Object Oriented Programming Job Sheet Overloading and Overriding

From:

AL AZHAR RIZQI RIFA'I FIRDAUS

Class:

2 I

Absence:

01

Student Number Identity:

2241720263

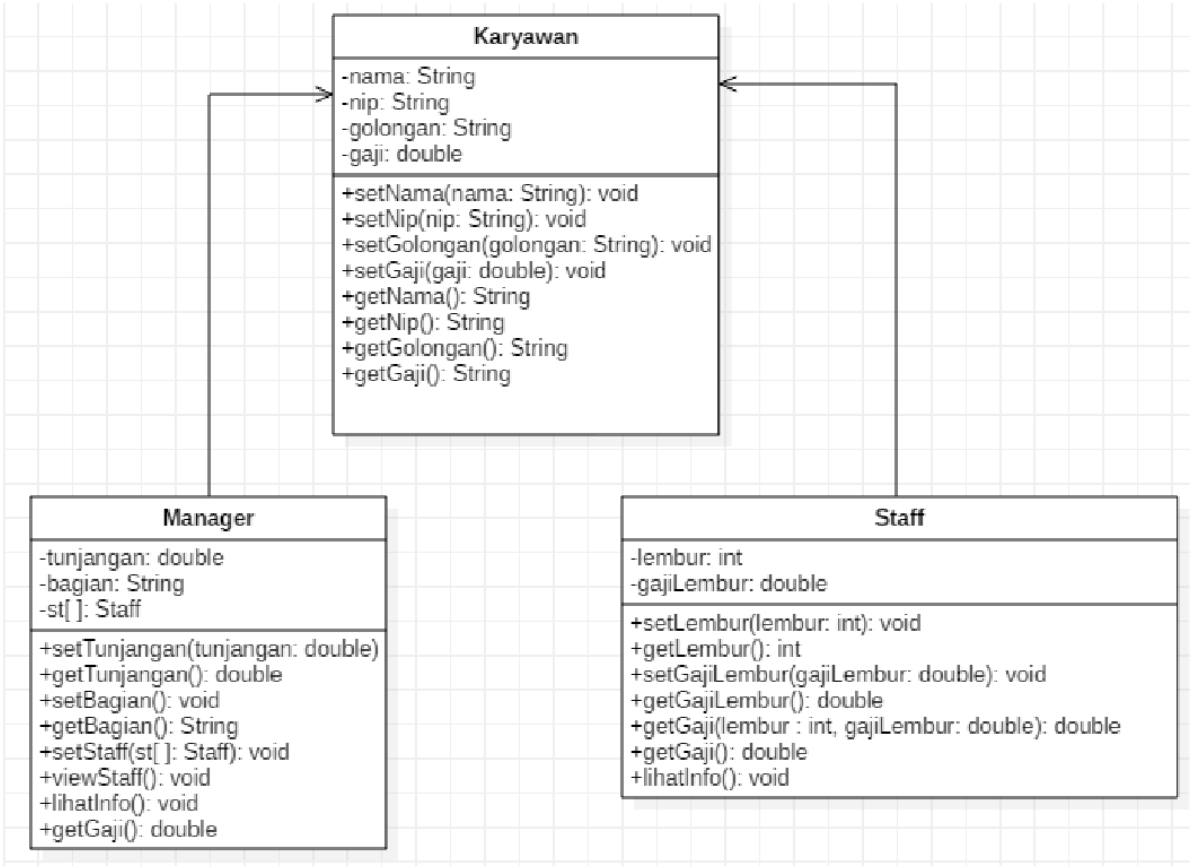
Department:

Information Technology

Study Program:

Informatics Engineering

Experiment

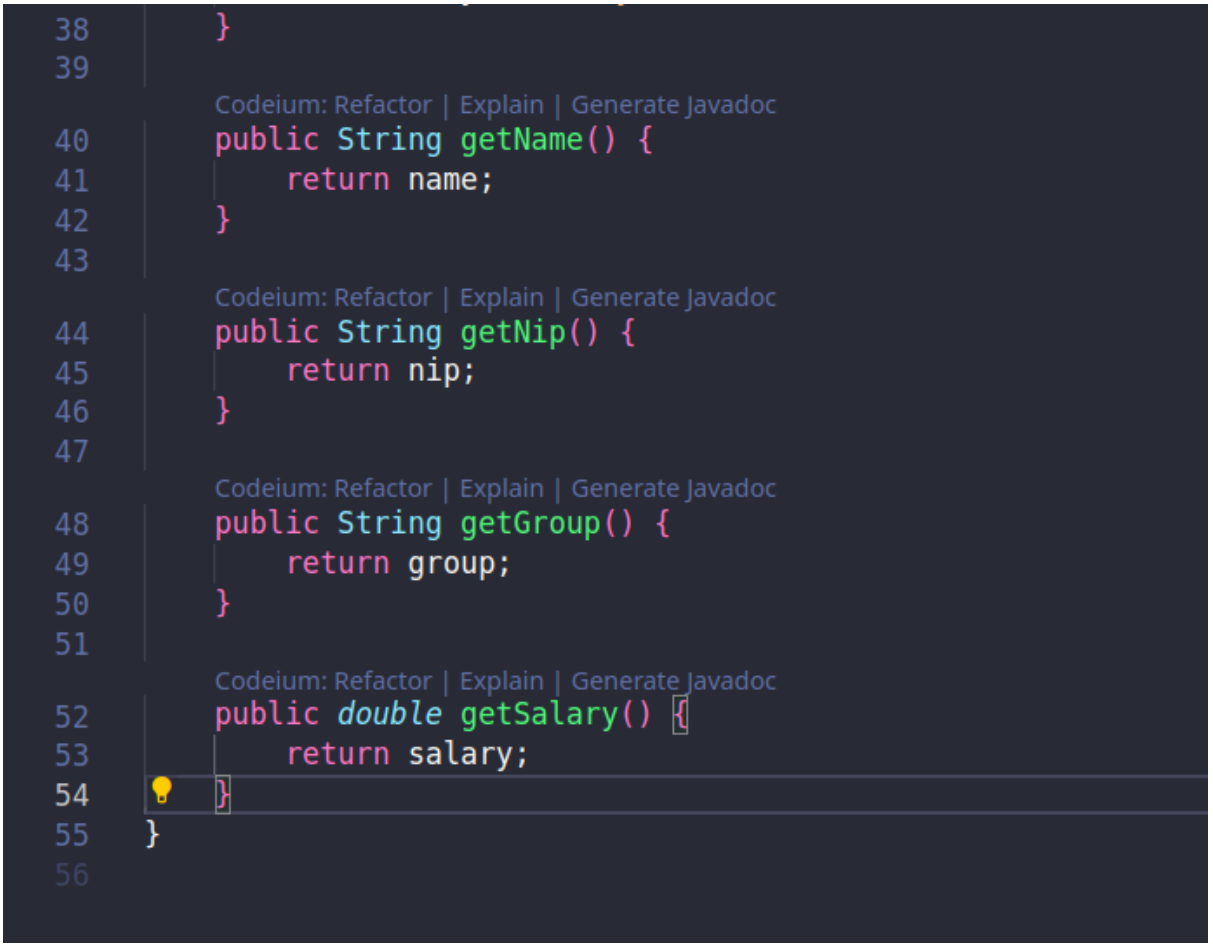


Code:

Employee.java | Staff.java | Manager.java | Main.java

src > main > java > com > azhar > exp > Employee.java > Employee > getSalary()

```java
package com.azhar.exp;

public class Employee {
    private String name, nip, group;
    private double salary;

    Codeium: Refactor | Explain | Generate Javadoc
    public void setName(String name) {
        this.name = name;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void setNip(String nip) {
        this.nip = nip;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void setGroup(String group) {
        this.group = group;
        switch(group.charAt(0)) {
            case '1':
                this.salary = 5_000_000;
                break;
            case '2':
                this.salary = 3_000_000;
                break;
            case '3':
                this.salary = 2_000_000;
                break;
            case '4':
                this.salary = 1_000_000;
                break;
            case '5':
                this.salary = 750_000;
                break;
        }
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void setSalary(double salary) {
        this.salary = salary;
```


}
Codeium: Refactor | Explain | Generate Javadoc
public String getName() {
return name;
}
Codeium: Refactor | Explain | Generate Javadoc
public String getNip() {
return nip;
}
Codeium: Refactor | Explain | Generate Javadoc
public String getGroup() {
return group;
}
Codeium: Refactor | Explain | Generate Javadoc
public double getSalary() {
return salary;
}
}

Employee.java | Staff.java | Manager.java | Main.java

src > main > java > com > azhar > exp > Staff.java > Staff > print()

```java
package com.azhar.exp;

public class Staff extends Employee {
    private int overTime;
    private double overTimeSalary;

    Codeium: Refactor | Explain | Generate Javadoc
    public void setOverTime(int overTime) {
        this.overTime = overTime;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void setOverTimeSalary(double overTimeSalary) {
        this.overTimeSalary = overTimeSalary;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public int getOverTime() {
        return overTime;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public double getOverTimeSalary() {
        return overTimeSalary;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public double getSalary(int overTime, double overTimeSalary) {
        return super.getSalary() + (overTime * overTimeSalary);
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public double getSalary() {
        return super.getSalary() + (overTime * overTimeSalary);
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void print() {
        System.out.println("Name : " + this.getName());
        System.out.println("NIP : " + this.getNip());
        System.out.println("Group : " + this.getGroup());
        System.out.println("Amount of over time : " + this.getOverTime());
        System.out.printf("Over time salary : %.0f\n", this.getOverTimeSalary());
        System.out.printf("Salary : %.0f\\n", this.getSalary());
    }
}
```

Employee.java | Staff.java | Manager.java | Main.java

src > main > java > com > azhar > exp > Manager.java > Manager > print()

```java
package com.azhar.exp;

public class Manager extends Employee {
    private double allowance;
    private String section;
    private Staff st[];

    Codeium: Refactor | Explain | Generate Javadoc
    public void setAllowance(double allowance) {
        this.allowance = allowance;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void setSection(String section) {
        this.section = section;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void setStaff(Staff[] st) {
        this.st = st;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public double getAllowance() {
        return allowance;

    }

    Codeium: Refactor | Explain | Generate Javadoc
    public String getSection() {
        return section;
    }

    Codeium: Refactor | Explain | Generate Javadoc
    public void viewStaff() {
        System.out.println("================");
        for (int i = 0; i < st.length; i++) {
            st[i].print();
        }
        System.out.println("================");
    }

```


Codeium: Refactor | Explain | Generate Javadoc
public void print() {
System.out.println("Manager : " + this.getSection());
System.out.println("Name : " + this.getName());
System.out.println("NIP : " + this.getNip());
System.out.println("Group : " + this.getGroup());
System.out.println("Section : " + this.getSection());
System.out.println("Allowance : " + this.getAllowance());
System.out.printf("Salary : %.0f\n", this.getSalary());
}
Codeium: Refactor | Explain | Generate Javadoc
public double getSalary() {
return super.getSalary() + allowance;
}
}

Employee.java | Staff.java | Manager.java | Main.java

src > main > java > com > azhar > exp > Main.java > Main > main(String[])

```java
package com.azhar.exp;

public class Main {
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) {
        System.out.println("Manager & Staff Testing Program");
        Manager[] managers = new Manager[2];
        Staff [] staffs1 = new Staff[2];
        Staff [] staffs2 = new Staff[3];

        managers[0] = new Manager();
        managers[0].setName(name:"John");
        managers[0].setNip(nip:"123");
        managers[0].setGroup(group:"1");
        managers[0].setAllowance(allowance:5_000_000);
        managers[0].setSection(section:"Administration");

        managers[1] = new Manager();
        managers[1].setName(name:"Jane");
        managers[1].setNip(nip:"456");
        managers[1].setGroup(group:"1");
        managers[1].setAllowance(allowance:2_500_000);
        managers[1].setSection(section:"Marketing");

        staffs1[0] = new Staff();
        staffs1[0].setName(name:"Sarah");
        staffs1[0].setNip(nip:"789");
        staffs1[0].setGroup(group:"2");
        staffs1[0].setOverTime(overTime:10);
        staffs1[0].setOverTimeSalary(overTimeSalary:10_000);

        staffs1[1] = new Staff();
        staffs1[1].setName(name:"Alex");
        staffs1[1].setNip(nip:"321");
        staffs1[1].setGroup(group:"2");
        staffs1[1].setOverTime(overTime:10);
        staffs1[1].setOverTimeSalary(overTimeSalary:55_000);

        staffs2[0] = new Staff();
        staffs2[0].setName(name:"Alex");
        staffs2[0].setNip(nip:"321");
```

```
41          staffs2[0].setGroup(group:"3");
42          staffs2[0].setOverTime(overTime:15);
43          staffs2[0].setOverTimeSalary(overTimeSalary:5_500);
44
45          staffs2[1] = new Staff();
46          staffs2[1].setName(name:"Alex");
47          staffs2[1].setNip(nip:"321");
48          staffs2[1].setGroup(group:"4");
49          staffs2[1].setOverTime(overTime:5);
50          staffs2[1].setOverTimeSalary(overTimeSalary:100_000);
51
52          staffs2[2] = new Staff();
53          staffs2[2].setName(name:"Alex");
54          staffs2[2].setNip(nip:"321");
55          staffs2[2].setGroup(group:"3");
56          staffs2[2].setOverTime(overTime:6);
57          staffs2[2].setOverTimeSalary(overTimeSalary:20_000);
58
59          managers[1].setStaff(staffs2);
60
61          managers[0].print();
62          managers[1].print();
63      }
64  }
65
```

Result:

```
→ zharsuke@box  ~/Documents/College/Semester_3/oop/meet-9/coding git:(master) ✗
ExceptionMessages -cp /home/zharsuke/Documents/College/Semester_3/oop/meet-9/cod
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Manager & Staff Testing Program
Manager : Administration
Name : John
NIP : 123
Group : 1
Section : Administration
Allowance : 5000000.0
Salary : 10000000
Manager : Marketing
Name : Jane
NIP : 456
Group : 1
Section : Marketing
Allowance : 2500000.0
Salary : 7500000
→ zharsuke@box  ~/Documents/College/Semester_3/oop/meet-9/coding git:(master) ✗
```

Exercise

Code:

MyMultiplication.java

src > main > java > com > azhar > exercise > MyMultiplication.java > MyMultiplication > main

```java
package com.azhar.exercise;

public class MyMultiplication {
    Codeium: Refactor | Explain | Generate Javadoc
    void multiply(int a, int b) {
        System.out.println(a * b);
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void multiply(int a, int b, int c) {
        System.out.println(a * b * c);
    }

    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) {
        MyMultiplication myMultiplication = new MyMultiplication();
        myMultiplication.multiply(a:25, b:43);
        myMultiplication.multiply(a:34, b:23, c:56);
    }
}
```

Result:

```
→ zharsuke@box  ~/Documents/College/Semester_3/oop/meet-9/coding git:(master) ✗
ExceptionMessages -cp /home/zharsuke/Documents/College/Semester_3/oop/meet-9/cod
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
1075
43792
→ zharsuke@box  ~/Documents/College/Semester_3/oop/meet-9/coding git:(master) ✗
```

1. Where is the overloading located in the source coding above?

- The overloading are located at second multiply method that has three parameters that is int a, int b, int c.

2. If there is overloading, how many different parameters are there?

- The different between both is the first method has 2 parameters that is a, b. And the second method has 3 parameters that is a, b, c.

Code:

```java
MyMultiplication.java
src > main > java > com > azhar > exercise > MyMultiplication.java > MyMultiplication > main(String[])
package com.azhar.exercise;

public class MyMultiplication {
    Codeium: Refactor | Explain | Generate Javadoc
    void multiply(int a, int b) {
        System.out.println(a * b);
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void multiply(double a, double b) {
        System.out.println(a * b);
    }

    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) {
        MyMultiplication myMultiplication = new MyMultiplication();
        myMultiplication.multiply(a:25, b:43);
        myMultiplication.multiply(a:34.56, b:23.73);
    }
}
```

Result:

```
→ zharsuke@box  ~/Documents/College/Semester_3/oop/meet-9/coding git:(master) ✗
ExceptionMessages -cp /home/zharsuke/Documents/College/Semester_3/oop/meet-9/codi
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
1075
819.072
→ zharsuke@box  ~/Documents/College/Semester_3/oop/meet-9/coding git:(master) ✗
```

3. From the source coding above, where is the overloading located?

- The overloading are located at second multiply method that has 2 parameters that is double a, and double b.

4. If there is overloading, how many different types of parameters are there?

- Both method has 2 parameters which is same, but there are different in data type parameter. The first method has variable a and b with int data type, then the second method has variable a and be with double data type.

Code:

MyMultiplication.java | Fish.java | Piranha.java | Main.java

src > main > java > com > azhar > exercise > Fish.java > Fish > swim()

```java
package com.azhar.exercise;

public class Fish {
    Codeium: Refactor | Explain | Generate Javadoc
    public void swim() {
        System.out.println("Fish can swim");
    }
}
```

MyMultiplication.java | Fish.java | Piranha.java | Main.java

src > main > java > com > azhar > exercise > Piranha.java > Piranha > swim()

```java
package com.azhar.exercise;

public class Piranha extends Fish {
    Codeium: Refactor | Explain | Generate Javadoc
    public void swim() {
        System.out.println("Piranha can swim");
    }
}
```

MyMultiplication.java | Fish.java | Piranha.java | Main.java

src > main > java > com > azhar > exercise > Main.java > Main > main(String[])

```java
package com.azhar.exercise;

public class Main {
    Run | Debug | Codeium: Refactor | Explain | Generate Javadoc
    public static void main(String[] args) {
        Fish a = new Fish();
        Fish b = new Piranha();
        a.swim();
        b.swim();
    }
}
```

Result:

```
→ zharsuke@box  ~/Documents/College/Semester_3/oop/meet-9/coding git:(master) ✗
ExceptionMessages -cp /home/zharsuke/Documents/College/Semester_3/oop/meet-9/cod
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
Fish can swim
Piranha can swim
→ zharsuke@box  ~/Documents/College/Semester_3/oop/meet-9/coding git:(master) ✗
```

5. Where is the overriding located in the source coding above?

- The overriding are located at swim method inside Piranha class.

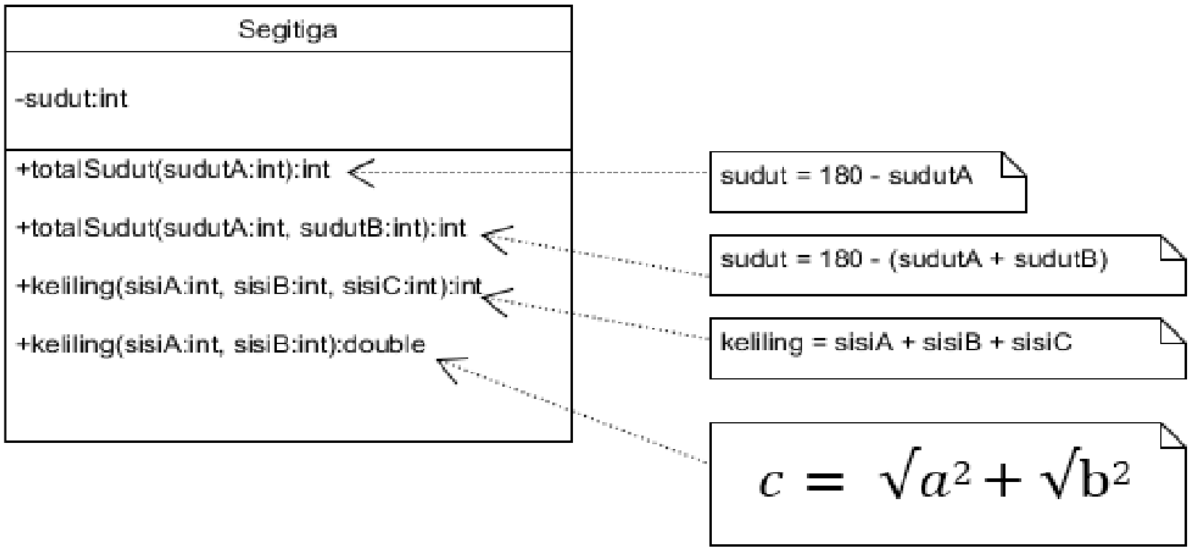
6. Explain if the source coding above if there is overriding?

- There is Fish class that has one method that is swim. Fish are parent class. Then, there is child class that is Piranha that has one method also that is swim, which is it overriding method from parent class. Both method has same name, but has different in their implementations.

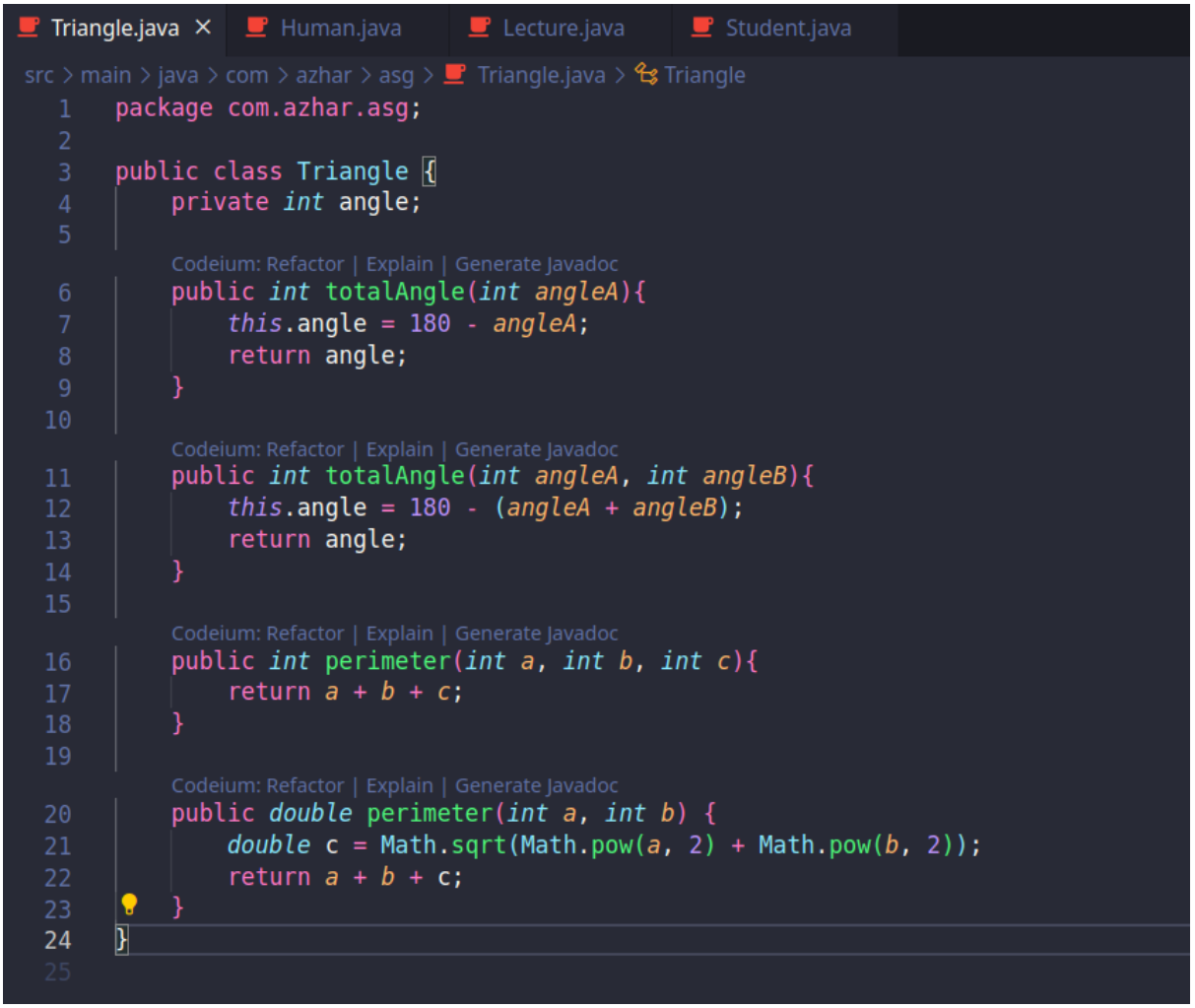
Assignment

1. Overloading

Implement the concept of overloading in the class diagram below:

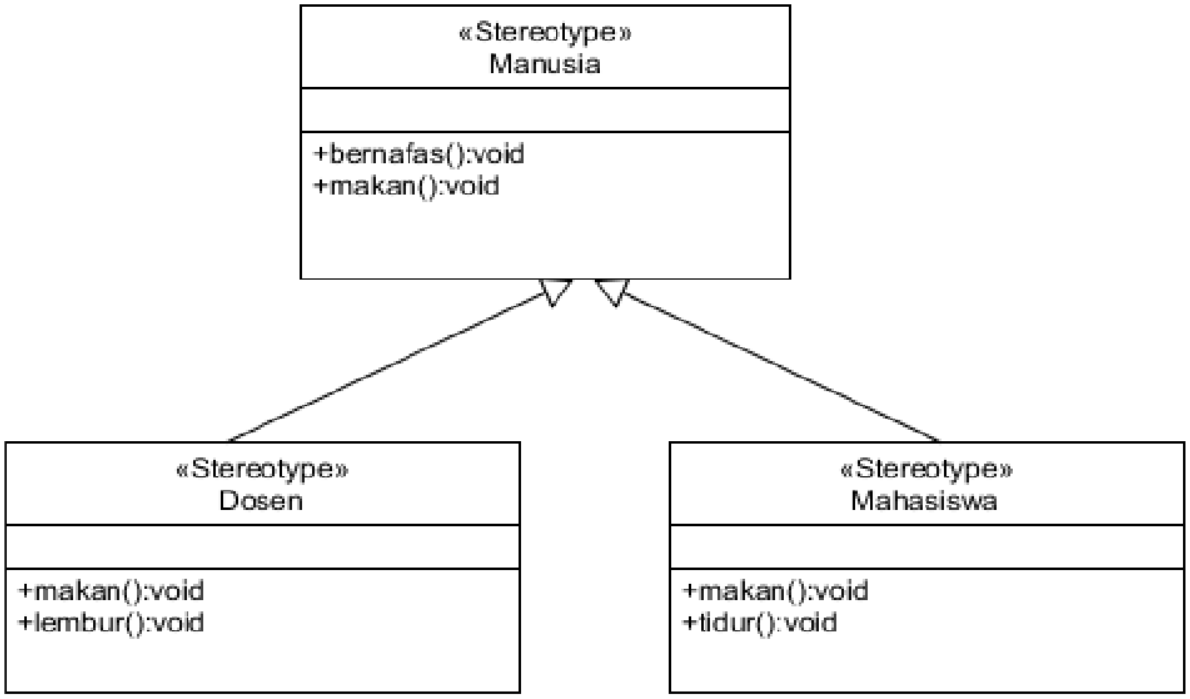


Code:



2. Overriding

Implement the class diagram below using dynamic technique method dispatch technique:


«Stereotype»
Manusia
+bernafas():void
+makan():void
«Stereotype»
Dosen
+makan():void
+lembur():void
«Stereotype»
Mahasiswa
+makan():void
+tidur():void

Code:

Triangle.java | Human.java × | Lecture.java | Student.java

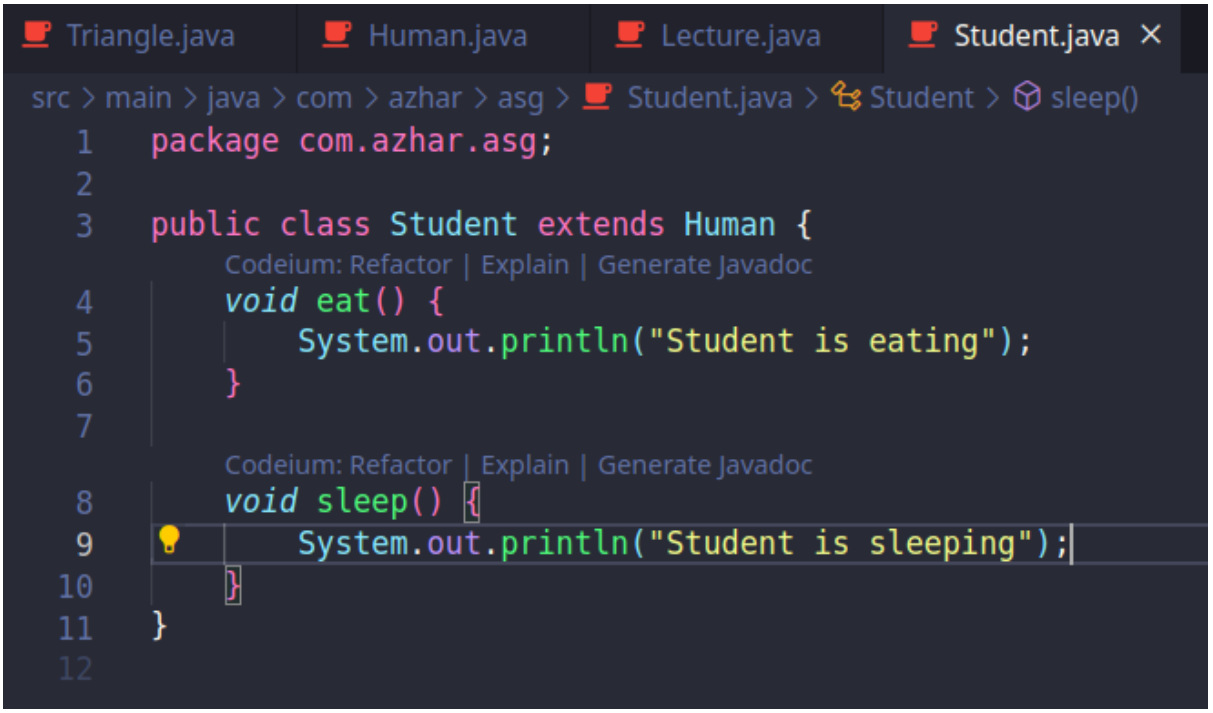
src > main > java > com > azhar > asg > Human.java > Human > eat()

```java
package com.azhar.asg;

public class Human {
    Codeium: Refactor | Explain | Generate Javadoc
    void breath() {
        System.out.println("Human is breathing");
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void eat() {
        System.out.println("Human is eating");
    }
}
```

Triangle.java | Human.java | Lecture.java × | Student.java

src > main > java > com > azhar > asg > Lecture.java > Lecture > overTime()

```java
package com.azhar.asg;

public class Lecture extends Human {
    Codeium: Refactor | Explain | Generate Javadoc
    void eat() {
        System.out.println("Lecture is eating");
    }

    Codeium: Refactor | Explain | Generate Javadoc
    void overTime() {
        System.out.println("Lecture is over time");
    }
}
```


Triangle.java
Human.java
Lecture.java
Student.java
src > main > java > com > azhar > asg > Student.java > Student > sleep()
1 package com.azhar.asg;
2
3 public class Student extends Human {
Codeium: Refactor | Explain | Generate Javadoc
4 void eat() {
5 System.out.println("Student is eating");
6 }
7
Codeium: Refactor | Explain | Generate Javadoc
8 void sleep() {
9 System.out.println("Student is sleeping");
10 }
11 }
12