

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 **InternshipEmploye**) ?

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?

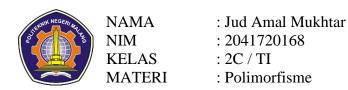
Jawab:

'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



Percobaan 2

Pertanyaan:

 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'

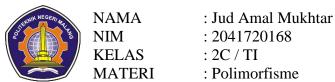
2. Mengapa pemanggilan method **e.getEmployeeInfo()** disebut sebagai pemanggilan method virtual (virtual method invication), sedangkan **pEmp.getEmployeeInfo()** tidak?

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.



: 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 biisi dengan objek-objek dengan tipe yang berbeda, yaitu objek pEmp (objek dari PermanentEmployee) dan objek eBill (objek dari ElectricityBilling)?

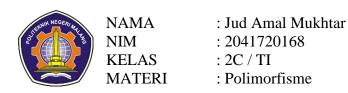
Jawab:

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'



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

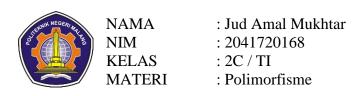
4. Perhatikan class **Owner**, diperlukan untuk apakah sintaks **p** instanceof ElectricityBill pada baris ke-6?

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



TUGAS

1. IDestroyable

```
package destroyzombie;
public interface IDestroyable {
    public abstract void destroyed();
}
```

2. Barrier

```
package destroyzombie;
public class Barrier implements IDestroyable{
    private int strength;

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

public int getStrength() {
    return strength;
}

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

public void destroyed() {
    strength*=0.9;
}

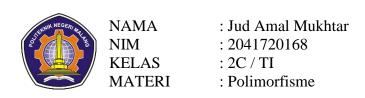
public String getBarrierInfo(){
    return "Barrier Strength = "+strength+"\n";
}

public String getBarrierInfo(){
```

3. Zombie

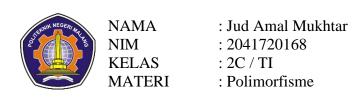
```
package destroyzombie;
public abstract class Zombie implements IDestroyable{
    protected int health,level;
    public abstract void heal();

@Override
    public abstract void destroyed();
public String getZombieInfo(){
    return "Health = "+health+"\nLevel = "+level+"\n";
}
```



4. WalkingZombie

```
package destroyzombie;
      public class WalkingZombie extends Zombie{
          public WalkingZombie(int health,int level) {
              this.level=level;
          @Override
          public void heal() {
 0
              switch(level){
                  case 1:health*=1.1;break;
                  case 2:health*=1.3;break;
                  case 3:health*=1.4;break;
          @Override
          public void destroyed() {
0
              health-=health*20/100;
          public String getZombieInfo(){
%↓
              return "Walking Zombie Data =\n"+super.getZombieInfo();
```



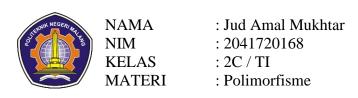
5. JumpingZombie

```
ckage destroyzombie;
      public class JumpingZombie extends Zombie{
          public JumpingZombie(int health,int level) {
               this.health=health;
              this.level=level;
          @Override
          public void heal() {
0
              switch(level){
                  case 1:health*=1.3;break;
                   case 2:health*=1.4;break;
                   case 3:health*=1.5;break;
14
           }
          @Override
          public void destroyed() {
0
              health-=health*10/100;
          public String getZombieInfo(){
<u>۾</u>.
              return "Jumping Zombie Data =\n"+super.getZombieInfo();
```

6. Plant

```
package destroyzombie;
public class Plant {

public void doDestroy(IDestroyable d) {
    if(d instanceof WalkingZombie) {
        ((WalkingZombie) d).destroyed();
    }else if(d instanceof JumpingZombie) {
        ((JumpingZombie) d).destroyed();
    }else if(d instanceof Barrier) {
        ((Barrier)d).destroyed();
    }
}
```



7. Tester

```
package destroyzombie;
      public class Tester {
           public static void main(String[] args) {
               WalkingZombie wz=new WalkingZombie(100,1);
               JumpingZombie jz=new JumpingZombie(100,2);
               Barrier b=new Barrier(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++){</pre>
                    p.doDestroy(wz);
                    p.doDestroy(jz);
14
                    p.doDestroy(b);
               System.out.println(""+wz.getZombieInfo());
               System.out.println(""+jz.getZombieInfo());
               System.out.println(""+b.getBarrierInfo());
```

8. Output