

Subscribe to DeepL Pro to edit this document.  
Visit [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document) for more information.

# JOBSHEET W06

**INHERITANCE**

# COMPETENCY

* 1. Understand the basic concepts of inheritance.
  2. Able to create a subclass of a certain superclass.
  3. Able to implement the concept of hierarchical inheritance
  4. Able to create objects from a subclass and access attributes and methods either owned or derived from its superclass.

# INTRODUCTION

**Inheritance** in object oriented programming is the concept of **inheritance** from a more general class to a more specific class. The class that inherits is called the **base** class/super **class/parent** class, while the class that is inherited is called the **derived** class/sub class/child **class**. Each **subclass** will "inherit" attributes and methods from the **superclass** that are either *public* or *protected*. The benefit of inheritance is *reusability* or reuse of lines of code.

In the Java programming language, inheritance declaration is done by adding the **extends** keyword after the class name declaration, followed by the parent class name. The extends keyword tells the Java compiler that we want to **extend** the class. Here is an example of an inheritance declaration.

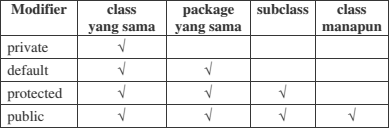
public class B extends A {

...

}

The example above tells the Java compiler that class B **extends** class A. That is, class B is a subclass of class A by extension. This extension will be done by adding special attributes and methods that are only owned by class B.

A parent class can restrict the attributes and methods that will be inherited by its subclasses. This restriction is done by determining the access level modifier. In java, attribute and method access level modifiers are summarized in the following table:



The attributes and methods that will be inherited from parent class to child class are attributes and methods with protected or public modifiers.

**The this** keyword is used to refer to the current object/class. While the **super** keyword is used to refer to the parent object/class. The writing format is as follows:

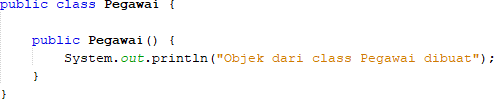
### super.<attribute name>

Accessing the parent attribute

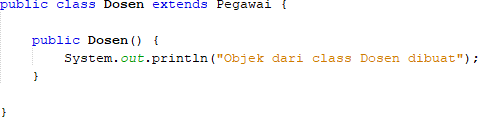
### super.<method name>()

Calling the parent method

1. **TRY 1 (extends)**
   1. **STAGES OF THE EXPERIMENT**
      1. Create a parent class with the name Employee. Then create a parameterless constructor with the following lines of code:



* + 1. Create a subclass of the Employee class with the name Lecturer, then also create a constructor without parameters with the following lines of code:



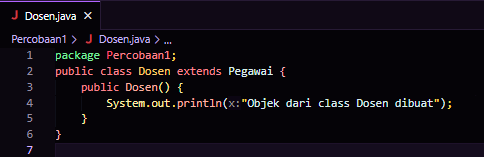
* + 1. Create a main class, for example InheritanceDemo.java, instantiate a new object named dosen1 from class Dosen as follows:



* + 1. Run the program and observe the results.
  1. **QUESTION**
     1. In experiment 1 above, determine the child class and parent class!

**The Lecturer and Employee classes are two separate classes that do not show any direct inheritance relationship.**

* + 1. What keyword makes the child class and parent class related?



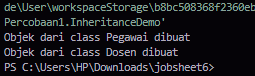
**extends keyword is required. If Lecturer is a child class of Employee**

* + 1. Based on the results displayed by the program, how many constructors were executed? Which class constructor is executed first?

**because adding inheritance using the extends keyword then there will be two constructors executed:**

**The constructor of the Employee class will be executed first (because this class is the parent class).**

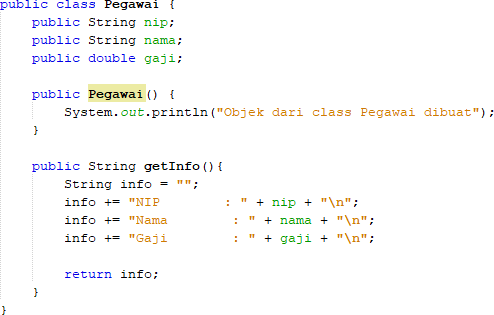
**After that, the constructor of the Dosen class (child class) will be executed**

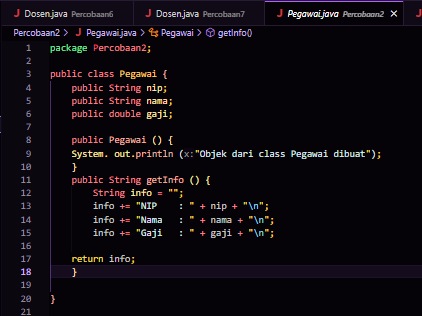
****

# TRY 2 (Inheritance)

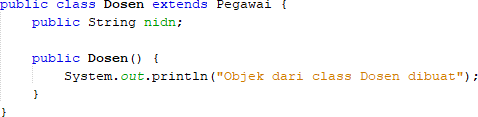
## STAGES OF THE EXPERIMENT

* + - * 1. Add the nip, name, and salary attributes and the getInfo() method to the Employee class.

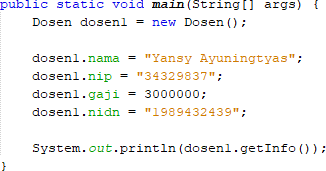


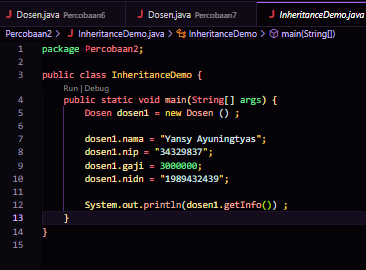


* + - * 1. Also add the NIDN attribute to the Lecturer class



* + - * 1. In the InheritanceDemo.java class, write the following lines of code:





* + - * 1. Run the program and observe the results

## QUESTION

* + - * 1. In experiment 2 above, did the program run successfully or did an error occur?

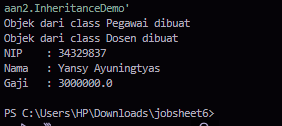
**the program will run successfully and no errors will occur.**

* + - * 1. If the program runs successfully, why is there no error in the assignment/filling of the values of the nip, salary, and NIDN attributes on the lecturer1 object when there is no declaration of these three attributes in the Lecturer class?

**The nip, name, and salary attributes can be accessed by the dosen1 object because the modifier is public, and the nidn attribute is declared directly in the Dosen class.**

* + - * 1. If the program runs successfully, why is there no error on the getInfo() method call by object dosen1 when there is no getInfo() method declaration in class Dosen?

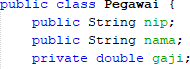
**There is no error when calling the getInfo() method by the lecturer1 object because getInfo() can be called by the lecturer1 object because the Lecturer class inherits this method from the Employee class, and the method is public.**

****

# TRIAL 3 (Access rights)

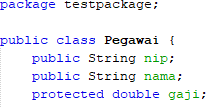
## STAGES OF THE EXPERIMENT

* + 1. Modify the access level modifier on the salary attribute to private in class Employee.java



* + 1. Run the program and observe the results.
    2. Change the access level modifier of the salary attribute to protected then move the Employee class.

to a new package, for example "testpackage".



* + 1. Import the Employee class from the testpackage on the Lecturer class.



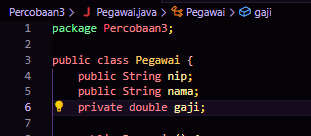
* + 1. Access the salary attribute in the Lecturer class by trying to print the salary attribute in the Lecturer constructor



* + 1. Change the access level modifier back to public and return the Employee class to its original package.

## QUESTION

* + 1. In step 1 above, an error occurs because the lecturer1 object cannot access the salary attribute. Even though salary is an Employee attribute which is the parent class of Lecturer. Why does this happen?



**because the salary attribute in the Employee class is declared with the private modifier. private can only be accessed in the class where the attribute is declared**

* + 1. In step 5, after the Employee class moves to a different package, the Lecturer class can still access the salary attribute. Why?

**The Lecturer class can still access the salary attribute after the Employee class is moved to a different package because the salary is changed to protected. The protected modifier allows access by the class itself, subclasses, and other classes in the same package. Thus, attribute inheritance still works even though it is in a different package.**

* + 1. Based on the experiment, how to determine the attributes and methods that will be inherited by the parent class to the child class?

**The attributes and methods inherited by a child class are determined by its access modifiers:**

**Private is not inherited.**

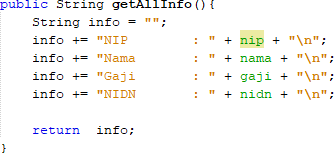
**Protected allows access in subclasses, even in different packages.**

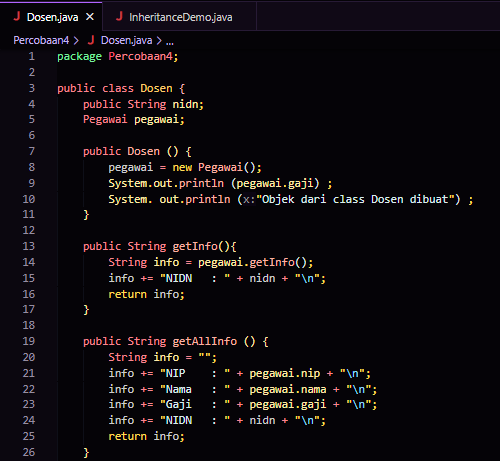
**Public is inherited and can be accessed from anywhere.**

# TRY 4 (Super - attributes)

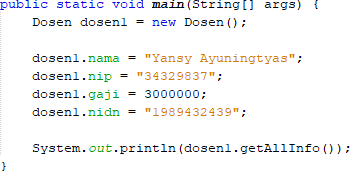
## STAGES OF THE EXPERIMENT

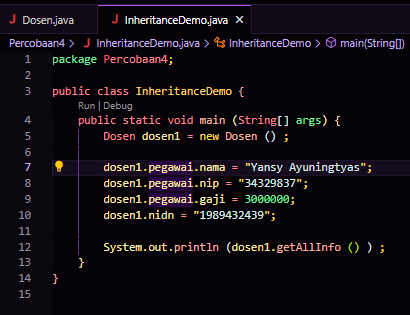
* + 1. But the getAllInfo() method on the Lecturer class



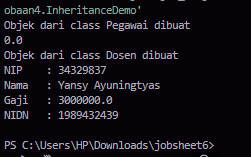


* + 1. Call the getAllInfo() method by the lecturer1 object in class InheritanceDemo.java

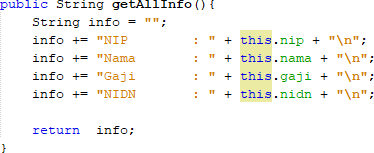




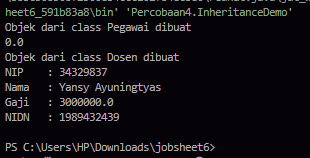
* + 1. Run the program and observe the results



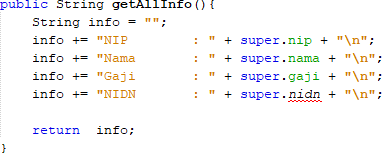
* + 1. Modify the getAllInfo() method on the Lecturer class

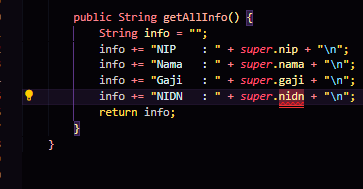


* + 1. Run the program and compare the results with step 2.

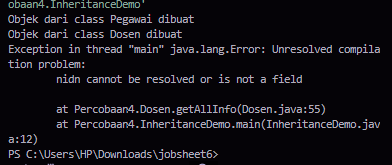


* + 1. Modify the getAllInfo() method on the Lecturer class again

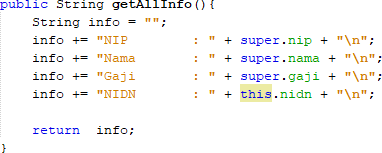


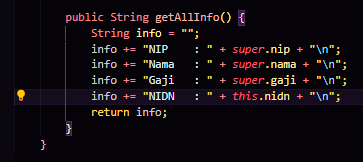


* + 1. Run the program and compare the results with the programs in No. 1 and No. 4.



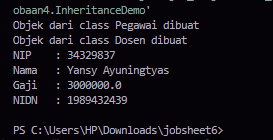
* + 1. Modify the getAllInfo() method on the Lecturer class again





* + 1. Run the program then compare the results with the programs in no 2 and no 4.

**programs 1 and 4 have the same results, all information such as NIP, Name, Salary, and NIDN are displayed correctly. While in program 8, initially there were no values for NIP, Name, Salary, and NIDN displayed.**



## QUESTION

* + 1. Are there any differences in the name, nip, and salary results displayed in programs 1, 4, and 8? Why?

**There is a Difference: The results displayed in programs 1 and 4 are the same, while program 8 experiences an error in displaying the initial value.**

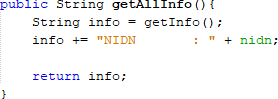
* + 1. Why does the error occur in program number 6?

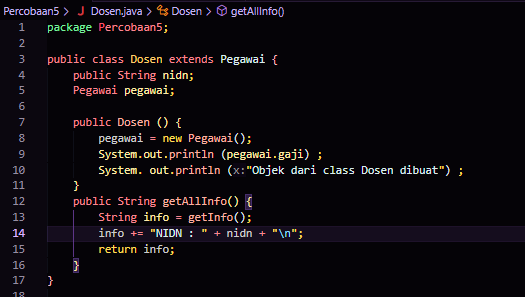
**Incorrect use of super and this**

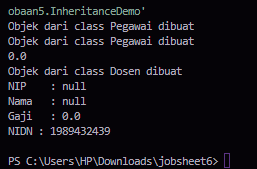
# TRIAL 5 (super & ov e r r id i n g )

## STAGES OF THE EXPERIMENT

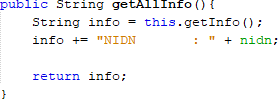
* + 1. Modify the getAllInfo() method again. Run the program and observe the results

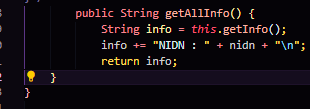


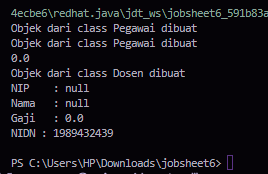




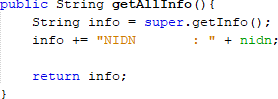
* + 1. Modify the getAllInfo() method again. Run the program and observe the results

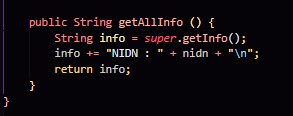


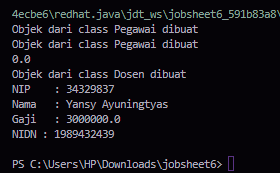




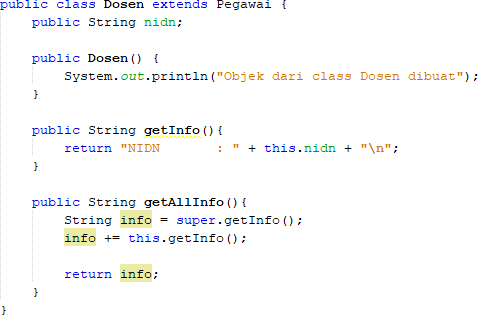
* + 1. Modify the getAllInfo() method again. Run the program and observe the results

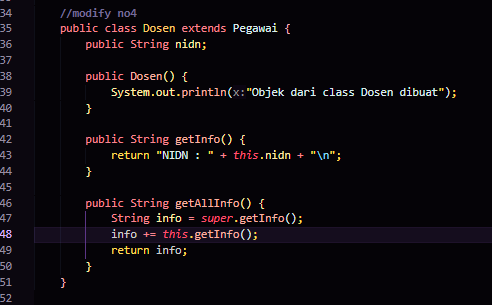


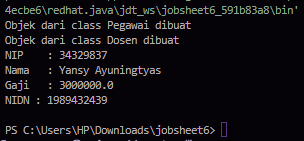




* + 1. Add the getInfo() method to the Lecturer class and modify the getAllInfo() method as follows







## QUESTION

* + 1. Is there any difference in the getInfo() method accessed in steps 1, 2, and 3?

**Step 1 and 2: Calling getInfo() from Lecturer without setting the attribute, so the result is the default value.**

**Step 3: Calling super.getInfo() but because the Lecturer attribute value is already set, the correct information is displayed.**

* + 1. Is there any difference between the super.getInfo() and this.getInfo() methods called in the getAllInfo() method in step 4? Explain!

**The main difference is where the methods are called from. super.getInfo() accesses a method from the parent class, while this.getInfo() accesses a method from the current class (Dosen), which means it returns more specific information.**

* + 1. In which method does overriding occur? Explain!

**Overriding: Is a process in which a subclass (in this case, Lecturer) provides a new implementation for a method that has been defined in the superclass (in this case, Employee).**

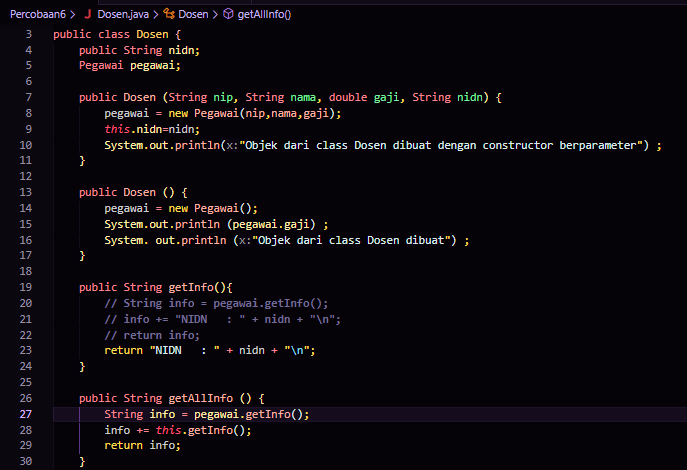
**Overriding occurs in the getInfo() method in the Dosen class.**

# TRIAL 6 (overloading)

## STAGES OF THE EXPERIMENT

* + 1. Add a new constructor for the Lecturer class as follows

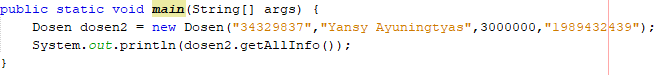


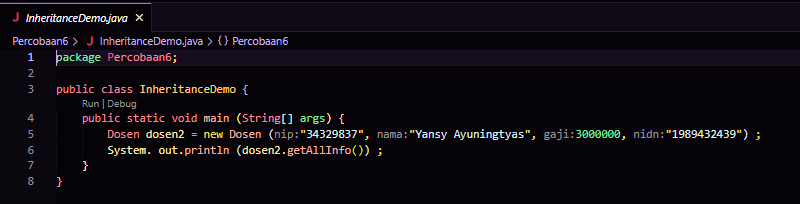


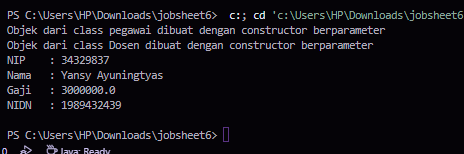
* + 1. Modify the InheritanceDemo class to instantiate a new object named dosen2 with a parameterized constructor. Run the program and observe the results.

**The parameterized constructor in the Dosen class is used to directly set the values of NIP, Name, Salary, and NIDN when the object is created.**

**Create a dosen2 object with a parameterized constructor and display complete lecturer information using the getAllInfo() method.**







## QUESTION

* + 1. What is the result of the nip, name, salary, and nidn values displayed in step 2? Why is this the case?

**The NIP, Name, Salary, and NIDN values will be displayed according to the parameters given when creating the dosen2 object. This is because the constructor with parameters directly initializes these values when the object is created.**

* + 1. Explain whether the parameterless constructor and the Lecturer class constructor created in step 1 have the same signature?

**The parameterless constructor and the constructor created in step 1 do not have the same signature. The signatures are different because the parameterless constructor does not accept arguments, while the constructor in step 1 accepts four parameters (nip, name, salary, and nidn).**

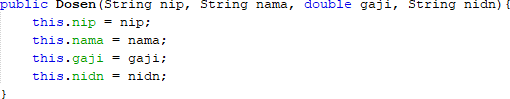
* + 1. What concept in OOP allows a class to have a constructor or method with the same name and different signature in one class?

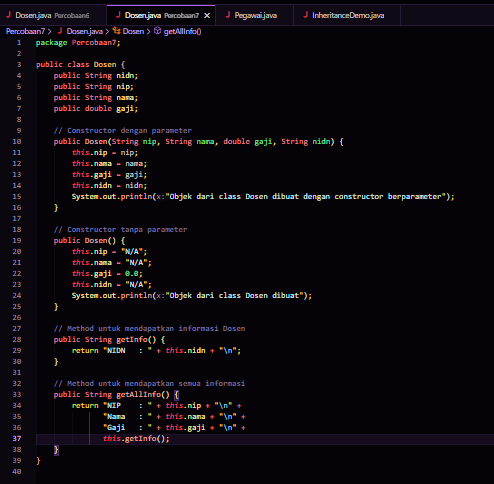
Overloading allows a class to have multiple methods or constructors with the same name, but with different signatures (number or type of parameters).

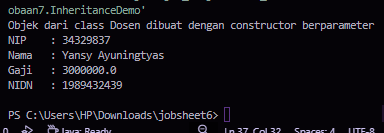
# TRY 7 (super - constructor)

## STAGES OF THE EXPERIMENT

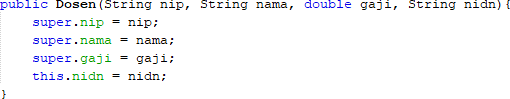
* + 1. Modify the constructor of the Lecturer class as follows. Run the program then observe the result.

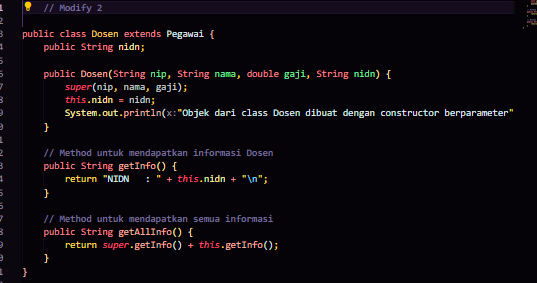


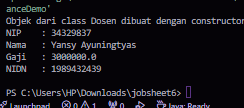




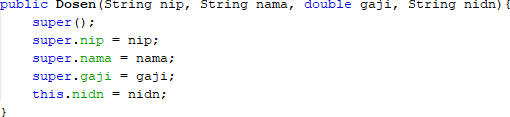
* + 1. Modify the constructor of the Lecturer class as follows. Run the program then observe the result.

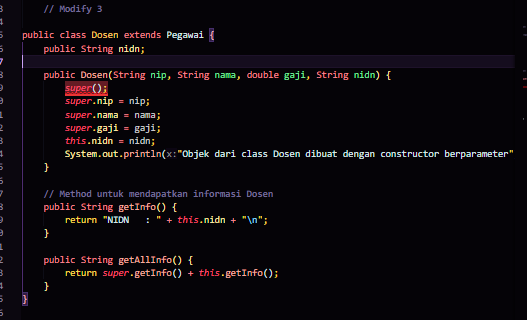


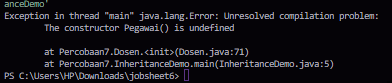




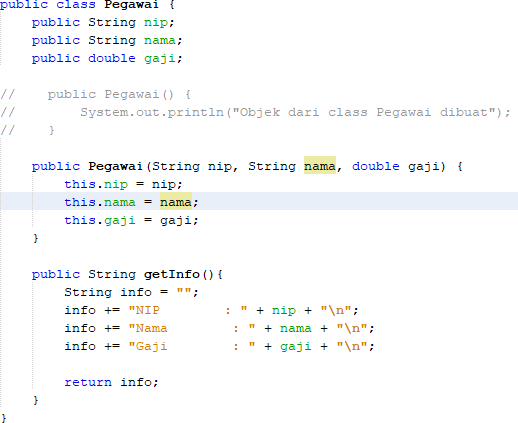
* + 1. Modify the constructor of the Lecturer class as follows. Run the program then observe the result.

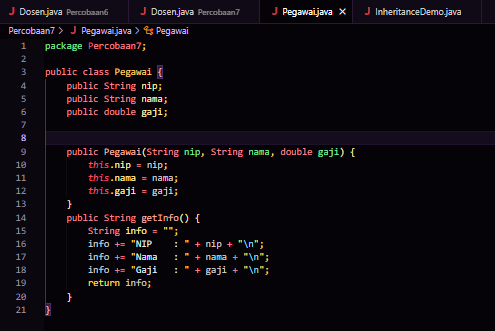


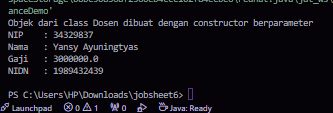




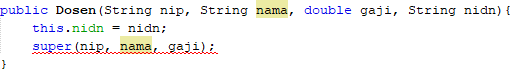
* + 1. Remove/comment the parameterless constructor of the Employee class. Add a new constructor for the Employee class as follows. Run the program and observe the result.

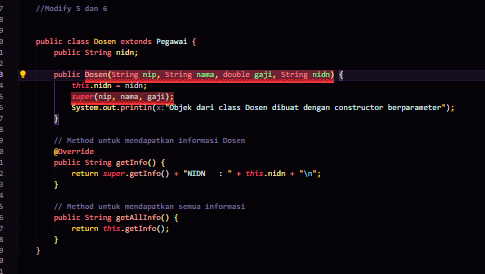


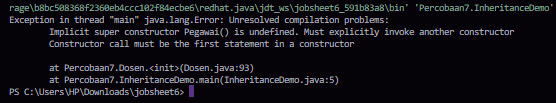




* + 1. Modify the constructor of the Lecturer class as follows. Run the program then observe the result.

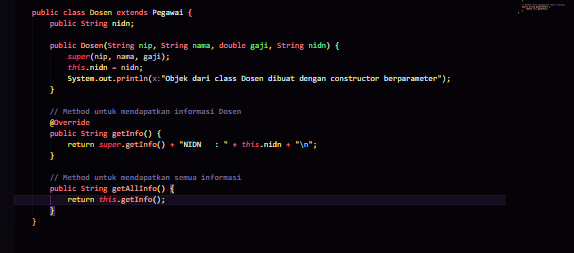


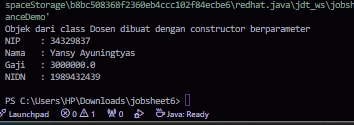




* + 1. Modify the constructor of the Lecturer class as follows. Run the program then observe the results.







## QUESTION

* + 1. Are there any differences in the results of steps 1 and 2? Explain!

**None, results when displayed are the same**

* + 1. Are there any differences in the results of steps 2 and 3? Explain!

**Yes, the difference occurs when Step 2 can display the result, while Step 3 has an error** because The constructor Employee() is undefined**.**

* + 1. Why is there an error in step 4?

**The error in step 4 occurs because you are calling the parameterless constructor of the Employee class implicitly with super(), but that constructor is not defined.**

* + 1. What is the difference between super() called in steps 3 and 6?

**Constructor Call: In step 3, super() without parameters is invalid if the default constructor is missing from the parent class. In contrast, in step 6, super(nip, name, salary) calls the appropriate constructor.**

**Proper Initialization: Step 6 ensures that all parent class attributes are properly initialized, whereas step 3 does not do so effectively.**

* + 1. Why is there an error in step 5?

**this.nidn = nidn; is now set first to initialize the nidn attribute.**

**super(nip, name, salary); is called afterward to call the constructor of the Employee class and initialize the nip, name, and salary attributes.**

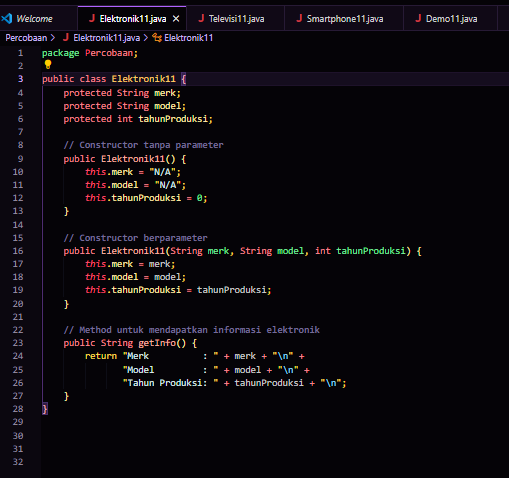
# TASK

* 1. Define a class that is an instance of another class.
  2. Create 3 attributes in the parent class then add at least 1 attribute to the child class.
  3. Perform method overloading by creating 2 constructors, namely the constructor without

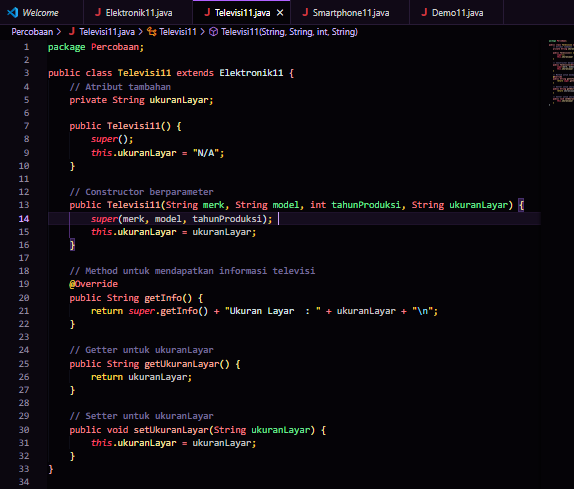
parameters and parameterized constructor in each class. Call the parameterized super() constructor to create an object of the parent class in the child class constructor.

* 1. Implement the class diagram created in the PBO theory course
  2. Create a Demo class then instantiate the child class object in the main function
  3. Try to modify the attribute value (either declared in the child class or inherited from it) and print the info.

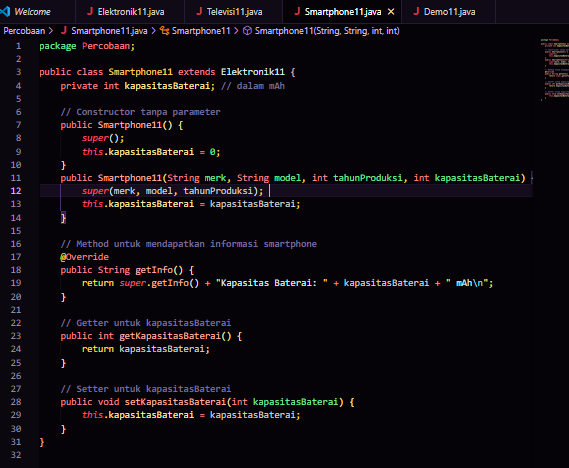
1. Electronic Classroom



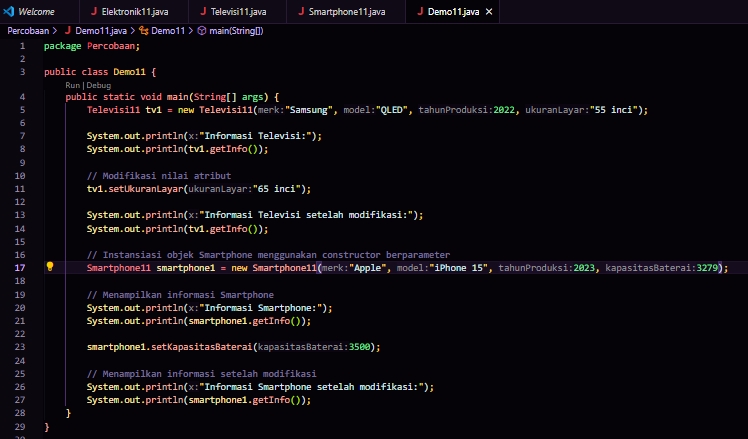
1. Television Class



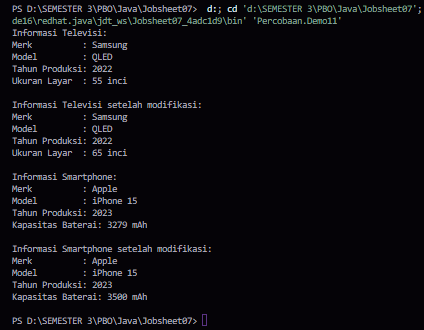
1. Smartphone class



1. Demo class



1. Rum Results



**Getters and Setters: Getters are used to get the value of a private attribute, and setters are used to change the value of that attribute. This follows the principle of encapsulation in object-oriented programming.**

**Using Setters: In the Demo class, we can now use setters to modify the values of the ScreenSize and BatteryCapacity attributes.**

### --- good work----