**JOBSHEET 7 OVERLOADING AND OVERRIDING**

1. **Competence**

After taking this subject, students are able to:

1. Understand the concepts of overloading and overriding,
2. Understand the difference between overloading and overriding,
3. Accuracy in identifying overriding and overloading methods
4. Accuracy in practicing instructions on the jobsheet
5. Implement overloading and overriding methods.
6. **Introduction**
   1. **Overloading**

is to rewrite a method with the same name on a class. The goal is to facilitate the use/invocation of methods with similar functionality. The Overloading method declaration rules are as follows:

* + - The method name must be the same.
    - The list of parameters should be different.
    - The return type can be the same, or it can be different.

There are several lists of parameters on overloading can be seen as follows:

* + - The difference in the list of parameters does not only occur in the difference in the number of parameters, but also in the order of the parameters.
    - For example, the following two parameters:
      * Function\_member (int x, string n)
      * Function\_member (String n, int x)
    - The two parameters are also considered different in the list of parameters.
    - The parameter list is not related to the naming of the variables present in the parameter.
    - For example, the following 2 list of parameters:
      * function\_member(int x)
      * function\_member(int y)
    - The two lists of parameters above are considered the same because the only difference is the naming of the variable parameters.

Overloading can also occur between the parent class and its subclass if it meets all three overload conditions. There are several overloading rules, namely:

* + - Primitive widening conversions take precedence over overloading over boxing and var args.
    - We can't do the widening process from one wrapper type to another (changing the Integer to Long).
    - We can't do the widening process followed by boxing (from int to Long)
    - We can do boxing followed by widening (int can be an Object via an Integer)
    - We can combine var args with either widening or boxing
  1. **Overriding**

is a Subclass that seeks to modify behaviors inherited from super classes. The goal is that the subclass can have more specific behavior so that it can be done by redeclaring the parent class's method in the subclass.

The method declaration in the subclass must be the same as the one in the super class. Similarities on:

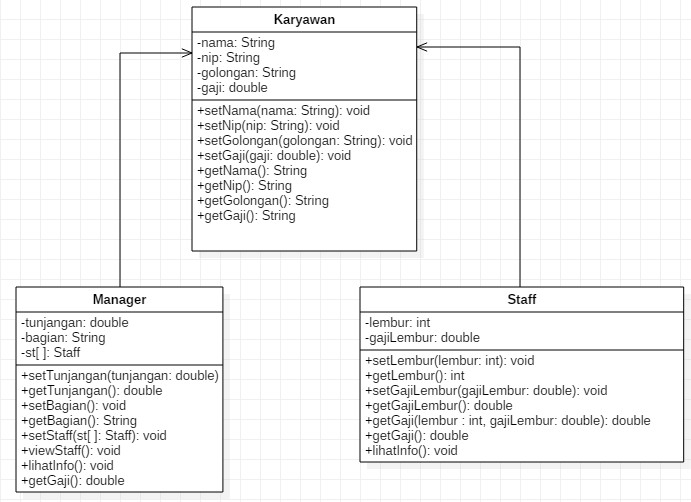
* + - Name
    - Return type (for return type: class A or is a subclass of class A)
    - List of parameters (number, type and order)

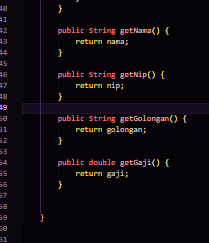
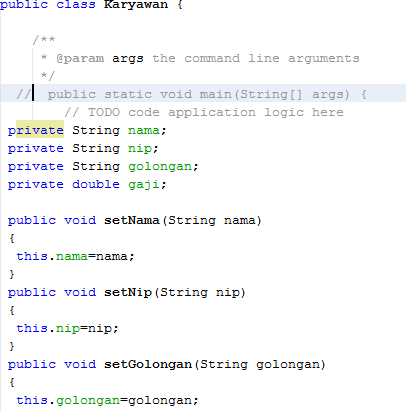
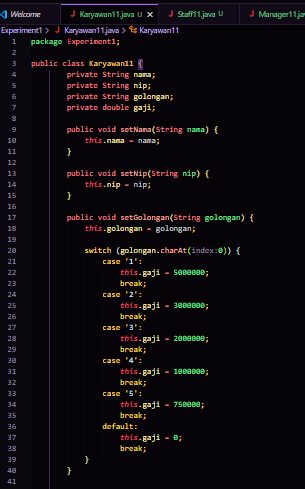
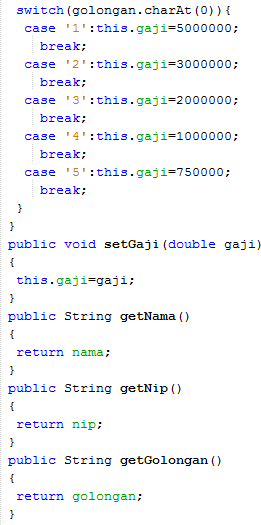
So that the method in the parent class is called the overridden method and the method in the subclass is called the overriding method. There are several method rules in overriding:

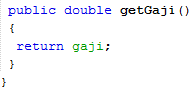
* + - The access mode of the overriding method must be the same or broader than the overridden method.
    - A subclass can only override a superclass method once, there must not be more than one method in the exact same class.
    - The overriding method must not throw checked exceptions that are not declared by the overridden method.

1. **Practicum**
   1. **Experiment 1**

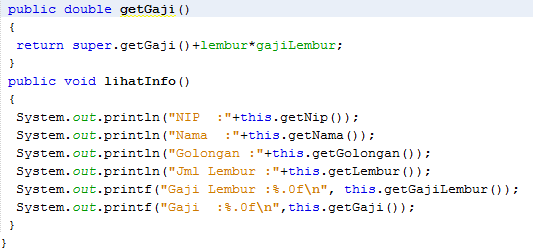
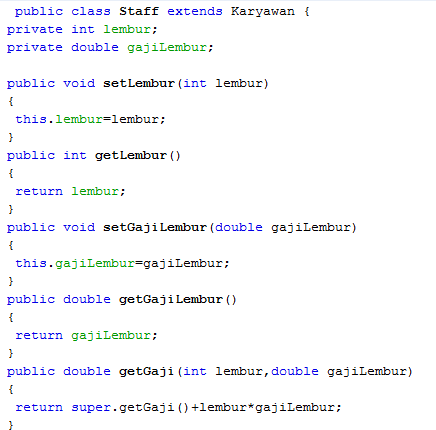
For the following example case, there are three classes, namely Karyawan, Manager, and Staff. Employee Class is a superclass of Manager and Staff where the Manager and Staff subclasses have different methods for calculating salaries.



* 1. **Karyawan**
  2. ********



* 1. **Staff**

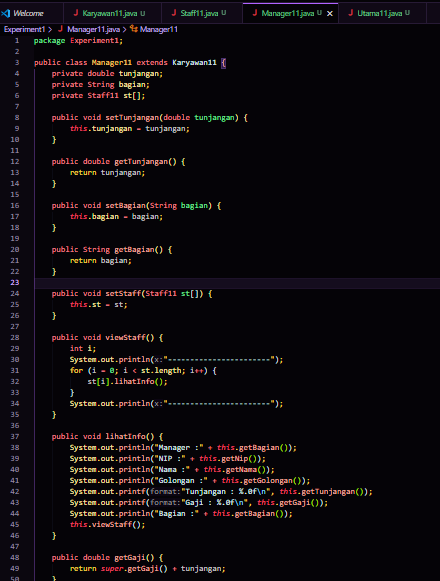


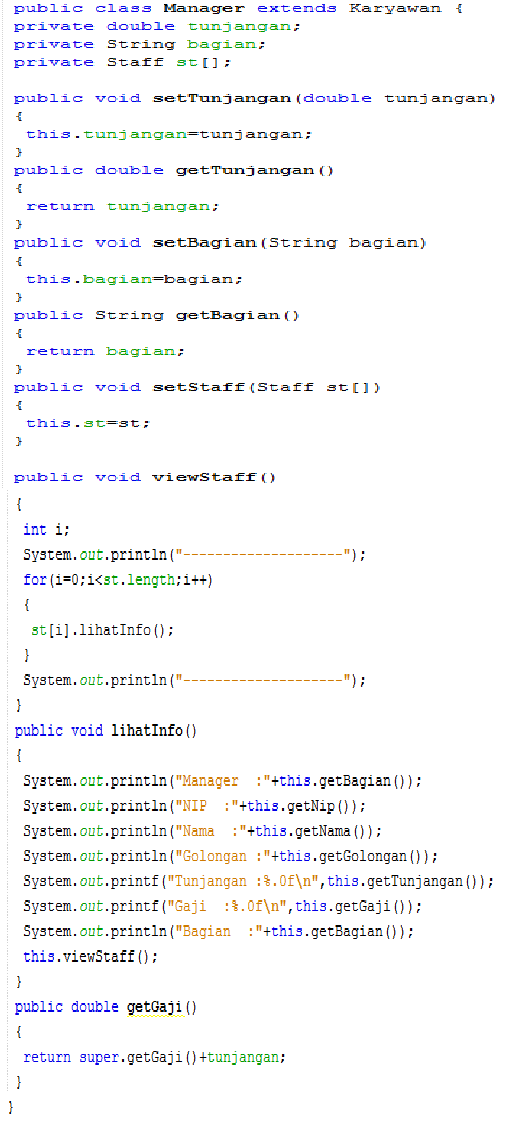
Overriding

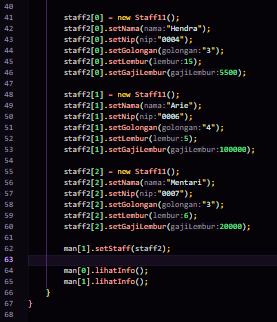
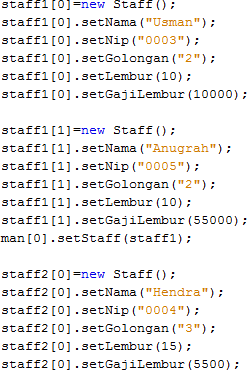
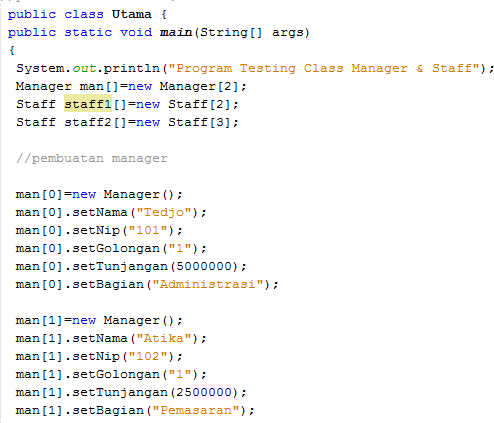
Overloading

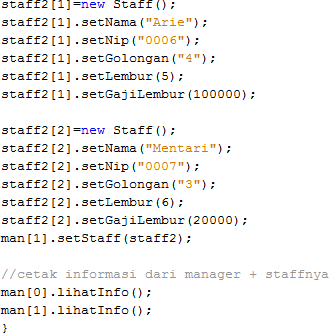
****

* 1. **Manager**

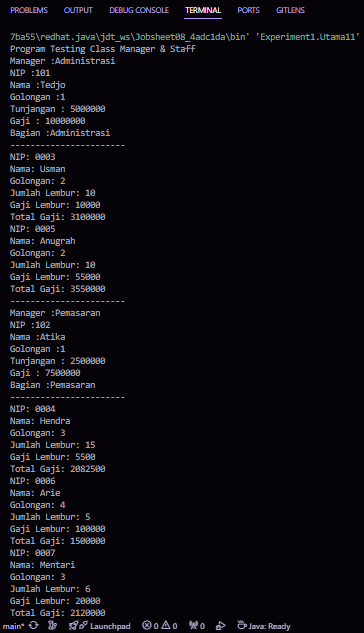




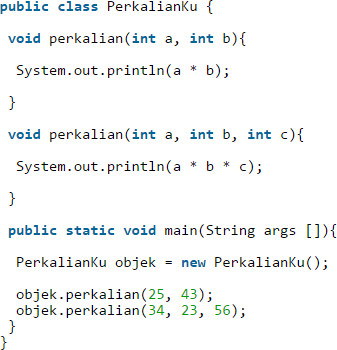
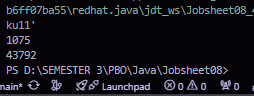
* 1. **Main**
  2. ****

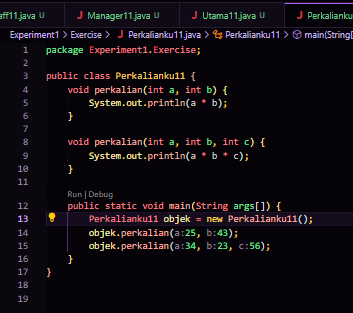


Hasil Run :



1. **Exercise**



****

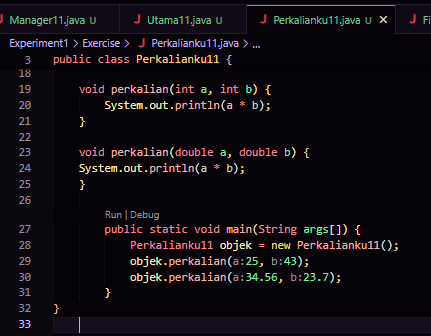
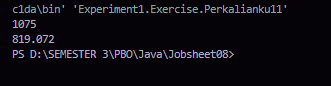
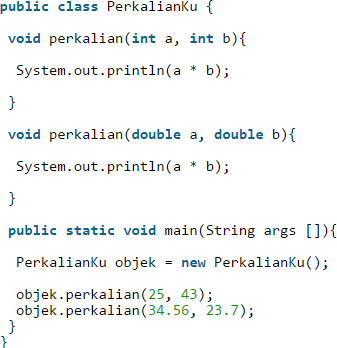
* 1. From the source coding above, where is the overloading?

**Overloading happens in the multiplication method**

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

**Overloading can be achieved in two ways:**

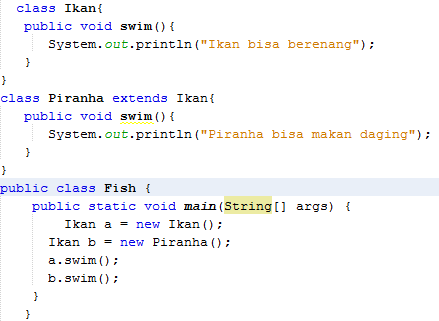
* **by altering the number of parameters, or**
* **by changing the parameter types, even if the number remains the same.**

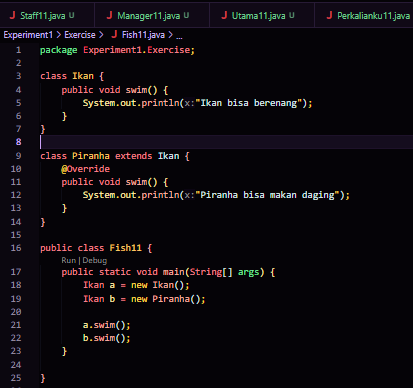
****

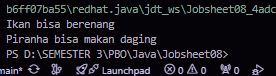
* 1. From the source coding above, where is the overloading?

"**Overloading takes place in the multiplication method."**

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

**There are two different parameter types yaitu int and double**.





* 1. From the source coding above, where is the overriding?

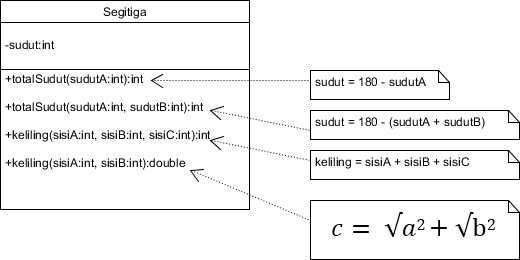
**The Piranha class contains the overriding, as it alters the swim() method from the Fish class to output a different message."**

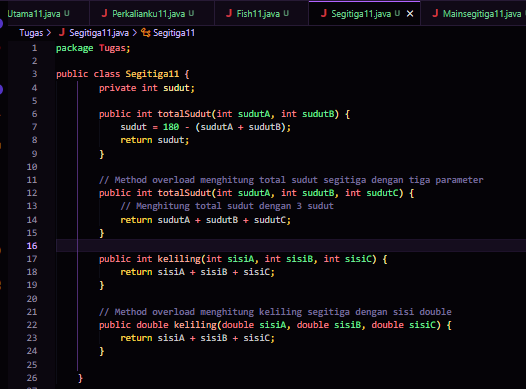
* 1. Describe when sourcoding above if there is overriding?

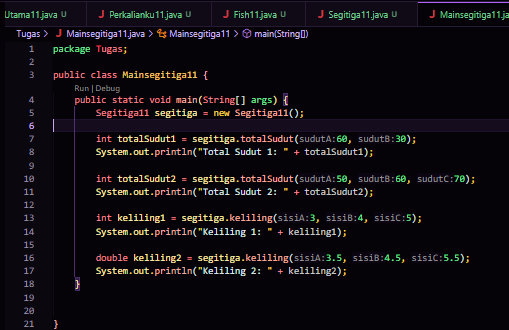
**Fish is a superclass with a swim() method, and Piranha is a subclass that overrides it. Overriding occurs when a subclass provides a new implementation for an inherited method, allowing Piranha to change the swimming behavior of Fish."**

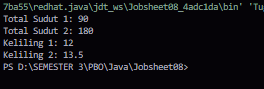
1. **Tasks**
   1. **Overloading**

Implement the overloading concept in the diagram class below:







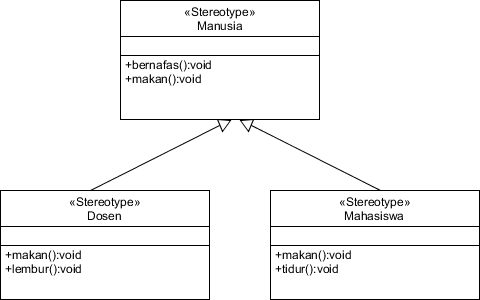


**Attributes: The Triangle class has angle attributes.**

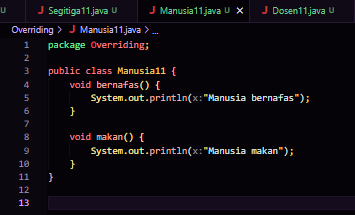
**Methods:**

* **totalAngles(int angleA, int angleB): Calculates the third angle of a triangle.**
* **totalAngles(int angleA, int angleB, int angleC): Calculates the total angle of a triangle with three angles (overload).**
* **perimeter(int sideA, int sideB, int sideC): Calculates the circumference of a triangle with rounded sides.**
* **perimeter(double sideA, double sideB, double sideC): Calculates the circumference of a triangle with double sides (overload).**
  1. **Overriding**

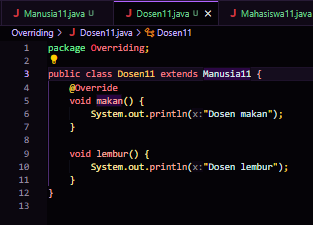
Implement the diagram class below using the dynamic method dispatch technique:



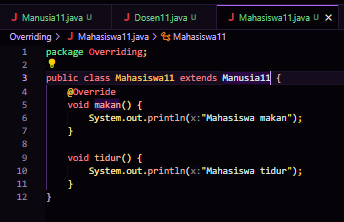
1. Class Human



1. Class lecturer



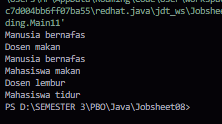
1. Class student



1. Main



1. Output



**Human : Base class that defines the methods breathe() and eat().**

**Lecturer : Derived class that overrides the eat() method and adds the overtime() method.**

**Student : Derived class that overrides the eat() method and adds the sleep() method.**

**Main : Main class that tests all methods using dynamic method dispatch.**

**This code demonstrates the use of inheritance and method overriding in Java, as well as the application of dynamic method dispatch and casting to call specific methods from subclasses**.