JOBSHEET 10 - POLIMORFISME

Object oriented Programming

AYU JOVITA WIDYADHARI
2241720219 / 2- I

1st Practicum

Employee Class

```
public class Employee {
    protected String name;

    public String getEmployeeInfo(){
        return "Name = "+name;
    }
    public interface Payable {
        public int getPaymentAmount();
    }
}
```

InternshipEmployee

```
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;
    }
}
```

Payable

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

PermanentEmployee class

```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 getSlary(){
        return salary;
    }
    public void setSlary(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;
    }
}
```

ElectricityBill Class

```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(){
        this.kwh = kwh;
    }
    public String getCategory(){
        return category;
    }
    public void setCategory(){
        this.category = category;
    }
```

```
32          @Override
33          public int getPaymentAmount() {
34              return kwh*getBasePrice();
35          }
36          public int getBasePrice(){
37              int bPrice = 0;
38              switch(category){
39                  case "R-1" : bPrice = 100;
40                  break;
41                  case "R-2" : bPrice = 200;
42                  break;
43              }
44              return bPrice;
45          }
46          public String getBillInfo(){
47              return "kWH = "+kwh+"\n"+
48                      "Category = "+category+"("+getBasePrice()+" per kWH)\n";
49          }
50      }
```

Terste1 class

```
public class Tester1 {
    public static void main(String[] args) {
        PermanentEmployee pEmp = new PermanentEmployee(name: "Dedik", salary: 500);
        InternshipEmployee iEmp = new InternshipEmployee(name: "Sunarto", length: 5);
        ElectricityBill eBill = new ElectricityBill(kwh:5, category: "A-1");
        Employee e;
        Payable p;
        e = pEmp;
        e = iEmp;
        p = pEmp;
        p = eBill;
    }
}
```

Pertanyaan:

1. Class apa sajakah yang merupakan turunan dari class Employee?
   Jawab:
   Class turunan dari class Employee adalah InternshipEmployee dan PermanentEmployee

2. Class apa sajakah yang implements ke interface Payable?
   Jawab:
   PermanentEmoloyee dan ElectricityBill merupakan implementasi dari interface Payable.

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:
   Di baris 10 dan 11, e merupakan variable dari tipe Employee yang mana merupakan superclass dari PermanentEmployee dan InternshipEmployee.
   Pada konsep polimorfisme, objek dari subclass dapat diassign ke variable superclass.
4. Perhatikan class Tester1, baris ke-12 dan 13. Mengapa p, bisa diisi denganobjekpEmp (merupakan objek dari class PermanentEmployee) dan objek eBill (merupakan objek dari class ElectricityBill)
   Jawab:

Pada baris 12 dan 13, p adalah variable dari tipe Payable yang merupakan tipe dari interface Payable. Objek dari kelas **PermanentEmployee** dan **ElectricityBill** dapat diassign ke variabel **p**, karena keduanya mengimplementasikan interface **Payable**.

5. Coba tambahkan sintaks: p = iEmp; e = eBill; pada baris 14 dan 15 (baris terakhir dalam method main) ! Apa yang menyebabkan error?
   Jawab:
   Jika Anda menambahkan sintaks **p = iEmp;** dan **e = eBill;**, akan terjadi error. Hal ini karena **iEmp** adalah objek dari **InternshipEmployee**, yang tidak mengimplementasikan interface **Payable**. Begitu juga dengan **eBill**, meskipun ia mengimplementasikan **Payable**, tetapi tidak bisa diassign ke **e**, yang merupakan variabel dengan tipe **Employee**.

6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!
   Jawab:
   olimorfisme adalah konsep dalam pemrograman berorientasi objek di mana objek dari kelas-kelas yang berbeda dapat diakses dan diolah menggunakan antarmuka yang sama. polimorfisme terlihat dalam kemampuan untuk mengassign objek dari subclass ke variabel superclass dan variabel bertipe antarmuka yang diimplementasikan oleh berbagai kelas. Ini memungkinkan kode untuk menjadi lebih fleksibel dan dapat beradaptasi dengan perubahan dalam hierarki kelas atau implementasi antarmuka.

2nd Practicum

```
public class Tester2 {
    public static void main(String[] args){
        jobsheet.pkg10.PermanentEmployee pEmp = new jobsheet.pkg10.PermanentEmployee(name: "Dedik", salary: 500);
        Employee e;
        e = pEmp;
        System.out.println(""+e.getEmployeeInfo());
        System.out.println(x: "--------------------");
        System.out.println(""+pEmp.getEmployeeInfo());
    }
}
```

Result:

```
Output - Jobsheet 10 (run)
run:
Name = Dedik
Registered as permanent employee with salary 500

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

BUILD SUCCESSFUL (total time: 0 seconds)
```

Pertanyaan:

1. Perhatikan class Tester2 di atas, mengapa pemanggilan e.getEmployeeInfo() pada baris 8 dan pEmp.getEmployeeInfo() pada baris 10 menghasilkan hasil sama?
   Pada baris 8, pemanggilan **e.getEmployeeInfo()** menghasilkan hasil yang sama dengan pemanggilan **pEmp.getEmployeeInfo()** pada baris 10 karena **e** merujuk pada objek dari kelas **PermanentEmployee**. Meskipun tipe referensi adalah **Employee**, pemanggilan metode akan tetap merujuk pada implementasi metode di kelas aktual objek yang ditunjuk oleh referensi tersebut. Dalam hal ini, meskipun variabel **e** dideklarasikan sebagai **Employee**, ia sebenarnya

merujuk pada objek yang merupakan instans dari **PermanentEmployee**, sehingga metode yang dipanggil adalah implementasi dari **PermanentEmployee**.

2. Mengapa pemanggilan method e.getEmployeeInfo() disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan pEmp.getEmployeeInfo() tidak?
   Pemanggilan e.getEmployeeInfo() disebut sebagai pemanggilan metode virtual karena metode yang dipanggil ditentukan saat runtime berdasarkan objek yang sebenarnya ditunjuk oleh referensi (e dalam hal ini). Dengan kata lain, implementasi metode dipilih dinamis pada saat runtime berdasarkan tipe objek yang sebenarnya, bukan pada saat kompilasi.
   Pada kasus pEmp.getEmployeeInfo(), meskipun tipe referensi adalah PermanentEmployee, ini juga dapat dianggap sebagai pemanggilan metode virtual karena metode yang dipanggil tetap tergantung pada objek sebenarnya yang ditunjuk oleh referensi pada saat runtime.
3. Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?
   Virtual method invocation (pemanggilan metode virtual) merujuk pada kemampuan pemrograman objek di mana metode yang akan dipanggil ditentukan pada saat runtime berdasarkan tipe objek aktual, bukan tipe referensi yang digunakan untuk memanggil metode tersebut.
   Istilah "virtual" digunakan karena pemilihan metode terjadi secara dinamis pada saat runtime, dan komputer harus "mengerti" atau "memahami" objek yang sebenarnya ditunjuk oleh referensi. Dalam konteks ini, "virtual" menunjukkan bahwa pemilihan metode tidak sepenuhnya ditentukan pada saat kompilasi, melainkan dapat berubah pada saat runtime tergantung pada objek yang sebenarnya. Oleh karena itu, metode yang akan dipanggil dianggap "virtual" atau dapat berubah sesuai konteks objek yang sebenarnya.

3rd Practicum

```
10  public class Tester3 {
11      public static void main(String[] args){
12          PermanentEmployee pEmp = new PermanentEmployee(name: "Dedik", salary: 500);
13          InternshipEmployee iEmp = new InternshipEmployee(name: "Sunarto", length: 5);
14          ElectricityBill eBill = new ElectricityBill(kwh:5, category: "A-1");
            Employee e[] = {pEmp, iEmp};
            Payable p[] = {pEmp, eBill};
            Employee e2[] = {pEmp, iEmp, eBill};
18      }
19  }
```

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) ?
   Pada baris ke-8, array **e** bisa diisi dengan objek-objek yang memiliki tipe yang berbeda (objek **pEmp** dan **iEmp**) karena keduanya merupakan turunan dari kelas **Employee** atau mengimplementasikan interface **Employee**. Dengan menggunakan polimorfisme, objek dari subclass atau implementor interface dapat dianggap sebagai objek dari tipe superclass atau interface yang lebih umum.
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) ?
   Pada baris ke-9, array **p** bisa diisi dengan objek-objek yang memiliki tipe yang berbeda (objek **pEmp** dan **eBill**) karena keduanya merupakan turunan dari kelas **Payable** atau mengimplementasikan interface **Payable**. Dengan menggunakan polimorfisme, objek dari subclass atau implementor interface dapat dianggap sebagai objek dari tipe superclass atau interface yang lebih umum.
3. Perhatikan baris ke-10, mengapa terjadi error?

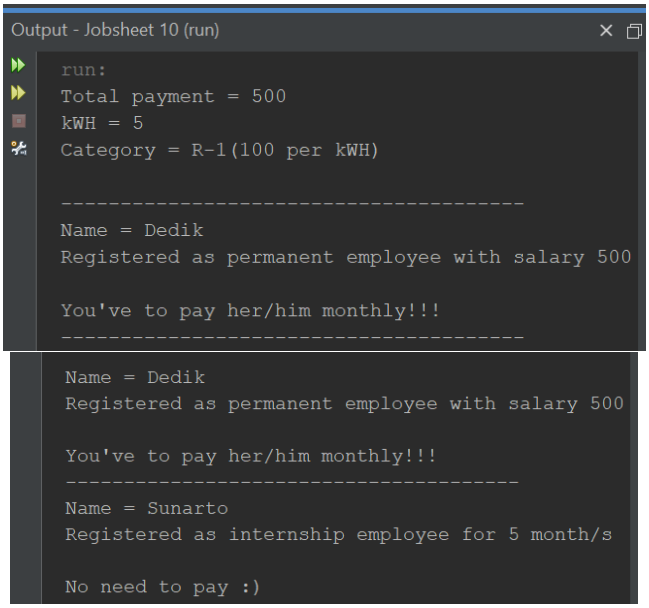
Pada baris ke-10, terjadi error karena array **e2** dideklarasikan sebagai array dari tipe **Employee**, sementara objek **eBill** merupakan objek dari kelas **ElectricityBill** dan tidak menurunkan atau mengimplementasikan interface **Employee**. Oleh karena itu, objek **eBill** tidak dapat dimasukkan ke dalam array **e2** yang diharapkan hanya berisi objek-objek yang merupakan turunan dari kelas **Employee** atau mengimplementasikan interface **Employee**.

4th Practicum

```
public class Owner {
    public void pay(Payable p){
        System.out.println("Totak 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 showEmployee(Employee e){
        System.out.println(""+e.getEmployeeInfo());
        if(e instanceof PermanentEmployee)
            System.out.println(x: "You've to pay her/him monthly!!!");
        else
            System.out.println(x: "No need to pay :)");
    }
}
```

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

        PermanentEmployee pEmp = new PermanentEmployee(name: "Dedik", salary: 500);
        ow.showEmployee(e: pEmp);//show information (Permanent Employee)
        System.out.println(x: "----------------------------------------");

        InternshipEmployee iEmp = new InternshipEmployee(name: "Sunarto", length: 5);
        ow.showEmployee(e: pEmp);//show information (Permanent Employee)
        System.out.println(x: "----------------------------------------");
        ow.showEmployee(e: iEmp);//show internship Employee info
    }
}
```

Result



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:
   Pemanggilan **ow.pay(eBill)** dan **ow.pay(pEmp)** dapat dilakukan karena, meskipun tipe parameter pada metode **pay** adalah **Payable**, objek **eBill** dan **pEmp** dapat diserahkan sebagai argumen karena keduanya mengimplementasikan interface **Payable**. Dalam pemrograman berorientasi objek, prinsip ini disebut **polimorfisme**, di mana objek dari kelas yang berbeda dapat dianggap sebagai objek dari tipe yang lebih umum (dalam hal ini, **Payable**).

2. Jadi apakah tujuan membuat argument bertipe Payable pada method pay() yang ada di dalam class Owner?
   Jawab:
   Tujuan dari membuat parameter bertipe **Payable** pada metode **pay()** di dalam class **Owner** adalah untuk menerima objek dari kelas-kelas yang mengimplementasikan interface **Payable**. Dengan menggunakan tipe interface sebagai parameter, class **Owner** dapat menerima berbagai jenis objek yang memiliki kemampuan untuk melakukan pembayaran (implementasi dari **Payable**). Ini mendukung konsep **polimorfisme**, di mana objek-objek dari kelas yang berbeda dapat dianggap sebagai objek dari tipe yang lebih umum (interface **Payable** dalam hal ini).

3. Coba pada baris terakhir method main() yang ada di dalam class Tester4 ditambahkan perintah ow.pay(iEmp);
   Jawab:
   Terjadi error pada baris **ow.pay(iEmp);** karena metode **pay()** di dalam class **Owner** mengharapkan parameter bertipe **Payable**. Meskipun **InternshipEmployee** (**iEmp**) mungkin memiliki implementasi **Payable**, hal itu tidak mencukupi untuk memasukkannya langsung sebagai argumen ke metode **pay()**.

4. Perhatikan class Owner, diperlukan untuk apakah sintaks p instanceof ElectricityBill pada baris ke-6 ?
   Jawab:
   Pemeriksaan **p instanceof ElectricityBill** pada baris ke-6 digunakan untuk memeriksa apakah objek yang diwakili oleh variabel **p** adalah instance dari kelas **ElectricityBill** atau kelas turunannya. Ini dapat berguna untuk menghindari kesalahan casting yang tidak aman. Dengan menggunakan **instanceof**, Anda dapat memastikan bahwa objek yang akan dicasting memiliki tipe yang benar sebelum melakukan casting. Jika **p** bukan merupakan instance dari **ElectricityBill**, maka operasi casting akan menghasilkan **ClassCastException**.

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:
   Casting objek dengan sintaks **(ElectricityBill) p** diperlukan karena variabel **p** dideklarasikan sebagai tipe **Payable**, sedangkan pada baris ke-7, kita ingin mengakses metode atau properti yang spesifik untuk kelas **ElectricityBill**.
   Sebagai contoh, jika **Payable** adalah sebuah interface yang diterapkan oleh **ElectricityBill**, kita mungkin memiliki beberapa metode tambahan yang spesifik untuk **ElectricityBill** di sana. Dengan casting, kita memberitahu kompiler bahwa kita yakin objek **p** (yang secara umum bertipe **Payable**) sebenarnya adalah objek **ElectricityBill**, sehingga kita dapat mengakses metode atau properti khusus **ElectricityBill** tersebut.
   Namun, perlu diingat bahwa penggunaan casting perlu dilakukan dengan hati-hati dan sebaiknya setelah memastikan bahwa objek dapat dicasting ke tipe yang diinginkan. Jika casting dilakukan tanpa memperhatikan tipe sebenarnya objek, dan objek tidak sesuai, akan terjadi **ClassCastException**. Oleh karena itu, penggunaan **instanceof** sebelum melakukan casting adalah praktik yang baik.

Assignment

Destroyable: interface

```
    interface Destroyable {
        void destroyed();
13  }
```

Class Zombie

```
public class Zombie implements Destroyable {
12      protected int health;
13      protected int level;
14
15      public Zombie(int health, int level) {
16          this.health = health;
17          this.level = level;
18      }
19
20      @Override
22      public void destroyed() {
23          // Default behavior for destroyed method
24          System.out.println(x: "Zombie destroyed!");
25      }
27      public void heal() {
28          // Default behavior for heal method
29          System.out.println(x: "Zombie healed!");
30      }
31      public String getZombieInfo() {
32          return "Health: " + health + "\nLevel: " + level;
33      }
34  }
```

Barrier class

```java
public class Barrier implements Destroyable {
    private int strength;

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

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

    public int getStrength() {
        return strength;
    }

    @Override
    public void destroyed() {
        System.out.println(x: "Barrier destroyed!");
    }

    public void destroy() {
        destroyed();
    }

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

WalkingZombie class

```java
public class WalkingZombie extends Zombie {
    public WalkingZombie(int health, int level) {
        super(health, level);
    }

    @Override
    public void heal() {
        double healingPercentage = 0.2 * level;
        health += (int) (health * healingPercentage);
        System.out.println(x: "Walking Zombie healed!");
    }

    @Override
    public void destroyed() {
        double destructionPercentage = 0.02;
        health -= (int) (health * destructionPercentage);
        System.out.println(x: "Walking Zombie destroyed!");
    }
}
```

JumpingZombie class

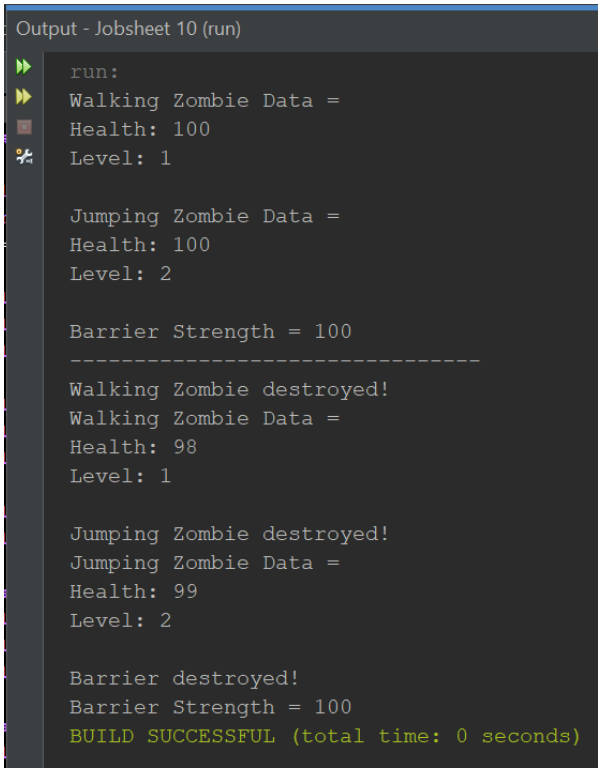
```java
public class JumpingZombie extends Zombie {
    public JumpingZombie(int health, int level) {
        super(health, level);
    }

    @Override
    public void heal() {
        double healingPercentage = 0.3 * level;
        health += (int) (health * healingPercentage);
        System.out.println(x: "Jumping Zombie healed!");
    }

    @Override
    public void destroyed() {
        double destructionPercentage = 0.01;
        health -= (int) (health * destructionPercentage);
        System.out.println(x: "Jumping Zombie destroyed!");
    }
}
```

Plant class

```java
class Plant {
    public void doDestroy(Destroyable d) {
        d.destroyed();
    }
}
```

```java
public class Tester {
    public static void main(String[] args) {
        // Example usage
        WalkingZombie walkingZombie = new WalkingZombie(health: 100, level: 1);
        JumpingZombie jumpingZombie = new JumpingZombie(health: 100, level: 2);
        Barrier barrier = new Barrier(strength: 100);

        System.out.println(x: "Walking Zombie Data = ");
        System.out.println(x: walkingZombie.getZombieInfo());
        System.out.println();

        System.out.println(x: "Jumping Zombie Data = ");
        System.out.println(x: jumpingZombie.getZombieInfo());
        System.out.println();

        System.out.println("Barrier Strength = " + barrier.getStrength());
        System.out.println(x: "---------------------------------");

        walkingZombie.destroyed();
        System.out.println(x: "Walking Zombie Data = ");
        System.out.println(x: walkingZombie.getZombieInfo());
        System.out.println();

        jumpingZombie.destroyed();
        System.out.println(x: "Jumping Zombie Data = ");
        System.out.println(x: jumpingZombie.getZombieInfo());
        System.out.println();

        barrier.destroy();
        System.out.println("Barrier Strength = " + barrier.getStrength());
```

Result:


Output - Jobsheet 10 (run)
run:
Walking Zombie Data =
Health: 100
Level: 1
Jumping Zombie Data =
Health: 100
Level: 2
Barrier Strength = 100
Walking Zombie destroyed!
Walking Zombie Data =
Health: 98
Level: 1
Jumping Zombie destroyed!
Jumping Zombie Data =
Health: 99
Level: 2
Barrier destroyed!
Barrier Strength = 100
BUILD SUCCESSFUL (total time: 0 seconds)

PPT ASSIGNMENT

EXERCISE 1

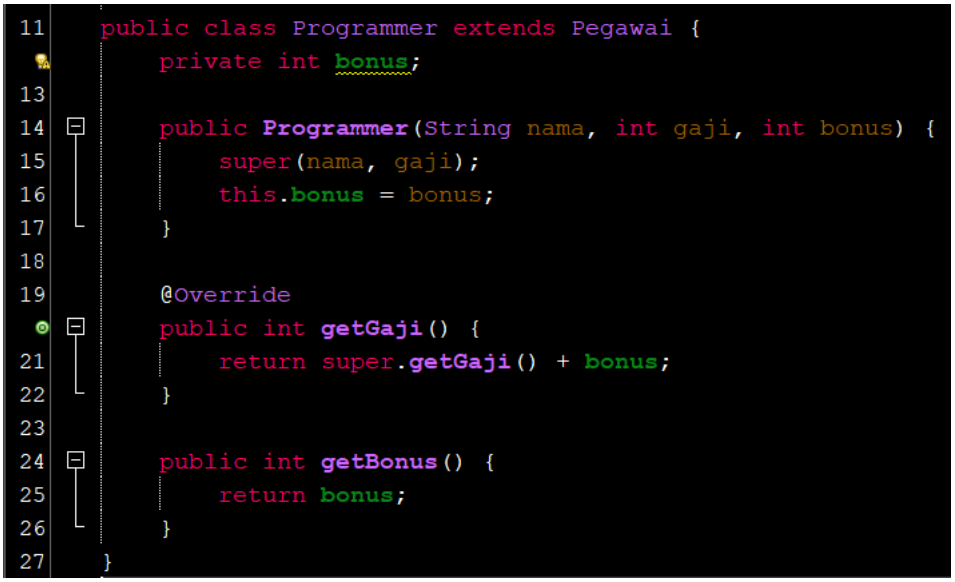
Pegawai class

```java
public class Pegawai {
    private String nama;
    private int gaji;

    public Pegawai() {
    }

    public Pegawai(String nama, int gaji) {
        this.nama = nama;
        this.gaji = gaji;
    }

    public int getGaji() {
        return gaji;
    }
}
```

Manajer class

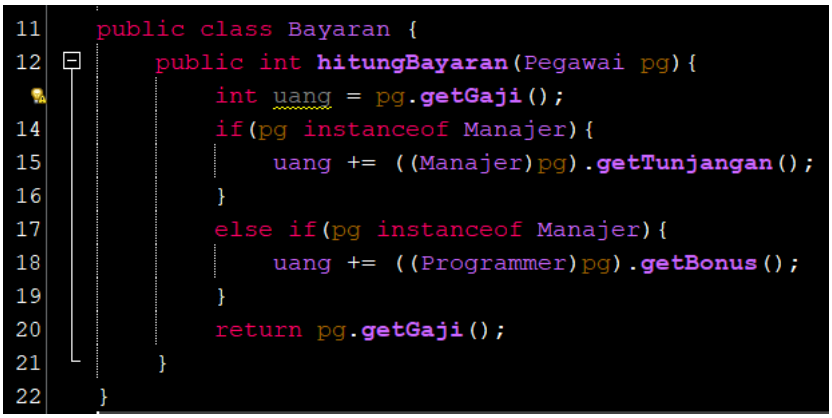
```java
public class Manajer extends Pegawai {
    private int tunjangan;

    public Manajer(String nama, int gaji, int tunjangan) {
        super(nama, gaji);
        this.tunjangan = tunjangan;
    }

    @Override
    public int getGaji() {
        return super.getGaji() + tunjangan;
    }

    public int getTunjangan() {
        return tunjangan;
    }
}
```

Programmer class:



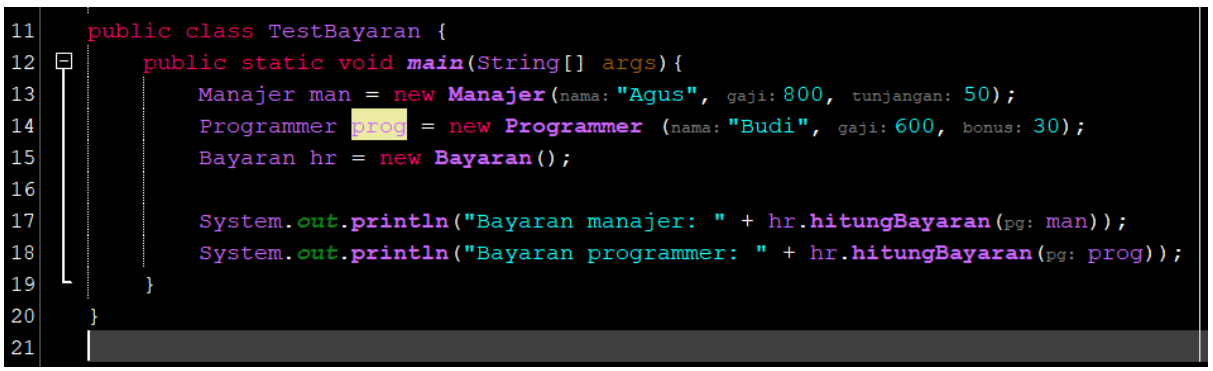
```
11  public class Programmer extends Pegawai {
12      private int bonus;
13
14      public Programmer(String nama, int gaji, int bonus) {
15          super(nama, gaji);
16          this.bonus = bonus;
17      }
18
19      @Override
20      public int getGaji() {
21          return super.getGaji() + bonus;
22      }
23
24      public int getBonus() {
25          return bonus;
26      }
27  }
```

Bayaran class:



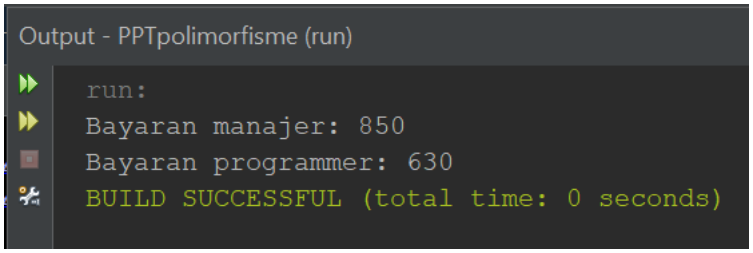
```
11  public class Bayaran {
12      public int hitungBayaran(Pegawai pg){
13          int uang = pg.getGaji();
14          if(pg instanceof Manajer){
15              uang += ((Manajer)pg).getTunjangan();
16          }
17          else if(pg instanceof Manajer){
18              uang += ((Programmer)pg).getBonus();
19          }
20          return pg.getGaji();
21      }
22  }
```

Main class:



```
11  public class TestBayaran {
12      public static void main(String[] args){
13          Manajer man = new Manajer(nama: "Agus", gaji: 800, tunjangan: 50);
14          Programmer prog = new Programmer (nama: "Budi", gaji: 600, bonus: 30);
15          Bayaran hr = new Bayaran();
16
17          System.out.println("Bayaran manajer: " + hr.hitungBayaran(pg: man));
18          System.out.println("Bayaran programmer: " + hr.hitungBayaran(pg: prog));
19      }
20  }
21
```

Result:



```
Output - PPTpolimorfisme (run)
run:
Bayaran manajer: 850
Bayaran programmer: 630
BUILD SUCCESSFUL (total time: 0 seconds)
```

EXERCISE 2

Elektronika class

```
public class Elektronika {
    private int voltase;

    public Elektronika(int voltase) {
        this.voltase = voltase;
    }

    public int getVoltase() {
        return voltase;
    }
}
```

TelevisiJadul class

```
public class TelevisiJadul extends Elektronika {
    private String modelInput;

    public TelevisiJadul(String modelInput) {
        super(voltase:110);
        this.modelInput = modelInput;
    }

    public String getModelInput() {
        return modelInput;
    }
}
```

TelevisiModern class

```
public class TelevisiModern extends Elektronika
    private String modelInput;

    public TelevisiModern(String modelInput) {
        super(voltase:220);
        this.modelInput = modelInput;
    }

    public String getModelInput() {
        return modelInput;
    }
}
```

Manusia class:

```
11    public class Manusia {
12        public void nyalakanPerangkat(Elektronika perangkat) {
13            if (perangkat instanceof TelevisiJadul) {
14                System.out.println("Nyalakan televisi jadul dengan input: " +
15                        ((TelevisiJadul) perangkat).getModelInput());
16            } else if (perangkat instanceof TelevisiModern) {
17                System.out.println("Nyalakan televisi modern dengan input: " +
18                        ((TelevisiModern) perangkat).getModelInput());
19            }
              System.out.println("Voltase televisi: " + perangkat.getVoltase());
21        }
22    }
```

Main class:

```
public class TestElektronik {
    public static void main(String[] args) {
        Manusia indro = new Manusia();
        TelevisiJadul tvJadul = new TelevisiJadul(modelInput:"DVI");
        TelevisiModern tvModern = new TelevisiModern(modelInput:"HDMI");

        indro.nyalakanPerangkat(perangkat: tvJadul);
        indro.nyalakanPerangkat(perangkat: tvModern);
    }
}
```

Result:

```
Output - PPTpolimorfisme (run)
run:
Nyalakan televisi jadul dengan input: DVI
Voltase televisi: 110
Nyalakan televisi modern dengan input: HDMI
Voltase televisi: 220
BUILD SUCCESSFUL (total time: 0 seconds)
```