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 Class:** `Pegawai`
   - **Child Class:** `Dosen`
2. What keywords make the child class and parent class have a relationship?
   - The keyword that establishes the relationship between the child class and the parent class in Java is **extends**. In this case, Dosen extends Pegawai, indicating that Dosen is a subclass of Pegawai.
3. Based on the results displayed by the program, how many constructors are executed? Which constructor class is executed first?

- **Number of Constructors Executed:** 2

  The constructor of the parent class (Pegawai) and the constructor of the child class (Dosen) are both executed.

- **Order of Constructor Execution:**

  The constructor of the parent class (Pegawai) is executed first, followed by the constructor of the child class (Dosen). This is because when a subclass is instantiated, the constructor of the parent class is called first to ensure that the parent part of the object is initialized before the child part.

4. EXPERIMENT 2 (Inheritance)

A. TRIAL STAGES

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

```java
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

```java
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:

```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.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 can run successfully. No error will occur because all the necessary elements are in place for inheritance

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?
    - The reason there is no error is due to inheritance. The `Dosen` class inherits from the `Pegawai` class. Therefore, all public or protected fields of the `Pegawai` class (like `nip`, `gaji`, and `nama`) are accessible in the `Dosen` class, even though they are not directly declared in the `Dosen` class itself.
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?
    - The `getInfo()` method is defined in the parent class `Pegawai`. Since the `Dosen` class inherits from `Pegawai`, it also inherits all of its methods, including `getInfo()`. Therefore, the `dosen1` object can call the `getInfo()` method even though it is not explicitly defined in the `Dosen` 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.
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 error occurs because the `gaji` attribute in the `Pegawai` class is marked as `protected`. The `protected` keyword allows the attribute to be accessed within the same package or by subclasses of `Pegawai`, but not by code outside the package unless accessed through inheritance. In your code, the `Dosen` class is in a different package (`inheritance`), so direct access to the `gaji` attribute from outside the `testpackage` package is not allowed unless it's through an inherited method or constructor.

2. In step 5, after the Pegawai class moves to a different package, the Dosen class can still access the gaji attributes. Why?
    - Even though the `Pegawai` class is in a different package, the `gaji` attribute is still accessible to the `Dosen` class because the `Dosen` class inherits it. The `protected` modifier allows access to the attribute for subclasses (`Dosen` in this case), even when the subclass is in a different package. So, as long as the `Dosen` class accesses the `gaji` attribute through inheritance (and not directly as if it were in the same package), the program will work.

3. Based on the experiment, how to determine the attributes and methods that will be inherited by the parent class to the child class?
   The inheritance of attributes and methods depends on their access modifiers:
    - public: Public members are inherited and accessible from anywhere, including from different packages.
    - protected: Protected members are inherited and accessible within the same package or by subclasses, even if they are in different packages.
    - default (no modifier): Members without an access modifier are accessible only within the same package. Subclasses in different packages cannot access them.
    - private: Private members are not inherited by subclasses. They are only accessible within the class where they are declared.

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

```
Objek dari dari Pegawai dibuat
0.0
Objek dari class Dosen dibuat
NIP       : 343298837
Nama      : Yansy Ayuningtyas
Gaji      : 3000000.0
NIDN      : 1989432439
```

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

```
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.

```
Objek dari dari Pegawai dibuat
0.0
Objek dari class Dosen dibuat
NIP       : 343298837
Nama      : Yansy Ayuningtyas
Gaji      : 3000000.0
NIDN      : 1989432439
```

6. 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        : " + super.nidn + "\n";

    return  info;
}
```

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

```
Objek dari dari Pegawai dibuat
0.0
Objek dari class Dosen dibuat
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        nidn cannot be resolved or is not a field

        at inheritance.Dosen.getAllInfo(Dosen.java:19)
        at InheritanceDemo.main(InheritanceDemo.java:14)
```

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 are same output

2. Why does the error occur in program no 6?
    - Because tries to access the nidn field using super. However, nidn is not a field in the Pegawai class (the parent class), but it is defined in the Dosen class itself. The keyword super refers to the parent class, and nidn is not inherited from the Pegawai class, so super.nidn results in a compilation error nidn cannot be resolved or is not a field

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;
}
```

```
Objek dari dari Pegawai dibuat
Objek dari class Dosen dibuat
NIDN    : 1989432439
NIDN        : 1989432439
```

- The method calls getInfo(), which is defined in the Pegawai class. This method returns the information related to nip, nama, and gaji. Then, the method appends the NIDN field from the Dosen class to the info string.

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;
}
```

```
Objek dari dari Pegawai dibuat
Objek dari class Dosen dibuat
NIDN    : 1989432439
NIDN        : 1989432439
```

- The second method is functionally the same as the first. It also calls the getInfo() method from Pegawai to get nip, nama, and gaji, and then adds the NIDN information from Dosen.
- The only difference is the explicit use of this.getInfo(). In this context, the this keyword is redundant because there is no ambiguity about which getInfo() method is being called. Since Dosen does not override getInfo(), the method from Pegawai is called in both cases.

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;
}
```

```
Objek dari dari Pegawai dibuat
Objek dari class Dosen dibuat
NIP        : 343298837
Nama       : Yansy Ayuningtyas
Gaji       : 3000000.0
NIDN        : 1989432439
```

- By using super.getInfo(), the method fetches information from the parent class (Pegawai), and then appends the nidn field from the Dosen class. This is a typical way to combine parent and child class information in an inherited class structure.

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

```
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?
2. Is there a difference between the super.getInfo() and this.getInfo() methods called in the getAllInfo() method in step 4? Explain!
    - The difference is in otput where if the this and the getalinfo () he will only bring up the nidn sedangakn super he will display super complete variables such as the name of the salary nip
3. In what method does overriding occur? Explain!
    - Overriding occurs in the `getInfo` method. In the `Pegawai` class, the `getInfo` method is defined to return information specific to an employee (`Pegawai`). In

the `Dosen` class, which extends the `Pegawai` class, the `getInfo` method is overridden by the `getAllInfo` method. This new method calls `super.getInfo()` to retrieve the information from the parent class (`Pegawai`) and then adds specific information related to the `Dosen` class, such as `nidn` (the lecturer's ID).

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?
    - When `Dosen dosen2 = new Dosen("34329837", "Yansy Ayuningtyas", 3000000, "1989432439");` is executed, the program will not compile successfully if no corresponding constructor exists in the `Dosen` class that accepts these four parameters (`nip`, `nama`, `gaji`, `nidn`).
2. Explain whether the parameterless constructor and the Dosen class constructor created in step 1 have the same signature?
    - No, the parameterless constructor and the new `Dosen` constructor (if added) **do not have the same signature**.
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?
    - In the case of constructors, overloading allows a class like `Dosen` to have both a **parameterless constructor** and a **parameterized constructor** that take in specific arguments for initialization.

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;
}
```

```
Objek dari dari Pegawai dibuat
NIP       : 34329837
Nama      : Yansy Ayuningtyas
Gaji      : 3000000.0
NIDN       : null
```

- The new constructor in the `Dosen` class takes four parameters: `nip`, `nama`, `gaji`, and `nidn`. It assigns these values to the corresponding fields of the `Dosen` object.

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;
}
```

```
Objek dari dari Pegawai dibuat
NIP       : 34329837
Nama      : Yansy Ayuningtyas
Gaji      : 3000000.0
NIDN       : 1989432439
```

- The use of `super` helps in maintaining clear separation of responsibilities between the parent class (`Pegawai`) and the subclass (`Dosen`), showcasing proper inheritance practices.

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;
}
```

```
Objek dari dari Pegawai dibuat
NIP       : 34329837
Nama      : Yansy Ayuningtyas
Gaji      : 3000000.0
NIDN       : 1989432439
```

- In the latest modification, the Faculty constructor uses super () to call the parameterless constructor of the employee parent class, which prints "object from from employee created". After that, the nip attribute, name, and salary are initialized from the employee, while the nidn is initialized in the lecturer. When the program is run, will be displayed complete information from lecturers with additional 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.

```
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;
    }
}
```

- The Pegawai class no longer prints "Objek dari dari Pegawai dibuat" because the parameterless constructor was removed. The new constructor in Pegawai initializes nip, nama, and gaji via the Dosen constructor using super(nip, nama, gaji). The program runs smoothly, showing the expected information for both Pegawai and Dosen.

5. 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.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, when a `Dosen` object was created, it printed `"Objek dari dari Pegawai dibuat"` from the superclass constructor. In Step 2, this message is no longer printed because the parameterless constructor was removed, leading to a cleaner output with no initialization message. The output now only shows the `NIP`, `Nama`, `Gaji`, and `NIDN` without the initial message from the superclass.
2. Is there a difference in the results in steps 2 and 3? Explain!
    - In Step 2, all constructors were correctly set up, and the values displayed correctly. If there was an error in Step 3 (for example, if the constructor signature didn't match), it would lead to compilation or runtime errors. The specific error in Step 3 depends on how the constructors were altered or invoked.
3. Why did the error occur in step 4?
    - An error in Step 4 could occur if:

    I tried to create an instance of Dosen without using the correct constructor.

    The Pegawai class was called without the necessary parameters (if you commented out the constructor and didn't provide a proper one).

    Common Error Causes:

    Calling a non-existent parameterless constructor.

    Not passing the required arguments for initialization.

    If Dosen was instantiated like new Dosen(), it would cause an error because the constructor requires four parameters, which were not provided.
4. What is the difference between super() called in steps 3 and 6?
    - In Step 3, using super() without parameters in an incorrect context (like calling a non-existent parameterless constructor) would lead to an error. In Step 6, the super(nip, nama, gaji) call correctly matches the constructor of the superclass Pegawai, leading to successful initialization and no errors.
5. Why did the error occur in step 5?
    - because the Super variable must be called first

10. ASSIGNMENT

1. Define a class that is a derivative of another class.
    - I have a parent class `Product` and two child classes: `Electronics` and `Clothing`.

2. Create 3 attributes in the parent class then add at least 1 attribute in the child class.

   Three in `Product` and one in each child class.

Product class :

```java
/**
 * Product
 */
public class Product {

    // Attributes
    private String productName;
    private double price;

    // Parameterless constructor
    public Product() {
        this.productName = "Unknown";
        this.price = 0.0;
    }

    // Parameterized constructor
    public Product(String productName, double price) {
        this.productName = productName;
        this.price = price;
    }

    // Method to display product info
    public void displayProduct() {
        System.out.println("Product Name: " + productName);
        System.out.println("Price: $" + price);
    }
}
```

Electronics class:

```java
/**
 * Electronics
 */
public class Electronics extends Product {
    // Additional attribute for Electronics
    private String brand; // Brand of the electronic product

    // Parameterless constructor
    public Electronics() {
        super(); // Call the parameterless constructor of Product
```

```
        this.brand = "Unknown";
    }

    // Parameterized constructor
    public Electronics(String productName, double price, String
brand) {
        super(productName, price);
        this.brand = brand;
    }

    // Method to display electronic info
    public void displayElectronic() {
        super.displayProduct(); // Call the parent class method
        System.out.println("Brand: " + brand);
    }
}
```

Clothing class:

```
public class Clothing extends Product {
    // Additional attribute for Clothing
    private String size; // Size of the clothing item

    // Parameterless constructor
    public Clothing() {
        super(); // Call the parameterless constructor of Product
        this.size = "Unknown";
    }

    // Parameterized constructor
    public Clothing(String productName, double price, String size) {
        super(productName, price); // Call the parameterized
constructor of Product
        this.size = size;
    }

    // Method to display clothing info
    public void displayClothing() {
        super.displayProduct(); // Call the parent class method
        System.out.println("Size: " + size);
    }
}
```

3. Perform the overloading method by creating 2 constructors, namely a parameterless 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.

Both the parent and child classes will have parameterless and parameterized constructors.

4. Implement the class diagram made in the theoretical PBO course

   The classes defined above closely resemble the structure outlined in your class diagram. The Product class contains the relevant attributes and methods, while the Electronics and Clothing classes inherit from it.

5. Create a Demo class then instantiate the child class object in the main function

Demo :

```
public class Demo {
    public static void main(String[] args) {
        // Instantiate an Electronics object using the parameterized
constructor
        Electronics laptop = new Electronics("Laptop", 999.99,
"Dell");
        System.out.println("Electronics Info:");
        laptop.displayElectronic();

        System.out.println();

        // Instantiate a Clothing object using the parameterized
constructor
        Clothing tShirt = new Clothing("T-Shirt", 19.99, "Medium");
        System.out.println("Clothing Info:");
        tShirt.displayClothing();
    }
}
```

Output :

```
Electronics Info:
Product Name: Laptop
Price: RP.200000.0
Brand: Dell

Clothing Info:
Product Name: T-Shirt
Price: RP.100000.0
Size: Medium
```

6. Try modifying the attribute values (both those declared in the child class and those inherited from the print info.

   I added setter for modify

In Electronics class

```
    public void setBrand(String brand) {
        this.brand = brand;
    }
```

In Clothing class

```
    public void setSize(String size) {
        this.size = size;
    }
```

Demo :

```
        // Modify attributes
        laptop.setBrand("HP"); // Change the brand
        System.out.println("\nModified Electronics Info:");
        laptop.displayElectronic();

        Clothing tShirt = new Clothing("T-Shirt", 19.99, "Medium");
        System.out.println("\nClothing Info:");
        tShirt.displayClothing();

        // Modify attributes
        tShirt.setSize("Large"); // Change the size
        System.out.println("\nModified Clothing Info:");
        tShirt.displayClothing();
```

--- happy working----