**JOBSHEET**

**POLIMORFISME**

# 1. Kompetensi

Setelah melakukan percobaan pada jobsheet ini, diharapkan mahasiswa mampu:

1. Memahami konsep dan bentuk dasar polimorfisme

1. Memahami konsep virtual method invication

1. Menerapkan polimorfisme pada pembuatan heterogeneous collection

1. Menerapkan polimorfisme pada parameter/argument method

1. Menerapkan object casting untuk meng-ubah bentuk objek

# 2. Pendahuluan

Polimorfisme merupakan kemampuan suatu objek untuk memiliki banyak bentuk. Penggunaan polimorfisme yang paling umum dalam OOP terjadi ketika ada referensi super class yang digunakan untuk merujuk ke objek dari sub class. Dengan kata lain, 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 (relasi generalisasi atau IS-A).

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.

Untuk mengilustrasikan uraian di atas, diberikan contoh sebagai berikut ini. Terdapat interface **Vegetarian**, dan super class **Animal**. Kemudian dibuat class **Deer** yang merupakan sub-class dari **Animal** dan implements ke **Vegetarian**. Sedangkan class **Lion** sub-class dari **Animal**, dan tidak implements ke **Vegetarian**.

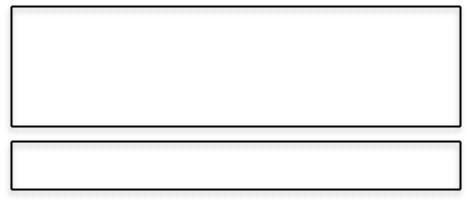
public interface Vegetarian{} public class Animal{}

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

Dari deklarasi class di atas, contoh deklarasi objek di bawah ini akan menunjukan mana yang valid dan yang tidak valid berdasarkan konsep polimorfisme.

Deer d = new Deer();

Lion l = new Lion();

Animal a = d;

Animal a2 = l; ***valid***

Vegetarian v = d;

Vegetarian v2 = l; ***tidak valid***

Dari contoh di atas, **a** (bertipe **Animal**) bisa digunakan untuk mereferensi ke objek **d** (merupakan objek dari **Deer**), karena class **Deer** merupakan turunan dari **Animal**. Demikian juga **a2** (bertipe **Animal**), juga bisa digunakan untuk mereferensi ke objek **l** (merupakan objek dari **Lion**), karena class **Lion** merupakan turunan dari **Animal**. Objek **v** (dideklarasikan dari interface **Vegetarian**) bisa juga digunakan untuk mereferensi ke objek **d** (objek dariclass **Deer**), hal ini bisa dilakukan karena class **Deer** implements ke interface **Vegetarian**.

Sedangkan objek **v2** (didekarasikan dari interface **Vegetarian**) **tidak bisa** digunakan untuk mereferensi objek **l** (objek dari class **Lion**), karena class **Lion tidak implements** ke interface **Vegetarian**. Ilustrasi tersebut bisa menunjukkan konsep dan bentuk dasar dari polimorfisme.

# Virtual method Invocation

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.

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”);

}

Ketika ada suatu objek polimorfisme **a**, misalkan:

Deer d = new Deer();

Animal a = d;

Kemudian dipanggil method overriding darinya, maka saat itu terjadi pemanggilan method virtual, seperti:

a.walk();

Saat compile time, compiler akan mengenali method **walk()** yang akan dipanggil adalah method **walk()** yang ada di class **Animal**, karena objek **a** bertipe **Animal**. Tetapi saat dijalankan (run time), maka yang dijalankan oleh JVM justru method **walk()** yang ada di class **Deer**. Akan berbeda halnya jika pemanggilan method walk() dilakukan dari objek **d** (bukan objek polimorfisme), seperti

d.walk();

maka method **walk()** yang dikenali saat compile time oleh compiler dan yang dijalankan saat runtime oleh JVM adalah sama-sama method **walk()** yang ada di class **Deer** (karena objek **d** dideklarasikan dari class **Deer**).

# Heterogeneous Collection

Dengan adanya konsep polimorfisme, maka variabel array bisa dibuat heterogen. Artinya di dalam array tersebut bisa berisi berbagai macam objek yang berbeda. Contoh:

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

Dari contoh tersebut data pertama dari array **animalArray** berisi objek **Deer**, dan data kedua dari **animalArray** berisi objek **Lion**. Hal ini bisa dilakukan karena array **animalArray** dideklarasikan dari class **Animal** (superclass dari **Deer** dan **Lion**).

# Polymorphic Argument

Polimorfisme juga bisa diterapkan pada argument suatu method. Tujuannya agar method tersebut bisa menerima nilai argument dari berbagai bentuk objek. Misalkan dibuat class baru sebagai berikut:

public class Human{

public void drive(Animal anim){ anim.walk();

}

}

Perhatikan method **drive()**, ia memiliki argument berupa **Animal**. Karena **Animal** memiliki subclass **Lion** dan **Deer**, maka method **drive()** tersebutakan bisa menerima argument berupa objek dari **Deer** maupun objek dari **Lion**.

Deer d = new Deer();

Lion l = new Lion();

Human hum = new Human();

hum.drive(d); hum.drive(l);

# Operator instanceof

Operator instanceof bisa digunakan untuk mengecek apakah suatu objek merupakan hasil instansiasi dari suatu class tertentu. Hasil dari instanceof berupa nilai boolean. Misalkan dibuat objek d dan l.

Deer d = new Deer();

Lion l = new Lion();

Animal a1 = d;

Animal a2 = l;

Jika kemudian digunakan operator instanceof, misalkan

a1 instanceof Deer àà akan menghasilkan true a2 instanceof Lion àà akan menghasilkan true

# Object Casting

Casting objek digunakan untuk mengubah tipe dari suatu objek. Jika ada suatu objek dari subclass kemudian tipenya diubah ke superclass, maka hal ini termasuk ke upcasting. Contoh:

Deer d = new Deer();

Animal a1 = d; *// proses ini bisa disebut juga upcasting*

Downcast terjadi jika ada suatu objek superclass, kemudian diubah menjadi objek dari subclass. Contoh:

Deer d = new Deer();

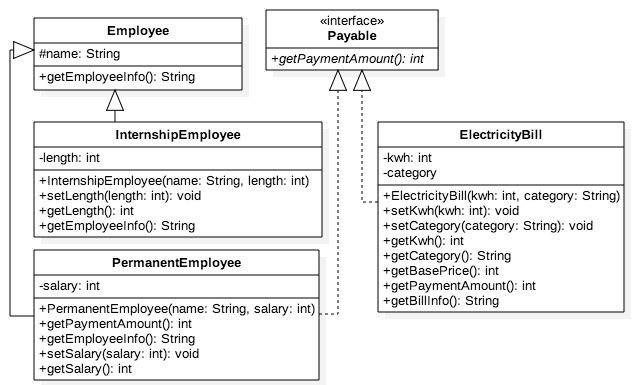
Animal a1 = d; *// proses ini bisa disebut juga upcasting*

Deer d = (Deer) a1; //proses downcasting

Proses downcasting sering disebut juga sebagai explicit casting, karena bentuk tujuan dari casting harus dituliskan dalam tanda kurung, di depan objek yang akan di-casting.

## 3. Studi Kasus

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

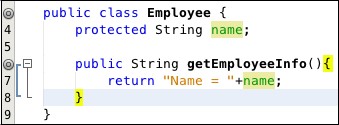


Dalam suatu perusahaan, pemilik pada tiap bulannya harus membayar gaji pegawai tetap dan rekening listrik. Selain pegawai tetap perusahaan juga memiliki pegawai magang, dimana pegawai ini tidak mendapatkan gaji.

## 4. Percobaan 1 – Bentuk dasar polimorfisme

### 4. 1. Langkah Percobaan

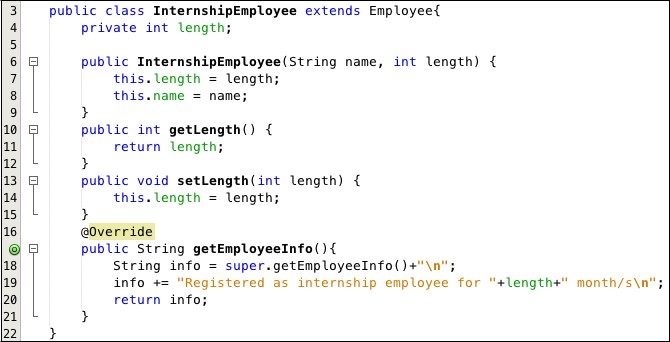
1. Buat class **Employee**



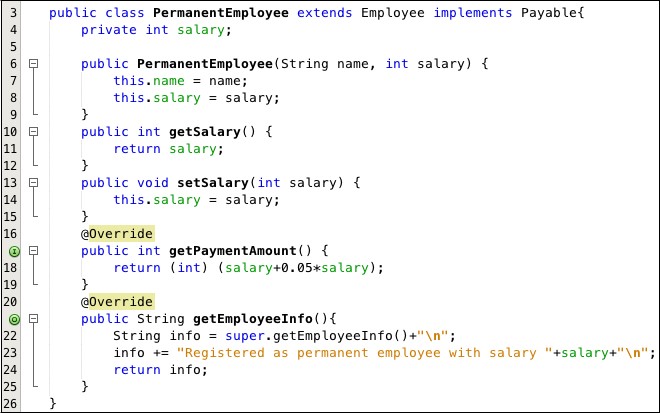
1. Buat interface **Payable**



1. Buat class **InternshipEmployee**, subclass dari **Employee**



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

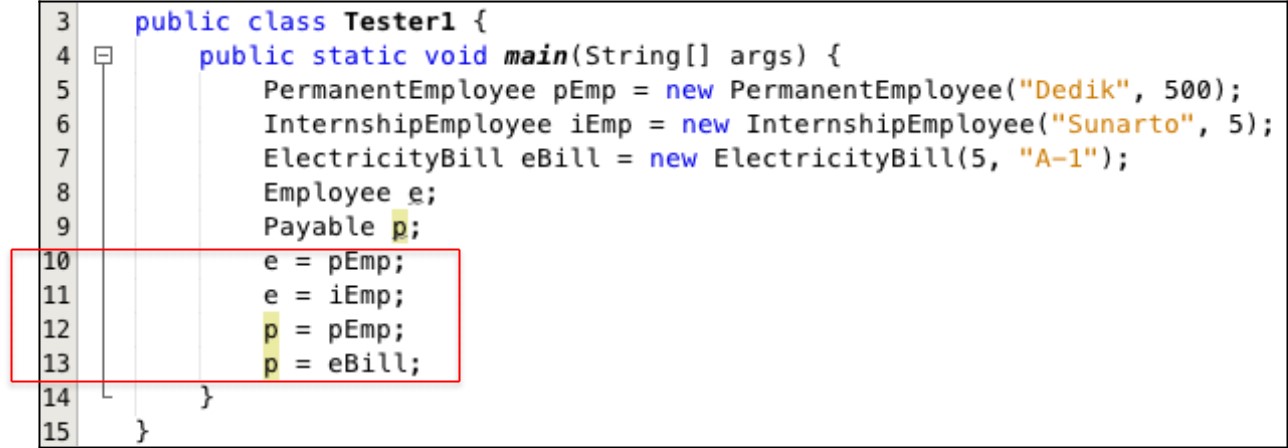


1. Buat class **ElectricityBill** yang implements ke interface

**Payable**



1. Buat class **Tester1**



#### 4.2. Pertanyaan

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

* Turunannya adalah InternshipEmployee dan PermanentEmployee.

1. Class apa sajakah yang implements ke interface **Payable**?

* Yang mengimplementasikan Payable adalah ElectricityBill dan PermanentEmployee.

1. 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**) ?

* Karena dalam class antara class internship dan permanent terdapat perbedaan atribut yang mana class permanent dapat menerapkan atau menginherit class employee.

1. 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**) ?

* Karena class permanent sendiri mengimplentasi class payable didalamnya, begitu juga class electricityBill, sehingga hal tersebut dapat dilakukan.

1. Coba tambahkan sintaks: **p = iEmp; e = eBill;** pada baris 14 dan 15 (baris terakhir dalam method **main**) ! Apa yang menyebabkan error?

* Terjadi error karena class internship tidak menerapkan payable didalamnya yang mana hal ini adalah kebalikan dari class permanent sehingga terjadi error.

1. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!

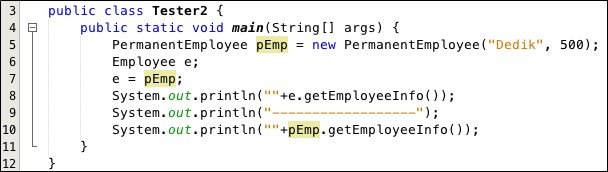
- Polimorfisme merupakan pengubahan suatu fungsi menyesuaikan kebutuhan.

## 5. Percobaan 2 – Virtual method invocation

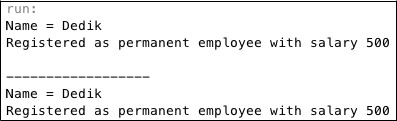
### 5. 1. Langkah Percobaan

1. Pada percobaan ini masih akan digunakan class-class dan interface yang digunakan pada percobaan sebelumnya.

1. Buat class baru dengan nama **Tester2**.



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



#### 5.2. Pertanyaan

1. Perhatikan class **Tester2** di atas, mengapa pemanggilan

**e.getEmployeeInfo()** pada baris 8 dan **pEmp.getEmployeeInfo()** pada baris 10 menghasilkan hasilsama?

* Karena objek yang dibuat dari class Employee didefinisikan dengan objek yang dibuat dari class Permanent yang mana membuat alamat dari objek dari class Employee sama dengan alamat objek yang dibuat dari class Permanent.

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

* Karena **e.getEmployeeInfo()** merupakan method dari class Employee yang alamatnya sama dengan alamat class Permanent yang artinya class Employee tidak memiliki alamat sendiri dan hanya menumpang dari alamat Permanent.

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

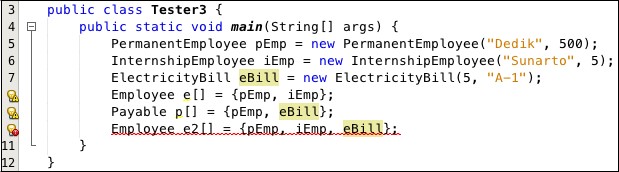
- VMI merupakan sebuah method dari class yang didefinisikan class lain. Jadi saat sebuah class memanggil methodnya, method yang dijalankan adalah method yang ada dalam class asli yang dijadikan definisi.

## 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.

1. Buat class baru **Tester3**.



#### 6.2. Pertanyaan

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**) ?

* Hal ini dapat terjadi karena kedua class terebut(Permanent dan Internship) memiliki kesamaan yaitu menginherit class yang sama(Employee).

1. 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**) ?

* Karean pEmp dan eBill mengimplementasikan Payable sehingga bisa masuk kedalam array objek Payable.

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

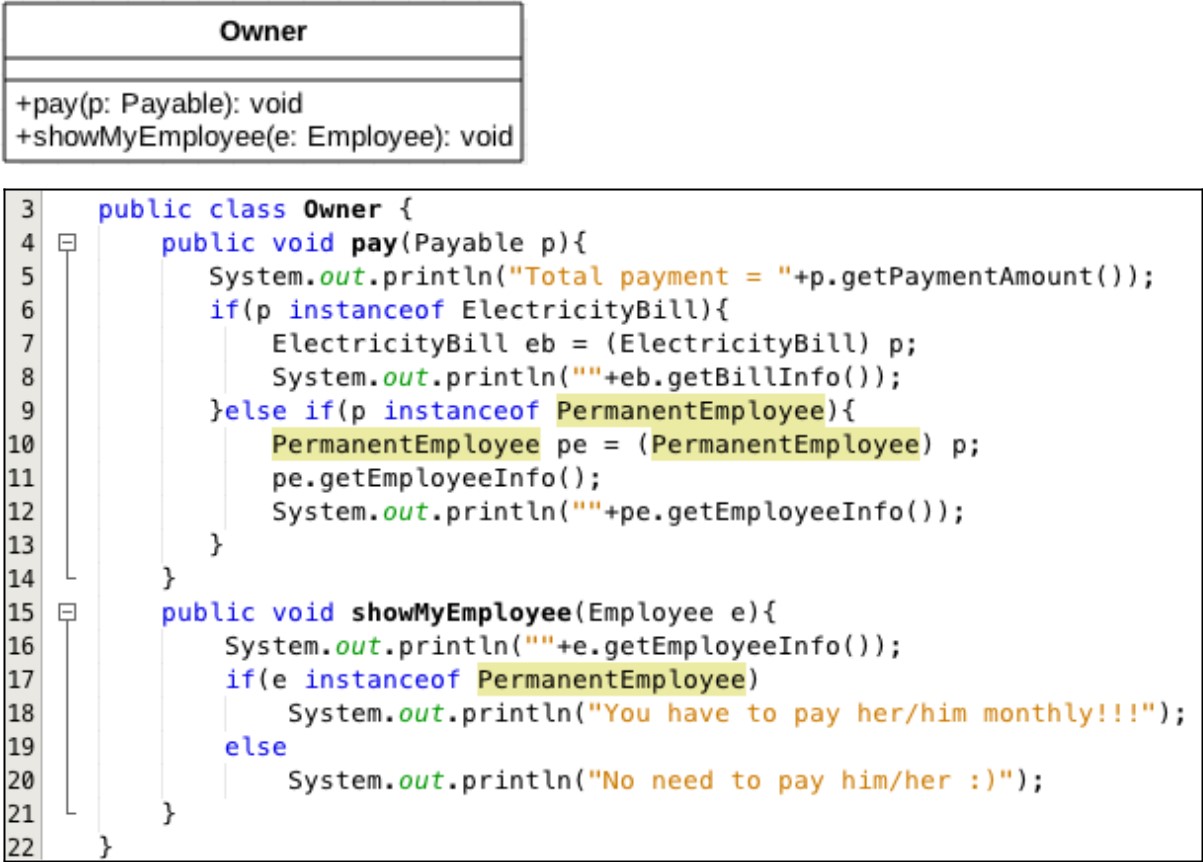
* Terjadi error karena perbedaan yaitu pengimplementasian dalam masing-masing class, dimana dalam class Internship tidak mengimplementasikan class Payable, sedangkan class Employee mengimplementasikannya. Dan juga pada class Electricity tidak menginherit class Employee.

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

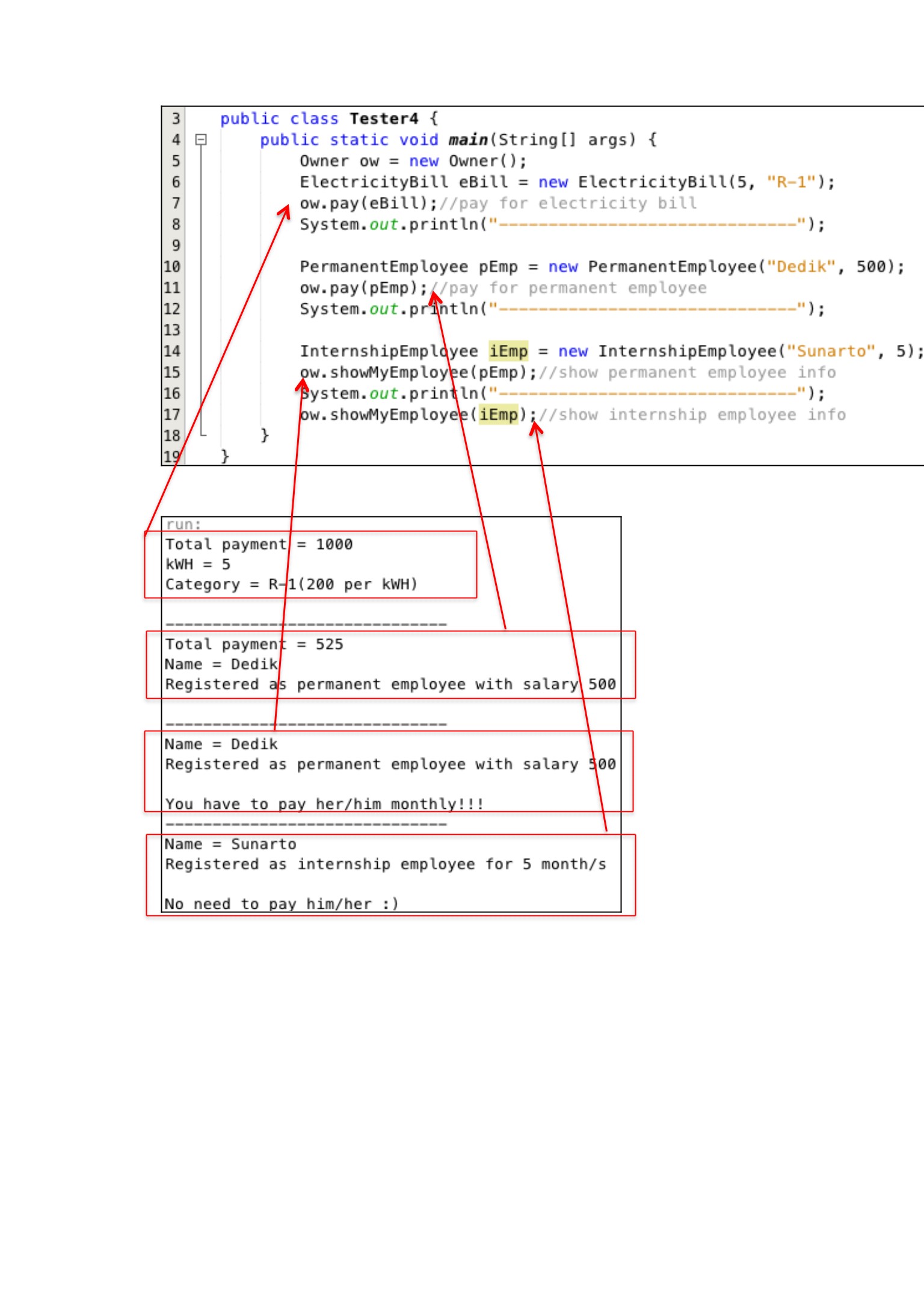
### 7.1. Langkah Percobaan

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()**.



2. Buat class baru **Tester4**.



3.

Jalankan class

**Tester4**

, dan akan didaptkan hasil sebagai berikut:

**7.2.**

**Pertan**

**yaan**

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

**Owne**

**r**

memiliki

argument/parameter

bertipe

**Payab**

**le**

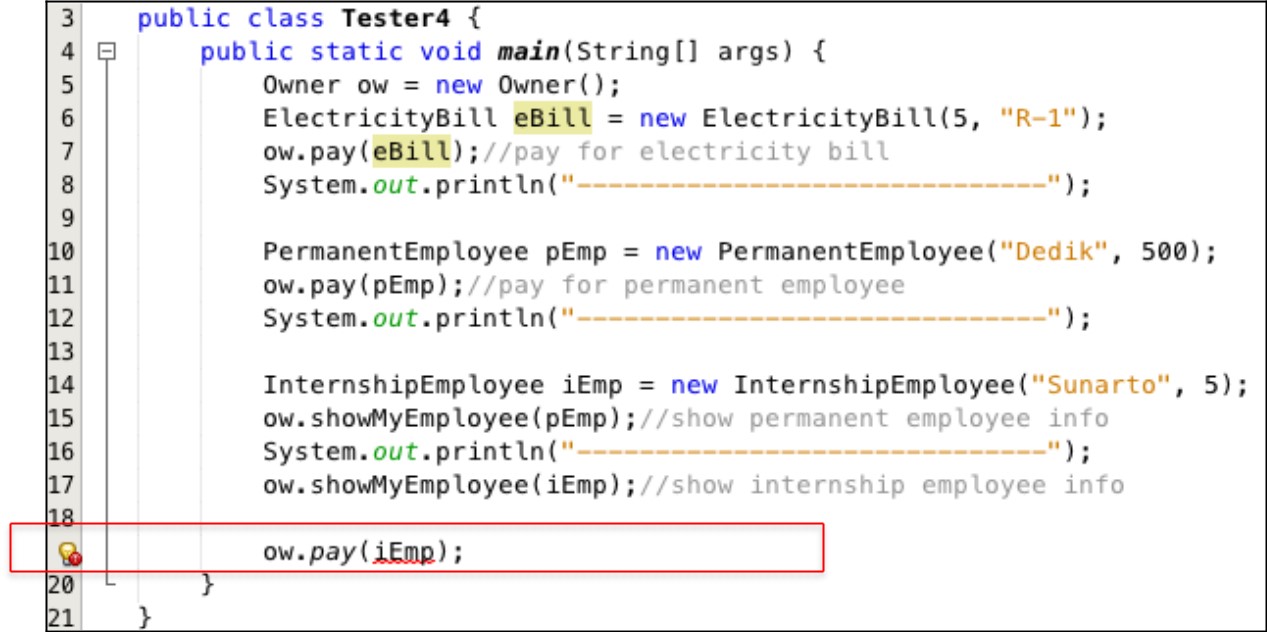
?

Jika diperhatikan lebih detil eBill merupakan objek dari **ElectricityBill** dan pEmp merupakan objek dari **PermanentEmployee**?

* Karena Electricity mengimplementasikan class Payable sehingga dapat digunakan dalam method pay().

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

* Untuk melakukan pembayaran kepada objek dari class yang dituju.

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

Mengapa terjadi error?

* Karena dalam class owner, method pay menggunakan instanceof class Permanent, bukan menggunakan Internship.

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

* Untuk membayar Electricity yang dimiliki oleh owner.

1. 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 ?**

- Agar bisa mengakses method-method yang mungkin tidak ada dalam Payable sedangkan ada dalam Electricity.

## 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

* + 1. 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%

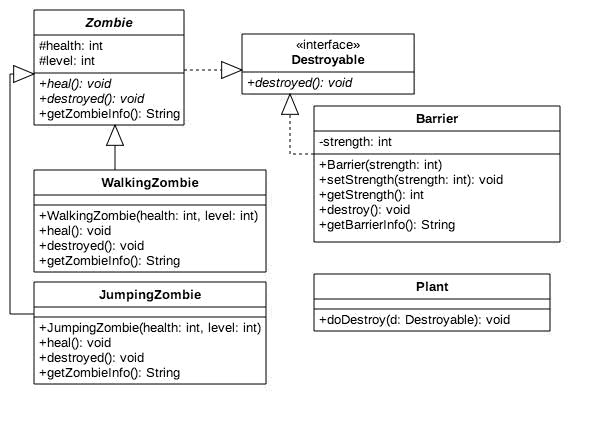
* Pada Jumping Zombie
  + Penyembuhan : Penyembuhan ditentukan berdasar level zombie yang bersangkutan
    1. Jika zombie level 1, maka setiap kali penyembuhan, health akan bertambah 30%

§§ Jika zombie level 2, maka setiap kali penyembuhan, health akan bertambah 40%

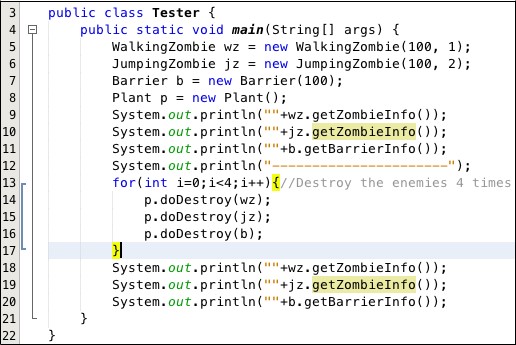
§§ Jika zombie level 3, maka setiap kali penyembuhan, health akan bertambah 50%

* + Penghancuran : setiap kali penghancuran, health akan berkurang 1%

Buat program dari class diagram di bawah ini!



**Contoh**: jika class Tester seperti di bawah ini:



Akan menghasilkan output:



* Main class :

package tugas;

public class Main {

    public static void main(String[] args) {

        WalkingZombie wz = new WalkingZombie(100, 2);

        JumpingZombie jz = new JumpingZombie(100, 2);

        Barrier b = new Barrier(200);

        Plant p = new Plant();

        wz.status();

        jz.status();

        b.status();

        for (int i = 0; i < 45; i++) {

            p.attacking(wz);

            p.attacking(jz);

            p.attacking(b);

        }

        System.**out**.println("-------------------------------");

        wz.status();

        jz.status();

        b.status();

        System.**out**.println("-------------------------------");

        wz.heal();

        jz.heal();

        b.heal();

        System.**out**.println("-------------------------------");

        wz.status();

        jz.status();

        b.status();

    }

}

* Hasil run :

