VERSION 1.6

25 February 2025

OBJECT ORIENTED PROGRAMMING

MODULE 2 – OOP (CLASS, OBJECT, METHOD, ATTRIBUTE)

COMPILED BY:

WIRA YUDHA AJI PRATAMA

KEN ARYO BIMANTORO

AUDITED BY:

Ir. Galih Wasis Wicaksono, S.Kom, M.Cs.

PRESENTED BY: TIM LAB. IT

UNIVERSITAS MUHAMMADIYAH MALANG

INTRODUCTION

OBJECTIVE

1. Students understand the basic concepts of Object Oriented Programming (OOP) in Java.
2. Students understand how to define and use classes, objects, methods, and attributes in Java programs.

MODULE TARGET

1. Students can create simple programs that implement classes, objects, methods, and attributes in Java.

INTRUDUCTION

1. Device (Laptop/PC)
2. IDE (IntelliJ)

KEYWORDS

OOP Java, Class, Methods, Object, Attribute

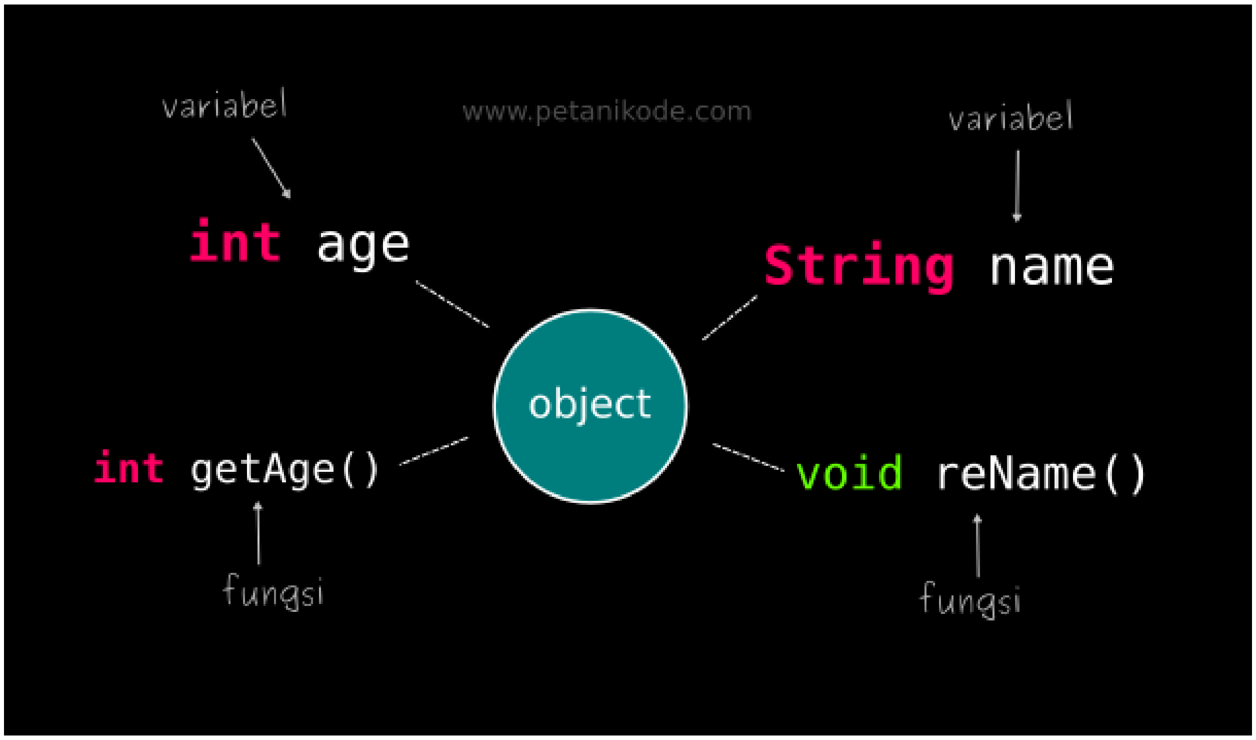
TABLE OF CONTENTS

OOP (Object Oriented Programming)

THEORY

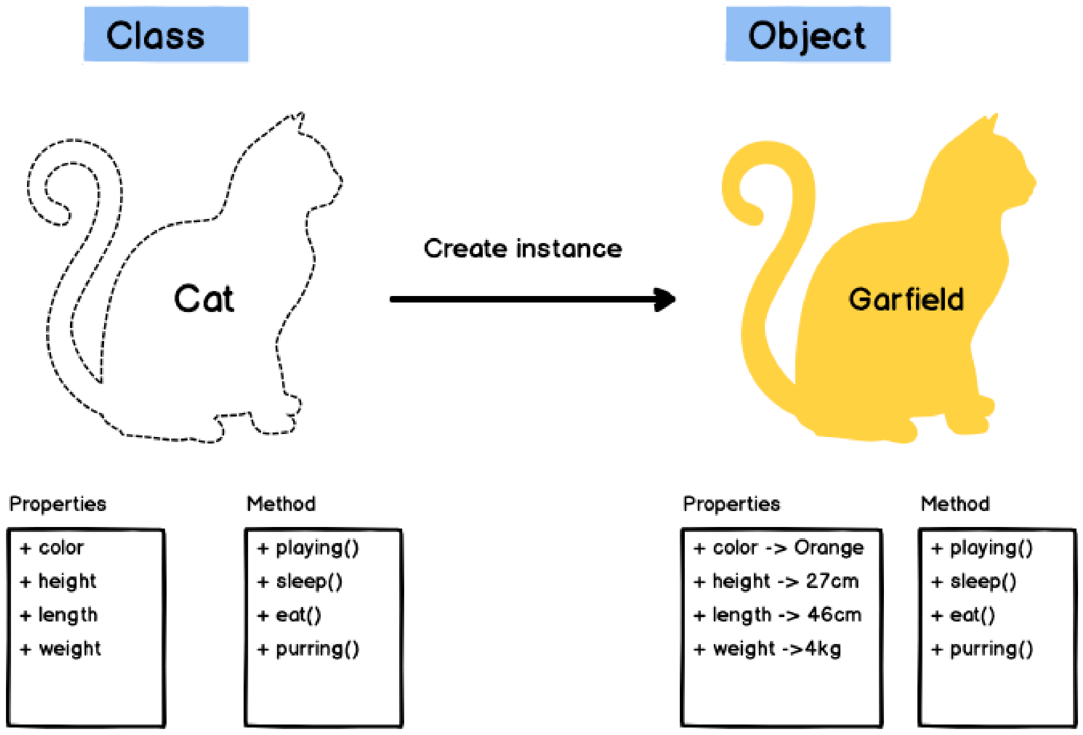
The concept of OOP (Object Oriented Programming)

Object-Oriented Programming (OOP) is a programming paradigm that focuses on objects. An object is an entity that combines related data and functions. OOP helps us create programs that are more structured, easy to understand, and easy to change. Existing programs are a combination of several small components that already exist. This can make it easier for a programmer to develop a program. Objects that are interrelated and arranged into one group are called classes. Later, these objects will interact with each other to solve complex program problems.

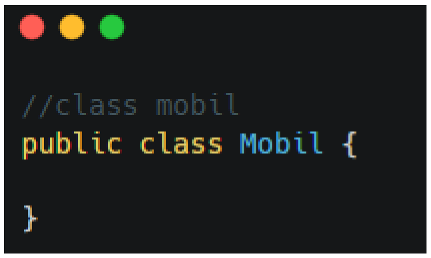


MATERIALS

Class



Class is a "blueprint" to create an object that defines the structure and behavior of an object in object oriented programming (OOP). Class actually serves to collect functions/methods and variables in one place. In the example here we liken the class to a car, which means that a car can have a color variable, name, speed and can also have a method to do something like move forward, backward, and turn. For example if we create a Car class:



```
//class mobil
public class Mobil {

}
```

Class actually contains variables called attributes or properties and functions called methods. A class is different from a function, a class does not require a () but directly in the class body which starts with ”{” and ends with ”}”.

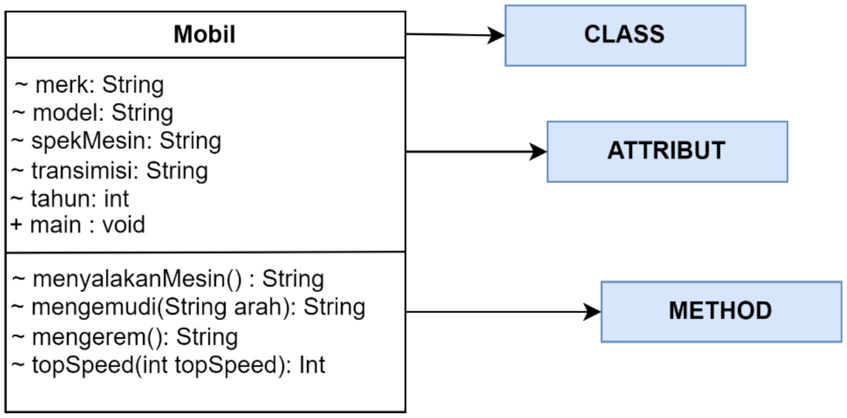
- Diagram class

  Classes can be likened to molds for creating objects. These molds define the characteristics and capabilities that all objects created have. A class diagram is a drawing that shows the structure and relationships between classes. Like a house plan, this diagram helps us understand how the various parts of the system are connected.

  Important parts of a class diagram:

  1. Name (and stereotype): Identity and type of class
  2. Attribute: characteristics possessed by the class
  3. Method: capabilities possessed by the class

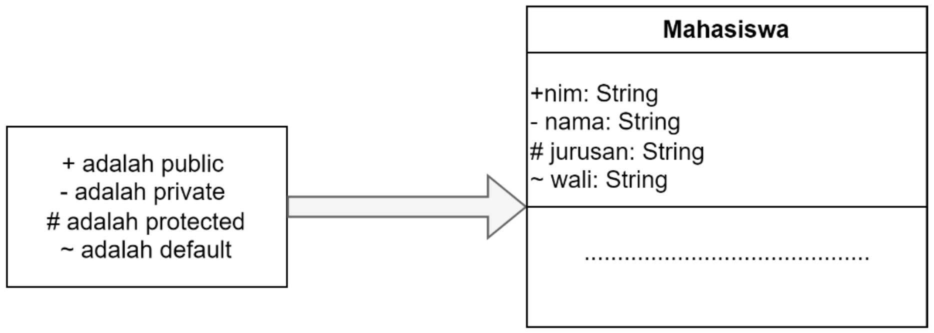
  Contoh:



  Explanation:

  - The data type after the method name is the return type of the function/method
  - Access modifier. The function of the access modifier in Java is to limit the scope of a class, constructor, attribute, method, or other data member contained in a Java class. More about modifiers will be explained in the next module.

  Example:



- Relationship between classes

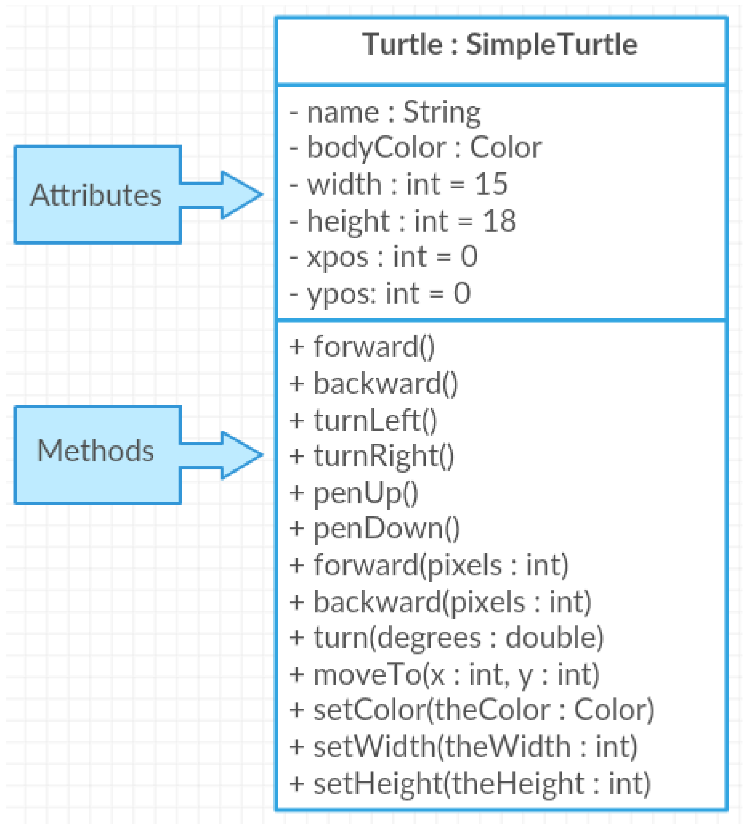
  Association, which is a static relationship between classes that usually describes a class that has attributes in the form of another class. There are several types of associations, such as:

  1. Simple Association: Simple association form (―――――)
  2. Aggregation is a relationship that states parts, usually the relationship between master data and its details. For example, one purchase consists of a number of items (◆―――――).
  3. Navigability: shows the direction of queries/communication between objects, can be one or two directions, seen by the arrow (―――――>)
  4. Mixture / Composite : mixture of associations (◆―――――>)

  Benefits of class diagrams:

  1. Understanding system structure: Class diagrams help us see the big picture of the system and how its various parts are interconnected.
  2. Improve communication: Class diagrams help development teams to communicate more effectively about system design.
  3. Facilitates development: Class diagrams help developers to create more structured and easy-to-understand code.

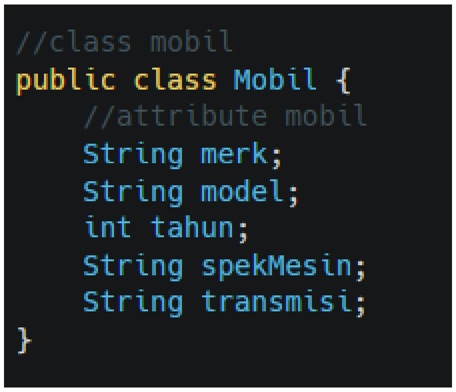
Attribute



Attributes or Properties are the identity of data or information from the object class that we have created, or in the previous C language it was called a variable. There are several things related to attributes, namely:

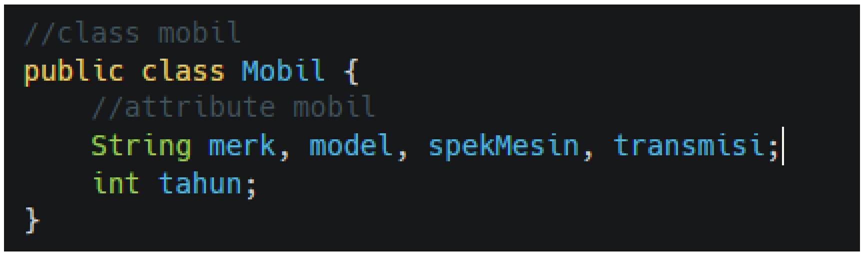
1. Attributes are data or properties owned by a class.
2. Attributes can be variables such as name, age, color, and so on.
3. Attributes define the state of an object.

For example, a car has specifications such as brand, year, speed, color. Then we can write in the Car class like this:



```java
//class mobil
public class Mobil {
    //attribute mobil
    String merk;
    String model;
    int tahun;
    String spekMesin;
    String transmisi;
}
```

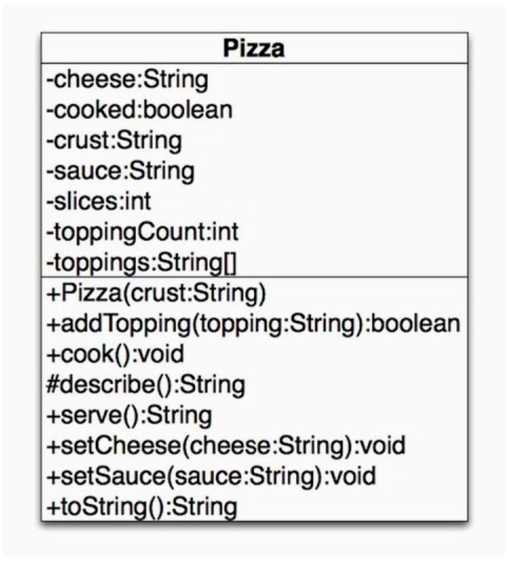
Or you can also do it like the image below, this can also be done to reduce code repetition:



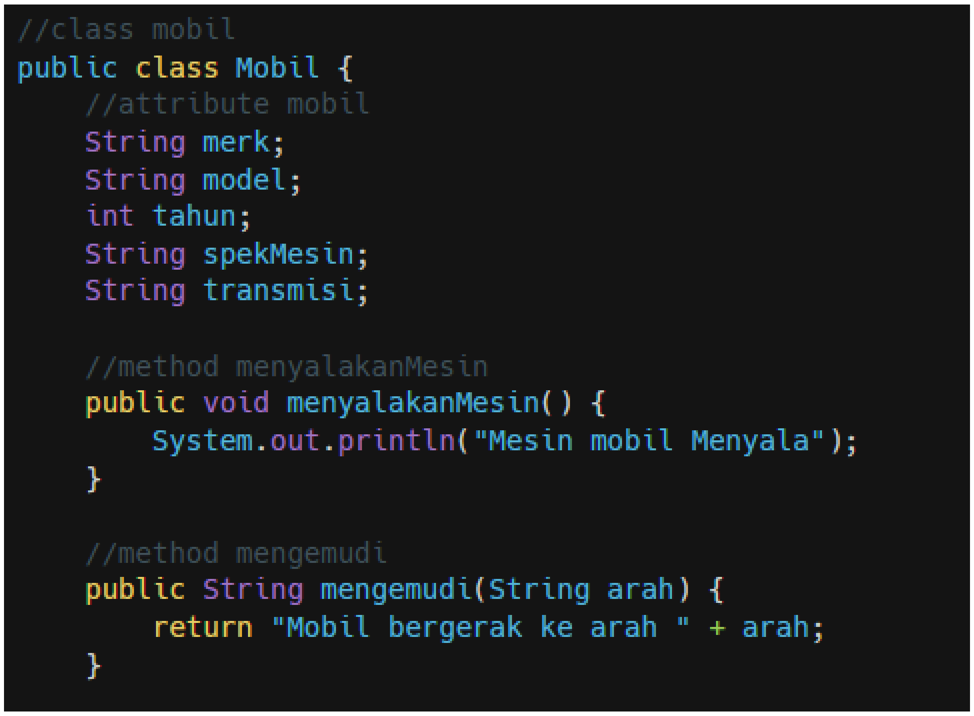
```java
//class mobil
public class Mobil {
    //attribute mobil
    String merk, model, spekMesin, transmisi;
    int tahun;
}
```

Method

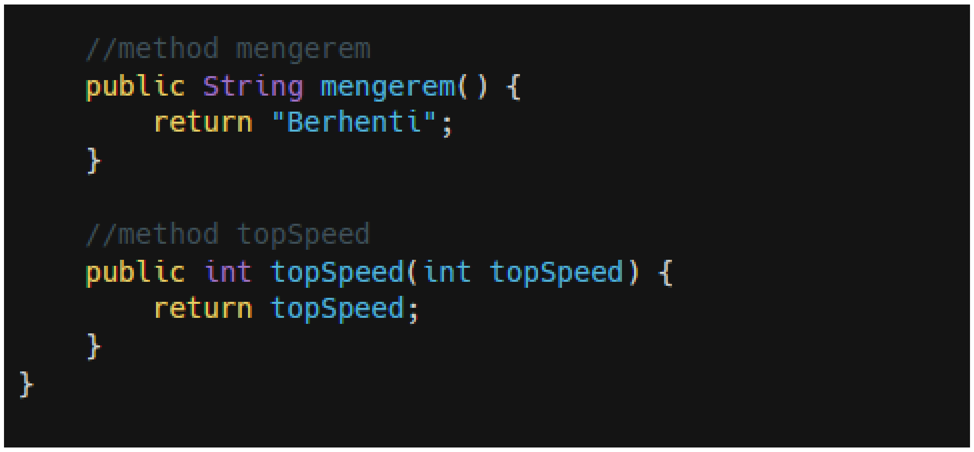
By looking at the illustration in the previous sub-chapter about attributes, can you say which is the method/behavior in the image below?



Method or can also be called behavior is an action or behavior that can be done by an object that is an instance of a class. Method represents how an object works or how it can run or operate. Example:



```java
//class mobil
public class Mobil {
    //attribute mobil
    String merk;
    String model;
    int tahun;
    String spekMesin;
    String transmisi;

    //method menyalakanMesin
    public void menyalakanMesin() {
        System.out.println("Mesin mobil Menyala");
    }

    //method mengemudi
    public String mengemudi(String arah) {
        return "Mobil bergerak ke arah " + arah;
    }
```



```
    //method mengerem
    public String mengerem() {
        return "Berhenti";
    }

    //method topSpeed
    public int topSpeed(int topSpeed) {
        return topSpeed;
    }
}
```

Actually, method is the same as function (in C language) that can do something. For the code example above, method is something that can be done by the car object instance later, here is the explanation:

- turn onEngine() : is a method that creates an instance that can later turn on the engine
- drive() : moves the car forward, backward, and also turns
- brake() : stops the car

Method characteristics:

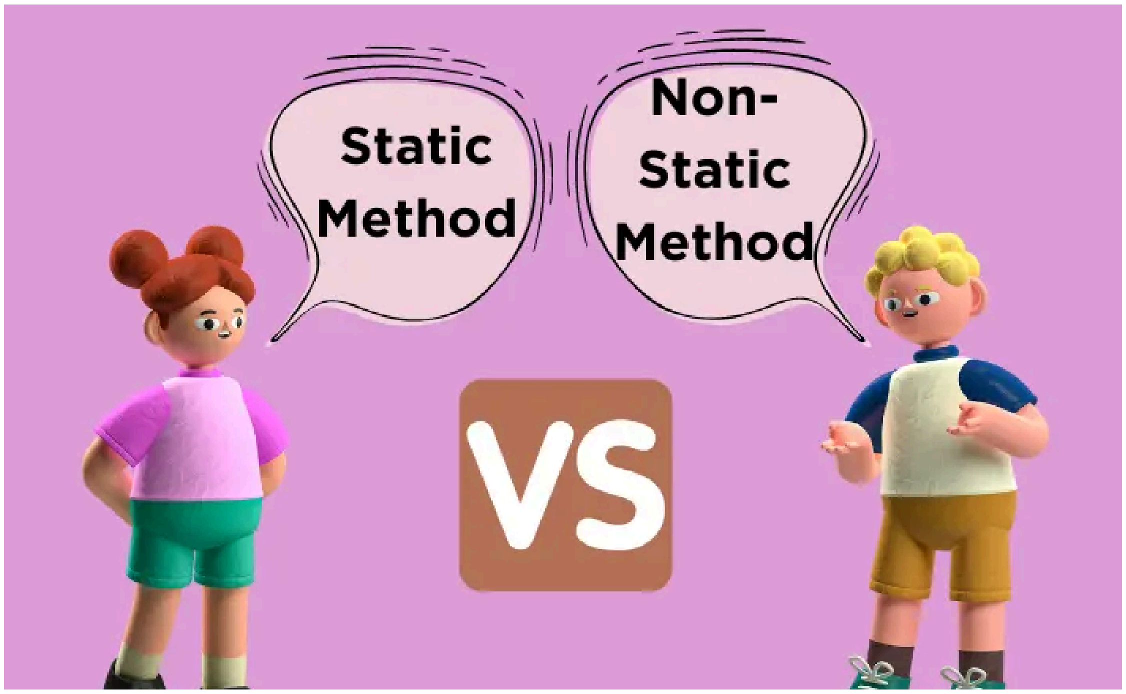
- Name : the method name is the same as the function name that can be used to do something. In the example here, a car can do drive(), brake(), and turn onEngine().
- Parameter : a parameter is data that is input when a method is called. In the example here, it is in the drive(String direction) method, where the String direction is a parameter.
- Return type: the return type determines the data produced by a method. In the example here, it is the topSpeed(int topSpeed) method, where this method returns an integer value.
- Body: the body contains the code that will be run by a method. The body of a method begins with "{" and ends with "}".

Benefits of using methods:

- Increase code reusability. For example, the drive() function can be used repeatedly to drive a car.
- Increase code readability. The code will be better structured, making the code easier to read and understand.

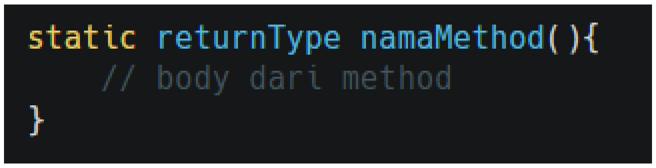
- ○ Helps divide the program into smaller parts. This makes the program easier to manage and change.
- ○ Increase maintainability. Programs that use methods will be easier to repair and maintain in the future.

- **Static and Non-static methods**



Static method is a method that can be executed or called without having to create an object instance. While non-static method is a method that must create an object instance to use or call it, where this method will be closely related to the OOP concept.

The way to declare a static method in Java is by adding the static keyword. For example like this:



```
static returnType namaMethod( ){
    // body dari method
}
```

An example of the implementation of the static method, we will try to change the braking method with static to be like this:

```java
static String mengerem() {
    return "Berhenti";
}
```

The way to call it is like this in the main method:

```java
public static void main(String[] args) {
    // tidak perlu membuat instance objek mobil
    System.out.println(mengerem());
}
```

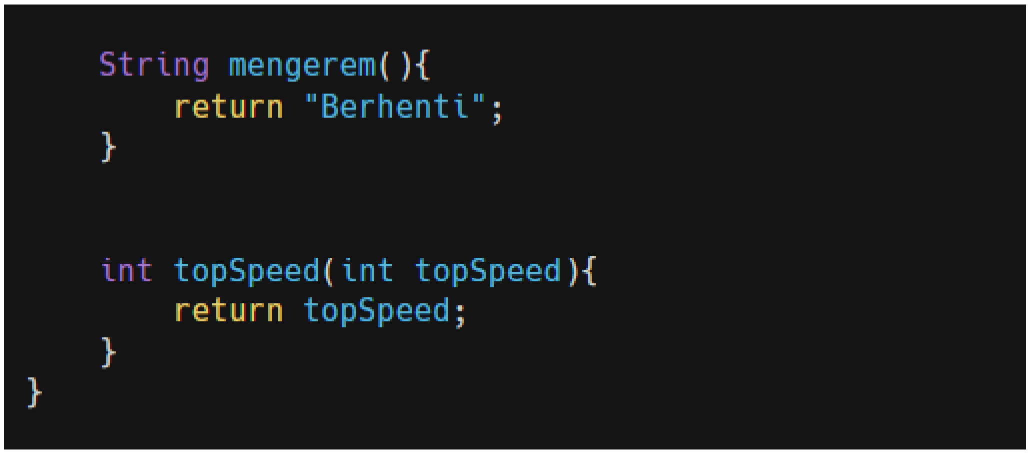
In the code above, we can directly call or use the brake() method without creating an object instance of the Car class.

For the process of declaring a static method, in the material above we have actually created a non-static method. To create a static method, it is declared without using the static keyword like this:

```java
returnType namaMethod(){
    // body method
}
```

For the implementation, you can check again on the Car class that we created previously like this:

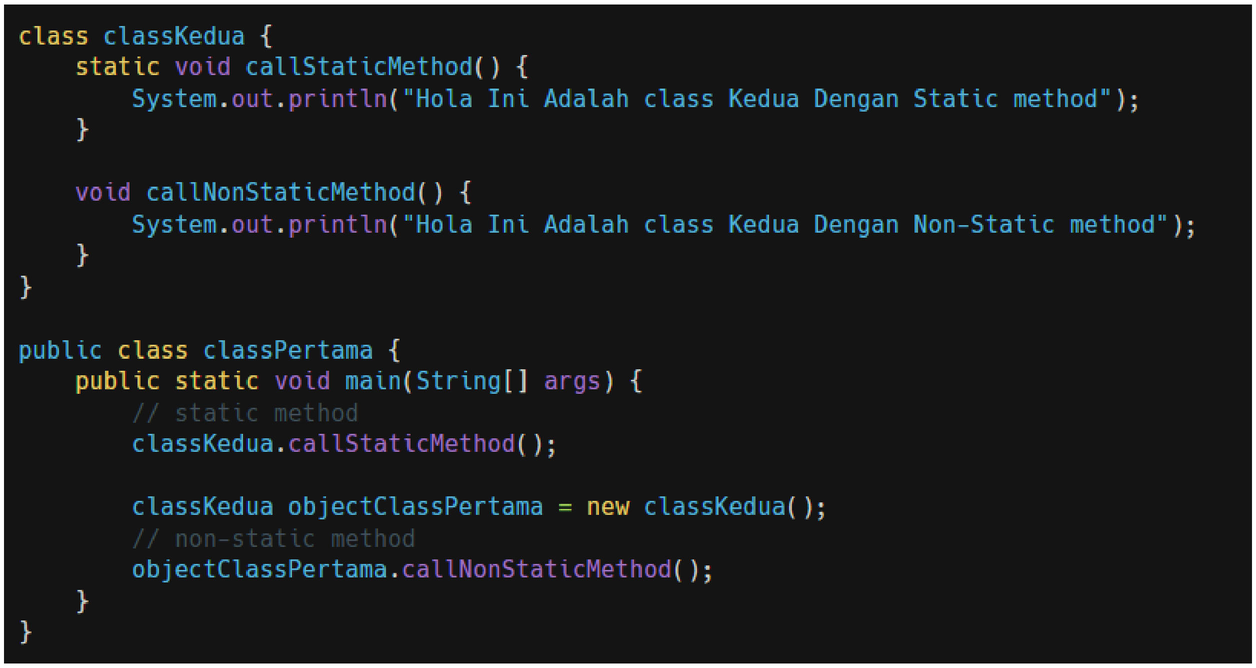
```java
public class Mobil{
    String merk;
    String model;


    void menyalakanMesin(){
        System.out.println("Mesin mobil menyala");
    }


    String mengemudi(String arah){
        return "Mobil bergerak ke arah" + arah;
    }
```



```java
    String mengerem(){
        return "Berhenti";
    }


    int topSpeed(int topSpeed){
        return topSpeed;
    }
}
```

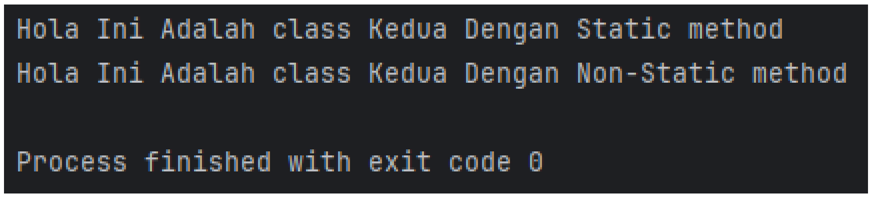
In the Car class code above, all methods are non-static. The main thing about non-static methods is that they are called using '.' after the object name.

Note for static methods, they can be called directly with the method name as long as it is still in the scope of the class (one file) and if from another class we can call it with the class name then the method name. For example:



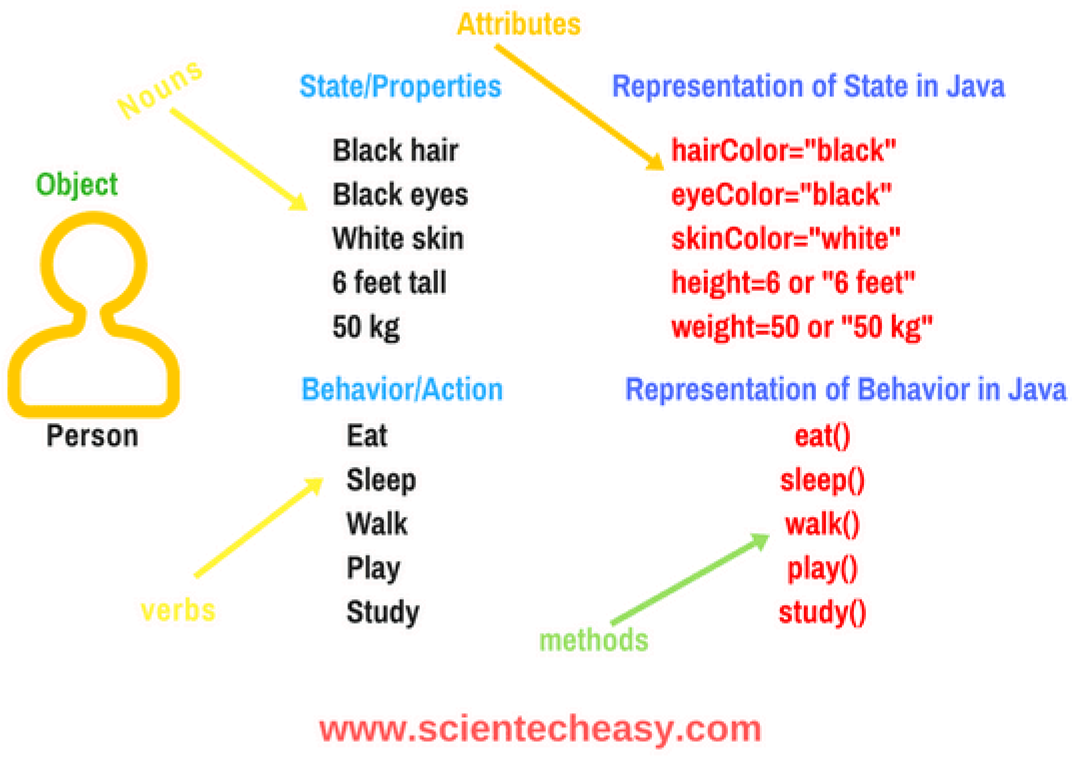
```java
class classKedua {
    static void callStaticMethod() {
        System.out.println("Hola Ini Adalah class Kedua Dengan Static method");
    }

    void callNonStaticMethod() {
        System.out.println("Hola Ini Adalah class Kedua Dengan Non-Static method");
    }
}

public class classPertama {
    public static void main(String[] args) {
        // static method
        classKedua.callStaticMethod();

        classKedua objectClassPertama = new classKedua();
        // non-static method
        objectClassPertama.callNonStaticMethod();
    }
}
```

Output program:

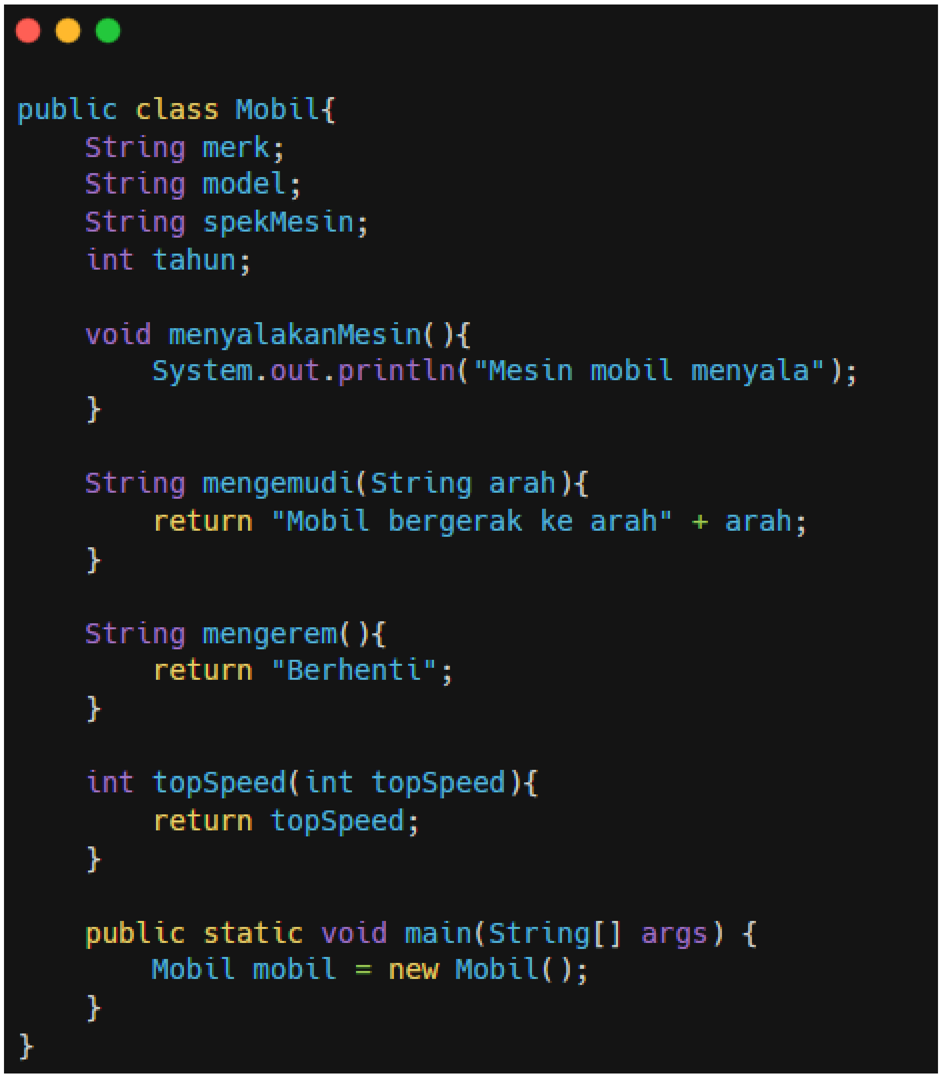


```
Hola Ini Adalah class Kedua Dengan Static method
Hola Ini Adalah class Kedua Dengan Non-Static method

Process finished with exit code 0
```

Object

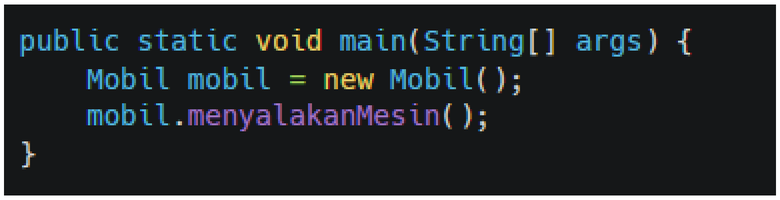


Objects are the result of a declaration or instance of a class. In objects, we can access and manipulate the contents of a class that we have previously created. For example, if we have a Car class before, then we can create an object of that class in the main method like this:



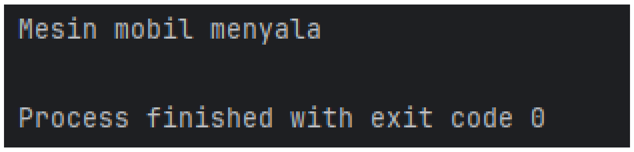
```java
public class Mobil{
    String merk;
    String model;
    String spekMesin;
    int tahun;

    void menyalakanMesin(){
        System.out.println("Mesin mobil menyala");
    }

    String mengemudi(String arah){
        return "Mobil bergerak ke arah" + arah;
    }

    String mengerem(){
        return "Berhenti";
    }

    int topSpeed(int topSpeed){
        return topSpeed;
    }

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

When we run the program, nothing will happen, because we only create an object instance from the Car class with the object name car. After we create an object, we can call the method that has been created in this way in the main method:



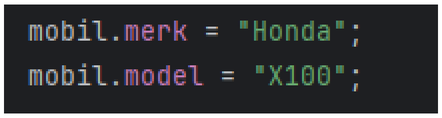
```java
public static void main(String[] args) {
    Mobil mobil = new Mobil();
    mobil.menyalakanMesin();
}
```

Then the program output will be like this:



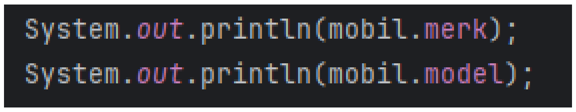
```
Mesin mobil menyala

Process finished with exit code 0
```

In addition, if we change the contents of the attributes in the Car class like this:



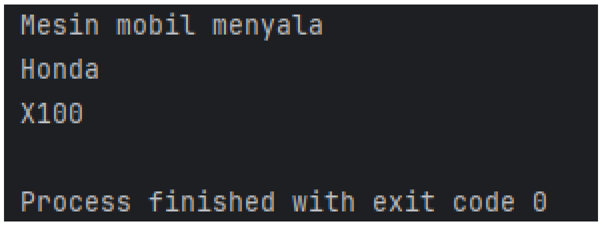
```
mobil.merk = "Honda";
mobil.model = "X100";
```

So we can also call the contents of each attribute in this way:



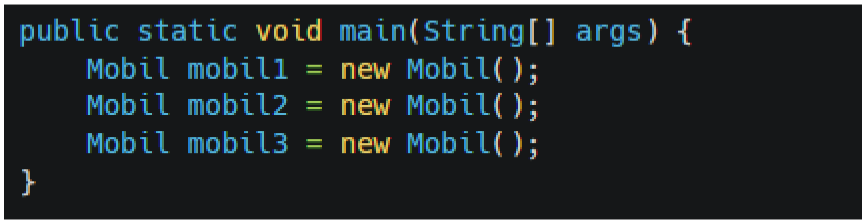
```
System.out.println(mobil.merk);
System.out.println(mobil.model);
```

Output of the program when run:



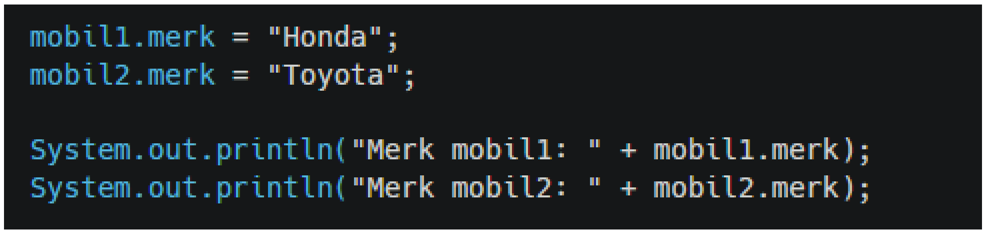
```
Mesin mobil menyala
Honda
X100

Process finished with exit code 0
```

When we create a class, can we only create 1 object instance? The answer is no. We can create as many object instances as we need, for example here we will try to create 3 object instances from the Car class, then we can write the code like this in the main method:



```
public static void main(String[] args) {
    Mobil mobil1 = new Mobil();
    Mobil mobil2 = new Mobil();
    Mobil mobil3 = new Mobil();
}
```

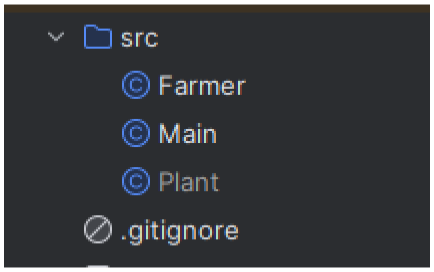
Each object instance is separate from each other and different, meaning that the car1 object stands alone as do the car2 and car3 objects. The way to call methods and attributes on each object is the same as before.



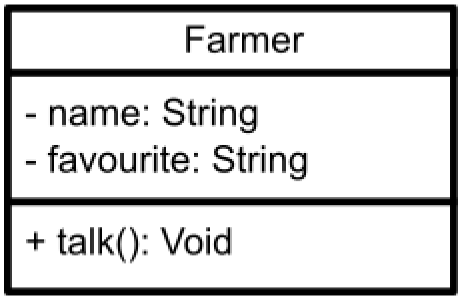
```
mobil1.merk = "Honda";
mobil2.merk = "Toyota";

System.out.println("Merk mobil1: " + mobil1.merk);
System.out.println("Merk mobil2: " + mobil2.merk);
```

PRACTICE

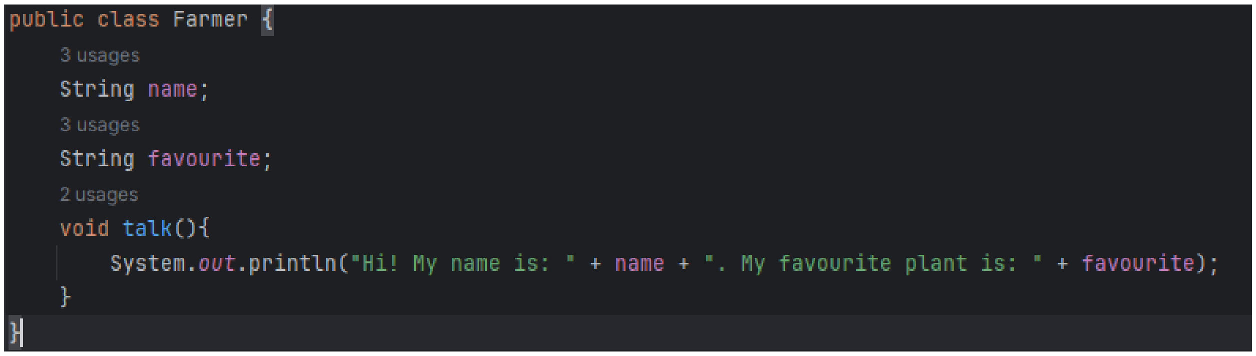
Let's do an experiment by continuing our practice in yesterday's module 1! Please open the project file from the previous module 1 practice that we have created. Then please create a new class with the name Plant and Farmer. Forgot how to create a class? Please go back to module 1 and read it again 😀. If it is certain it will look like this:



The Farmer class has the following specifications:

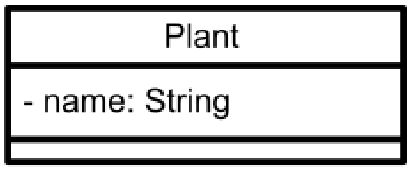


Or you can see it here:

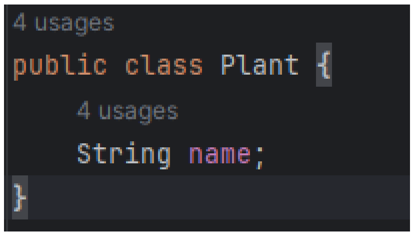


```
public class Farmer {
    3 usages
    String name;
    3 usages
    String favourite;
    2 usages
    void talk(){
        System.out.println("Hi! My name is: " + name + ". My favourite plant is: " + favourite);
    }
}
```

Meanwhile, the Plant class has the following specifications:

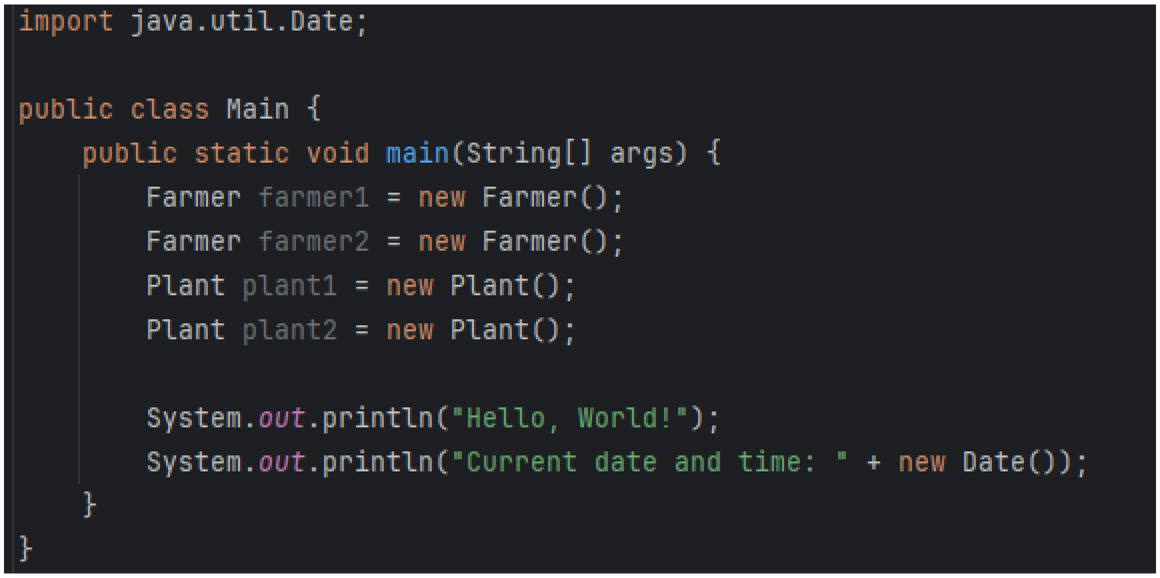


Or like this:



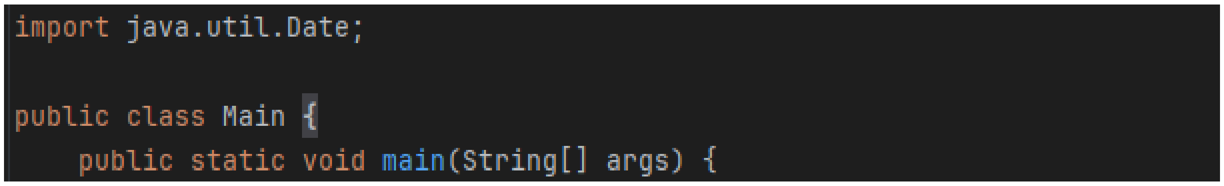
```
4 usages
public class Plant {
    4 usages
    String name;
}
```

Then in the main class, please create 2 objects from the Farmer and Plant classes like this:

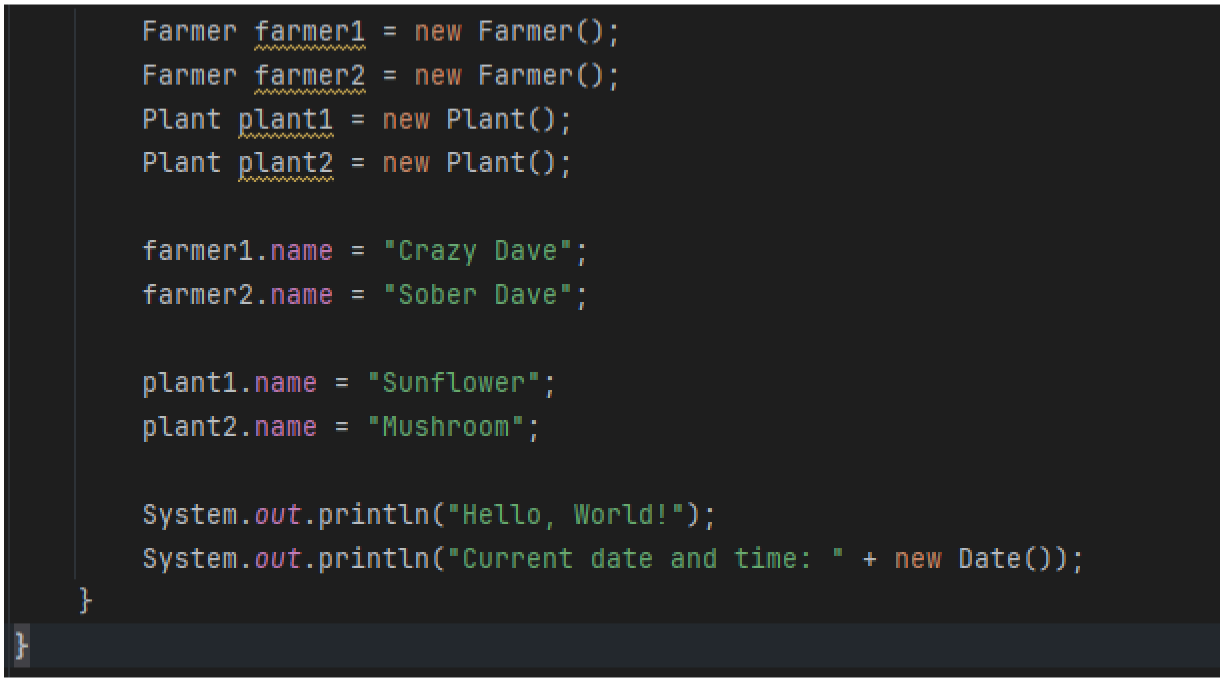


```
import java.util.Date;

public class Main {
    public static void main(String[] args) {
        Farmer farmer1 = new Farmer();
        Farmer farmer2 = new Farmer();
        Plant plant1 = new Plant();
        Plant plant2 = new Plant();

        System.out.println("Hello, World!");
        System.out.println("Current date and time: " + new Date());
    }
}
```

Then assign names to the four objects like this:

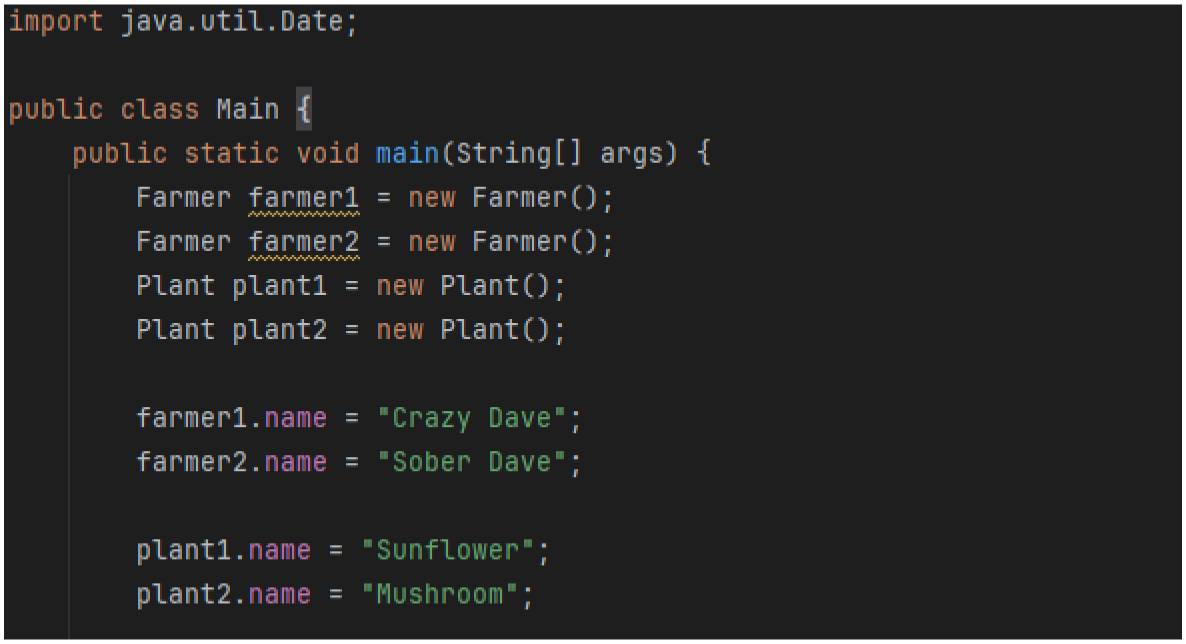


```
import java.util.Date;

public class Main {
    public static void main(String[] args) {
```

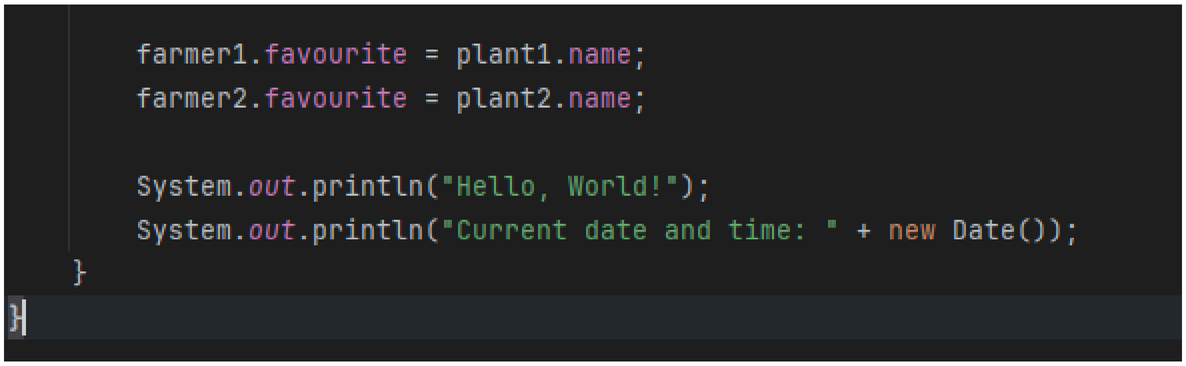


```java
        Farmer farmer1 = new Farmer();
        Farmer farmer2 = new Farmer();
        Plant plant1 = new Plant();
        Plant plant2 = new Plant();

        farmer1.name = "Crazy Dave";
        farmer2.name = "Sober Dave";

        plant1.name = "Sunflower";
        plant2.name = "Mushroom";

        System.out.println("Hello, World!");
        System.out.println("Current date and time: " + new Date());
    }
}
```

Don't forget to add a value to the Favourite variable in both Farmer objects like this:

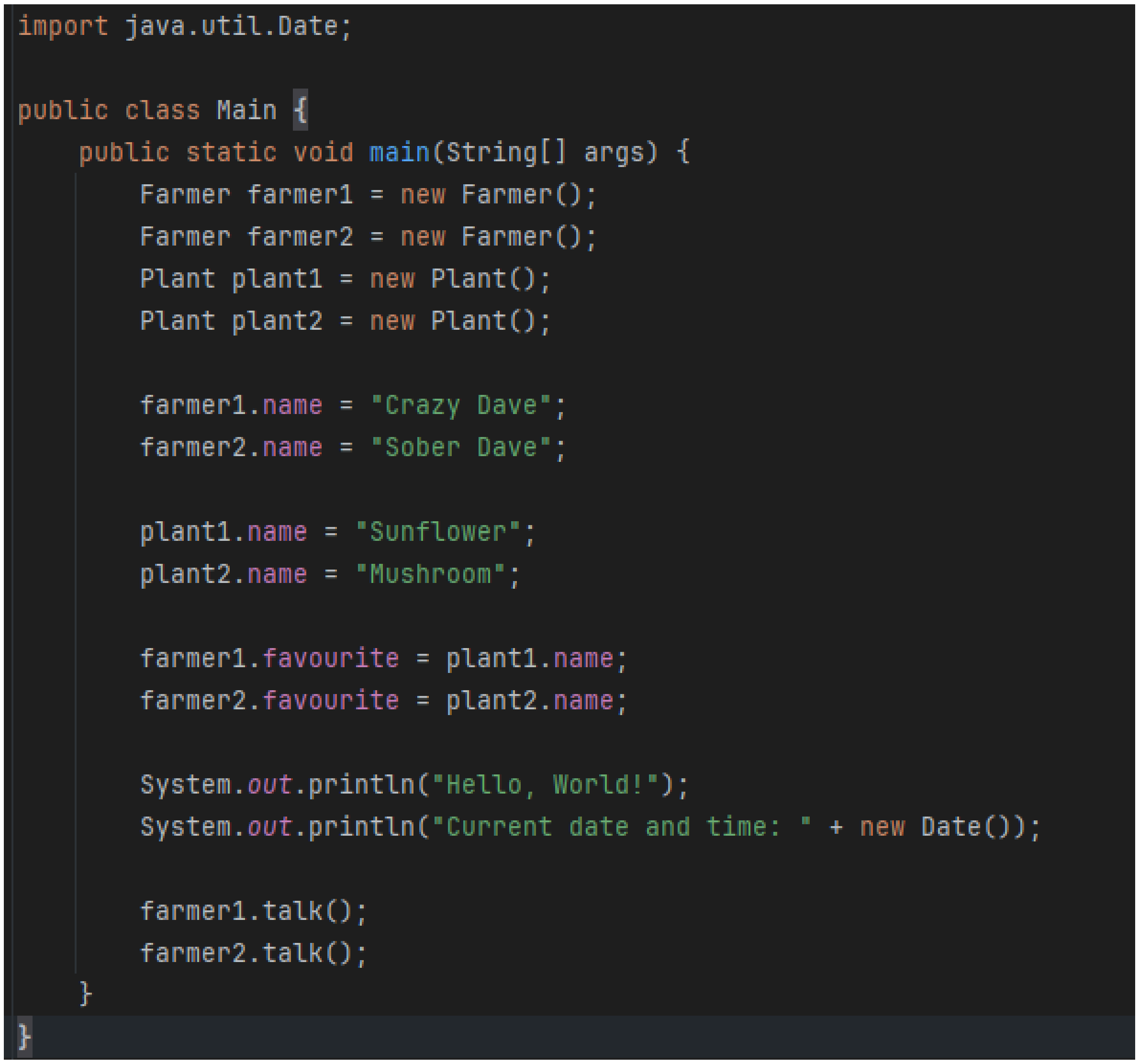


```java
import java.util.Date;

public class Main {
    public static void main(String[] args) {
        Farmer farmer1 = new Farmer();
        Farmer farmer2 = new Farmer();
        Plant plant1 = new Plant();
        Plant plant2 = new Plant();

        farmer1.name = "Crazy Dave";
        farmer2.name = "Sober Dave";

        plant1.name = "Sunflower";
        plant2.name = "Mushroom";
```



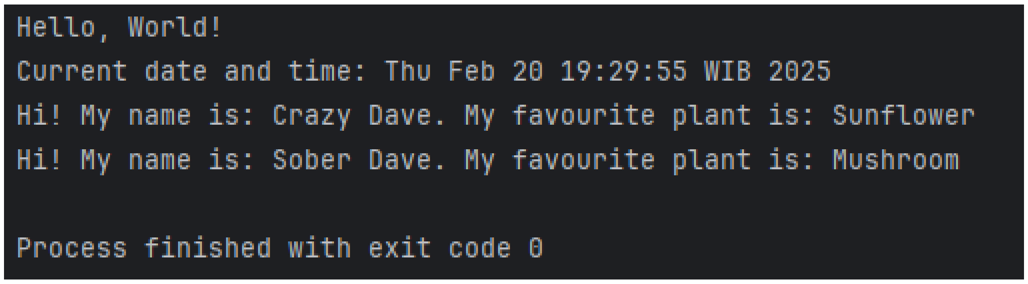
```java
        farmer1.favourite = plant1.name;
        farmer2.favourite = plant2.name;

        System.out.println("Hello, World!");
        System.out.println("Current date and time: " + new Date());
    }
}
```

What else is missing? Oh yeah, the method in the Farmer class hasn't been called yet. Please call the method on the two objects of the Farmer class like this.



```java
import java.util.Date;

public class Main {
    public static void main(String[] args) {
        Farmer farmer1 = new Farmer();
        Farmer farmer2 = new Farmer();
        Plant plant1 = new Plant();
        Plant plant2 = new Plant();

        farmer1.name = "Crazy Dave";
        farmer2.name = "Sober Dave";

        plant1.name = "Sunflower";
        plant2.name = "Mushroom";

        farmer1.favourite = plant1.name;
        farmer2.favourite = plant2.name;

        System.out.println("Hello, World!");
        System.out.println("Current date and time: " + new Date());

        farmer1.talk();
        farmer2.talk();
    }
}
```

Then run the program



```
Hello, World!
Current date and time: Thu Feb 20 19:29:55 WIB 2025
Hi! My name is: Crazy Dave. My favourite plant is: Sunflower
Hi! My name is: Sober Dave. My favourite plant is: Mushroom

Process finished with exit code 0
```

TIPS

Short video introduction to PBO:

Video

Short video on class and object concepts

Video

Short video on properties and methods:

Video

CODELAB & TASK

CODELAB 1

Create a program in Java that consists of two classes, namely Animal and Main.

1. The Animal class must have three attributes with the String data type, namely:
   - Name
   - Type
   - Sound
2. The Animal class must also have a displayInfo() method that prints animal information such as Name, Type, and Sound.
3. The Main class must have a method main(String[] args), where:
   - Created two Animal objects, namely animal 1 and animal 2.
   - animal 1 has the following attributes:
     Name = "Cat"

     Type = "Mammal"

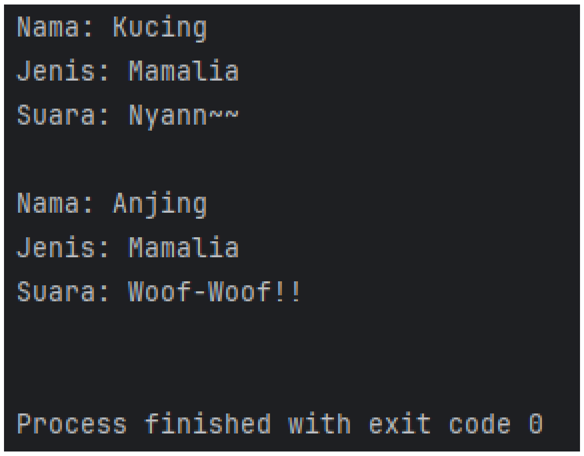
Sound = "Nyann~~"

- animal has attributes:
  name = "Dog"

  Type = "Mammal"

  Sound = "Woof-Woof!!"
- Call the showInfo() method of both objects.

4. Example of expected output:



```
Nama: Kucing
Jenis: Mamalia
Suara: Nyann~~

Nama: Anjing
Jenis: Mamalia
Suara: Woof-Woof!!


Process finished with exit code 0
```

CODELAB 2

Create a program in Java that consists of two classes, namely BankAccount and Main.

Bank Account Class Terms:

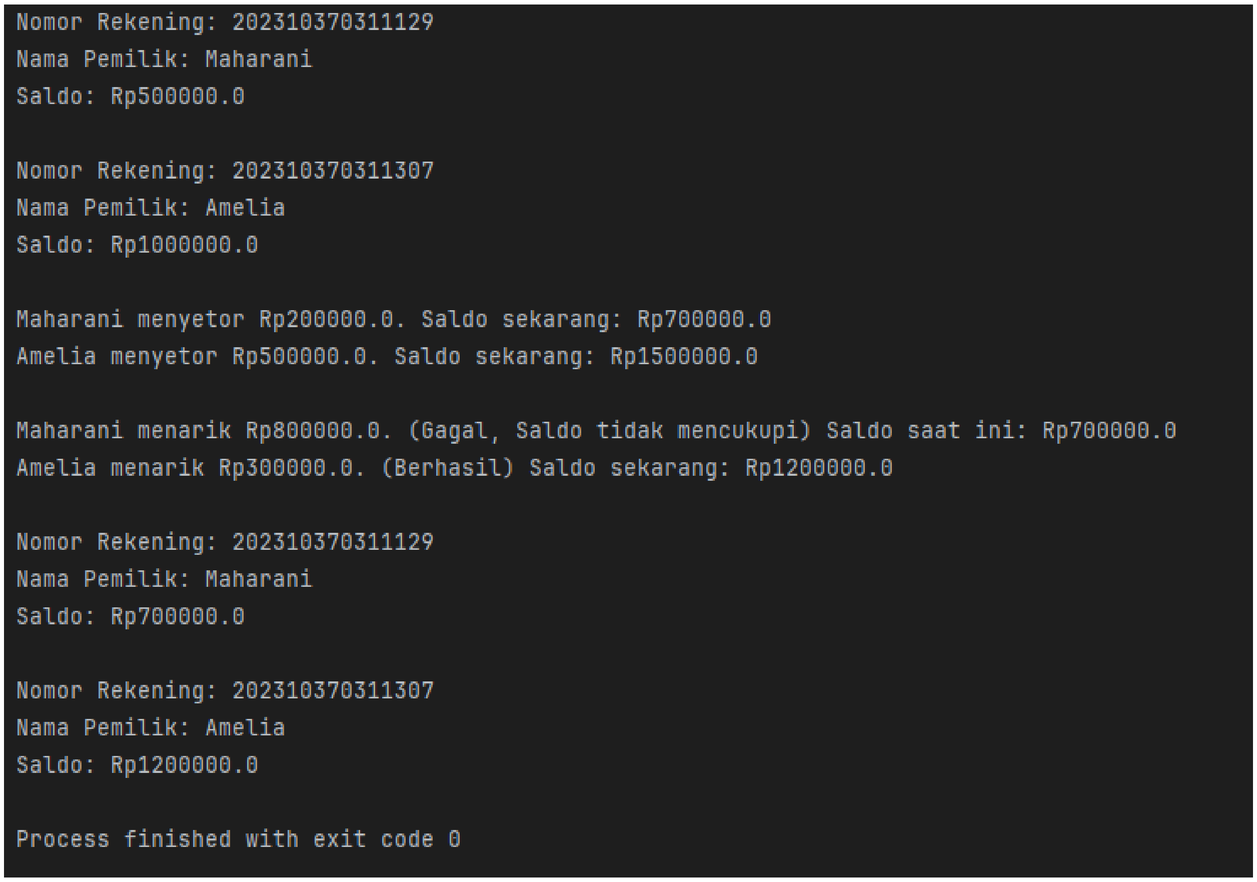
1. Three attributes as follows:
   - accountNumber (String type) → Customer account number
   - ownerName (String type) → Account owner name
   - balance (double type) → Balance in the account
2. Three methods as follows:
   - displayInfo() → Displays account information.
   - depositMoney(double amount) → Adds an amount to the balance, and displays transaction information.
   - withdrawMoney(double amount) → Subtracts an amount from the balance if the balance is sufficient, and displays transaction information.

Main Class Requirements

1. This class must have a main(String[] args) method
2. Create two Bank Account objects, namely account1 and account2.
3. account1 has the attributes:
   accountnumber = "Your NIM"

   ownername = "Your name"
   balance = "up to you, according to your ATM balance is also okay (must be a positive number)"
4. account2 has the attributes:
   accountnumber = "Your friend's NIM"

   ownername = "Your friend's name"
   balance = "up to you"
5. Then call the method on both objects. Example of expected output:



```
Nomor Rekening: 202310370311129
Nama Pemilik: Maharani
Saldo: Rp500000.0

Nomor Rekening: 202310370311307
Nama Pemilik: Amelia
Saldo: Rp1000000.0

Maharani menyetor Rp200000.0. Saldo sekarang: Rp700000.0
Amelia menyetor Rp500000.0. Saldo sekarang: Rp1500000.0

Maharani menarik Rp800000.0. (Gagal, Saldo tidak mencukupi) Saldo saat ini: Rp700000.0
Amelia menarik Rp300000.0. (Berhasil) Saldo sekarang: Rp1200000.0

Nomor Rekening: 202310370311129
Nama Pemilik: Maharani
Saldo: Rp700000.0

Nomor Rekening: 202310370311307
Nama Pemilik: Amelia
Saldo: Rp1200000.0

Process finished with exit code 0
```

TASK

In this assignment, the login system that was previously developed in one file, is now changed to an object-based system with better class separation.

Program Structure

- **Admin Class**
  - Contains the username and password attributes that have been defined in the previous module.
  - Has a login() method that verifies user input with valid admin data.
- **Student Class**
  - Contains the name and student ID attributes that have been defined in the previous module.
  - Has a login() method that verifies user input with valid student data.
  - Has a displayInfo() method to display student information after successful login.
- **LoginSystem Class (main class)**
  - Contains the main logic of the program, including input from the user.
  - Creates Admin and Student objects to access their respective login methods.
  - Provides a menu for users to choose to login as Admin or Student.
  - Uses methods from Admin and Student objects to verify login.

How the Program Works

- The program asks the user to choose the login type: Admin or Student.
- If you choose Admin:
  - The program asks for username and password input.
  - If it matches the admin data, the login is successful.
  - If it does not match, an error message appears.
- If you choose Student:
  - The program asks for name and student ID input.
  - If it matches the student data, the login is successful and the student information is displayed.

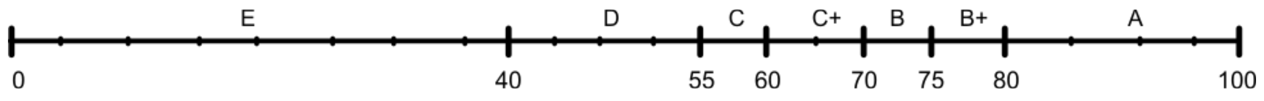
  - If it does not match, an error message appears.
- If the selected input is invalid, the program displays an error message.

EVALUATION

ASSESSMENT RUBRIC

Assessment Aspects	**Poin**
CODELAB 1	**Total 20%**
Code neatness	5%
Code accuracy & output	5%
Code creativity	5%
Original code (not copied)	5%
CODELAB 2	**Total 20%**
Code neatness	5%
Code accuracy & output	5%
Code creativity	5%
Original code (not copied)	5%
TASK	**Total 60%**
Code neatness	5%
Code accuracy & output	10%
Original code (not copied)	5%
Ability to explain	20%
Answer questions	20%
TOTAL	**100%**

ASSESSMENT SCALE



A = (81 - 100) → Excellent

B+ = (75 - 80) → Very Good

B = (70 - 74) → Good

C+ = (60 - 69) → Fairly Good

C = (55 - 59) → Fair

D = (41 - 54) → Poor

E = (0 - 40) → Bro really...

END MODULE SUMMARY

Okay, how is module 2? Still feels easy? Actually, if you pay attention, the tasks in modules 1 and 2 produce the same output. However, there is something different. What do you think? Right, the source code. In module 2, you are asked to apply the concept of OOP.

Maybe you will ask: *"Bro, why use a difficult way if there is an easier way and produces the same output?"* Well, your question will be answered later in Module 6.

Don't worry if you find this concept difficult to learn. The more you practice, the more visible the pattern. Like learning to ride a bicycle, no matter how many books you read about how to ride a bicycle, it will be useless if you don't start pedaling.

If you are confused, don't hesitate to ask. That's all thanks.