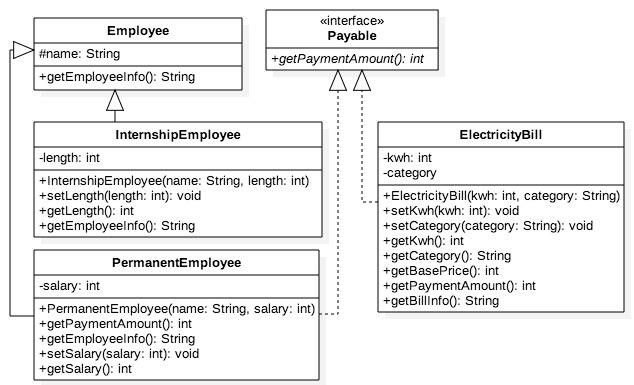
**JOBSHEET**

**POLIMORFISME**

## 3. Studi Kasus

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

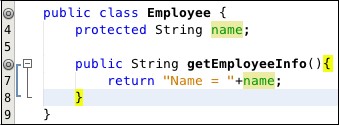


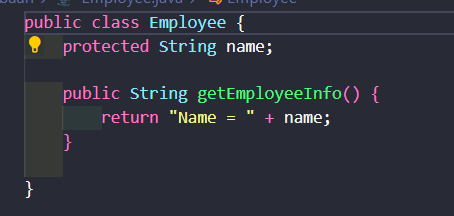
Dalam suatu perusahaan, pemilik pada tiap bulannya harus membayar gaji pegawai tetap dan rekening listrik. Selain pegawai tetap perusahaan juga memiliki pegawai magang, dimana pegawai ini tidak mendapatkan gaji.

## 4. Percobaan 1 – Bentuk dasar polimorfisme

### 4. 1. Langkah Percobaan

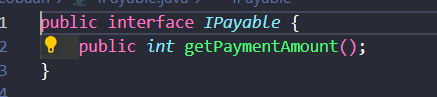
1. Buat class **Employee**



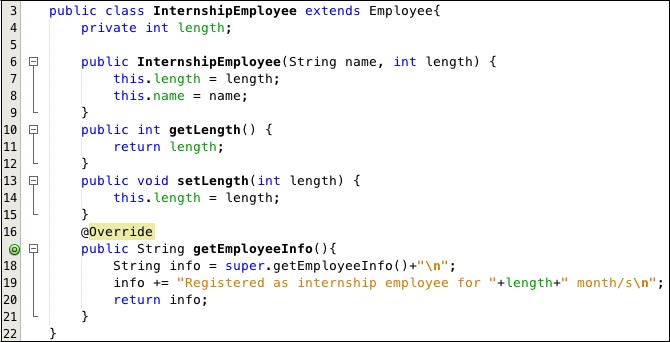


1. Buat interface **Payable**



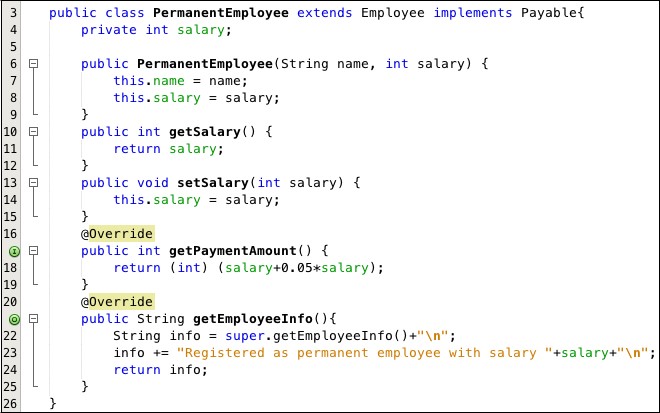


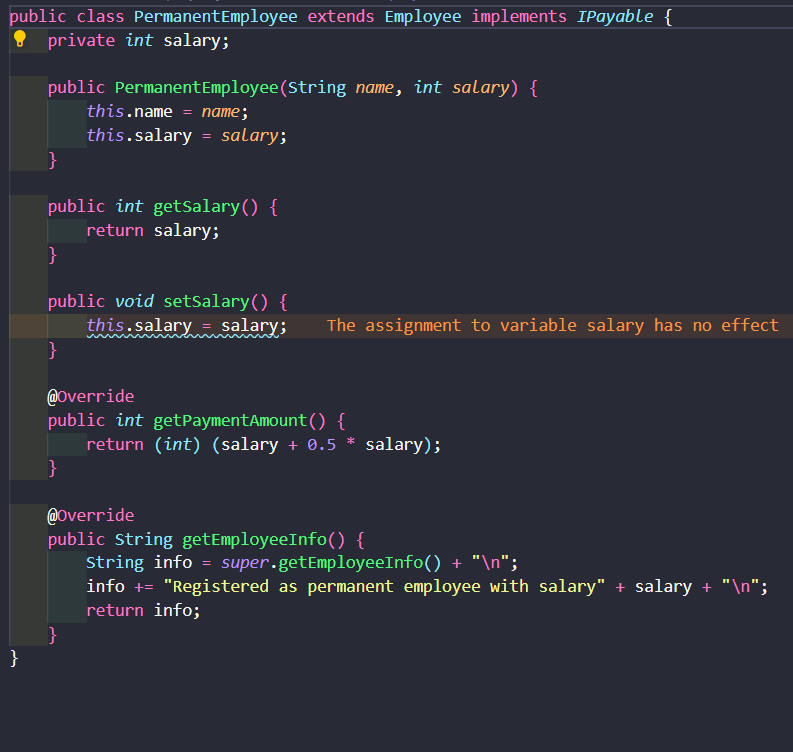
1. Buat class **InternshipEmployee**, subclass dari **Employee**



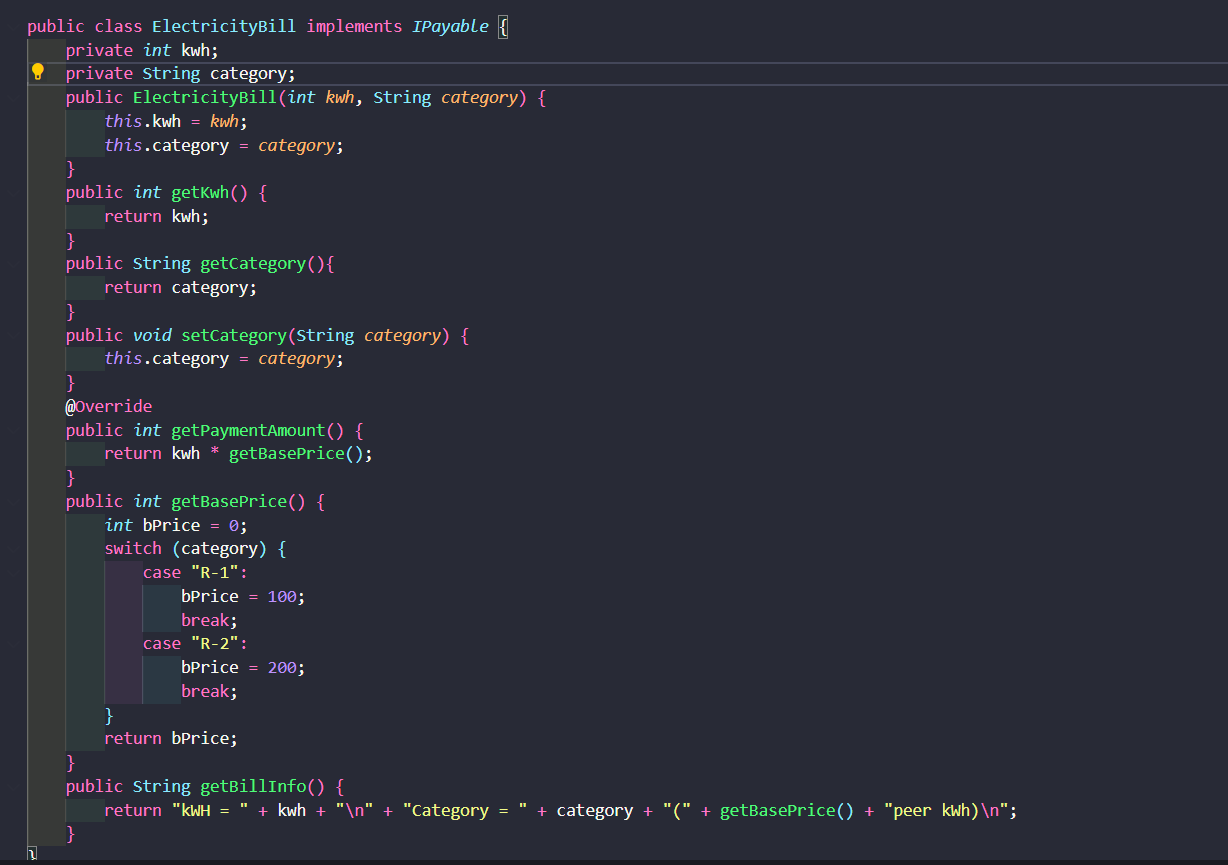


1. Buat class **PermanentEmployee**, subclass dari **Employee** dan implements ke **Payable**

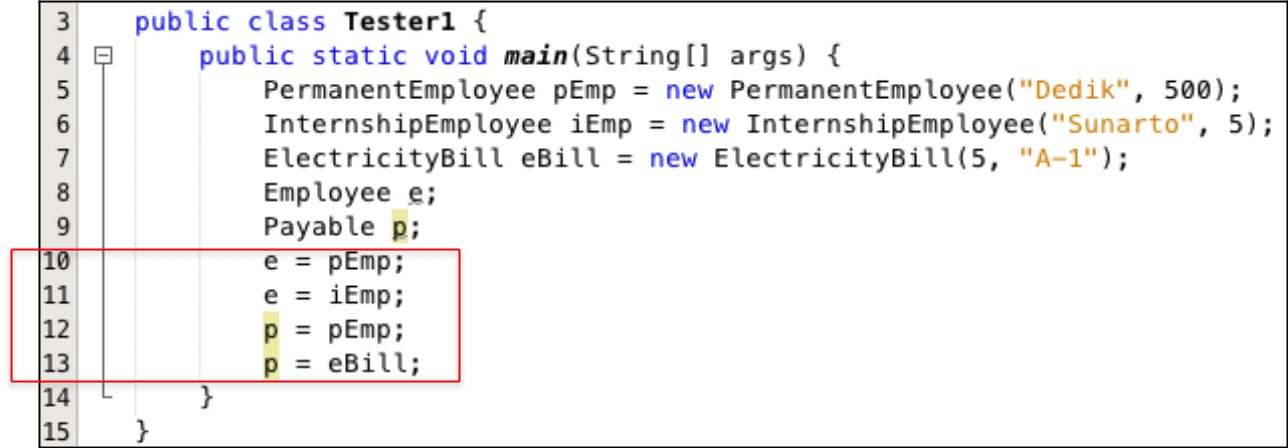


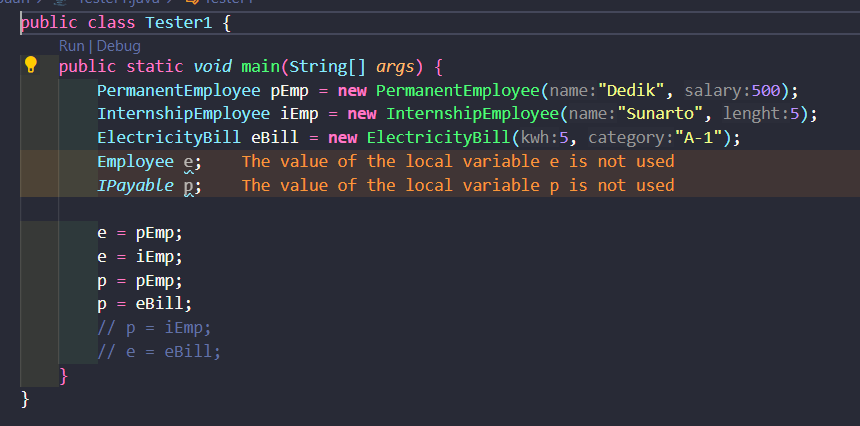


1. Buatt class **ElectricityBill** yang implements ke interface **Payable**



1. Buat class **Tester1**





#### 4.2. Pertanyaan

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

**Jawab :** Class InternshipEmplouyee dan PermanentEmployee

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

**Jawab :** 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**) ?

**Jawab :** Karena class pEmp dan iEmp merupakan class turunan dari Employee. Dan e merupakan objek Employee.

1. 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.

1. 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

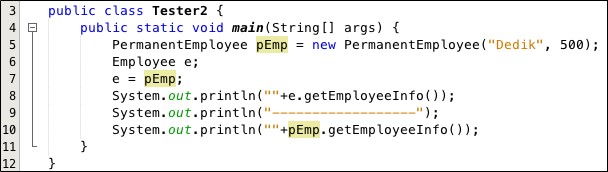
1. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!
2. Polimorfisme adalah kemampuan untuk menggunakan satu antarmuka atau nama Metode yang sama untuk berbagai tipe objek atau kelas yang berbeda. Polimorfisme memungkin objek yang berasal dari kelas yang berbeda untuk merespons panggilan Metode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

## 5. Percobaan 2 – Virtual method invocation

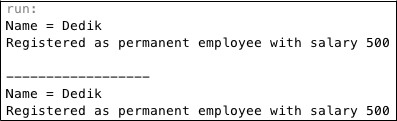
### 5. 1. Langkah Percobaan

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

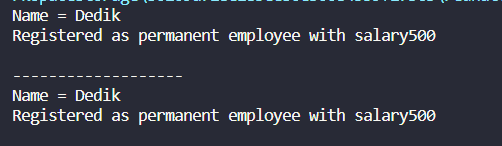
1. Buat class baru dengan nama **Tester2**.



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?

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

1. 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.

1. 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

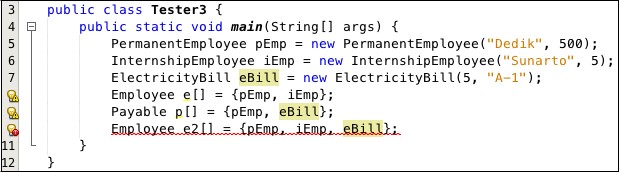
dari berbagai kelas yang berbeda secara seragam, mengikuti prinsip "program ke antarmuka, bukan ke implementasi".

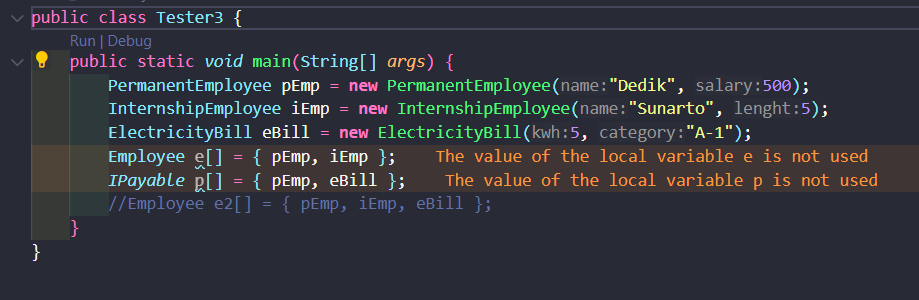
## 6. Percobaan 3 – Heterogenous Collection

### 6. 1. Langkah Percobaan

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

1. Buat class baru **Tester3**.





#### 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

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**) ?

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

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

**Jawab :** Karena objek eBill bukan turunan dari class Employee.

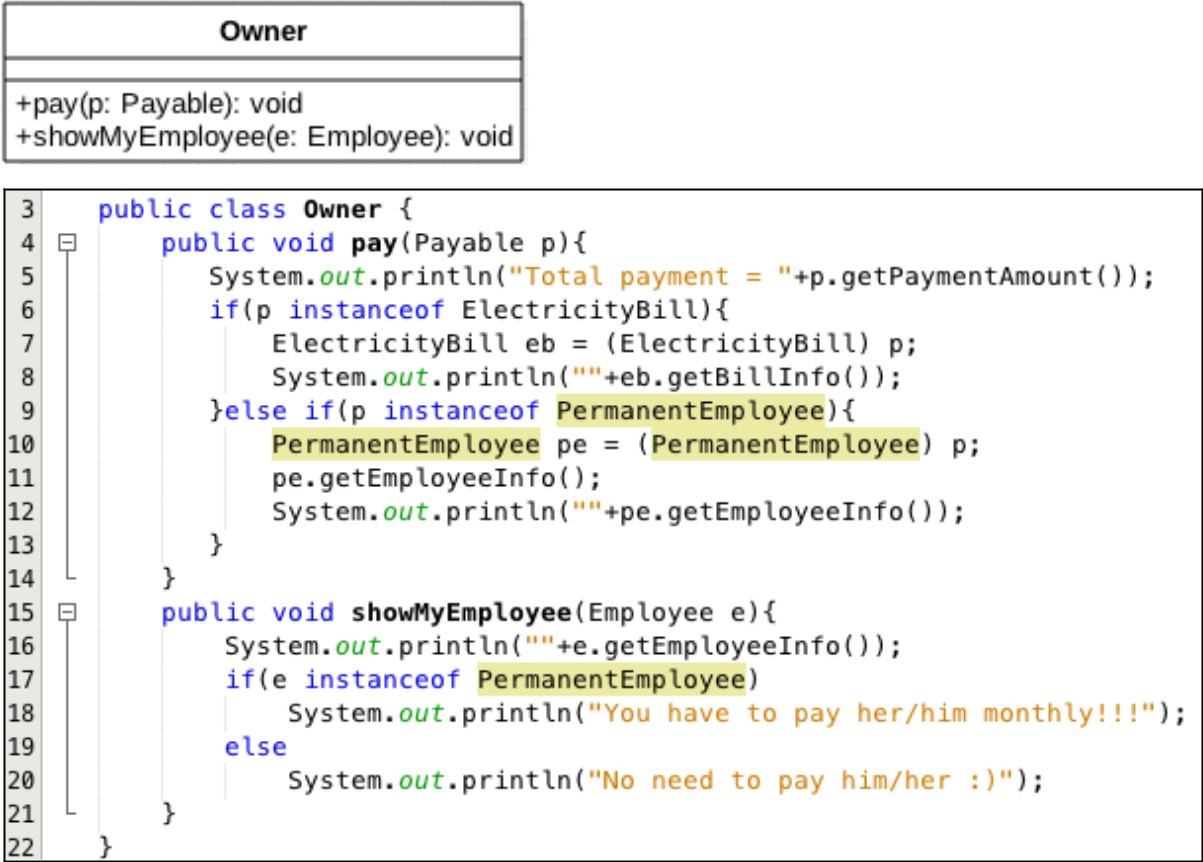
## 7. Percobaan 4 – Argumen polimorfisme, instanceof dan casting objek

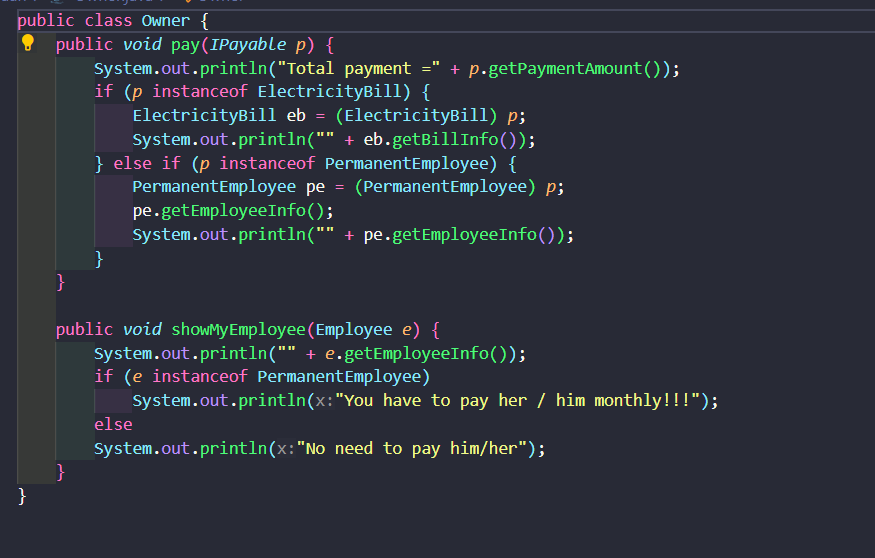
### 7.1. Langkah Percobaan

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

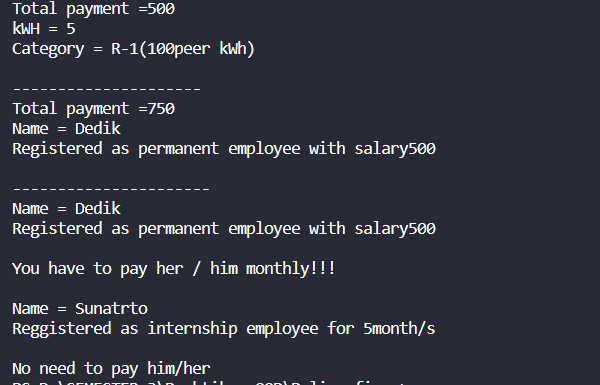
permanen maupun pegawai magang melalui method **showMyEmployee()**.

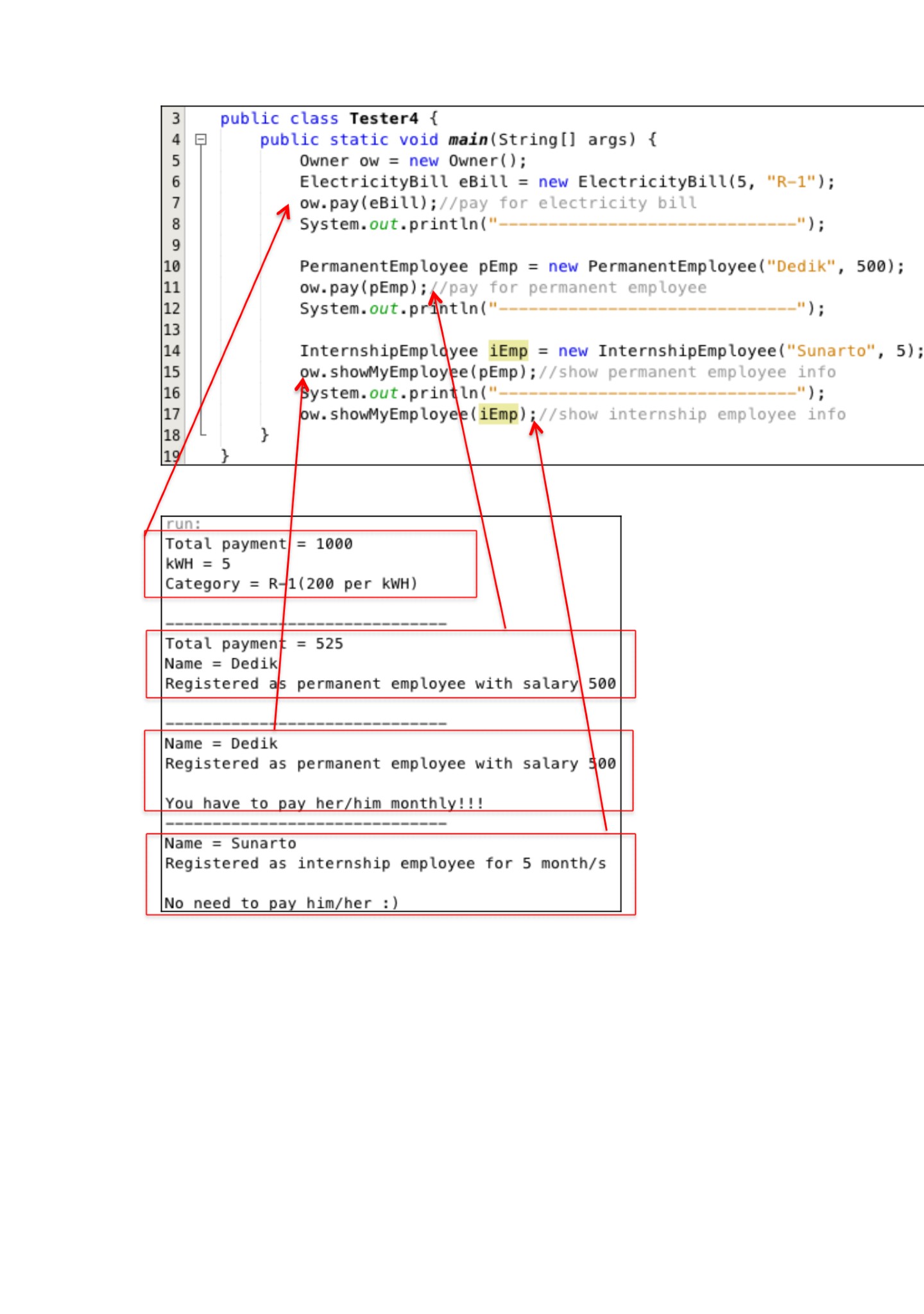




2. Buat class baru **Tester4**.







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

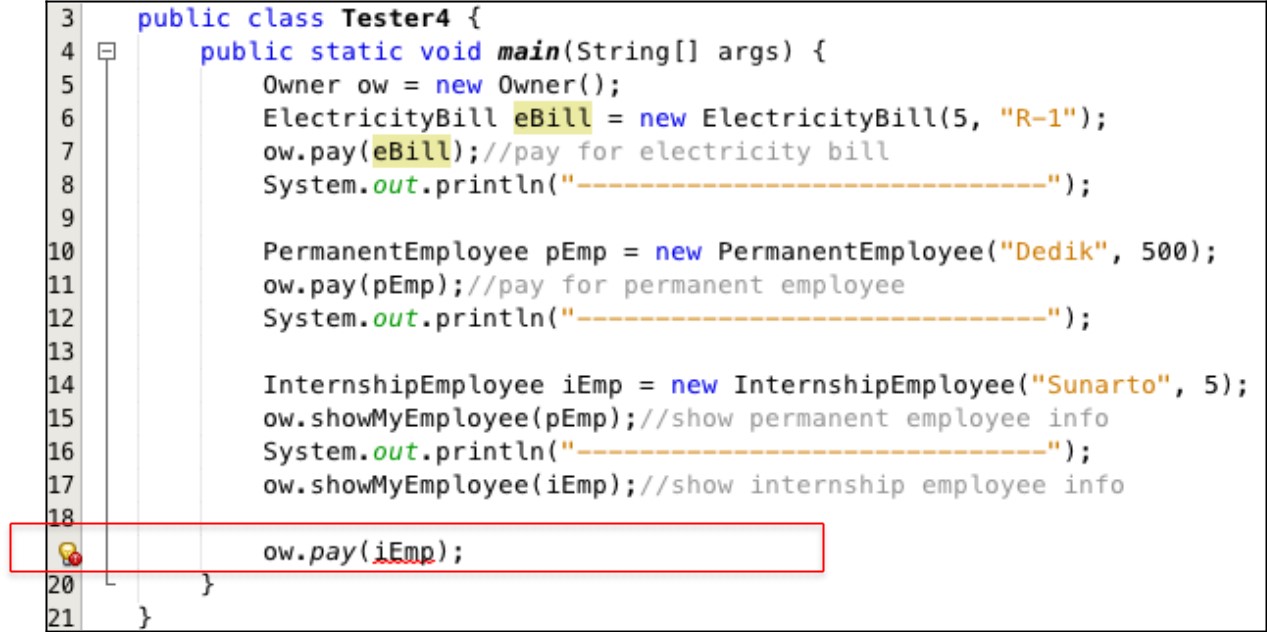
**PermanentEmployee**?

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

1. 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.

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

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

Mengapa terjadi error?

**Jawab :** Karena objek iEmp tidak berhubungan atau menginterface kelas Payable

1. 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.

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 ?**

**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

* + 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%

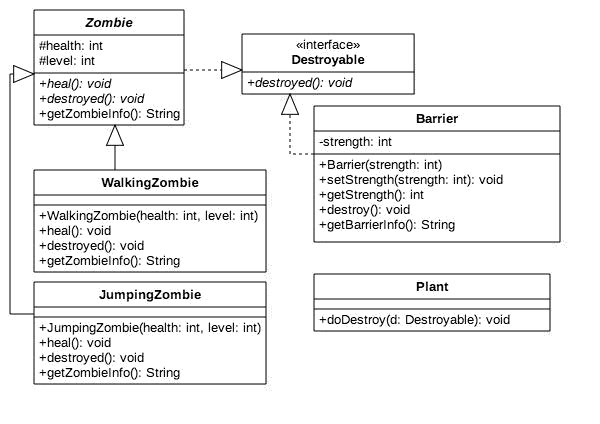
 Jika zombie level 2, maka setiap kali penyembuhan, health akan bertambah 40%

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

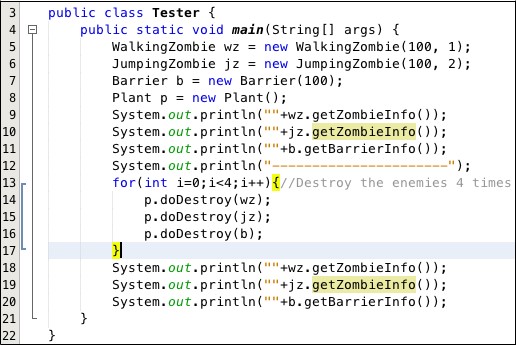
* + Penghancuran : setiap kali penghancuran, health akan berkurang

1%

Buat program dari class diagram di bawah ini!



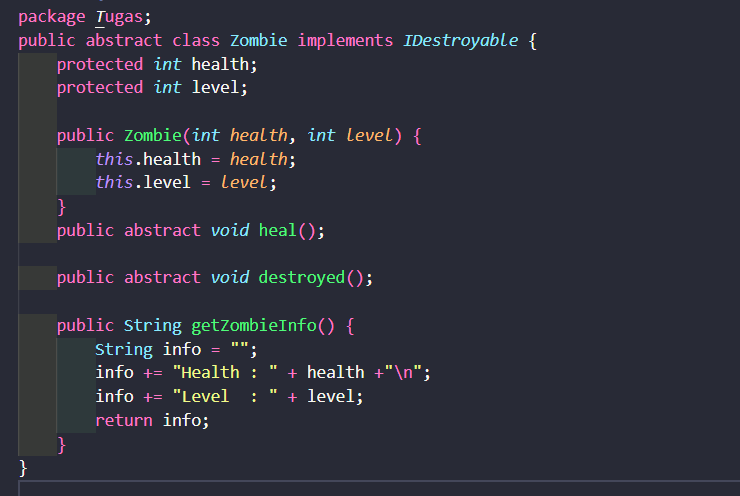
**Contoh**: jika class Tester seperti di bawah ini:



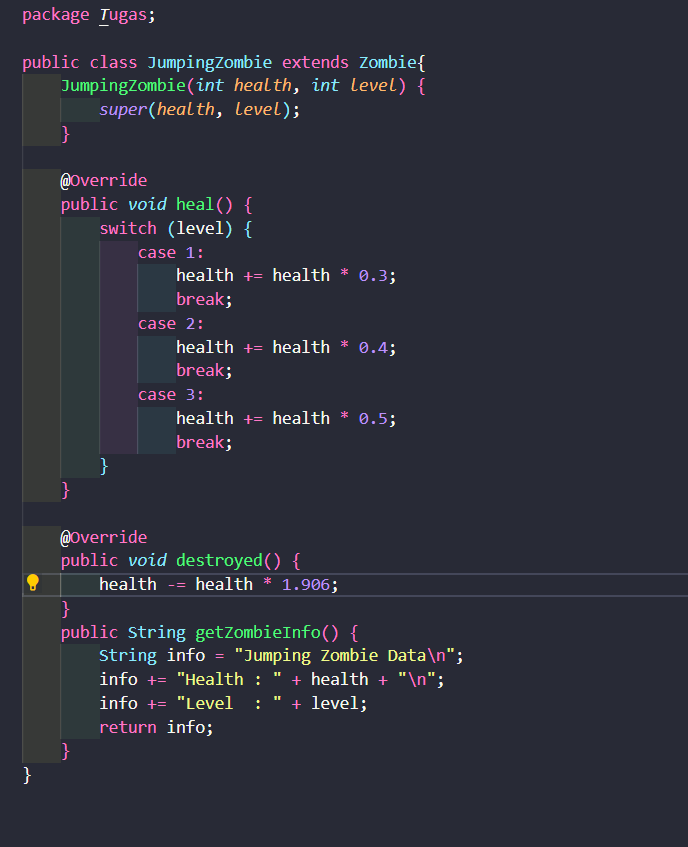
Akan menghasilkan output:



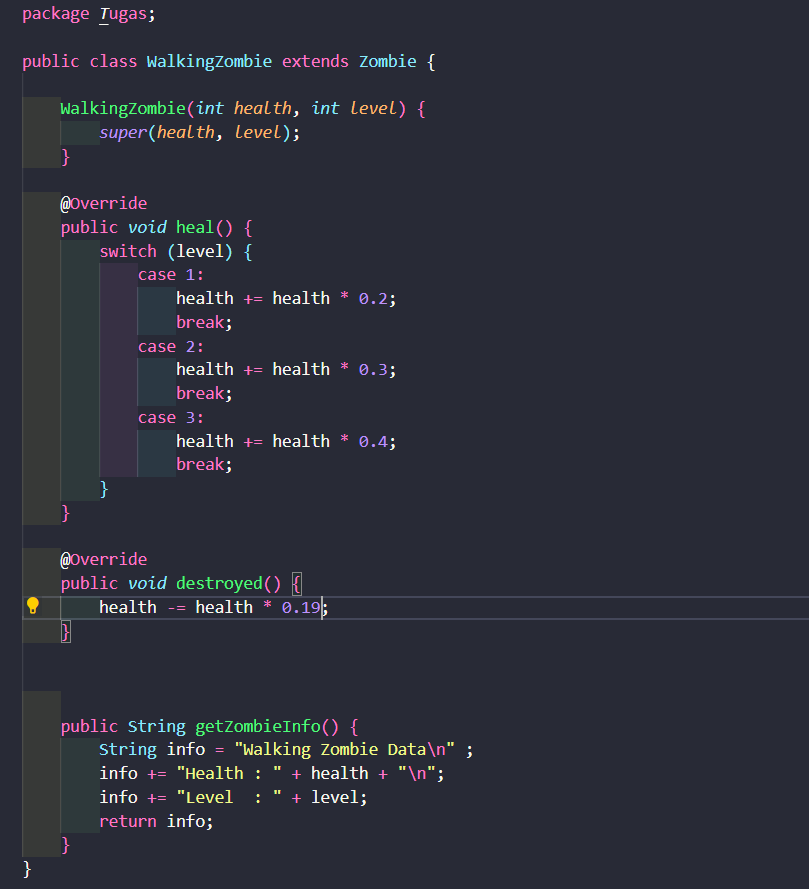
**Class Zombie**

****

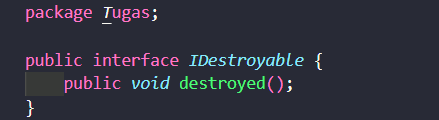
**Class JumpingZombie**

****

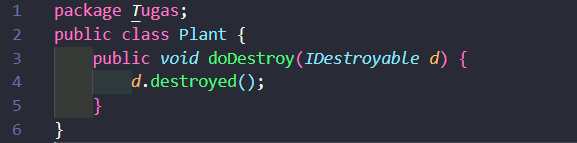
**Class Walking Zombie**

****

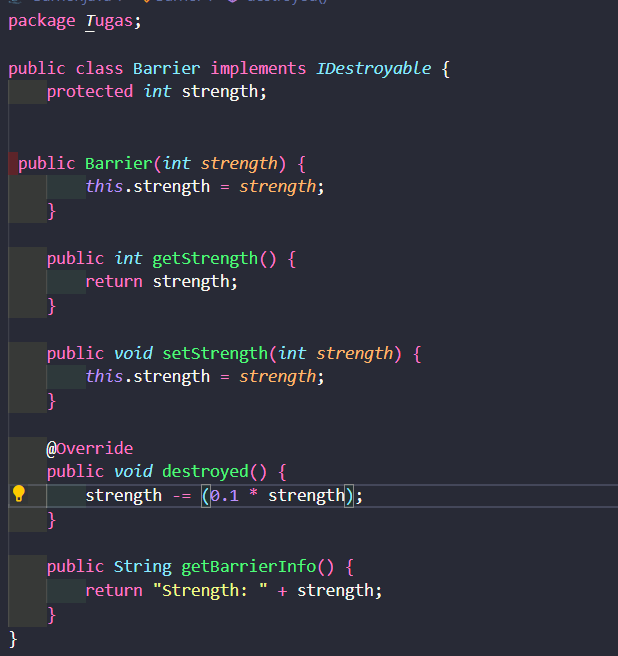
**Class interface Destroyable**

****

**Class Plant**

****

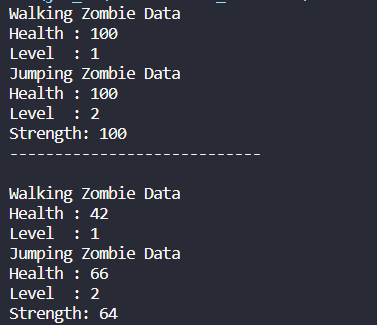
**Class Barrier**

****

**Class Tester sebagai class main**

****

**Output :**

****