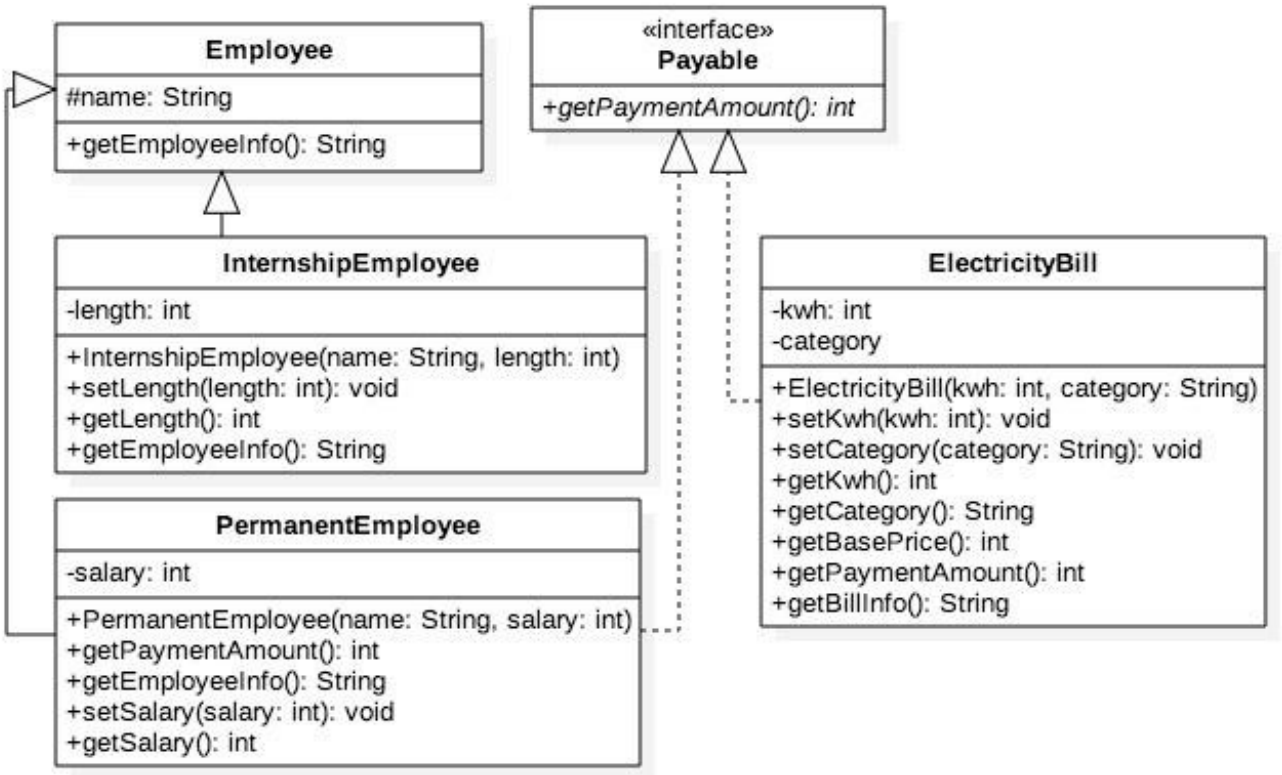
JOBSHEET

POLIMORFISME

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. Percobaan 1 – Bentuk dasar polimorfisme

4. 1. Langkah Percobaan

1. Buat class **`Employee`**

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

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

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

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

2. Buat interface **`Payable`**

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

```java
public interface IPayable {
    public int getPaymentAmount();
}
```

3. Buat class **`InternshipEmployee`**, subclass dari **`Employee`**

```java
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;
    }
}
```

```java
public class InternshipEmployee  extends Employee{
    private int lenght;

    public InternshipEmployee(String name, int lenght) {
        this.lenght = lenght;
        this.name = name;
    }

    public int getLenght() {
        return lenght;
    }

    public void setLenght(int lenght) {
        this.lenght = lenght;
    }

    @Override
    public String getEmployeeInfo() {
        String info = super.getEmployeeInfo() + "\n";
        info += "Reggistered as internship employee for " + lenght + "month/s\n";
        return info;
    }
}
```

4. Buat class **`PermanentEmployee`**, subclass dari **`Employee`** dan implements ke **`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;
    }
}
```

```java
public class PermanentEmployee extends Employee implements IPayable {
    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;    The assignment to variable salary has no effect
    }

    @Override
    public int getPaymentAmount() {
        return (int) (salary + 0.5 * salary);
    }

    @Override
    public String getEmployeeInfo() {
        String info = super.getEmployeeInfo() + "\n";
        info += "Registered as permanent employee with salary" + salary + "\n";
        return info;
    }
}
```

5. Buatt class **`ElectricityBill`** yang implements ke interface **`Payable`**

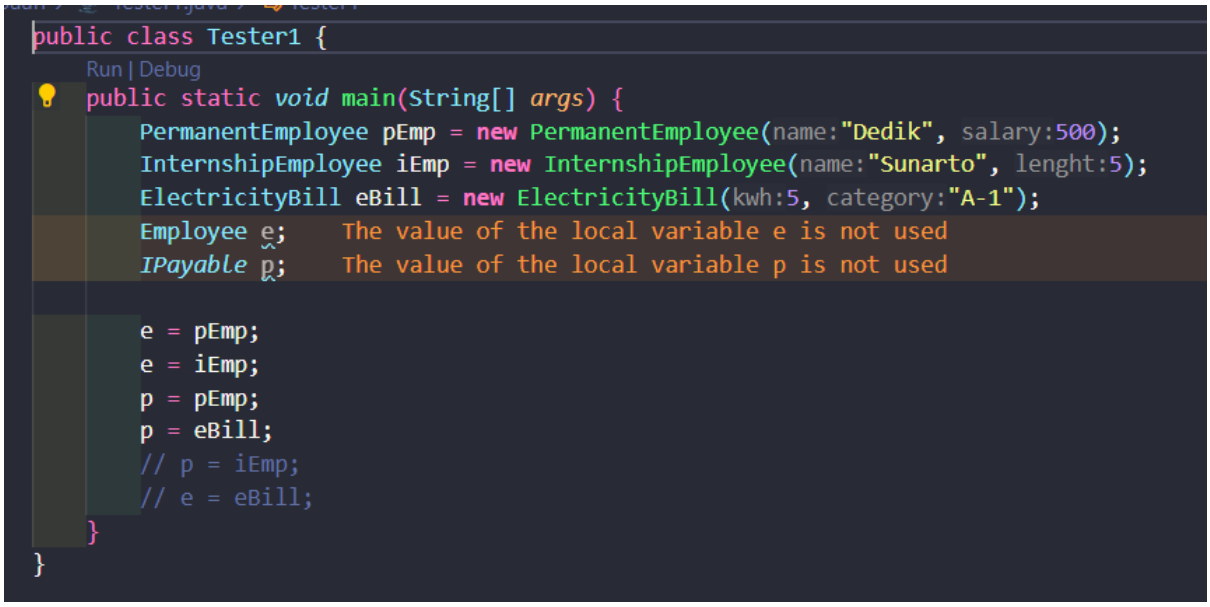
```java
public class ElectricityBill implements Payable{
    private int kwh;
    private String category;

    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";
    }
}
```

```java
public class ElectricityBill implements IPayable {
    private int kwh;
    private String category;
    public ElectricityBill(int kwh, String category) {
        this.kwh = kwh;
        this.category = category;
    }
    public int getKwh() {
        return 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() + "peer kWh)\n";
    }
}
```

6. Buat class **`Tester1`**

```
3   public class Tester1 {
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");
8           Employee e;
9           Payable p;
10          e = pEmp;
11          e = iEmp;
12          p = pEmp;
13          p = eBill;
14      }
15  }
```



4.2. Pertanyaan

1. Class apa sajakah yang merupakan turunan dari class **`Employee`**?
   Jawab : Class InternshipEmplouyee dan PermanentEmployee

2. Class apa sajakah yang implements ke interface **`Payable`**?
   Jawab : ElectricityBill dan 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`**) ?
   `Jawab` : Karena class pEmp dan iEmp merupakan class turunan dari Employee. Dan e merupakan objek Employee.

4. Perhatikan class **`Tester1`**, baris ke-12 dan 13. Mengapa **`p`**, bisa diisi Denganobjek **`pEmp`** (merupakan objek dari class **`PermanentEmployee`**) dan objek **`eBill`** (merupakan objek dari class **`ElectricityBill`**) ?
   `Jawab` : Karena class PermanentEmployee dan ElectricityBill mengimplement class Payable.

5. Coba tambahkan sintaks: **`p = iEmp;`**

   `e = eBill;`

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

   Jawab : Karena objek InternshipEmployee payable tidak mengimplement class IPayable dan class e Bill bukan turunan dari class employee

6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!
7. Polimorfisme adalah kemampuan untuk menggunakan satu antarmuka atau nama Metode yang sama untuk berbagai tipe objek atau kelas yang berbeda. Polimorfisme memungkin objek yang berasal dari kelas yang berbeda untuk merespons panggilan Metode dengan perilaku yang sesuai dengan jeis objek tersebut.

5. Percobaan 2 – Virtual method invocation

5. 1. 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`**.

```
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. 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
```

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

```
Name = Dedik
Registered as permanent employee with salary500

--------------------
Name = Dedik
Registered as permanent employee with salary500
```

5.2.
Pertanyaan

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

   Jawab : Karena class permanent Employee merupakan turunan dari class Employee.Dan terdapat method yang sama, sehingga menghasilkan output yang sama.

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

   Jawab : Karena pemanggilan method **e.getEmployeeInfo()** adalah pemanggilan method pada objek yang memiliki tingkat abstraksi yang lebih tinggi. Dan pemanggil **pEmp.getEmployeeInfo()**tidak menjadi pemanggilan virtual karena memiliki tipe yang lebih khusus.

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

   Jawab :

   Virtual method invocation adalah proses pemanggilan method pada objek berdasarkan tipe objek yang sebenarnya, bukan tipe variabel referensi yang digunakan untuk mengakses objek tersebut. Itu disebut "virtual" karena method yang akan dipanggil ditentukan secara dinamis selama runtime, sehingga tidak tergantung pada tipe variabel referensi yang digunakan. Hal ini memungkinkan polimorfisme, di mana Anda dapat memperlakukan objek

dari berbagai kelas yang berbeda secara seragam, mengikuti prinsip "program ke antarmuka, bukan ke implementasi".

6. Percobaan 3 – Heterogenous Collection

6. 1. Langkah Percobaan

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

2. Buat class baru **`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  }
```

```
public class Tester3 {
    Run | Debug
    public static void main(String[] args) {
        PermanentEmployee pEmp = new PermanentEmployee(name:"Dedik", salary:500);
        InternshipEmployee iEmp = new InternshipEmployee(name:"Sunarto", lenght:5);
        ElectricityBill eBill = new ElectricityBill(kwh:5, category:"A-1");
        Employee e[] = { pEmp, iEmp };    The value of the local variable e is not used
        IPayable p[] = { pEmp, eBill };    The value of the local variable p is not used
        //Employee e2[] = { pEmp, iEmp, eBill };
    }
}
```

6.2. Pertanyaan

1. Perhatikan array **`e`** pada baris ke-8, mengapa ia bisa diisi dengan objekobjek dengan tipe yang berbeda, yaitu objek **`pEmp`** (objek dari **`PermanentEmployee`**) dan objek **`iEmp`** (objek dari **`InternshipEmployee`**) ?

   `Jawab :` Karena array e merupakan objek dari class employee dan pEmp dan iEmp merupakan objek dari class turunan Employee

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`**) ?

   `Jawab :` Karena object pEmp dari class PermanentEmployee dan object eBill dari class ElectricityBilling menginterface kelas Payable

3. Perhatikan baris ke-10, mengapa terjadi error?
   Jawab : Karena objek eBill bukan turunan dari class Employee.

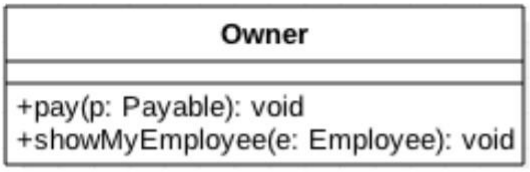
7. Percobaan 4 – Argumen polimorfisme, instanceof dan casting objek

7.1. Langkah Percobaan

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

4. 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()`**.




```
3   public class Owner {
4       public void pay(Payable p){
5           System.out.println("Total payment = "+p.getPaymentAmount());
6           if(p instanceof ElectricityBill){
7               ElectricityBill eb = (ElectricityBill) 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      public void showMyEmployee(Employee e){
16          System.out.println(""+e.getEmployeeInfo());
17          if(e instanceof PermanentEmployee)
18              System.out.println("You have to pay her/him monthly!!!");
19          else
20              System.out.println("No need to pay him/her :)");
21      }
22  }
```

```java
public class Owner {
    public void pay(IPayable 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(x:"You have to pay her / him monthly!!!");
        else
        System.out.println(x:"No need to pay him/her");
    }
}
```

2. Buat class baru **`Tester4`**.

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

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

        InternshipEmployee iEmp = new InternshipEmployee(name:"Sunatrto", lenght:5);
        ow.showMyEmployee(pEmp);
        System.out.println();
        ow.showMyEmployee(iEmp);

        //ow.pay(iEmp);

    }
}
```

```
Total payment =500
kWH = 5
Category = R-1(100peer kWh)

----------------------
Total payment =750
Name = Dedik
Registered as permanent employee with salary500

----------------------
Name = Dedik
Registered as permanent employee with salary500

You have to pay her / him monthly!!!

Name = Sunatrto
Reggistered as internship employee for 5month/s

No need to pay him/her
```

```
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. Jalankan class **`Tester4`**, dan akan didaptkan hasil sebagai berikut:

```
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. 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 : Karena method pay menggunakan parameter dengan tipe objek Payable. Dan object eBill, pEmp menerapkan interface dari class Payable.

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

   Jawab : Agar method pay bisa menerima nilai argument dari berbagai bentuk objek yang berhubungan dengan Payable.

3. Coba pada baris terakhir method **`main()`** yang ada di dalam class

```
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  }
```

`Tester4` ditambahkan perintah **`ow.pay(iEmp);`**

Mengapa terjadi error?

Jawab : Karena objek iEmp tidak berhubungan atau menginterface kelas Payable

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

   Jawab : Untuk mengecek objek p merupakan intansiasi dari class ElectricityBill atau bukan.

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 ?`**

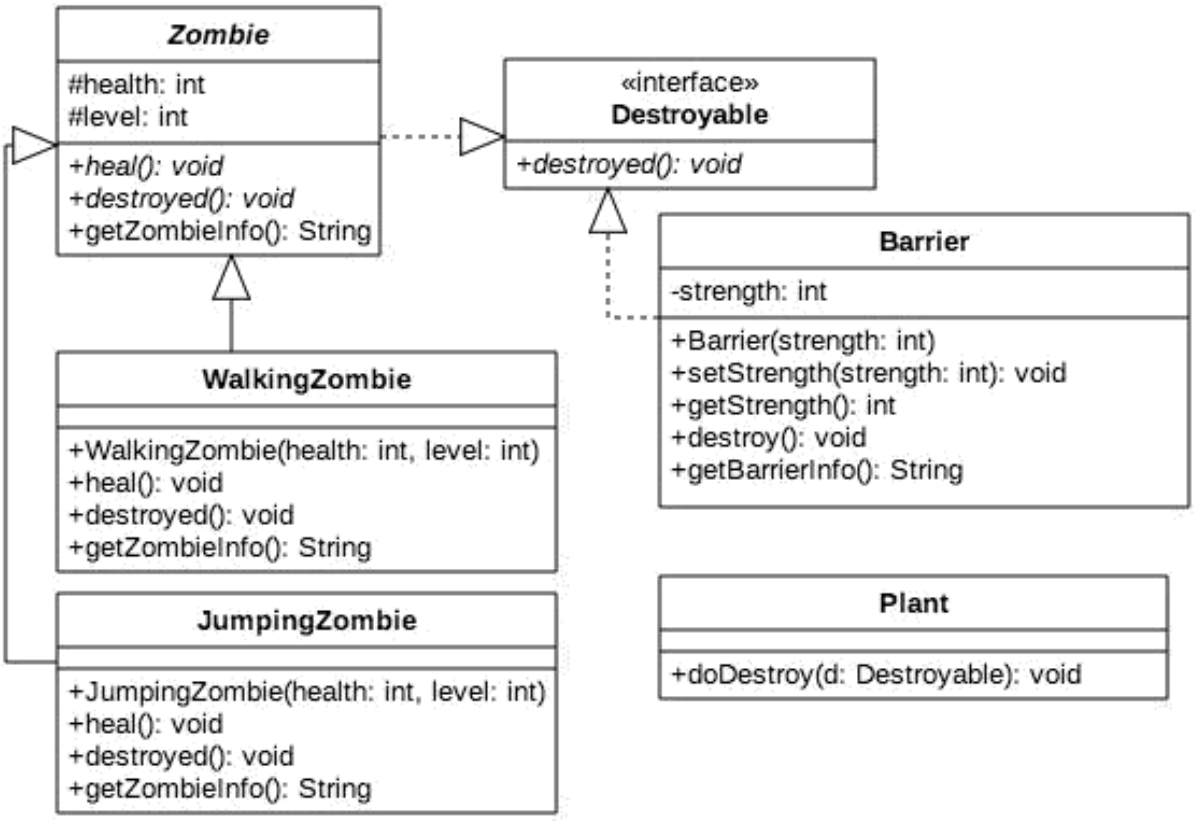
   Jawab : Untuk mengubah tipe dari objek superclass / payable ke objek dari subclass / ElectricityBill

8. 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 20%
        - 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!



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)
```

Class Zombie

```java
package Tugas;
public abstract class Zombie implements IDestroyable {
    protected int health;
    protected int level;

    public Zombie(int health, int level) {
        this.health = health;
        this.level = level;
    }
    public abstract void heal();

    public abstract void destroyed();

    public String getZombieInfo() {
        String info = "";
        info += "Health : " + health +"\n";
        info += "Level  : " + level;
        return info;
    }
}
```

Class JumpingZombie

```java
package Tugas;

public class JumpingZombie extends Zombie{
    JumpingZombie(int health, int level) {
        super(health, level);
    }

    @Override
    public void heal() {
        switch (level) {
            case 1:
                health += health * 0.3;
                break;
            case 2:
                health += health * 0.4;
                break;
            case 3:
                health += health * 0.5;
                break;
        }
    }

    @Override
    public void destroyed() {
        health -= health * 1.906;
    }
    public String getZombieInfo() {
        String info = "Jumping Zombie Data\n";
        info += "Health : " + health + "\n";
        info += "Level  : " + level;
        return info;
    }
}
```

Class Walking Zombie

```
package Tugas;

public class WalkingZombie extends Zombie {

    WalkingZombie(int health, int level) {
        super(health, level);
    }

    @Override
    public void heal() {
        switch (level) {
            case 1:
                health += health * 0.2;
                break;
            case 2:
                health += health * 0.3;
                break;
            case 3:
                health += health * 0.4;
                break;
        }
    }

    @Override
    public void destroyed() {
        health -= health * 0.19;
    }



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

Class interface Destroyable

```
package Tugas;

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

Class Plant

```
1   package Tugas;
2   public class Plant {
3       public void doDestroy(IDestroyable d) {
4           d.destroyed();
5       }
6   }
```

Class Barrier

```java
package Tugas;

public class Barrier implements IDestroyable {
    protected int strength;


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

    public int getStrength() {
        return strength;
    }

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

    @Override
    public void destroyed() {
        strength -= (0.1 * strength);
    }

    public String getBarrierInfo() {
        return "Strength: " + strength;
    }
}
```

Class Tester sebagai class main

```java
package Tugas;

public class Tester {
    Run | Debug
    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:"----------------------------\n");
        for (int i = 0; i < 4; i++) {
            p.doDestroy(wz);
            p.doDestroy(jz);
            p.doDestroy(b);
        }
        System.out.println("" + wz.getZombieInfo());
        System.out.println("" + jz.getZombieInfo());
        System.out.println("" + b.getBarrierInfo());
    }
}
```

Output :

```
Walking Zombie Data
Health : 100
Level  : 1
Jumping Zombie Data
Health : 100
Level  : 2
Strength: 100
----------------------------

Walking Zombie Data
Health : 42
Level  : 1
Jumping Zombie Data
Health : 66
Level  : 2
Strength: 64
```