

## Object Oriented Programming Job Sheet polymorphism



**From:**

AL AZHAR RIZQI RIFA'I FIRDAUS

**Class:**

2 I

**Absence:**

01

**Student Number Identity:**

2241720263

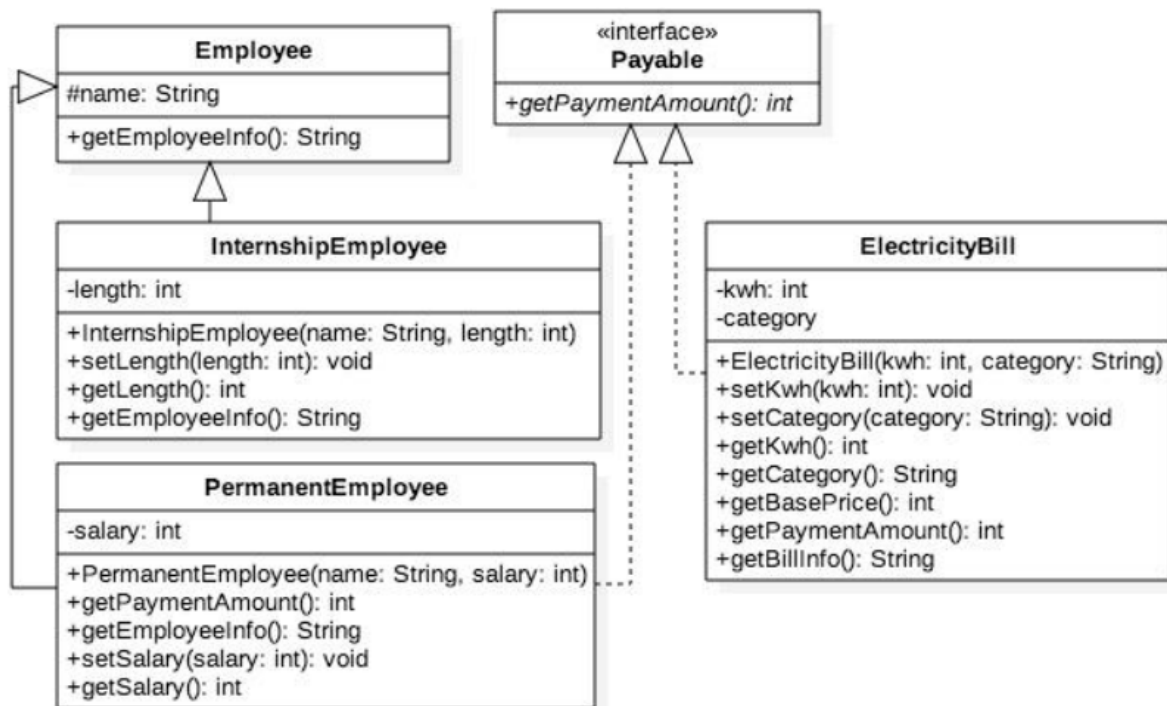
**Department:**

Information Technology

**Study Program:**

Informatics Engineering

## Case Study



## Experiment 1

Code:

src > main > java > com > azhar > exp > ☕ Employee.java > ...

```
1  package com.azhar.exp;
2
3  public class Employee {
4      protected String name;
5
6      public String getEmployeeInfo() {
7          return "Name = " + name;
8      }
9  }
10
```

src > main > java > com > azhar > exp > ☕ Payable.java > 🔗 Payable

```
1  package com.azhar.exp;
2
3  public interface Payable {
4
5      💡 public int getPaymentAmount();
6
7  }
```

```
src > main > java > com > azhar > exp > 📄 InternshipEmployee.java > 📁 InternshipEmployee > 📄 getEmployeeInfo()
1  package com.azhar.exp;
2
3  public class InternshipEmployee extends Employee {
4
5      private int length;
6
7      public InternshipEmployee(String name, int length) {
8          this.name = name;
9          this.length = length;
10     }
11
12     public int getLength() {
13         return length;
14     }
15
16     public void setLength(int length) {
17         this.length = length;
18     }
19
20     @Override
21     public String getEmployeeInfo() {
22         String info = super.getEmployeeInfo() + "\n";
23         info += "Registered as internship employee for " + length + " months/s\n";
24         return info;
25     }
26
27 }
28
```

```
src > main > java > com > azhar > exp > PermanentEmployee.java > PermanentEmployee > getPaymentAmount()
1  package com.azhar.exp;
2
3  public class PermanentEmployee extends Employee implements Payable {
4
5      private int salary;
6
7      public PermanentEmployee(String name, int salary) {
8          this.name = name;
9          this.salary = salary;
10     }
11
12     public int getSalary() {
13         return salary;
14     }
15
16     public void setSalary(int salary) {
17         this.salary = salary;
18     }
19
20     @Override
21     public int getPaymentAmount() {
22         return (int) (salary + 0.05 * salary);
23     }
24
25     @Override
26     public String getEmployeeInfo() {
27         String info = super.getEmployeeInfo() + "\n";
28         info += "Registered as permanent employee with salary " + salary + "\n";
29         return info;
30     }
31
32 }
33
```

```
src > main > java > com > azhar > exp > 📄 ElectriccityBill.java > 🔗 ElectriccityBill
1  package com.azhar.exp;
2  📌
3  public class ElectriccityBill implements Payable {
4      private int kwh;
5      private String category;
6
7      public ElectriccityBill(int kwh, String category) {
8          this.kwh = kwh;
9          this.category = category;
10     }
11
12     public int getKwh() {
13         return kwh;
14     }
15
16     public void setKwh(int kwh) {
17         this.kwh = kwh;
18     }
19
20     public String getCategory() {
21         return category;
22     }
23
24     public void setCategory(String category) {
25         this.category = category;
26     }
27
28     @Override
29     public int getPaymentAmount() {
30         return kwh * getBasePrice();
31     }
32
33     public int getBasePrice() {
34         int bPrice = 0;
35         switch (category) {
36             case "R-1":
37                 bPrice = 100;
38                 break;
39             case "R-2":
40                 bPrice = 200;
41                 break;
```

```

42     }
43     return bPrice;
44 }
45
46 public String getBillInfo() {
47     return "kWh = " + kWh + "\n" + "Category = " + category + "(" + getBasePrice() + " per kWh)\n";
48 }
49 }
50

```

src > main > java > com > azhar > exp >  Tester1.java > ...

```

1  package com.azhar.exp;
2
3  public class Tester1 {
4      Run | Debug
5      public static void main(String[] args) {
6          PermanentEmployee pEmp = new PermanentEmployee(name:"Dedik", salary:500);
7          InternshipEmployee iEmp = new InternshipEmployee(name:"Sunarto", length:5);
8          ElectriccityBill eBill = new ElectriccityBill(kwh:5, category:"A-1");
9          Employee e;
10         Payable p;
11         e = pEmp;
12         e = iEmp;
13         p = pEmp;
14         p = eBill;
15     }
16 }

```

## Questions

1. Which class is an instance of the Employee class?

Classes that inherit from the Employee class are:

- InternshipEmployee
- PermanentEmployee

2. Which class implements the Payable interface?

Classes that implement the Payable interface are:

- PermanentEmployee
- ElectriccityBill

3. Consider class Tester1, lines 10 and 11. Why e, can be filled with object pEmp (an object of class PermanentEmployee) and object iEmp (an object of class InternshipEmployee)?

- In the Tester1 class, lines 10 and 11 assign objects to e. Both pEmp (an object of the PermanentEmployee class) and iEmp (an object of the InternshipEmployee class) can be assigned to e because Java allows a reference variable of a superclass (or interface) type to hold an object of any subclass (or class implementing the interface). This is called polymorphism through inheritance.

4. Consider class Tester1, lines 12 and 13. Why can p be filled with pEmp object (an object of class PermanentEmployee) and eBill object (an object of class ElectricityBill)?

- In the Tester1 class, lines 12 and 13 assign objects to p. Both pEmp (an object of the PermanentEmployee class) and eBill (an object of the ElectriccityBill class) can be assigned to p because both classes implement the Payable interface. Java allows an interface reference

variable to refer to any object of a class that implements that interface. This is polymorphism through interfaces.

5. Try adding the syntax:

`p = iEmp;`

`e = eBill;`

on lines 14 and 15 (the last line in the main method)! What cause the error?

- Adding the syntax `p = iEmp;` at line 14 will cause an error because `iEmp` is an object of the `InternshipEmployee` class, which does not implement the `Payable` interface. Similarly, adding `e = eBill;` at line 15 will result in an error because `e` is a reference variable of type `Employee` and cannot directly refer to an object of class `ElectriccityBill`.

6. Give a conclusion about the basic concept/form of polymorphism!

Conclusion about the concept/basic form of polymorphism:

- Polymorphism in Java allows objects of different classes to be treated as objects of a common superclass or interface.
- Through inheritance, a superclass reference variable can refer to any subclass object, enabling code flexibility and reusability.
- Through interfaces, a reference variable of an interface type can refer to any object of a class implementing that interface, allowing for a standardized way to interact with different types of objects.

## Experiment 2

Code:



```
src > main > java > com > azhar > exp > Tester2.java > ...
```

```
1 package com.azhar.exp;
2
3 public class Tester2 {
4     Run | Debug
5     public static void main(String[] args) {
6         PermanentEmployee pEmp = new PermanentEmployee(name:"Dedik", salary:500);
7         Employee e;
8         e = pEmp;
9         System.out.println(" " + e.getEmployeeInfo());
10        System.out.println("-----");
11        System.out.println(" " + pEmp.getEmployeeInfo());
12    }
13 }
```

```
→ zharsuke@box ~/Documents/College/Semester_3/oop/meet-10/coding git:(master) x
sInExceptionMessages -cp /home/zharsuke/Documents/College/Semester_3/oop/meet-10/c
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Name = Dedik
Registered as permanent employee with salary 500

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

→ zharsuke@box ~/Documents/College/Semester_3/oop/meet-10/coding git:(master) x
```

## Questions

1. Consider the Tester2 class above, why do the calls `e.getEmployeeInfo()` on line 8 and `pEmp.getEmployeeInfo()` on line 10 return the same result?

- In the given code snippet, both `e.getEmployeeInfo()` and `pEmp.getEmployeeInfo()` produce the same result because `e` and `pEmp` reference the same object, specifically an instance of `PermanentEmployee`. When `e = pEmp;` is executed, `e` refers to the `pEmp` object. Despite `e` being of type `Employee`, due to polymorphism, the method invocation follows the actual implementation in the `PermanentEmployee` class.

2. Why is the `e.getEmployeeInfo()` method call referred to as a virtual method invocation, whereas `pEmp.getEmployeeInfo()` is not?




- The method call `e.getEmployeeInfo()` is referred to as a virtual method invocation because, at runtime, Java determines the actual method implementation to be executed based on the real object type that `e` refers to (polymorphism). On the other hand, `pEmp.getEmployeeInfo()` is not termed virtual because it directly calls the method on the `pEmp` object without any substitution or inheritance involved.

3. So what is a virtual method invocation? Why is called virtual?

- The method call `e.getEmployeeInfo()` is referred to as a virtual method invocation because, at runtime, Java determines the actual method implementation to be executed based on the real object type that `e` refers to (polymorphism). On the other hand, `pEmp.getEmployeeInfo()` is not termed virtual because it directly calls the method on the `pEmp` object without any substitution or inheritance involved.

## Experiment 3

Code:

```
src > main > java > com > azhar > exp >  Tester3.java >  Tester3 >  main(String[])
1  package com.azhar.exp;
2
3  public class Tester3 {
4      public static void main(String[] args) {
5          PermanentEmployee pEmp = new PermanentEmployee(name:"Dedik", salary:500);
6          InternshipEmployee iEmp = new InternshipEmployee(name:"Sunarto", length:5);
7          ElectriccityBill eBill = new ElectriccityBill(kwh:5, category:"A-1");
8          Employee e [] = {pEmp, iEmp};
9          Payable p [] = {pEmp, eBill};
10         Employee e2 [] = {pEmp, iEmp, eBill};
11     }
12 }
13
```

## Questions

1. Look at array e on line 8, why can it be filled with objects of different types, namely object pEmp (object of PermanentEmployee) and object iEmp (object of InternshipEmployee)?

- The array e on line 8 can contain objects of different types, such as pEmp (an object of type PermanentEmployee) and iEmp (an object of type InternshipEmployee), due to polymorphism. Both PermanentEmployee and InternshipEmployee classes extend from Employee, allowing an array of Employee type to hold references to objects of its subclasses.

2. Notice also the 9th row, why is the p array also filled with objects of different types, namely the pEmp object (object of PermanentEmployee) and the eBill object (object of ElectricityBilling)?

- Similarly, on line 9, the array p can contain different types of objects like pEmp (an object of type PermanentEmployee) and eBill (an object of type ElectricityBill). This is possible because both classes PermanentEmployee and ElectricityBill implement the Payable interface, allowing an array of Payable type to hold references to objects of classes that implement the Payable interface.

3. Look at line 10, why is there an error?

- The error occurs on line 10 because the array e2 attempts to contain objects of different types (pEmp, iEmp, and eBill) directly without being grouped under a common superclass or interface. Unlike the previous arrays (e and p), which hold references to objects of related types (Employee and Payable), e2 doesn't have a unifying type, resulting in a compilation error.

## Experiment 4

Code:

src > main > java > com > azhar > exp > Owner.java > Owner > pay(Payable)

```
1 package com.azhar.exp;
2
3 public class Owner {
4     public void pay(Payable p) {
5         System.out.println("Total payment = " + p.getPaymentAmount());
6         if (p instanceof ElectriccityBill) {
7             ElectriccityBill eb = (ElectriccityBill) p;
8             System.out.println("" + eb.getBillInfo());
9         } else if (p instanceof PermanentEmployee) {
10             PermanentEmployee pe = (PermanentEmployee) p;
11             pe.getEmployeeInfo();
12             System.out.println("" + pe.getEmployeeInfo());
13         }
14     }
15
16     public void showMyEmployee(Employee e) {
17         System.out.println("" + e.getEmployeeInfo());
18         if (e instanceof PermanentEmployee) {
19             System.out.println("You have to pay her/him monthly!!!");
20         } else {
21             System.out.println("No need to pay him/her :)");
22         }
23     }
24 }
25
```

```

src > main > java > com > azhar > exp > Tester4.java > Tester4 > main(String[])
1  package com.azhar.exp;
2
3  public class Tester4 {
4      public static void main(String[] args) {
5          Owner ow = new Owner();
6          ElectricityBill eBill = new ElectricityBill(kwh:5, category:"R-1");
7          ow.pay(eBill);
8          System.out.println("-----");
9
10         PermanentEmployee pEmp = new PermanentEmployee(name:"Dedik", salary:500);
11         ow.pay(pEmp);
12         System.out.println("-----");
13
14         InternshipEmployee iEmp = new InternshipEmployee(name:"Sunarto", length:5);
15         ow.showMyEmployee(iEmp);
16         System.out.println("-----");
17         ow.showMyEmployee(iEmp);
18     }
19 }
20

```

```

→ zharsuke@box ~/Documents/College/Semester_3/oop/meet-10/coding git:(master) x
p /home/zharsuke/Documents/College/Semester_3/oop/meet-10/coding/target/classes c
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Total payment = 500
kWH = 5
Category = R-1(100 per kWH)

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

-----
Name = Sunarto
Registered as internship employee for 5 months/s

No need to pay him/her :)
-----
Name = Sunarto
Registered as internship employee for 5 months/s

No need to pay him/her :)
→ zharsuke@box ~/Documents/College/Semester_3/oop/meet-10/coding git:(master) x

```

## Questions

1. Look at class Tester4 line 7 and line 11, why are the the calls `ow.pay(eBill)` and `ow.pay(pEmp)` can be made, even though if you look at the `pay()` method in the `Owner` class has an argument/parameter of type `Payable`? If we take a closer look, `eBill` is an object of `ElectricityBill` and `pEmp` is an object of `PermanentEmployee`?

- In Tester4, the calls `ow.pay(eBill)` and `ow.pay(pEmp)` are possible because `ElectriccityBill` and `PermanentEmployee` both implement the `Payable` interface. Even though the `pay` method in the `Owner` class specifies a `Payable` type argument, it can accept instances of classes that implement the `Payable` interface.

2. So what is the purpose of creating an argument of type Payable in the method pay() method in the Owner class?

- The purpose of using a Payable type argument in the pay() method of the Owner class is to enable the method to accept any object that implements the Payable interface. This enhances flexibility, allowing different types of payment-related objects to be processed uniformly through a common interface.

3. Try on the last line of the main() method inside the class Tester4, add the command ow.pay(iEmp);

```
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        ow.pay(iEmp);
20    }
21 }
```

Why does an error occur?

- Adding ow.pay(iEmp); to the end of the main() method in Tester4 results in an error because InternshipEmployee doesn't implement the Payable interface. Therefore, the pay method in Owner cannot accept an InternshipEmployee object as an argument.

4. Look at the Owner class, what is the syntax required for p instanceof ElectricityBill on line 6?

- The instanceof check (p instanceof ElectricityBill) in the Owner class is used to determine whether the object referenced by p is an instance of the ElectricityBill class. This check helps in deciding how to process or handle the object inside the pay method based on its type.

5. Look at the Owner class again in line 7, what is the object casting for there (ElectricityBill eb = (ElectricityBill) p) is required? Why does object p of type Payable need to be cast to into an eb object of type ElectricityBill?

- The casting (ElectricityBill eb = (ElectricityBill) p) in Owner class is necessary to explicitly convert the object referenced by p (of type Payable) to an ElectricityBill object. This is needed because at compile time, the pay method expects a Payable type argument, but at runtime, it may receive objects of specific implementing classes of Payable. Therefore, explicit casting is used to treat the object as an ElectricityBill inside the method.

## Assignment

In a game, Zombies and Barriers can be destroyed by Plants and can heal themselves. There are two types of Zombies, Walking Zombies and Jumping Zombie. Both Zombies have different ways of healing different ways of healing, as well as the way they are destroyed, which is determined by the following rules the following rules:

- Walking Zombies

o Healing: Healing is determined based on the level of the zombie level

§§ If the zombie is level 1, then every time it heals, health will increase by 20%

§§ If the zombie is level 2, then every time it heals, health will increase by 30%

§§ If the zombie is level 3, then each time it heals, health will increase by 40%

o Destruction: each time you destroy, health will be reduced by 2%

- On Jumping Zombie

o Healing: Healing is determined based on the level of the zombie level

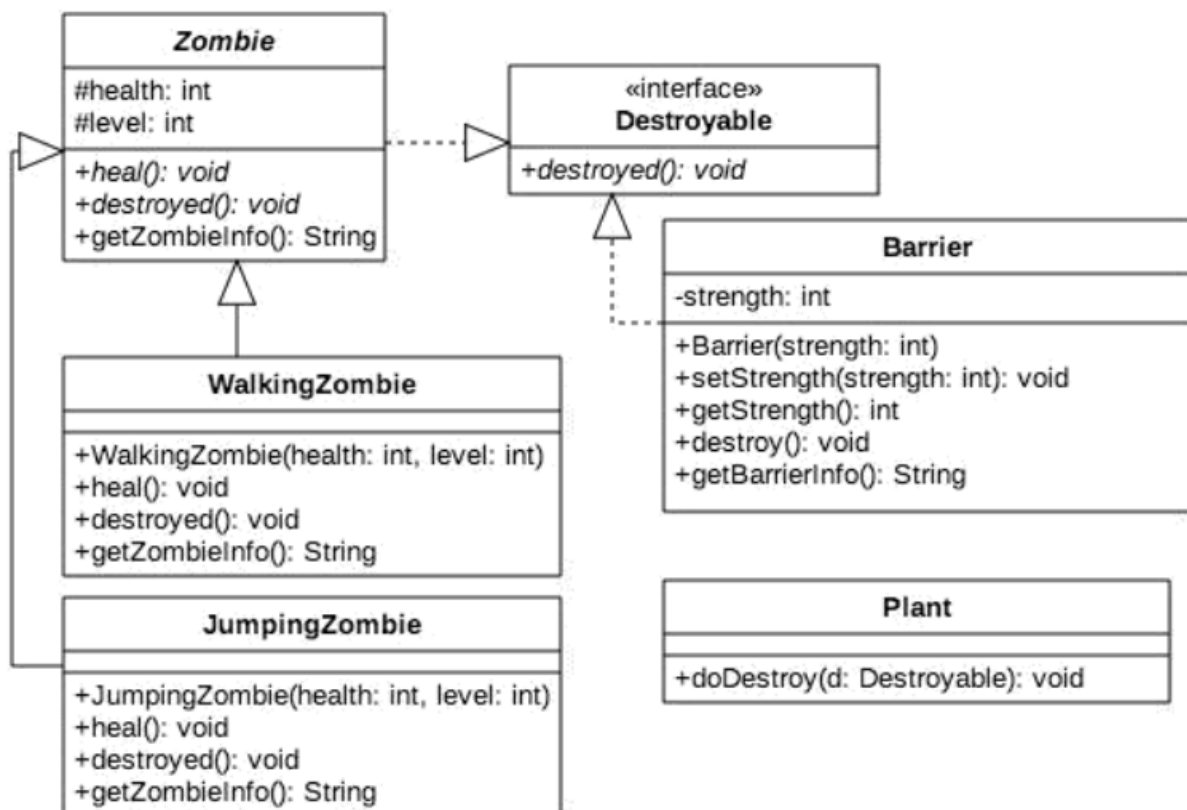
§§ If the zombie is level 1, then every time it heals, health will increase by 30%

§§ If the zombie is level 2, then every time it heals, health will increase by 40%

§§ If the zombie is level 3, then each time it heals, health will increase by 50%

o Destruction: each time you destroy, health will decrease by 1%

Create a program from the class diagram below!



```
src > main > java > com > azhar > asg > ☕ Destroyable.java > ...
```

```
1 package com.azhar.asg;
2
3 public interface Destroyable {
4     public void destroyed();
5 }
6
```

```
src > main > java > com > azhar > asg > ☕ Zombie.java > 🧟 Zombie > 📦 health
```

```
1 package com.azhar.asg;
2
3 public class Zombie implements Destroyable {
4     ⚡ protected int health;
5     protected int level;
6
7     public void heal() {
8         if (this.level == 1) {
9             this.health += this.health * 0.1;
10        } else if (this.level == 2) {
11            this.health += this.health * 0.3;
12        } else if (this.level == 3) {
13            this.health += this.health * 0.4;
14        }
15    }
16
17    public void destroyed() {
18        this.health -= this.health * 0.2;
19    }
20
21    public String getZombieInfo() {
22        return "\nHealth = " + this.health + "\nLevel = " + this.level;
23    }
24
25 }
26
```




src > main > java > com > azhar > asg > Barrier.java > Barrier > getBarrierInfo()

```
1  package com.azhar.asg;
2
3  public class Barrier implements Destroyable {
4      private int health;
5
6      public Barrier(int strength) {
7          this.health = strength;
8      }
9
10     public void setStrength(int strength) {
11         this.health = strength;
12     }
13
14     public int getStrength() {
15         return this.health;
16     }
17
18     public void destroyed() {
19         this.health -= this.health * 0.1;
20     }
21
22     public String getBarrierInfo() {
23         return "\nBarrier Strength = " + this.health + "\n";
24     }
25 }
26
```



src > main > java > com > azhar > asg > WalkingZombie.java > WalkingZombie

```
1  package com.azhar.asg;
2
3  public class WalkingZombie extends Zombie {
4      public WalkingZombie(int health, int level) {
5          this.health = health;
6          this.level = level;
7      }
8
9      public void heal() {
10         if (this.level == 1) {
11             this.health += this.health * 0.1;
12         } else if (this.level == 2) {
13             this.health += this.health * 0.3;
14         } else if (this.level == 3) {
15             this.health += this.health * 0.4;
16         }
17     }
18
19     public void destroyed() {
20         this.health -= this.health * 0.19;
21     }
22
23     public String getZombieInfo() {
24         return "\nWalking Zombie Data = " + super.getZombieInfo();
25     }
26
27
28 }
29
```

src > main > java > com > azhar > asg >  JumpingZombie.java >  JumpingZombie >  getZombieInfo()

```
1  package com.azhar.asg;
2
3  public class JumpingZombie extends Zombie {
4      public JumpingZombie(int health, int level) {
5          this.health = health;
6          this.level = level;
7      }
8
9      public void heal() {
10         if (this.level == 1) {
11             this.health += this.health * 0.1;
12         } else if (this.level == 2) {
13             this.health += this.health * 0.4;
14         } else if (this.level == 3) {
15             this.health += this.health * 0.5;
16         }
17     }
18
19     public void destroyed() {
20         this.health -= this.health * 0.095;
21     }
22
23     public String getZombieInfo() {
24         return "\nJumping Zombie Data = " + super.getZombieInfo();
25     }
26
27 }
28
```

src > main > java > com > azhar > asg >  Tester.java >  Tester >  main(String[])

```
1  package com.azhar.asg;
2
3  public class Tester {
4      Run | Debug
5      public static void main(String[] args) {
6          WalkingZombie wz = new WalkingZombie(health:100, level:1);
7          JumpingZombie jz = new JumpingZombie(health:100, level:2);
8          Barrier b = new Barrier(strength:100);
9          Plant p = new Plant();
10
11         System.out.println(wz.getZombieInfo());
12         System.out.println(jz.getZombieInfo());
13         System.out.println(b.getBarrierInfo());
14         System.out.println();
15
16         for (int i = 0; i < 4; i++) {
17             p.doDestroy(wz);
18             p.doDestroy(jz);
19             p.doDestroy(b);
20         }
21
22         System.out.println(wz.getZombieInfo());
23         System.out.println(jz.getZombieInfo());
24         System.out.println(b.getBarrierInfo());
25         System.out.println();
26     }
27 }
```

```
src > main > java > com > azhar > asg > Plant.java > Plant > doDestroy(Destroyable)
1  package com.azhar.asg;
2
3  public class Plant {
4      public void doDestroy(Destroyable d) {
5          d.destroyed();
6      }
7  }
8
```

```
→ zharsuke@box ~/Documents/College/Semester_3/oop/meet-10/coding git:(master) x
nExceptionMessages -cp /home/zharsuke/Documents/College/Semester_3/oop/meet-10/cod
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
```

```
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
```

```
→ zharsuke@box ~/Documents/College/Semester_3/oop/meet-10/coding git:(master) x
```