

Object Oriented Programming

Polimorfism



Name

Virza Aulia Rachman

NIM

2241720078

Class

1i

Department

Information Technology

Study Program

D4 Informatics Engineering

Practicum 1

```
package practicum1;

2 usages
public class ElectricityBill implements Employee.Payable {
    4 usages
    private int kwh;
    5 usages
    private String category;

    1 usage
    public ElectricityBill(int kwh,String category){
        this.kwh = kwh;
        this.category= category;
    }

    no usages
    public int getKwh() {
        return kwh;
    }

    no usages
    public String getCategory() {
        return category;
    }

    no usages
    public void setCategory(String category) {
        this.category = category;
    }

    no usages
    @Override
    public int getPaymentAmount() {
        return kwh+getBasePrice();
    }

    2 usages
    public int getBasePrice(){
        int bPrice = 0 ;
        switch(category){
            case "R-1" : bPrice = 100; break;
            case "R-2" : bPrice = 200; break;
        }
        return bPrice;
    }

    no usages
    public String getBillInfo(){
        return "kwh = "+kwh+"\n"+
            "category = "+category+"("+getBasePrice()+ " per kwh\n";
    }
}
```

```
package practicum1;

2 usages
public class InternshipEmployee extends Employee{
    4 usages
    private int length;

    1 usage
    public InternshipEmployee(String name,int length){
        this.length = length;
        this.name = name;
    }

    no usages
    public int getLength(){
        return length;
    }

    no usages
    public void setLength(int length) {
        this.length = length;
    }

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

```
package practicum1;

2 usages
public class PermanentEmployee extends Employee implements Employee.Payable {
    6 usages
    private int salary;

    1 usage
    public PermanentEmployee(String name,int salary){
        this.name = name;
        this.salary = salary;
    }

    no usages
    public int getSalary() {
        return salary;
    }

    no usages
    public void setSalary(int salary) {
        this.salary = salary;
    }

    no usages
    @Override
    public int getPaymentAmount() {
        return (int) (salary*0.5*salary);
    }

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

```
package practicum1;

6 usages 2 inheritors
public class Employee {
    3 usages
    protected String name;
    2 usages 2 overrides
    public String getEmployeeInfo(){
        return "Name = "+name;
    }

    3 usages 2 implementations
    public interface Payable{
        no usages 2 implementations
        public int getPaymentAmount();
    }
}
```

```
package practicum1;

public class Tester1 {
    public static void main(String[] args){
        PermanentEmployee pEmp = new PermanentEmployee("John", salary: 500);
        InternshipEmployee iEmp = new InternshipEmployee("Kevin", length: 5);
        ElectricityBill eBill = new ElectricityBill(1000, 5, category: "A-1");
        Employee e;
        Employee.Payable p;
        e = pEmp;
        e = iEmp;
        p = pEmp;
        p = eBill;
    }
}
```

Question

1. Class apa sajakah yang merupakan turunan dari class **Employee**?
2. Class apa sajakah yang implements ke 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 **InternshipEmployee**) ?
4. 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**) ?
5. Coba tambahkan sintaks:

```
p = iEmp;  
e = eBill;
```

pada baris 14 dan 15 (baris terakhir dalam method **main**) ! Apa yang menyebabkan error?
6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!

Answer:

1. **PermanentEmployee**, **InternshipEmployee**
2. **PermanentEmployee**, **ElectricityBill**
3. Karena objek **pEmp** dan **iEmp** adalah turunan dari class **Employee**, dan variabel **e** memiliki tipe data **Employee**. Dalam konsep polimorfisme, objek dari subclass dapat diassign ke variabel tipe superclass.
4. Karena objek **pEmp** dan **eBill** mengimplementasikan interface **Payable**, dan variabel **p** memiliki tipe data **Employee**. **Payable**. Dalam konsep polimorfisme, objek yang mengimplementasikan suatu interface dapat diassign ke variabel dengan tipe data interface tersebut.
5. a.) Hal ini akan menyebabkan error karena objek **iEmp** tidak mengimplementasikan interface **Payable**, sehingga tidak dapat diassign ke variabel **p** yang memiliki tipe data **Employee**. **Payable**.
b.) Objek **eBill** akan menyebabkan error karena variabel **e** memiliki tipe data **Employee**, sedangkan **eBill** bukan merupakan turunan dari **Employee**.
6. Conclusion:
 - Polimorfisme adalah konsep dalam pemrograman berorientasi objek yang memungkinkan objek dari kelas yang berbeda dapat diakses menggunakan antarmuka yang sama.
 - Dalam contoh ini, polimorfisme terlihat pada penggunaan variabel dengan tipe data yang sesuai (superclass atau interface) untuk merujuk pada objek yang berasal dari kelas yang berbeda.
 - Konsep ini memungkinkan fleksibilitas dalam penggunaan objek dan menyederhanakan pengkodean

Practicum 2

```
package practicum1;

import java.sql.SQLOutput;

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

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

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

Question

1. Perhatikan class **Tester2** di atas, mengapa pemanggilan **e.getEmployeeInfo()** pada baris 8 dan **pEmp.getEmployeeInfo()** pada baris 10 menghasilkan hasil sama?
2. Mengapa pemanggilan method **e.getEmployeeInfo()** disebut sebagai pemanggilan method virtual (virtual method invocation), sedangkan **pEmp.getEmployeeInfo()** tidak?
3. Jadi apakah yang dimaksud dari virtual method invocation? Mengapa disebut virtual?

Answer

1. Karena variabel `e` memiliki tipe data `Employee`, tetapi merujuk pada objek yang sebenarnya merupakan instance dari `PermanentEmployee`. Saat memanggil `getEmployeeInfo()` melalui variabel `e`, JVM akan mengeksekusi metode dari kelas aktual objek yang ditunjuk pada saat runtime, yaitu `PermanentEmployee`. Ini disebut polimorfisme, di mana metode yang dipanggil tergantung pada jenis objek aktual, bukan jenis variabel.
2. Pemanggilan `e.getEmployeeInfo()` disebut pemanggilan method virtual karena metode yang akan dijalankan ditentukan pada saat runtime dan bergantung pada jenis objek aktual yang dimiliki oleh variabel `e` pada saat itu.
Pemanggilan `pEmp.getEmployeeInfo()` juga merupakan pemanggilan method virtual karena metode yang dieksekusi bergantung pada jenis objek aktual yang dimiliki oleh variabel `pEmp` pada saat runtime. Dalam kasus ini, objek yang dimiliki oleh `pEmp` adalah `PermanentEmployee`.
3. Virtual method invocation adalah konsep di pemrograman berorientasi objek di mana pemanggilan suatu metode ditentukan pada saat runtime, dan metode yang dieksekusi bergantung pada jenis objek aktual yang dimiliki oleh variabel pada saat itu.
Kata "virtual" mengacu pada kenyataan bahwa metode yang akan dijalankan tidak ditentukan secara statis selama kompilasi, tetapi pada saat runtime. Ini memberikan fleksibilitas dan kemampuan untuk menerapkan polimorfisme, di mana metode dapat disesuaikan dengan perilaku objek aktual yang dijalanannya.

Practicum 3

```
package practicum1;

no usages
public class Tester3 {
    3 usages
    PermanentEmployee pEmp = new PermanentEmployee( name: "Dedik", salary: 500);
    1 usage
    InternshipEmployee iEmp = new InternshipEmployee( name: "Sunarto", length: 5);
    1 usage
    ElectricityBill eBill = new ElectricityBill( kwh: 5, category: "A-1");
    no usages
    Employee e[] = {pEmp,iEmp};
    no usages
    Employee.Payable p[] = {pEmp,eBill};
    no usages
    Employee e2[] = {[pEmp,iEmp,eBill]};
}
```

Question

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**) ?
2. Perhatikan juga baris ke-9, mengapa array **p** juga diisi dengan objek-objek dengan tipe yang berbeda, yaitu objek **pEmp** (objek dari **PermanentEmployee**) dan objek **eBill** (objek dari **ElectricityBilling**) ?
3. Perhatikan baris ke-10, mengapa terjadi error?

Answer

1. Karena e adalah array yang berisi objek dari tipe Employee, dan baik PermanentEmployee maupun InternshipEmployee adalah turunan dari Employee. Dalam konsep polimorfisme, objek dari subclass dapat dimasukkan ke dalam array dengan tipe superclass
2. Karena p adalah array yang berisi objek dari tipe Employee.Payable, dan baik PermanentEmployee maupun ElectricityBill mengimplementasikan interface Employee.Payable. Dalam konsep polimorfisme, objek yang mengimplementasikan suatu interface dapat dimasukkan ke dalam array dengan tipe interface tersebut.
3. Terjadi error karena sintaks inisialisasi array e2 tidak benar. Seharusnya, sintaks inisialisasi array menggunakan kurung kurawal {} dan dipisahkan dengan koma, bukan menggunakan tanda kurung siku [].

Practicum 4

```
package practicum1;

2 usages
public class Owner {
    2 usages
    public void pay(Employee.Payable 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;
            System.out.println(" "+pe.getEmployeeInfo());
        }
    }
    2 usages
    public void showEmployee(Employee e){
        System.out.println(" "+e.getEmployeeInfo());
        if(e instanceof PermanentEmployee){
            System.out.println("You have to pay her/him monthly!");
        }else{
            System.out.println("No need to pay him/her");
        }
    }
}
```

```
package practicum1;

public class Tester4 {
    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("-----");

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

        InternshipEmployee iEmp= new InternshipEmployee( name: "Sunarto", length: 5);
        ow.showEmployee(pEmp);
        System.out.println("-----");
        ow.showEmployee(iEmp);
    }
}
```

```
Total payment = 105
kwh = 5
category = R-1(100 per 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 = Sunarto
Registered as internship employee for 5month/s

No need to pay him/her
```


Question

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**?

2. Jadi apakah tujuan membuat argument bertipe **Payable** pada method **pay()** yang ada di dalam class **Owner**?
3. Coba pada baris terakhir method **main()** yang ada di dalam class **Tester4** ditambahkan perintah **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        ow.pay(iEmp);
19    }
20 }
21 }
```

Mengapa terjadi error?

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

Answer

1. Karena baik ElectricityBill maupun PermanentEmployee mengimplementasikan interface Payable. Dalam pemanggilan ow.pay(eBill), objek eBill memiliki tipe Payable melalui implementasi interface, dan hal yang sama berlaku untuk pEmp.
2. Tujuan dari membuat argument bertipe Payable pada method pay() adalah untuk memberikan fleksibilitas, sehingga method tersebut dapat menerima objek dari kelas apa pun yang mengimplementasikan interface Payable. Dengan demikian, class Owner dapat bekerja dengan objek yang memiliki kemampuan pembayaran, baik itu objek dari kelas ElectricityBill maupun PermanentEmployee.
3. Terjadi error karena iEmp adalah objek dari kelas InternshipEmployee, yang tidak mengimplementasikan interface Payable. Sebagai hasilnya, objek iEmp tidak dapat diterima sebagai argumen pada method pay() yang membutuhkan objek bertipe Payable
4. Sintaks p instanceof ElectricityBill digunakan untuk memeriksa apakah objek yang disimpan dalam variabel p merupakan instance dari kelas ElectricityBill. Dalam konteks ini, ini dilakukan untuk menentukan jenis objek yang dikirim ke method pay() agar tindakan yang sesuai dapat diambil.
5. Casting objek (ElectricityBill eb = (ElectricityBill) p) diperlukan karena variabel p memiliki tipe data Payable, dan kita perlu mengakses metode dan properti yang spesifik untuk kelas ElectricityBill. Oleh karena itu, kita menggunakan casting untuk mengubah tipe data dari Payable menjadi ElectricityBill, sehingga kita dapat mengakses metode dan properti spesifik dari kelas ElectricityBill.

Assignment

- Zombie.java

```
package Assignment;


2 usages 2 inheritors
public class Zombie implements Destroyable{
    27 usages
    protected int health;
    12 usages
    protected int level;

    no usages 2 overrides
    public void heal() {
        if (level == 1) {
            health += health * 0.1;
        } else if (level == 2) {
            health += health * 0.3;
        } else if (level == 3) {
            health += health * 0.4;
        }
    }

    1 usage 2 overrides
    public void destroyed() {
        health -= health * 0.02;
    }

    6 usages 2 overrides
    public String getZombieInfo() {
        return "\nHealth = " + health + "\nLevel = " + level + "\n";
    }
}
```

- Destroyable.java



```
package Assignment;
```

3 usages 4 implementations

```
public interface Destroyable {  
    1 usage 4 implementations  
    public void destroyed();  
}
```

- Barrier.java

```
package Assignment;

2 usages
public class Barrier implements Destroyable{
    6 usages
    private int health;

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

    no usages
    public void setStrength(int strength) {
        this.health = strength;
    }

    no usages
    public int getStrength() {
        return health;
    }

    1 usage
    public void destroyed() {
        health -= health * 0.1;
    }

    2 usages
    public String getBarrierInfo() {
        return "\nBarrier Strength = " + health + "\n";
    }
}
```

- WalkingZombie.java

```
package Assignment;

2 usages
public class WalkingZombie extends Zombie{
    1 usage
    public WalkingZombie(int health, int level) {
        this.health = health;
        this.level = level;
    }

    no usages
    public void heal() {
        if (level == 1) {
            health += health * 0.1;
        } else if (level == 2) {
            health += health * 0.3;
        } else if (level == 3) {
            health += health * 0.4;
        }
    }

    1 usage
    public void destroyed() {
        health -= health * 0.19;
    }

    6 usages
    public String getZombieInfo() {
        return "\nWalking Zombie Data = " + super.getZombieInfo();
    }
}
```

- JumpingZombie.java

```
package Assignment;
```

2 usages

```
public class JumpingZombie extends Zombie {
```

1 usage

```
    public JumpingZombie(int health, int level) {  
        this.health = health;  
        this.level = level;  
    }
```

no usages

```
    public void heal() {  
        if (level == 1) {  
            health += health * 0.3;  
        } else if (level == 2) {  
            health += health * 0.4;  
        } else if (level == 3) {  
            health += health * 0.5;  
        }  
    }
```

1 usage

```
    public void destroyed() {  
        health -= health * 0.095;  
    }
```

6 usages

```
    public String getZombieInfo() {  
        return "\nJumping Zombie Data = " + super.getZombieInfo();  
    }  
}
```

- Plant.java

```

package Assignment;

2 usages
public class Plant {
    3 usages
    public void doDestroy(Destroyable d) {
        d.destroyed();
    }
}

```

- Tester.java and result

```

package Assignment;

public class Tester {
    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("-----");
        for (int i = 0; i < 4; i++) { // destroying the enemies 4 times
            p.doDestroy(wz);
            p.doDestroy(jz);
            p.doDestroy(b);
        }
        System.out.println(wz.getZombieInfo());
        System.out.println(jz.getZombieInfo());
        System.out.println(b.getBarrierInfo());
    }
}

```

```

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

```