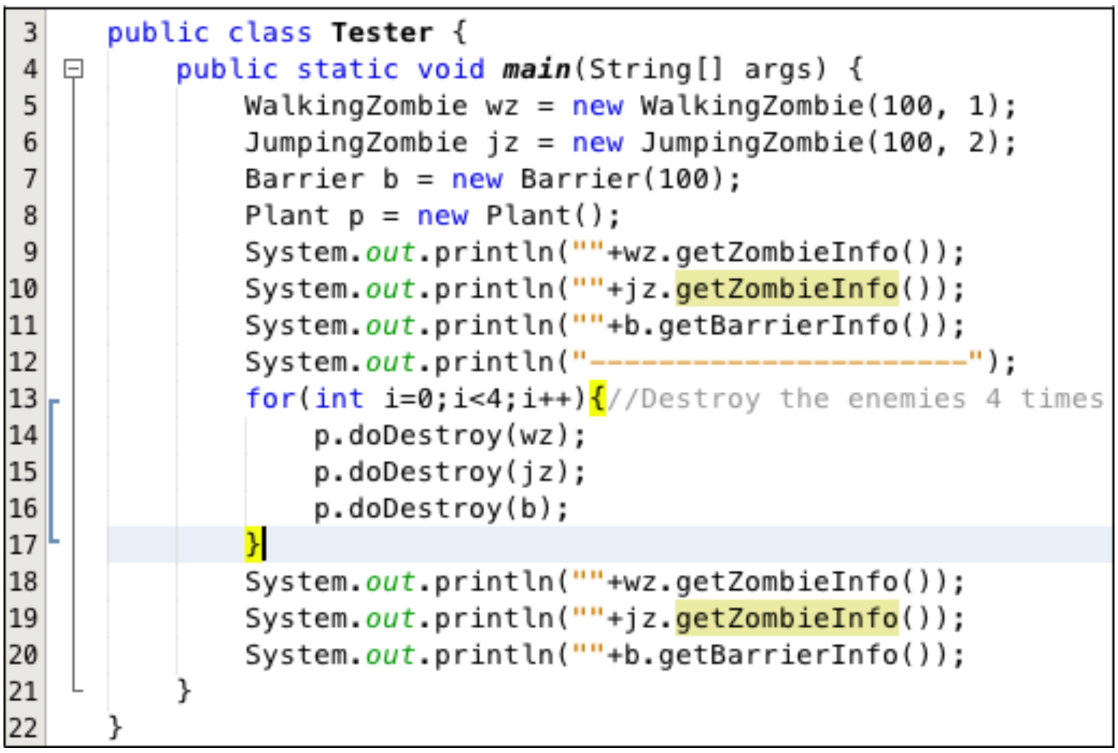
In a game, zombie and barrier can be destroyed by a plant and can be heal itself. There is two type of zombie, walking zombie and jumping zombie. They has their healing method, also way to destroy them like this:

- Walking Zombie
  - Heal: Based on zombie level
    - Zombie level 1, health increased 20%
    - Zombie level 2, health increased 30%
    - Zombie level 3, health increased 40%
  - Destroy: every attack received, health decreased 2%
- Jumping Zombie
  - Heal: Based on zombie level
    - Zombie level 1, health increased 30%
    - Zombie level 2, health increased 40%
    - Zombie level 3, health increased 50%
  - Destroy : every attack received, health decreased 1%

Make a program based on this description !

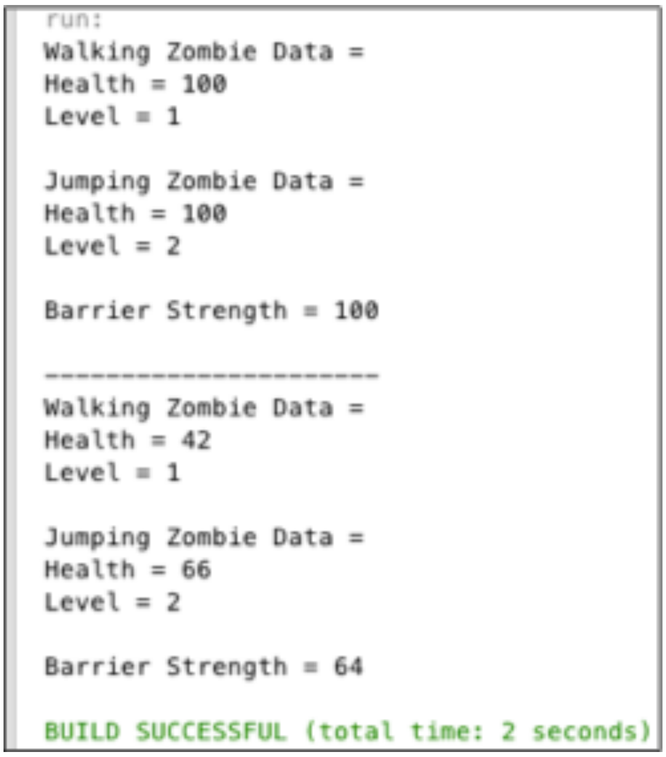


Example: if class Tester like this one:



```
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++){//Destroy the enemies 4 times
            p.doDestroy(wz);
            p.doDestroy(jz);
            p.doDestroy(b);
        }
        System.out.println(""+wz.getZombieInfo());
        System.out.println(""+jz.getZombieInfo());
        System.out.println(""+b.getBarrierInfo());
    }
}
```

The result will be like this:



```
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)
```