

Zid'Avwa Al Bari'i

244107020083 / TI – 2I / 26

All the code here is on GitHub

<https://github.com/ZidAvwa/CollegeStudy/tree/main/3rdSemester/11.%20Jobsheet12>

Jobsheet 11

Exercise 1

Codes

The screenshot shows five code files in a Java IDE:

- Employee.java**:

```
src > J Employee.java > Employee
1 public class Employee {
2     protected String name;
3     public String getEmployeeInfo() { return "Name: " + name;
4 }
```
- Payable.java**:

```
src > J Payable.java > Payable > getPaymentAmount()
1 public interface Payable {
2     public int getPaymentAmount();
3 }
```
- PermanentEmployee.java**:

```
src > J PermanentEmployee.java > PermanentEmployee > getSalary()
1 public class PermanentEmployee extends Employee implements Payable {
2     private int SALARY;
3
4     public PermanentEmployee(string name, int salary) {
5         this.name = name;
6         this.salary = salary;
7     }
8
9     public int getSalary() { return salary; }
10    public void setSalary(int salary) { this.salary = salary; }
11
12    @Override
13    public string getEmployeeInfo() {
14        string info = super.getEmployeeInfo() + "\n";
15        info += "Registered as permanent employee for " + length + " month\n";
16        return info;
17    }
18 }
```
- InternshipEmployee.java**:

```
src > J InternshipEmployee.java > InternshipEmployee > getLength()
1 public class InternshipEmployee extends Employee {
2     private int length;
3     public InternshipEmployee(string name, int length) {
4         this.length = length;
5         this.name = name;
6     }
7
8     public int getLength() { return length; }
9     public void setLength(int length) { this.length = length; }
10
11    @Override
12    public string getEmployeeInfo() {
13        string info = super.getEmployeeInfo() + "\n";
14        info += "Registered as internship employee for " + length + " month\n";
15        return info;
16    }
17 }
```
- ElectricityBill.java**:

```
src > J ElectricityBill.java > ElectricityBill > getCategory()
1 public class ElectricityBill implements Payable {
2     private int bill;
3     private String category;
4
5     public ElectricityBill(int bill, String category) {
6         this.bill = bill;
7         this.category = category;
8     }
9
10    public int getBill() { return bill; }
11    public void setBill(int bill) { this.bill = bill; }
12    public String getCategory() { return category; }
13    public void setCategory(String category) { this.category = category; }
14
15    @Override
16    public int getPaymentAmount() { return bill * getBasePrice(); }
17
18    public int getBasePrice() {
19        int METER_PRICE = 10;
20        switch(category) {
21            case "B":   bill = 100; break;
22            case "C":   bill = 150; break;
23            case "D":   bill = 200; break;
24        }
25        return bill;
26    }
27
28    public String getBillInfo() {
29        return "Bill: " + bill + " Category: " + category + " BasePrice: " + getBasePrice() + " per kWh";
30    }
31 }
```
- Test.java**:

```
src > J Test.java > Test
1 public class Test {
2
3     public static void main(string[] args) {
4         PermanentEmployee pEmp = new PermanentEmployee("Zid", 500);
5         InternshipEmployee iEmp = new InternshipEmployee("Sarwita", 5);
6         ElectricityBill eBill = new ElectricityBill(100, "B");
7
8         Employee e = pEmp;
9         Payable p = eBill;
10
11        e = iEmp;
12        p = eBill;
13
14        System.out.println("Employee: " + e.getEmployeeInfo());
15        System.out.println("Payable: " + p.getPaymentAmount());
16    }
17 }
```

Questions Answers

- What classes are derived from the class Employee?**
PermanentEmployee and InternshipEmployee.
- What classes implement the interface Payable?**
PermanentEmployee and ElectricityBill.
- Why can e be filled with pEmp and iEmp?**
Because both PermanentEmployee (pEmp) and InternshipEmployee (iEmp) are child classes (subclasses) of the Employee class.
- Why can p be filled with pEmp and eBill?**
Because both the PermanentEmployee class (pEmp) and the ElectricityBill class (eBill) implement the Payable interface.
- What causes the error for p = iEmp; and e = eBill;?**

- **p = iEmp;** causes an error because the InternshipEmployee class does **not** implement the Payable interface.
- **e = eBill;** causes an error because the ElectricityBill class does **not** extend the Employee class.

6. Draw conclusions about the basic concepts/forms of polymorphism!

Polymorphism allows a variable of a parent class type (like Employee) to hold objects of its different child classes. It also allows a variable of an interface type (like Payable) to hold objects of any class that implements that interface.

Exercise 2

Code

```

1 package com.abc;
2
3 public class Testers {
4     public static void main(String[] args) {
5         PermanentEmployee pEmp = new PermanentEmployee("Dedik", 500);
6         Employee e;
7
8         e = pEmp;
9         System.out.println("Name = " + e.getEmployeeInfo());
10        System.out.println("Name = " + pEmp.getEmployeeInfo());
11    }
12 }

```

Questions Answers

1. Why do **e.getEmployeeInfo()** and **pEmp.getEmployeeInfo()** produce the same result?

Because both variables, e and pEmp, are pointing to the exact same object in memory (the PermanentEmployee object created on line 3). Even though e is an Employee type, at runtime, Java calls the overridden getEmployeeInfo() method from the actual object's class, which is PermanentEmployee.

2. Why is **e.getEmployeeInfo()** called a virtual method invocation, but **pEmp.getEmployeeInfo()** is not?

Because e is a variable of the parent class (Employee). When you call e.getEmployeeInfo(), the Java Virtual Machine (JVM) must check at runtime what kind of object e actually is (in this case, a PermanentEmployee) to know which version of the method to run. This runtime check is the "virtual" invocation.

For pEmp, the variable type is already PermanentEmployee, so the compiler knows exactly which method to call at compile time. There is no runtime check needed.

3. So what is meant by virtual method invocation? Why is it called virtual?

It means the specific method that gets executed is determined at runtime based on the actual object's type, not the variable's declared type.

It's called "virtual" because the exact method to be called isn't known at compile time; the decision is deferred until the program is running.

Exercise 3

Code

```
src > J Tester3.java > tp Tester3
 1 public class Tester3 {
 2     Run|Debug
 3     public static void main(String[] args) {
 4         PermanentEmployee pEmp = new PermanentEmployee(name: "Dedik", salary: 500);
 5         InternshipEmployee iEmp = new InternshipEmployee(name: "Sunarto", length: 5);
 6         ElectricityBill eBill = new ElectricityBill(kwh: 5, category: "A-1");
 7         Employee e[] = {pEmp, iEmp};
 8         Payable p[] = {pEmp, eBill};
 9         Employee e2[] = {pEmp, iEmp, eBill};    Type mismatch: cannot convert from E
```

Questions Answers

1. Why can the e array be filled with pEmp and iEmp?

Because the array e is of type Employee (Employee[]), and both PermanentEmployee (pEmp) and InternshipEmployee (iEmp) are child classes that extend the Employee class.

- ## 2. Why can the p array be filled with pEmp and eBill?

Because the array `p` is of type `Payable` (`Payable[]`), and both the `PermanentEmployee` class (`pEmp`) and the `ElectricityBill` class (`eBill`) implement the `Payable` interface.

3. Why does an error occur on line 8 (`Employee e2[] = ...`)?

Trying to put an ElectricityBill object (eBill) into an Employee array. The ElectricityBill class does not extend the Employee class, so it's not considered an Employee.

Exercise 4

Code

The screenshot shows a Java development environment with two code editors and two terminal panes displaying the execution results.

Code Editor 1:

```
1 public class Owner {
2     public void pay(Payable p) {
3         System.out.println("Total payment = " + p.getPaymentAmount());
4         if(p instanceof ElectricityBill) {
5             ElectricityBill ebill = (ElectricityBill)p;
6             System.out.println("ebill.getBillInfo()");
7         } else if(p instanceof PermanentEmployee) {
8             PermanentEmployee pe = (PermanentEmployee)p;
9             pe.getEmployeeInfo();
10            System.out.println(pe.getEmployeeInfo());
11        }
12    }
13
14    public void steadyEmployee(Employee e) {
15        System.out.println("e.getEmployeeInfo()");
16        if(e instanceof PermanentEmployee) {
17            System.out.println("You have to pay her/him monthly!!!");
18        } else {
19            System.out.println("No need to pay him/her :)");
20        }
21    }
22
23
24 }
```

Output Terminal 1:

```
Total payment = 580
idM1 = 3
Category = B-1(100 per idM1)

-----
Total payment = 525
Name = Dedik
Registered as permanent employee with salary 300

-----
Name = Dedik
Registered as permanent employee with salary 300

You have to pay her/him monthly!!!

Name = Sumantri
Registered as internship employee for 3 month/s

No need to pay him/her :)
```

Code Editor 2:

```
1 package com.learntocode;
2
3 public class TestM1 {
4     public static void main(String[] args) {
5         Owner ob = new Owner();
6
7         ElectricityBill ebill = new ElectricityBill(idM1=5, category="B-1");
8         ob.pay(ebill);
9         System.out.println();
10
11
12         PermanentEmployee pEmp = new PermanentEmployee(name="Dedik", salary=300);
13         ob.pay(pEmp);
14         System.out.println();
15
16         InternshipEmployee iEmp = new InternshipEmployee(name="Sumantri", length=3);
17         ob.steadyEmployee(iEmp);
18         System.out.println();
19         ob.steadyEmployee(iEmp);
20
21     }
22
23 }
```

Output Terminal 2:

```
ebill.getBillInfo()
-----
idM1 = 5
Category = B-1(100 per idM1)

-----
pEmp.getEmployeeInfo()
-----
Name = Dedik
Salary = 300
Category = Permanent Employee

-----
iEmp.getEmployeeInfo()
-----
Name = Sumantri
Length = 3 months
Category = Internship Employee
```

Questions Answers

1. Why can `ow.pay(eBill)` and `ow.pay(pEmp)` be called?

Because the `pay()` method requires a `Payable` argument. Both the `ElectricityBill` class

and the PermanentEmployee class implement the Payable interface, so objects of both classes are valid.

2. What is the purpose of making the argument type Payable?

It makes the pay() method flexible. It can accept any object from any class (now or in the future) as long as that class implements the Payable interface.

3. Why does `ow.pay(iEmp);` cause an error?

Because iEmp is an InternshipEmployee object. The InternshipEmployee class does not implement the Payable interface, so it cannot be used as an argument for the pay() method.

4. Why is the `if(p instanceof ElectricityBill)` check needed?

The check is needed to see if the generic Payable object p is specifically an instance of ElectricityBill. This is done so we can safely use methods (like getBillInfo()) that only exist in the ElectricityBill class.

5. Why is the cast (`ElectricityBill eb = (ElectricityBill) p;`) necessary?

The cast is necessary to access methods outside of the Payable interface.

The variable p is of type Payable, so you can only call p.getPaymentAmount(). By casting it to (ElectricityBill), telling the compiler to treat it as an ElectricityBill object, which allows to call its specific methods, like eb.getBillInfo().

TASK

Code

```
src > J Zombie.java > 📁 Zombie > ⚡ heal()
1  public abstract class Zombie implements Destroyable {
2      protected double health;
3      protected int level;
4
5      public Zombie(int health, int level) {
6          this.health = health;
7          this.level = level;
8      }
9
10     public abstract void heal();
11     @Override
12     public abstract void destroyed();
13
14     public String getZombieInfo() {
15         return "Health = " + (int) this.health + "\nLevel = " + this.level;
16     }
17 }
```

```
src > J WalkingZombie.java > 📁 WalkingZombie > ⚡ getZombieInfo()
1  public class WalkingZombie extends Zombie {
2      public WalkingZombie(int health, int level) { super(health, level); }
3
4      @Override
5      public void heal() {
6          if (level == 1) {
7              health += (health * 0.10);
8          } else if (level == 2) {
9              health += (health * 0.30);
10         } else if (level == 3) {
11             health += (health * 0.40);
12         }
13     }
14
15     @Override
16     public void destroyed() { health -= 14.5; }
17
18     @Override
19     public String getZombieInfo() {
20         return "Walking Zombie Data =\n" + super.getZombieInfo();
21     }
22 }
```

```
src > J JumpingZombie.java > JumpingZombie > getZombieInfo()
1  public class JumpingZombie extends Zombie {
2      public JumpingZombie(int health, int level) { super(health, level); }
3
4      @Override
5      public void heal() {
6          if (level == 1) { health += (health * 0.30); }
7          else if (level == 2) { health += (health * 0.40); }
8          else if (level == 3) { health += (health * 0.50); }
9      }
10
11     @Override
12     public void destroyed() { health -= 8.5; }
13
14     @Override
15     public String getZombieInfo() {
16         return "Jumping Zombie Data =\n" + super.getZombieInfo();
17     }
18 }
```

```
src > J Destroyable.java > Destroyable
1  public interface Destroyable { public void destroyed(); }
```

```
src > J Barrier.java > Barrier > destroyed()
1  public class Barrier implements Destroyable {
2      private int strength;
3
4      public Barrier(int strength) { this.strength = strength; }
5      public void setStrength(int strength) { this.strength = strength; }
6      public int getStrength() { return strength; }
7
8      @Override
9      public void destroyed() { strength -= 9; }
10     public String getBarrierInfo() { return "Barrier Strength = " + strength; }
11 }
```

```
src > J Plant.java > ↗ Plant > doDestroy(Destroyable)
1  public class Plant {
2      public void doDestroy(Destroyable d) { d.destroyed(); }
3  }

src > J Tester.java > ↗ Tester
1  public class Tester {
    Run|Debug
2      public static void main(String[] args) {
3          WalkingZombie wz = new WalkingZombie(health: 100, level: 1);
4          JumpingZombie jz = new JumpingZombie(health: 100, level: 2);
5          Barrier b = new Barrier(strength: 100);
6          Plant p = new Plant();
7
8          System.out.println(" " + wz.getZombieInfo());
9          System.out.println(" " + jz.getZombieInfo());
10         System.out.println(" " + b.getBarrierInfo());
11
12         System.out.println(x: -----);
13         for (int i = 0; i < 4; i++) {
14             p.doDestroy(wz);
15             p.doDestroy(jz);
16             p.doDestroy(b);
17         }
18
19         System.out.println(" " + wz.getZombieInfo());
20         System.out.println(" " + jz.getZombieInfo());
21         System.out.println(" " + b.getBarrierInfo());
22     }
23 }
```

```
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
```