Name : Evan Diantha Fafian
Class : SIB 2G
Absent : -09
NIM : 2341760163

JOBSHEET W06
INHERITANCE

1. COMPETENCE

1. Understand the basic concept 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 own or derived from their superclass.

2. INTRODUCTION

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

In the Java programming language, inheritance declarations are made 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 do **an extension/extension** of 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 is extending class A. This means that class B is a subclass of class A by extension. This extension will be done by adding special attributes and methods that only class B has.

A parent class can limit the attributes and methods that will be inherited to its subclasses . The restriction is carried out through the determination of access level modifiers. In Java, the access level modifier attributes and methods are summarized in the following table:

Modifier	class yang sama	package yang sama	subclass	class manapun
private	√			
default	√	√		
protected	√	√	√	
public	√	√	√	√

Attributes and methods that will be inherited from parent class to child class are attributes and methods with a protected or public modifier.

The keyword **this** 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.<nameAttributes>**
  Accessing parent attributes
- **super.<nameMethod>()**

Calling the parent method

1. EXPERIMENT 1 (extends)

A. TRIAL STAGES

1. Create a parent class with the name of the Pegawai. Then create a parameterless constructor with the following line of code:

```
public class Pegawai {

    public Pegawai() {
        System.out.println("Objek dari class Pegawai dibuat");
    }
}
```

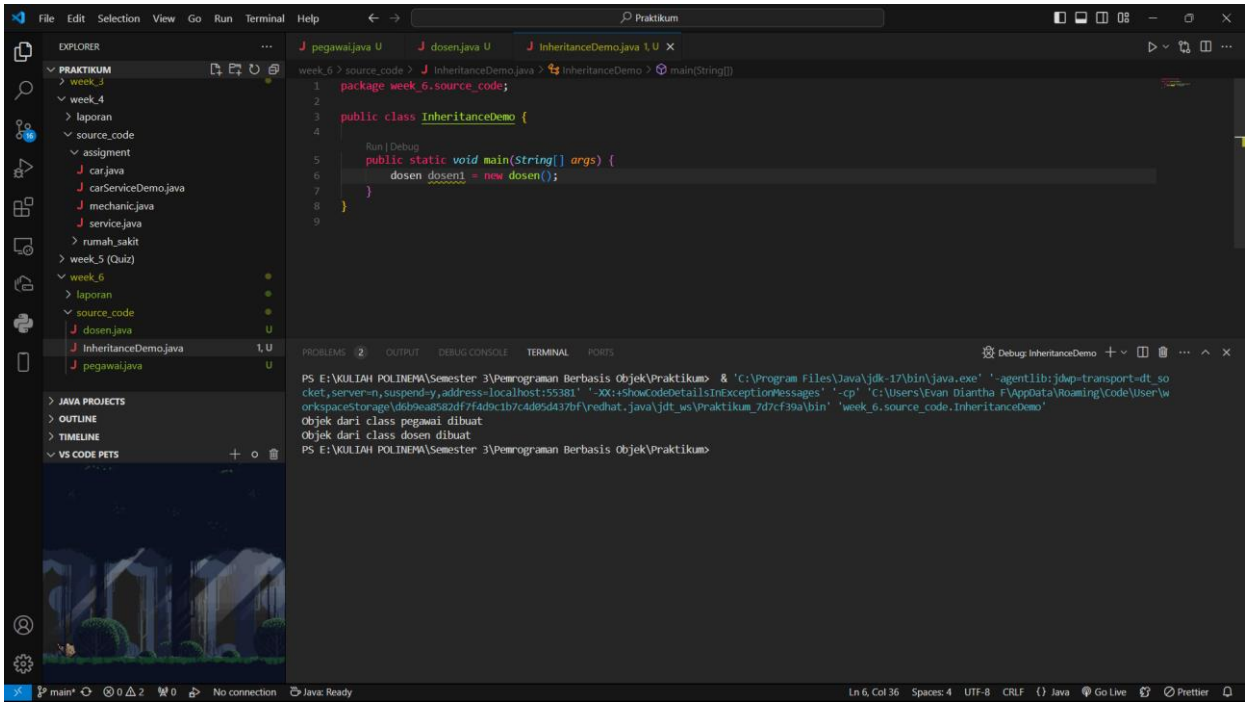
2. Create a subclass of the Pegawai class with the name Dosen, then also create a parameterless constructor with the following line of code:

```
public class Dosen extends Pegawai {

    public Dosen() {
        System.out.println("Objek dari class Dosen dibuat");
    }

}
```

3. Create a main class, for example InheritanceDemo.java, instantiate a new object named dosen1 from the lecturer class as follows:

```
public static void main(String[] args) {
    Dosen dosen1 = new Dosen();
}
```

4. Run the program and then observe the results.



B. QUESTION

1. In experiment 1 above, determine the child class and parent class!
   - parent : pegawai
   - child : dosen
2. What keywords make the child class and parent class have a relationship?
   - public class dosen **extends** pegawai {
3. Based on the results displayed by the program, how many constructors are executed? Which constructor class is executed first?
   - there are 2 constructors that are executed (the constructor of the **pegawai** class and constructor of the **dosen** class). The constructor of the **pegawai** class is executed first because the **dosen** class is a subclass of the **pegawai** class. When an object of the **dosen** class is created, the constructor of the **pegawai** class is executed first before the constructor of the **dosen** class is executed.

4. EXPERIMENT 2 (Inheritance)

A. TRIAL STAGES

1. Add the nip, nama, and gaji attributes and the getInfo() method to the Pegawai class

```
public class Pegawai {
    public String nip;
    public String nama;
    public double gaji;

    public Pegawai() {
        System.out.println("Objek dari class Pegawai dibuat");
    }

    public String getInfo(){
        String info = "";
        info += "NIP         : " + nip + "\n";
        info += "Nama         : " + nama + "\n";
        info += "Gaji         : " + gaji + "\n";

        return info;
    }
}
```

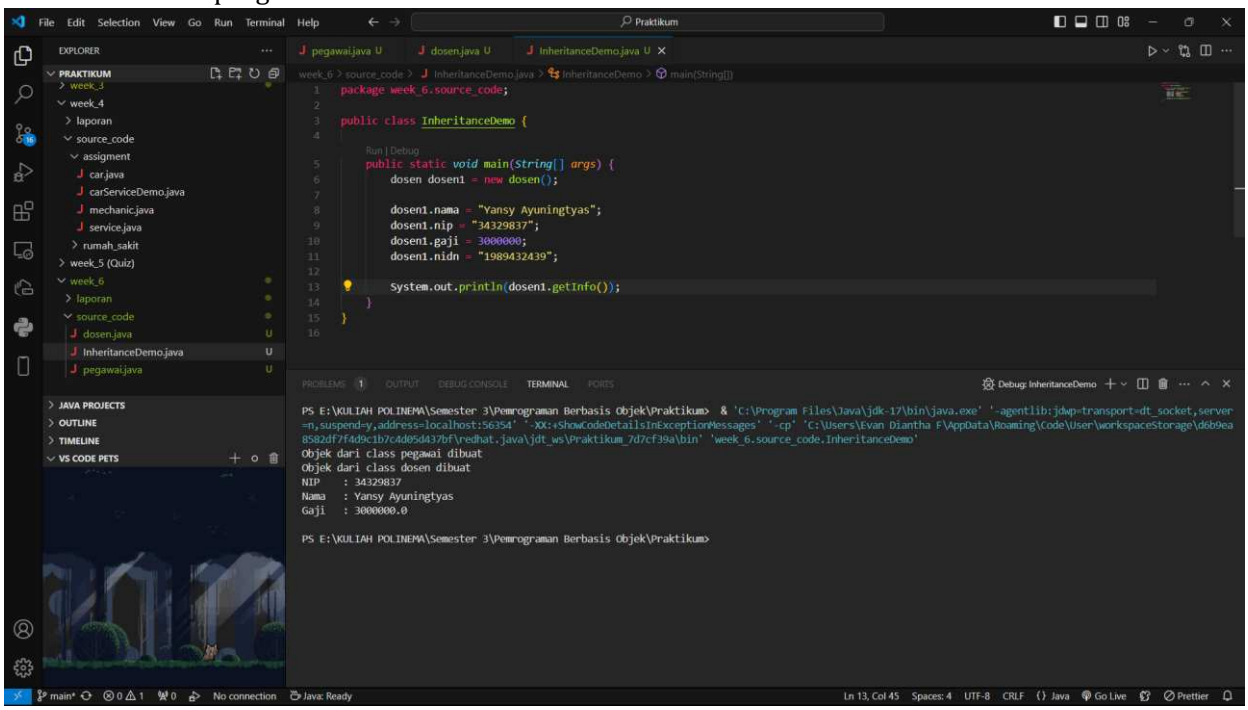
2. Also add the NIDN attribute to the Dosen class

```
public class Dosen extends Pegawai {
    public String nidn;

    public Dosen() {
        System.out.println("Objek dari class Dosen dibuat");
    }
}
```

3. In class InheritanceDemo.java write the following line of code:

```
public static void main(String[] args) {
    Dosen dosen1 = new Dosen();

    dosen1.nama = "Yansy Ayuningtyas";
    dosen1.nip = "34329837";
    dosen1.gaji = 3000000;
    dosen1.nidn = "1989432439";

    System.out.println(dosen1.getInfo());
}
```

4. Run the program then observe the results



B. QUESTION

1. In experiment 2 above, can the program run successfully or does an error occur?
   - The program will be executed successfully without any errors.
2. If the program is successfully executed, why is there no error in the assignment/filling in the values of the nip, gaji, and NIDN attributes on the lecturer object1 even though there is no declaration of these three attributes in the lecturer class?
   - This is because the Dosen class extends the Pegawai class, which means it inherits all the attributes and methods of the Pegawai class. The nip, gaji, and nidn attributes are declared in the Pegawai class, so they are also available in the Dosen class through inheritance.
3. If the program is successfully executed, why is there no error in the call of the getInfo() method by the lecturer1 object even though there is no getInfo() method declaration in the dosen class?
   - Again, this is because the Dosen class extends the Pegawai class, which means it inherits all the methods of the Pegawai class, including the getInfo() method. Even though the Dosen class doesn't declare its own getInfo() method, it can still call the getInfo() method inherited from the Pegawai class.

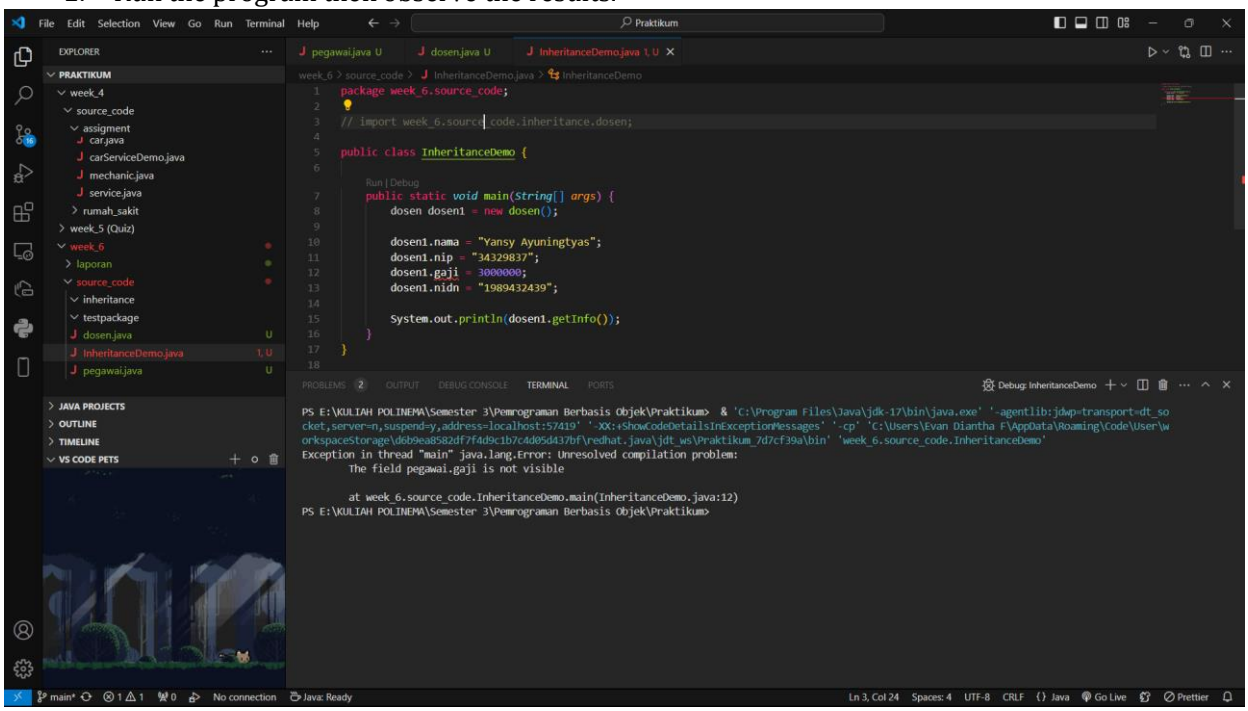
5. EXPERIMENT 3 (Access rights)

A. TRIAL STAGES

1. Modification of access level modifier on gaji attributes to private in class Pegawai.java

```
public class Pegawai {
    public String nip;
    public String nama;
    private double gaji;
```

2. Run the program then observe the results.



- The program will still run successfully, but the gaji attribute is now private, so it can only be accessed within the Pegawai class.

3. Change the access level modifier of the gaji attribute to protected and then move the Pegawai class to a new package, for example "testpackage".

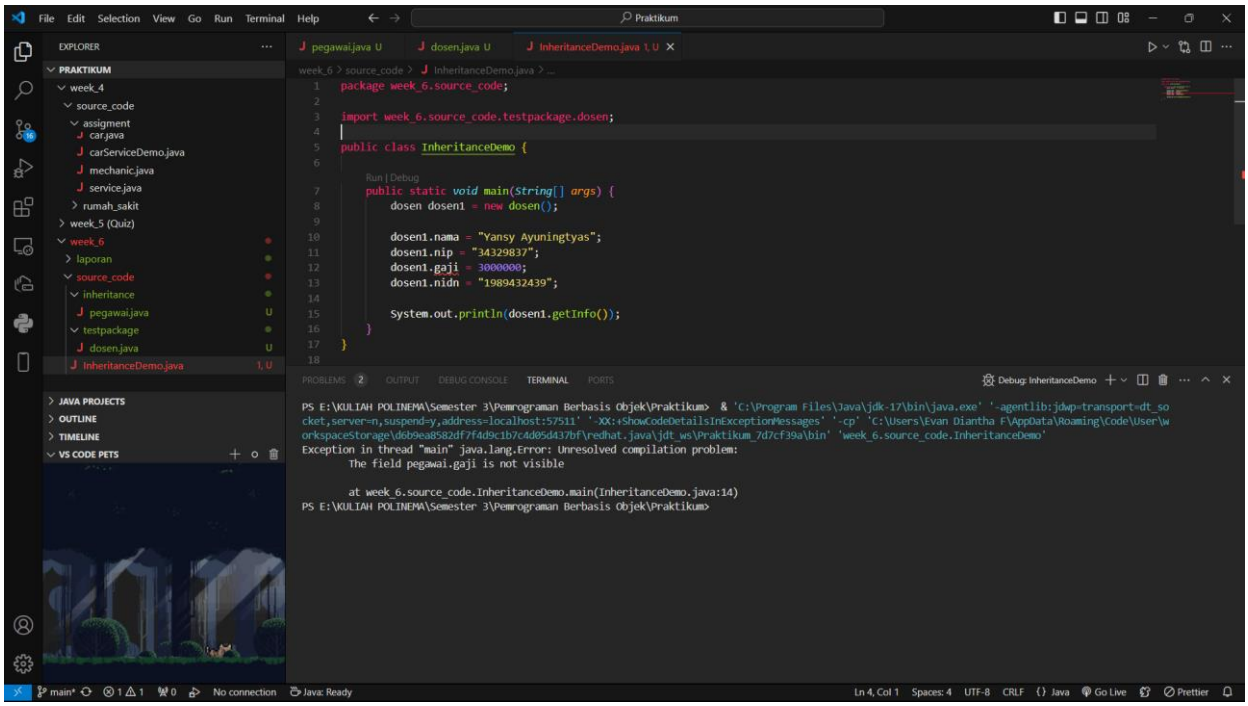
```
package testpackage;

public class Pegawai {
    public String nip;
    public String nama;
    protected double gaji;
```

4. Import the Pegawai class from the testpackage in the Dosen class.

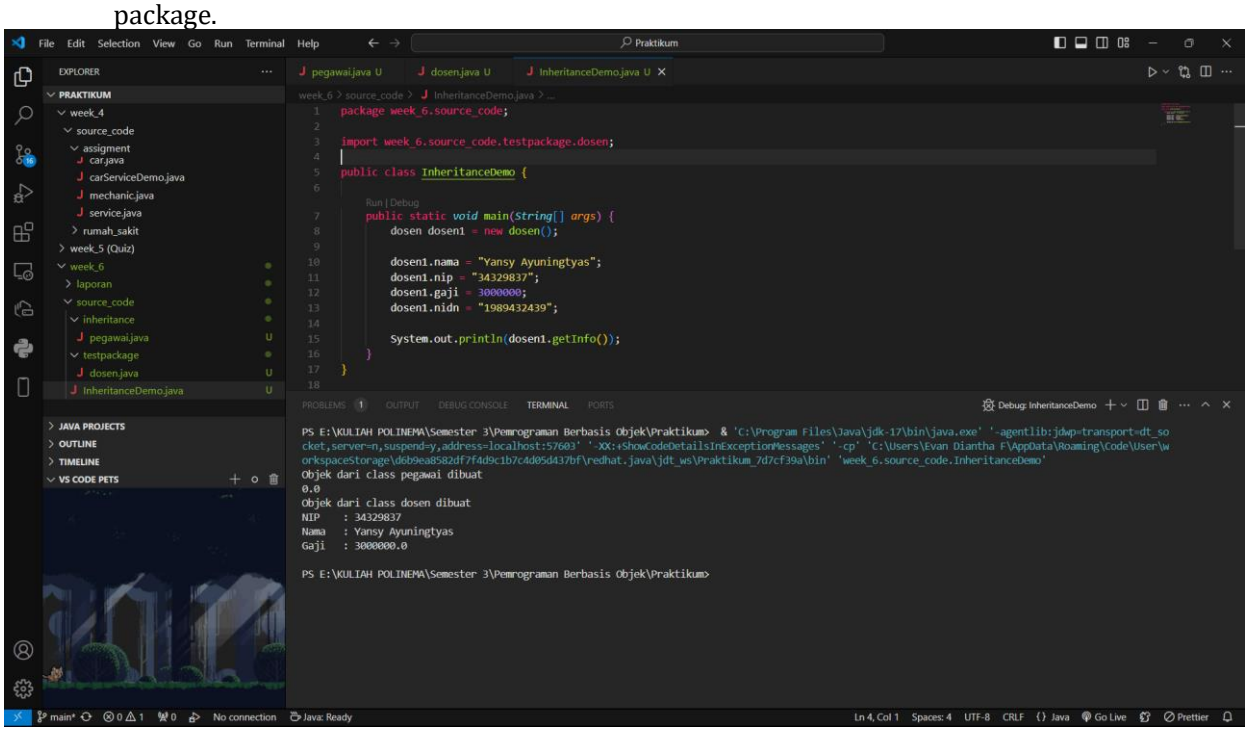
```
package inheritance;
import testpackage.Pegawai;
```

5. Access salary attributes in the Dosen class by trying to print the gaji attributes in the Lecturer constructor

```
public Dosen() {
    System.out.println(gaji);
    System.out.println("Objek dari class Dosen dibuat");
}
```



6. Change the access level modifier back to public and revert the Pegawai class to the original package.



B. QUESTION

1. In step 1 above, an error occurred because the dosen object1 could not access the gaji attributes. Even though gaji is an attribute of an pegawai who is the parent class of the dosen. Why does this happen?
   - The salary attribute is declared private in the employee class, so that the attribute can only be accessed in the employee class itself. Therefore, the lecturer1 object which is a child of the employee class cannot access the salary attribute directly.
2. In step 5, after the Pegawai class moves to a different package, the Dosen class can still access the gaji attributes. Why?
   - The salary attribute is declared as protected in the Employee class, so that this attribute can be accessed by the Lecturer class which is a subclass of the Employee class, even if the Lecturer class is in a different package.
3. Based on the experiment, how to determine the attributes and methods that will be inherited by the parent class to the child class?
   - Private

- Protected
- Public
- no modifier (*package-private*)

6. EXPERIMENT 4 (Super - attributes)

A. TRIAL STAGES

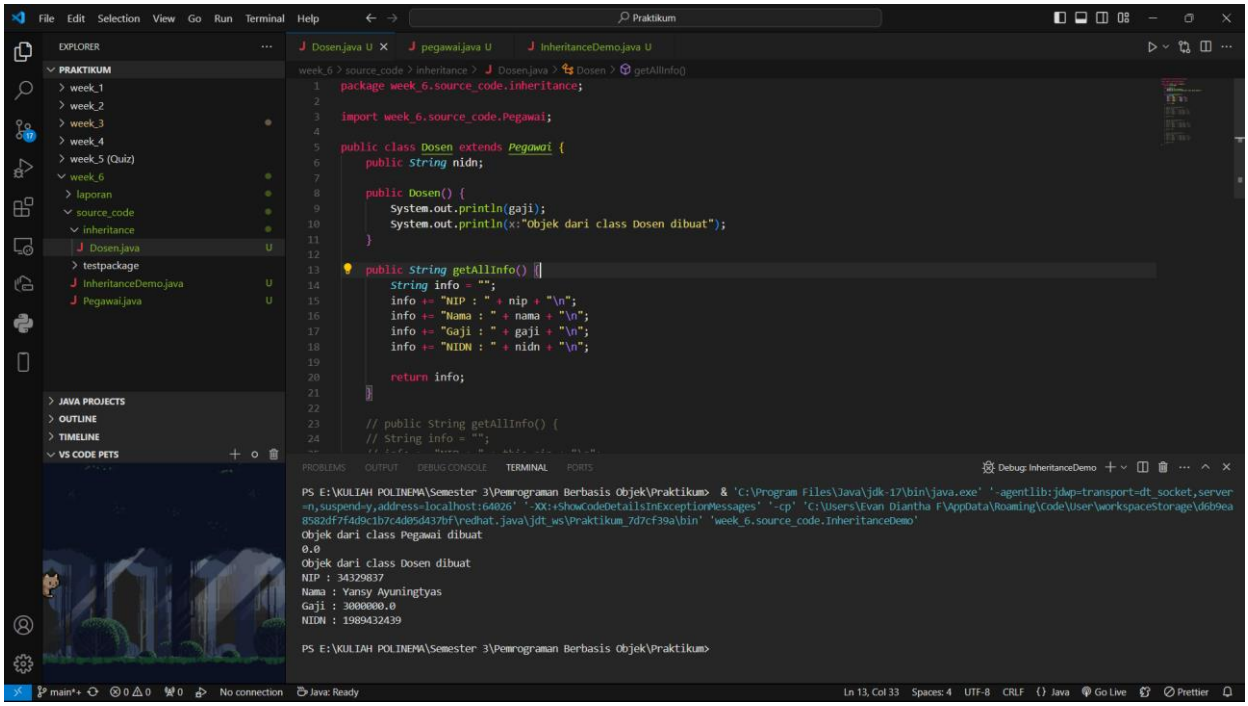
1. Use the getAllInfo() method in the Dosen class

```
public String getAllInfo(){
    String info = "";
    info += "NIP         : " + nip + "\n";
    info += "Nama        : " + nama + "\n";
    info += "Gaji        : " + gaji + "\n";
    info += "NIDN        : " + nidn + "\n";

    return  info;
}
```

2. Call the getAllInfo() method by the dosen1 object on class InheritanceDemo.java

```
public static void main(String[] args) {
    Dosen dosen1 = new Dosen();

    dosen1.nama = "Yansy Ayuningtyas";
    dosen1.nip = "34329837";
    dosen1.gaji = 3000000;
    dosen1.nidn = "1989432439";

    System.out.println(dosen1.getAllInfo());
}
```

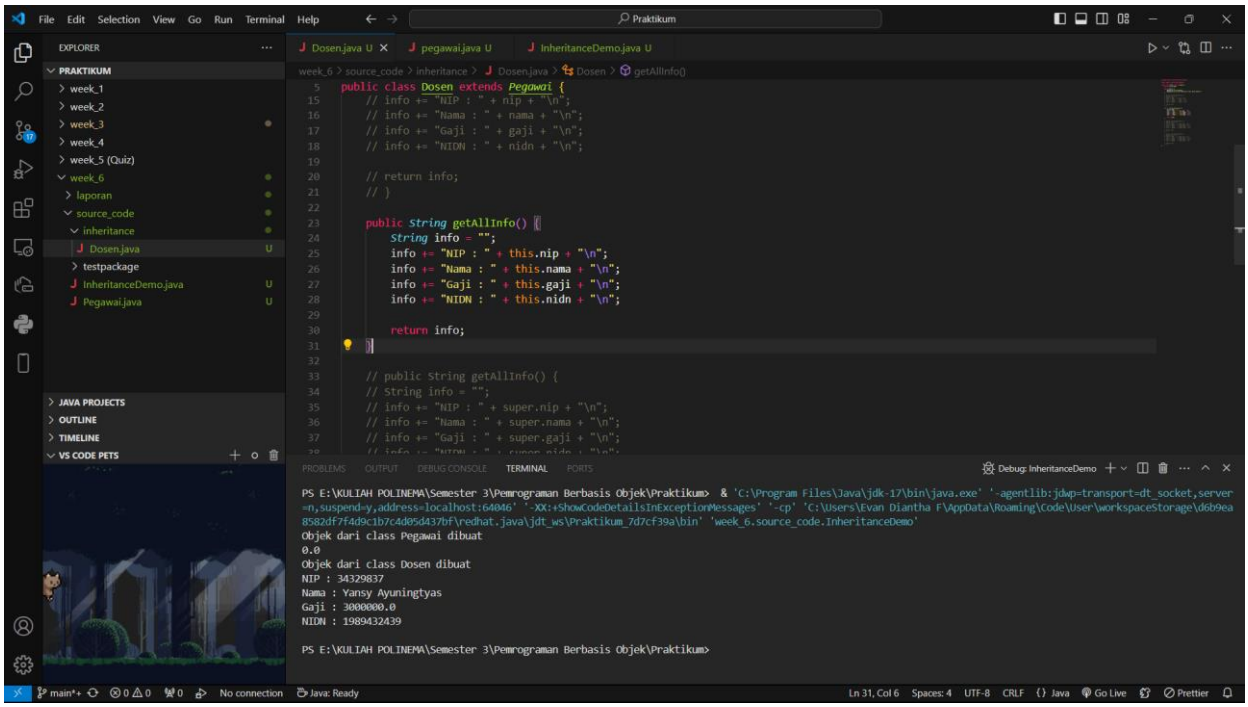
3. Run the program then observe the results



4. Modify the getAllInfo() method in the Dosen class

```java
public String getAllInfo(){
    String info = "";
    info += "NIP         : " + this.nip + "\n";
    info += "Nama        : " + this.nama + "\n";
    info += "Gaji        : " + this.gaji + "\n";
    info += "NIDN        : " + this.nidn + "\n";

    return  info;
}
```

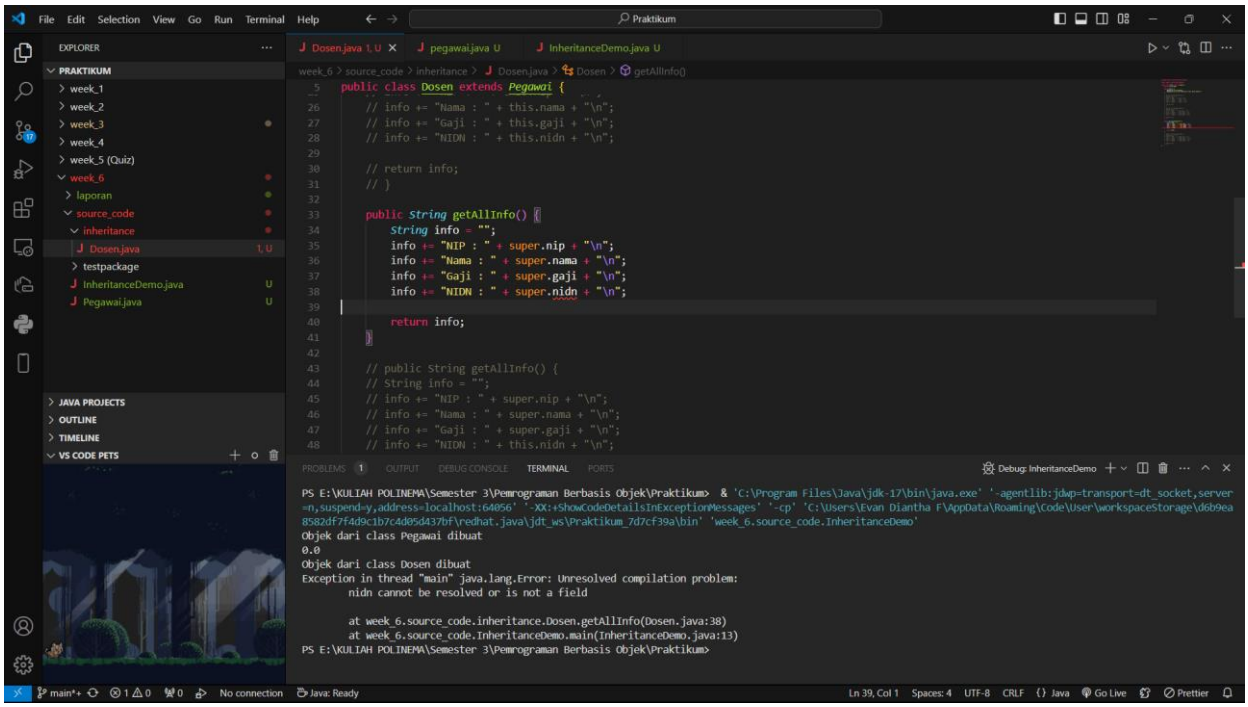
5. Run the program then compare the results with step no 2.



6. Modify the getAllInfo() method on the Dosen class again

```java
public String getAllInfo(){
    String info = "";
    info += "NIP         : " + super.nip + "\n";
    info += "Nama        : " + super.nama + "\n";
    info += "Gaji        : " + super.gaji + "\n";
    info += "NIDN        : " + super.nidn + "\n";

    return  info;
}
```

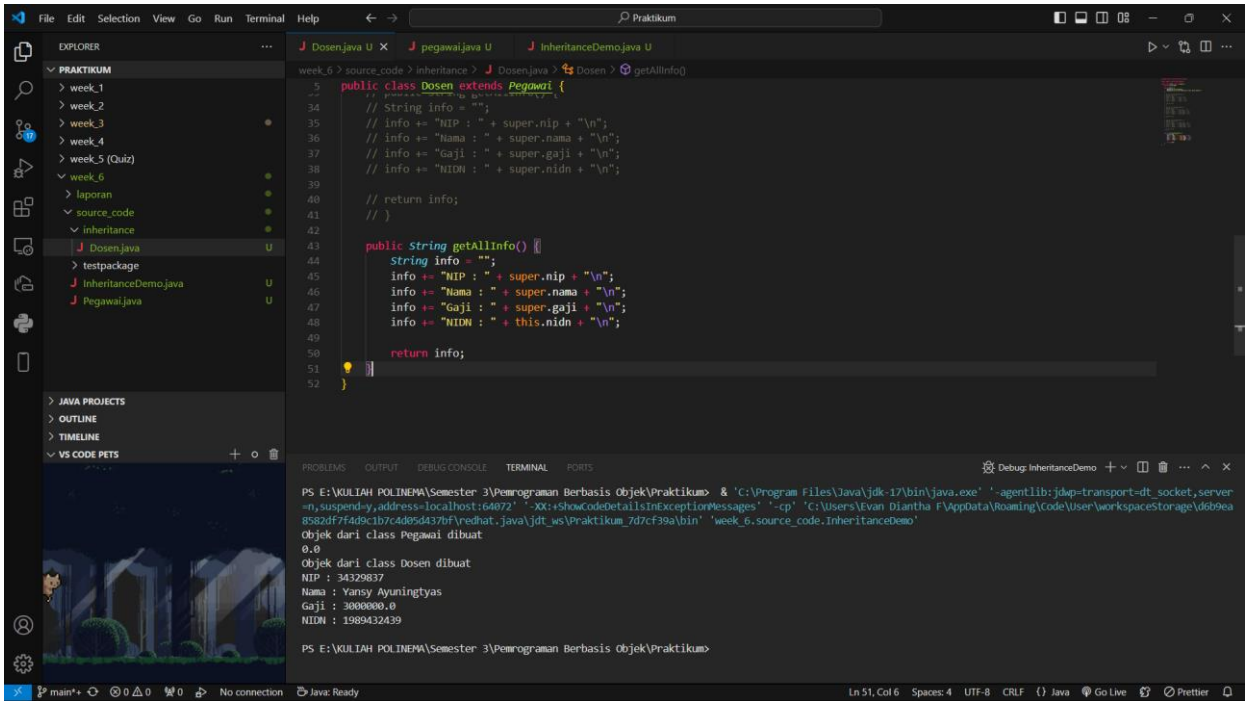
7. Run the program and then compare the results with the program at no 1 and no 4.



8. Modify the getAllInfo() method on the Dosen class again

```
public String getAllInfo(){
    String info = "";
    info += "NIP          : " + super.nip + "\n";
    info += "Nama         : " + super.nama + "\n";
    info += "Gaji         : " + super.gaji + "\n";
    info += "NIDN         : " + this.nidn + "\n";

    return  info;
}
```

9. Run the program and then compare the results with the program at number 2 and number 4.



B. QUESTION

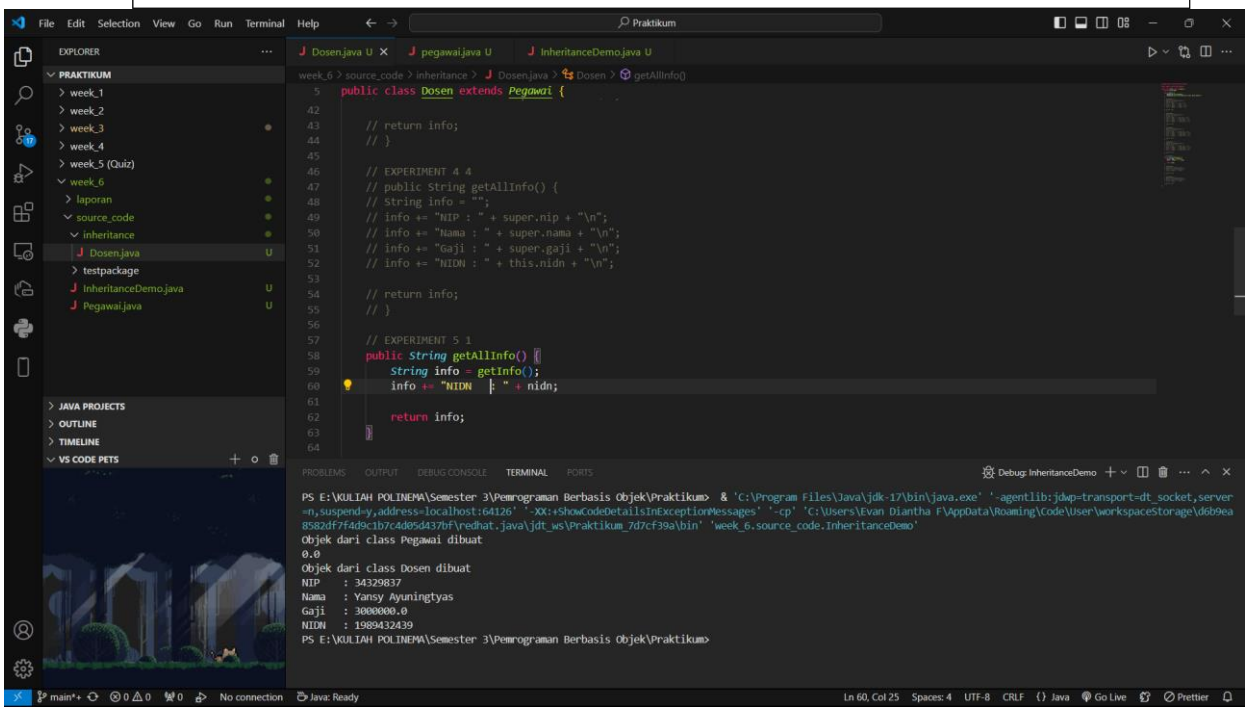
1. Are there any differences in the results of nama, nip, and gaji displayed in programs 1, 4, and 8? Why?
   - There is no difference in the results of name, nip, and salary displayed in programs 1, 4, and 8. In programs 1, 4, and 8, the nip, name, and salary attributes are accessed directly using the attribute names, namely nip, name, and salary. Because these attributes are declared as public in the employee class, they can be accessed directly by the lecturer class.
2. Why does the error occur in program no 6?
   - The error occurs in program number 6 because the nidn attribute is not declared in the employee class, but is declared in the lecturer class. Therefore, when the program tries to access the nidn attribute using the super keyword, an error will occur because the nidn attribute does not exist in the employee class.

7. EXPERIMENT 5 (super & overriding)

A. TRIAL STAGES

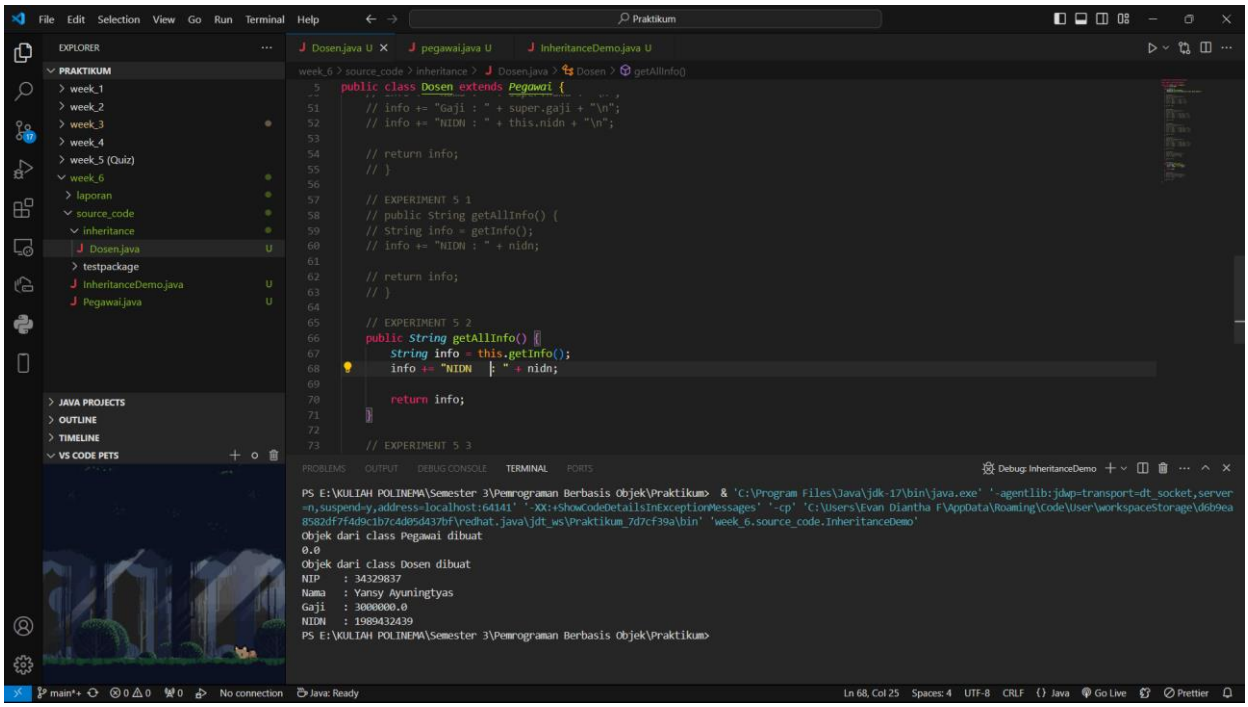
1. Modify the getAllInfo() method again. Run the program then observe the results

```
public String getAllInfo(){
    String info = getInfo();
    info += "NIDN        : " + nidn;

    return info;
}
```



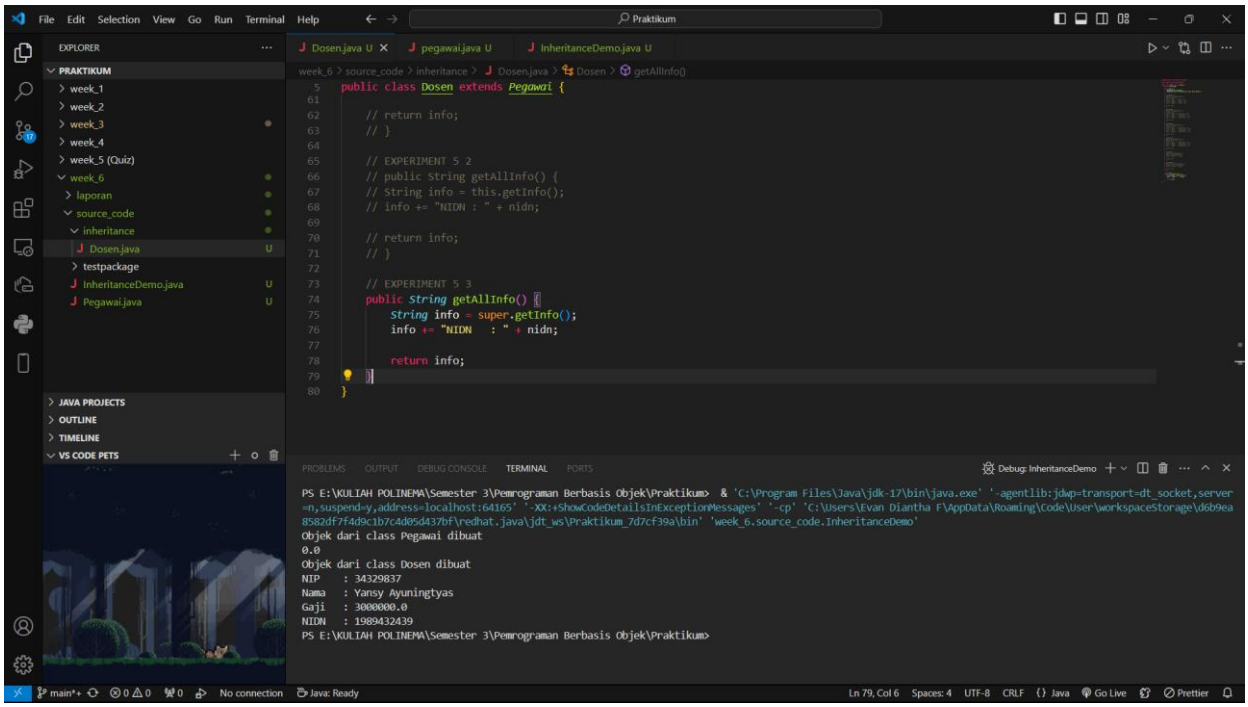
2. Modify the getAllInfo() method again. Run the program then observe the results

```
public String getAllInfo(){
    String info = this.getInfo();
    info += "NIDN        : " + nidn;

    return info;
}
```



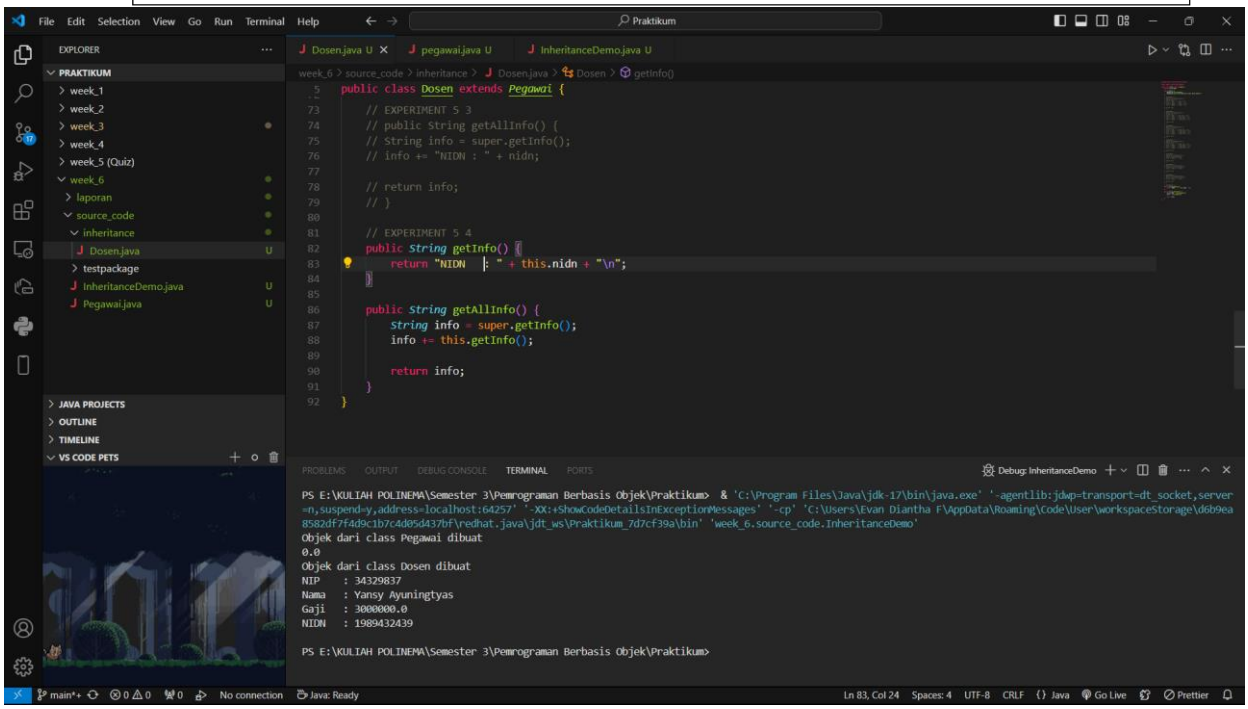
3. Modify the getAllInfo() method again. Run the program then observe the results

```
public String getAllInfo(){
    String info = super.getInfo();
    info += "NIDN        : " + nidn;

    return info;
}
```



4. Add the getInfo() method to the Dosen class and modify the getAllInfo() method as follows

```java
public class Dosen extends Pegawai {
    public String nidn;

    public Dosen() {
        System.out.println("Objek dari class Dosen dibuat");
    }

    public String getInfo(){
        return "NIDN       : " + this.nidn + "\n";
    }

    public String getAllInfo(){
        String info = super.getInfo();
        info += this.getInfo();

        return info;
    }
}
```



B. QUESTION

1. Are there any differences in the getInfo() methods accessed in steps 1, 2, and 3?
   - Step 1 and 2: getInfo() is called from the Lecturer class because the method is defined in the same class. The only difference is that the method in Step 1 uses a direct call (getInfo()), while in Step 2 it uses this.getInfo(). Both call the same method in the Lecturer class.
   - Step 3: super.getInfo() is called to retrieve the getInfo() implementation from the superclass (Employee). This means, if the getInfo() method is overridden in the Lecturer class, the super.getInfo() call will retrieve the getInfo() version from Employee, not from Lecturer.
2. Is there a difference between the super.getInfo() and this.getInfo() methods called in the getAllInfo() method in step 4? Explain!
   - super.getInfo(): Calls the getInfo() method of the superclass (Employee). If the Lecturer overrides the getInfo() method, then super.getInfo() will still retrieve the implementation from the superclass (Employee), not from the Lecturer.
   - this.getInfo(): Calls the getInfo() method of the Lecturer class. If getInfo() is overridden in the Lecturer class, then this.getInfo() will use the overridden version.

3. In what method does overriding occur? Explain!
    - Overriding occurs in the getInfo() method. Overriding is when a subclass (Lecturer) redefines a method that already exists in a superclass (Employee) in the same way (same signature). In this case, the getInfo() method is defined in Employee, and then overridden in the Lecturer class with a different implementation. Overriding allows subclasses to provide specific implementations of methods that have been defined by the superclass.

8. EXPERIMENT 6 (overloading)

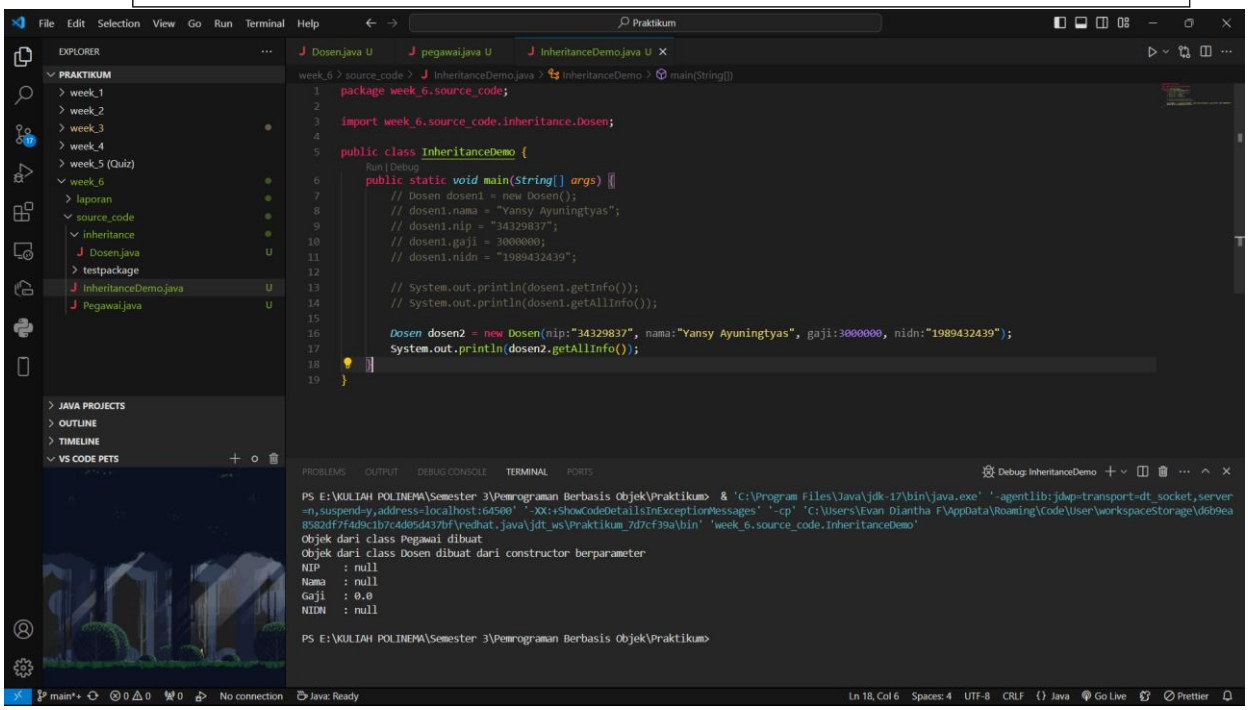
A. TRIAL STAGES

1. Add a new constructor for the Dosen class as follows

```
public Dosen(String nip, String nama, double gaji, String nidn){
    System.out.print("Objek dari class Dosen dibuat dengan constructor berparameter");
}
```

2. Modify the InheritanceDemo class to instantiate a new object with the name lecturer2 with a parameterized constructor. Run the program then observe the results.

```
public static void main(String[] args) {
    Dosen dosen2 = new Dosen("34329837","Yansy Ayuningtyas",3000000,"1989432439");
    System.out.println(dosen2.getAllInfo());
}
```



B. QUESTION

1. What are the results of the nip, nama, gaji, and nidn values displayed in step 2? Why is that?
   - The parameterized constructor (Dosen(String nip, String nama, double gaji, String nidn)) does not initialize the attributes nip, nama, gaji, and nidn. These attributes retain their default values: null for strings and 0.0 for the double.
   - Even though arguments are passed when the dosen2 object is created, the constructor doesn't map the arguments to the object's attributes.
2. Explain whether the parameterless constructor and the Dosen class constructor created in step 1 have the same signature?

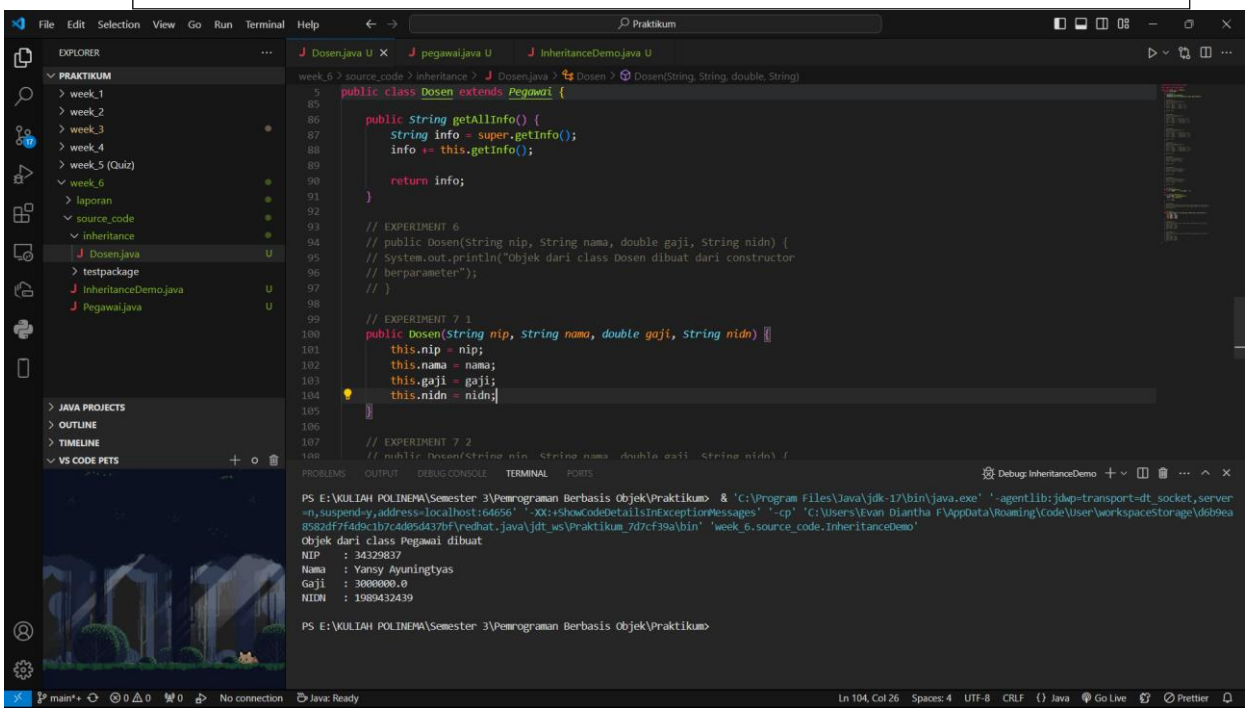
   No, the no-argument constructor and the parameterized constructor in the Dosen class do not have the same signature.
   - No-argument constructor: Dosen()
   - Parameterized constructor: Dosen(String nip, String nama, double gaji, String nidn)
3. What is the concept in OOP that allows a class to have a constructor or method with the same name and a different signature on a class?
   - Overloading allows a class to have multiple methods or constructors with the same name but different signatures (different numbers or types of parameters).
   - In this case, the Dosen class has two constructors with the same name (Dosen), but one takes parameters and the other does not.

9. EXPERIMENT 7 (super-constructor)

A. TRIAL STAGES

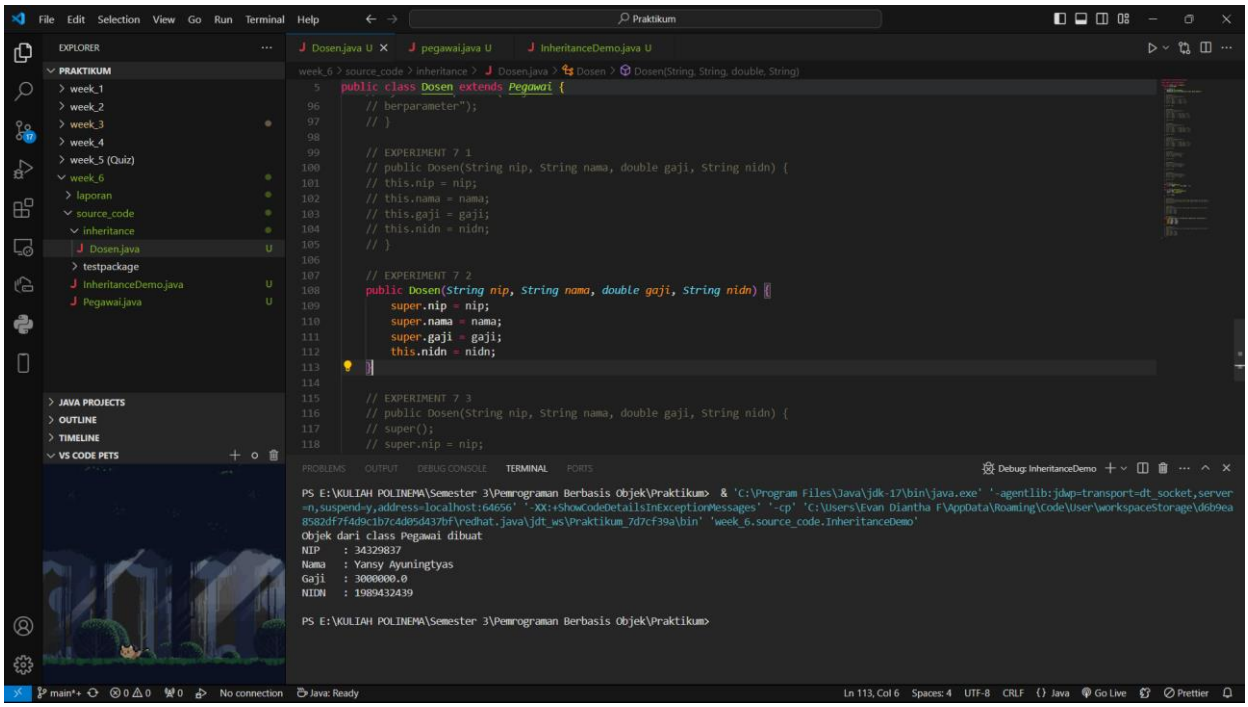
1. Constructor modifications in the Dosen class are as follows. Run the program then observe the results.

```
public Dosen(String nip, String nama, double gaji, String nidn){
    this.nip = nip;
    this.nama = nama;
    this.gaji = gaji;
    this.nidn = nidn;
}
```



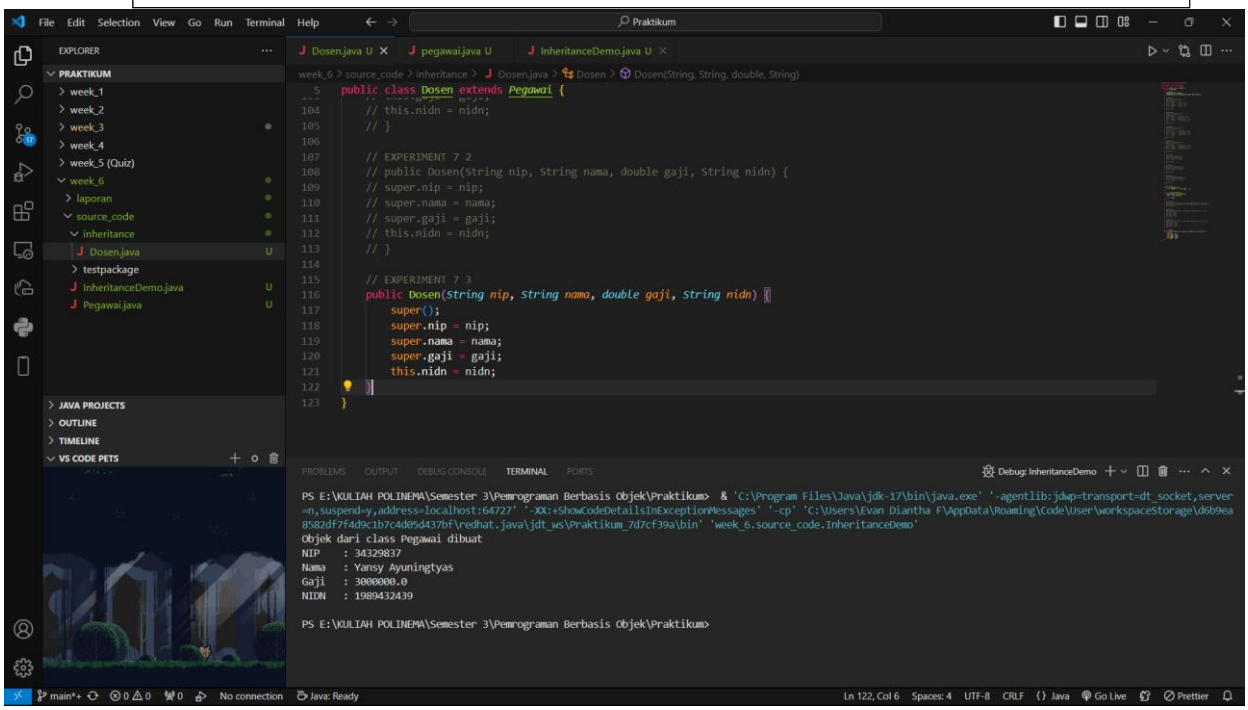
2. Constructor modifications in the Dosen class are as follows. Run the program then observe the results.

```
public Dosen(String nip, String nama, double gaji, String nidn){
    super.nip = nip;
    super.nama = nama;
    super.gaji = gaji;
    this.nidn = nidn;
}
```



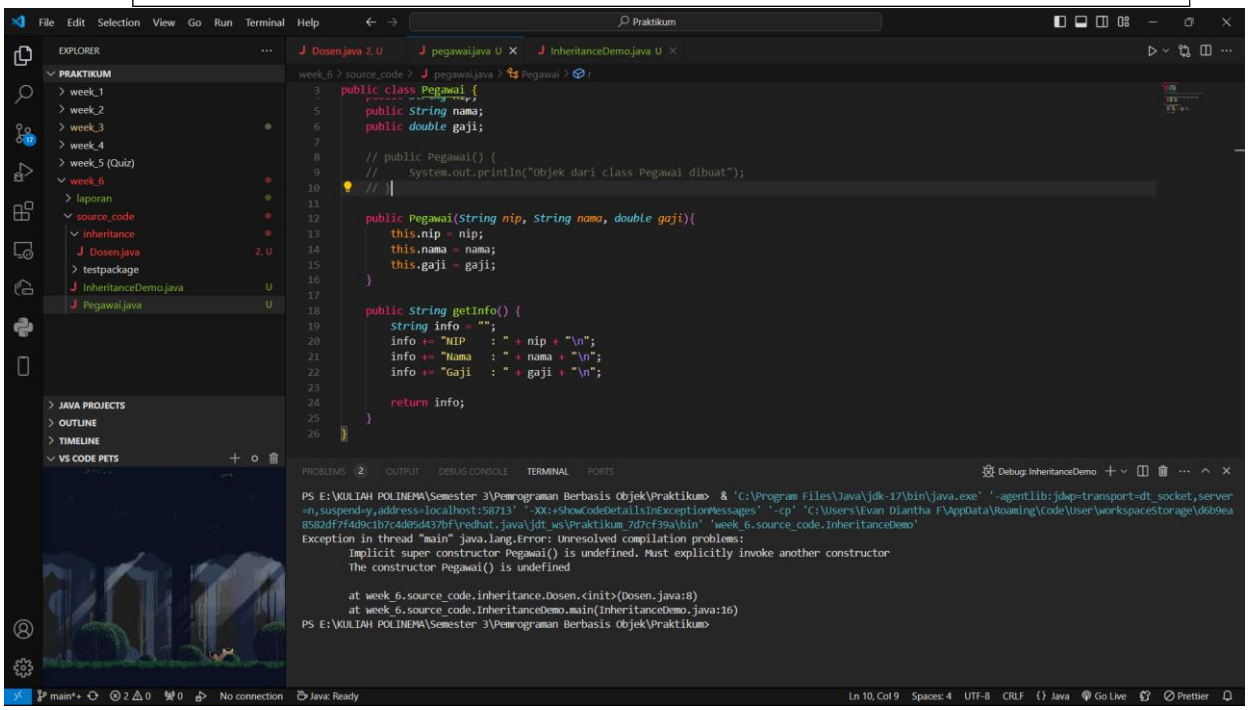
3. Constructor modifications in the Dosen class are as follows. Run the program then observe the results.

```
public Dosen(String nip, String nama, double gaji, String nidn){
    super();
    super.nip = nip;
    super.nama = nama;
    super.gaji = gaji;
    this.nidn = nidn;
}
```



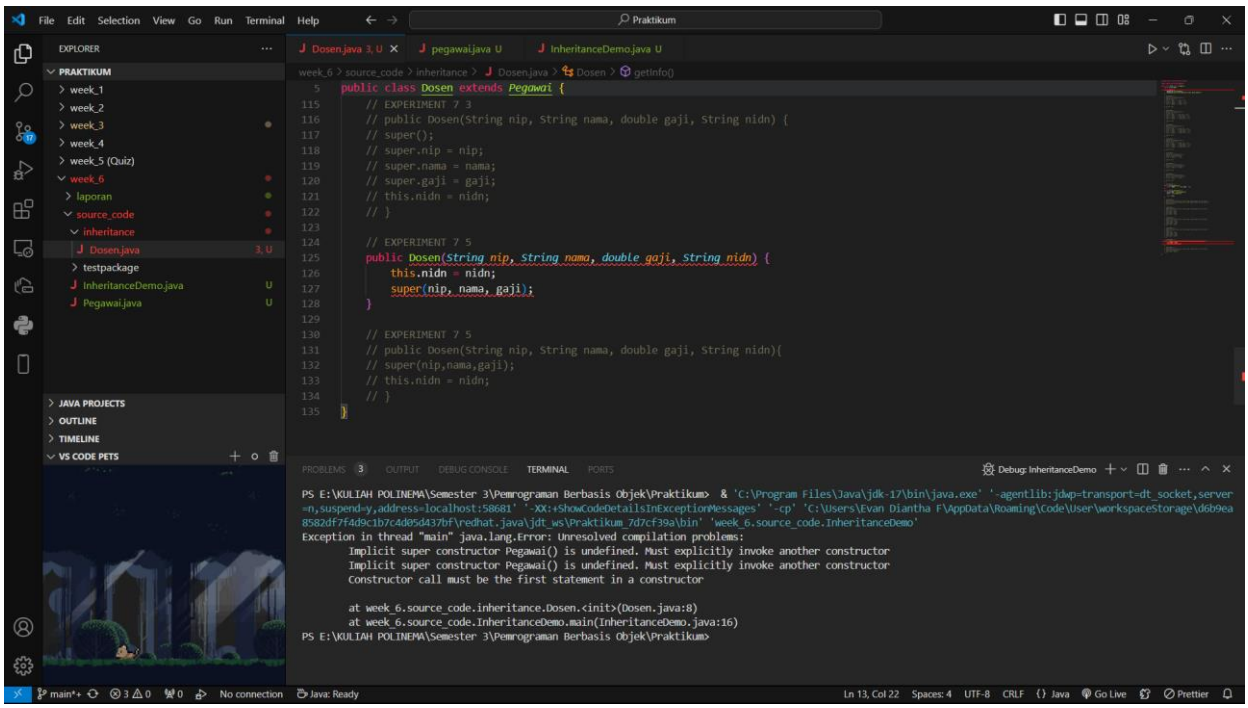
4. Remove/comment constructor without parameters from the Pegawai class. Add a new constructor for the Pegawai class as follows. Run the program then observe the results.

```java
public class Pegawai {
    public String nip;
    public String nama;
    public double gaji;

//     public Pegawai() {
//         System.out.println("Objek dari class Pegawai dibuat");
//     }

    public Pegawai(String nip, String nama, double gaji) {
        this.nip = nip;
        this.nama = nama;
        this.gaji = gaji;
    }

    public String getInfo(){
        String info = "";
        info += "NIP         : " + nip + "\n";
        info += "Nama        : " + nama + "\n";
        info += "Gaji        : " + gaji + "\n";

        return info;
    }
}
```



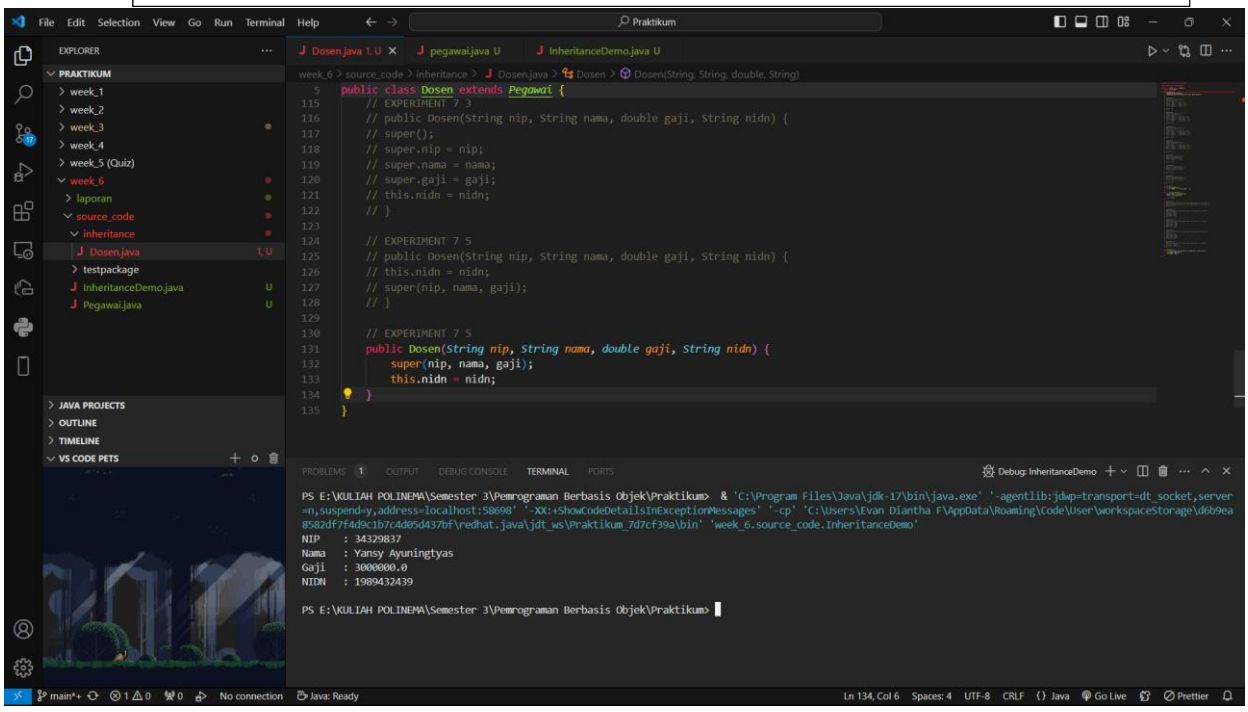
5. Constructor modifications in the Dosen class are as follows. Run the program then observe the results.

```java
public Dosen(String nip, String nama, double gaji, String nidn){
    this.nidn = nidn;
    super(nip, nama, gaji);
}
```



6. Constructor modifications in the Dosen class are as follows. Run the program then observe the results.

```
public Dosen(String nip, String nama, double gaji, String nidn){
    super(nip, nama, gaji);
    this.nidn = nidn;
}
```



B. QUESTION

1. Is there a difference in the results in steps 1 and 2? Explain!
   - In step 1, the attributes are initialized using this, directly referencing the current instance. In step 2, super is used, which only works if the attributes are accessible and inherited. If they are not, an error will occur.
2. Is there a difference in the results in steps 2 and 3? Explain!
   - Difference between Steps 2 and 3:
   - In step 3, super() is explicitly called before setting the attributes. If super() initializes properly and the superclass has those attributes, it should work as expected. If it's not, it will lead to an error.

3. Why did the error occur in step 4?

   If the default constructor from the Pegawai class is commented out or deleted, you can't create an instance of Pegawai without parameters. The code tries to instantiate Pegawai with parameters, leading to a compile-time error if no suitable constructor exists.

4. What is the difference between super() called in steps 3 and 6?
   - In step 3, super() is called but assigns values afterward. In step 6, the super() call initializes inherited attributes properly before the subclass initializes its own attributes, which is the correct practice

5. Why did the error occur in step 5?
   - In step 5, the error occurs because this.nidn = nidn; is written before calling super(). In Java, super() must be the first statement in the constructor.

10. ASSIGNMENT

1. Define a class that is a derivative of another class.
2. Create 3 attributes in the parent class then add at least 1 attribute in the child class.
3. Perform the overloading method by creating 2 constructors, namely a parameterless

I PUT THE ASSIGNMENT ON THE GITHUB LINK

https://github.com/rankadian/PEMROGRAMAN-BERBASIS-OBJEK.git

constructor and a parameterized constructor for each class. Call the parameterized super() constructor to create an object from the parent class on the child class constructor.

4. Implement the class diagram made in the theoretical PBO course
5. Create a Demo class then instantiate the child class object in the main function
6. Try modifying the attribute values (both those declared in the child class and those inherited from the print info.

--- happy working----