



NAMA : Jud Amal Mukhtar
NIM : 2041720168
KELAS : 2C / TI
MATERI : Polimorfisme

Percobaan 1

Pertanyaan :

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

Jawab :

InternshipEmployee 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 **InternshipEmployee**) ?

Jawab :

karena 'e' merupakan objek dari class Employee yang merupakan superclass dari objek 'pEmp' dan 'iEmp', yang masih memakai atribut dari parent-nya

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**) ?

Jawab :

karena 'p' yang merupakan objek dari class Payable merupakan subjek implementasi dari objek 'pEmp' dan 'eBill'

5. Coba tambahkan sintaks:

p = iEmp;

e = eBill;

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

Jawab :

'iEmp' tidak mengimplementasi objek 'p' dan 'eBill' bukan merupakan child dari 'e'

6. Ambil kesimpulan tentang konsep/bentuk dasar polimorfisme!

Jawab :

polimorfisme digunakan ketika sebuah method yang sama pada beberapa class tetapi isinya sedikit berbeda



NAMA : Jud Amal Mukhtar
NIM : 2041720168
KELAS : 2C / TI
MATERI : Polimorfisme

Percobaan 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 sama-sama menggunakan method `getEmployeeInfo` dari objek 'pEmp'

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

Jawab :

karena objek 'e' menggunakan method dari class turunan atau child-nya sedangkan 'pEmp' dari class-nya sendiri.

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

Jawab :

virtual method invocation adalah penggunaan method yang dilakukan superclass kepada subclass ketika adanya override method. Disebut virtual dikarenakan method tidak diakses langsung oleh class-nya sendiri, melainkan harus membuat objek yang mengarah pada method di luar class.



NAMA : Jud Amal Mukhtar
NIM : 2041720168
KELAS : 2C / TI
MATERI : Polimorfisme

Percobaan 3

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**) ?

Jawab :

karena 'pEmp' dan 'iEmp' merupakan subclass dari 'e' maka kedua objek tersebut dapat dimasukkan kedalam array-nya

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**) ?

Jawab :

karena 'pEmp' dan 'eBill' sama-sama mengimplementasikan objek 'p' maka bisa dimasukkan kedalam array-nya

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

Jawab :

karena 'eBill' bukan merupakan subclass dari objek 'e2'



NAMA : Jud Amal Mukhtar
NIM : 2041720168
KELAS : 2C / TI
MATERI : Polimorfisme

Percobaan 4

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 'pEmp' dan 'eBill' mengimplementasi Payable, oleh karena itu semua container yang menerima objek dari Payable dapat dimasukkan oleh objek yang mengimplementasikannya

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

Jawab :

agar dapat dimasuki objek yang akan dibayar atau yang mengimplementasikan Payable

3. Coba pada baris terakhir method **main()** yang ada di dalam class **Tester4** ditambahkan perintah **ow.pay(iEmp);**
Mengapa terjadi error?

Jawab :

objek 'iEmp' tidak terhubung dengan Payable atau tidak mengimplementasikan Payable

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

Jawab :

untuk mengecek apakah objek 'p'(objek masukan) merupakan objek yang sama dengan class ElectricityBill

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 mencegah keambiguan objek maka ketika objek 'p' di-casting maka yang masuk merupakan objek dari class ElectricityBill dan bukan dari class interface Payable



NAMA : Jud Amal Mukhtar
NIM : 2041720168
KELAS : 2C / TI
MATERI : Polimorfisme

TUGAS

1. IDestroyable

```
1 package destroyzombie;  
2 public interface IDestroyable {  
3     public abstract void destroyed();  
4 }
```

2. Barrier

```
1 package destroyzombie;  
2 public class Barrier implements IDestroyable{  
3     private int strength;  
4  
5     public Barrier(int strength) {  
6         this.strength = strength;  
7     }  
8  
9     public int getStrength() {  
10        return strength;  
11    }  
12  
13    public void setStrength(int strength) {  
14        this.strength = strength;  
15    }  
16  
17    @Override  
18    public void destroyed() {  
19        strength*=0.9;  
20    }  
21  
22    public String getBarrierInfo(){  
23        return "Barrier Strength = "+strength+"\n";  
24    }  
25  
26 }
```

3. Zombie

```
1 package destroyzombie;  
2 public abstract class Zombie implements IDestroyable{  
3     protected int health,level;  
4     public abstract void heal();  
5     @Override  
6     public abstract void destroyed();  
7     public String getZombieInfo(){  
8         return "Health = "+health+"\nLevel = "+level+"\n";  
9     }  
10 }
```



NAMA : Jud Amal Mukhtar
NIM : 2041720168
KELAS : 2C / TI
MATERI : Polimorfisme

4. WalkingZombie

```
1  package destroyzombie;|
2  public class WalkingZombie extends Zombie{
3
4      public WalkingZombie(int health,int level) {
5          this.health=health;
6          this.level=level;
7      }
8
9      @Override
10     public void heal() {
11         switch(level){
12             case 1:health*=1.1;break;
13             case 2:health*=1.3;break;
14             case 3:health*=1.4;break;
15         }
16     }
17
18     @Override
19     public void destroyed() {
20         health-=health*20/100;
21     }
22
23     public String getZombieInfo(){
24         return "Walking Zombie Data =\n"+super.getZombieInfo();
25     }
26
27 }
```



NAMA : Jud Amal Mukhtar
NIM : 2041720168
KELAS : 2C / TI
MATERI : Polimorfisme

5. JumpingZombie

```
1 package destroyzombie;  
2 public class JumpingZombie extends Zombie{  
3  
4     public JumpingZombie(int health,int level) {  
5         this.health=health;  
6         this.level=level;  
7     }  
8  
9     @Override  
10    public void heal() {  
11        switch(level){  
12            case 1:health*=1.3;break;  
13            case 2:health*=1.4;break;  
14            case 3:health*=1.5;break;  
15        }  
16    }  
17  
18    @Override  
19    public void destroyed() {  
20        health-=health*10/100;  
21    }  
22  
23    public String getZombieInfo(){  
24        return "Jumping Zombie Data =\n"+super.getZombieInfo();  
25    }  
26 }
```

6. Plant

```
1 package destroyzombie;  
2 public class Plant {  
3     public void doDestroy(IDestroyable d){  
4         if(d instanceof WalkingZombie){  
5             ((WalkingZombie) d).destroyed();  
6         }else if(d instanceof JumpingZombie){  
7             ((JumpingZombie) d).destroyed();  
8         }else if(d instanceof Barrier){  
9             ((Barrier)d).destroyed();  
10        }  
11    }  
12 }
```



NAMA : Jud Amal Mukhtar
NIM : 2041720168
KELAS : 2C / TI
MATERI : Polimorfisme

7. Tester

```
1 package destroyzombie;  
2 public class Tester {  
3     public static void main(String[] args) {  
4         WalkingZombie wz=new WalkingZombie(100,1);  
5         JumpingZombie jz=new JumpingZombie(100,2);  
6         Barrier b=new Barrier(100);  
7         Plant p=new Plant();  
8         System.out.println(""+wz.getZombieInfo());  
9         System.out.println(""+jz.getZombieInfo());  
10        System.out.println(""+b.getBarrierInfo());  
11        System.out.println("-----");  
12        for(int i=0;i<4;i++){  
13            p.doDestroy(wz);  
14            p.doDestroy(jz);  
15            p.doDestroy(b);  
16        }  
17        System.out.println(""+wz.getZombieInfo());  
18        System.out.println(""+jz.getZombieInfo());  
19        System.out.println(""+b.getBarrierInfo());  
20    }  
21 }  
22 }
```

8. 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: 1 second)
```