VERSION 1.6

20 February 2025

OBJECT ORIENTED PROGRAMMING

MODULE 1 – JAVA BASIC, JAVA API, GIT & GITHUB

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

OBJECTIVES

1. Students understand the basics of Java syntax (input/output, variables, data types, branching, loops).
2. Students understand the concept of API and its usage in Java programs.
3. Students understand the basic concept of version control with Git and working with GitHub in project collaboration.

MODULE TARGETS

1. Students can create simple programs that apply input/output, branching, and API in Java.
2. Students can perform commit, push, pull, and understand the basics of branching in GitHub.

PREPARATION

1. Device (Laptop/PC)
2. IDE (IntelliJ)
3. Internet
4. Web Browser
5. GitBash
6. GitHub Account

KEYWORDS

IDE, IntelliJ, JDK, GIT, GitBash

TABLE OF CONTENTS

JAVA BASIC

THEORY

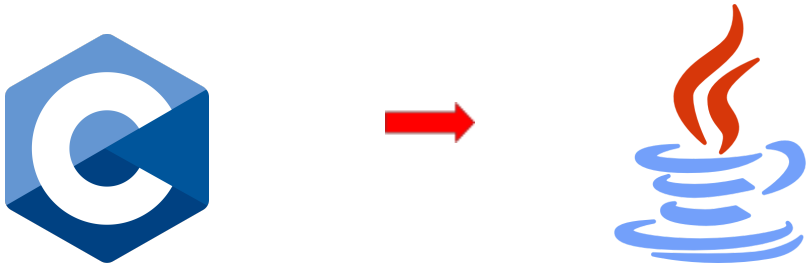
What Is Java?

- General Understanding

  Java is an Object-Oriented Programming (OOP) language, meaning Java is a collection of objects that communicate through method calls. Java is known for its flexibility and popularity. The language is widely used to develop various applications, from desktop and mobile applications to websites, large enterprise applications, and big data technology. Java's syntax is based on C and C++.

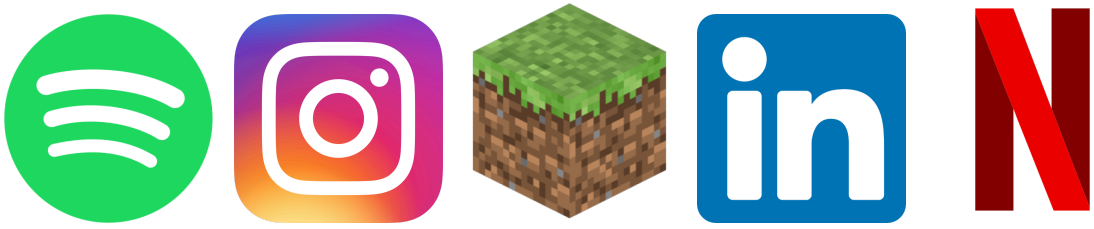
  Java is a case-sensitive programming language, meaning that uppercase and lowercase letters are distinguished. It's important to remember this when writing Java code, as capitalization errors can cause issues.

- Differences between Java and C



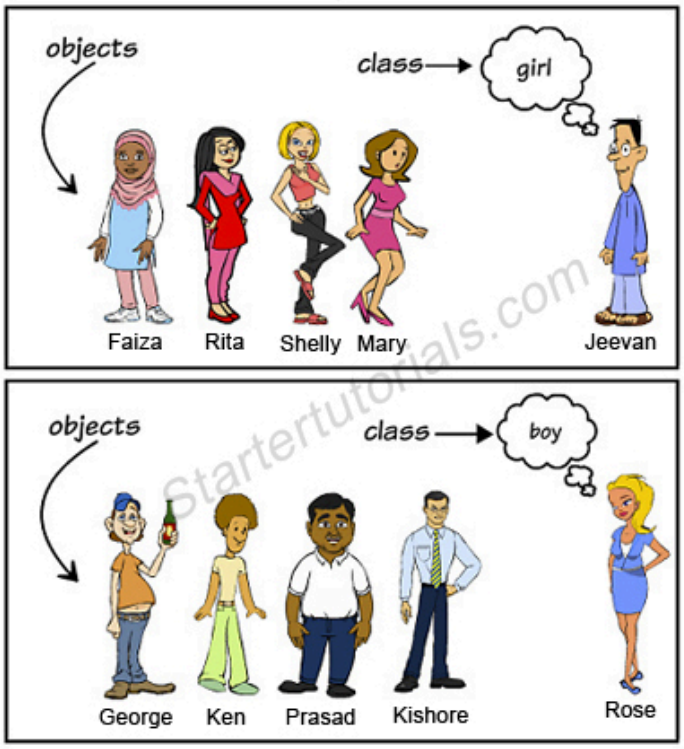
  Java and C are programming languages with different approaches. C is a procedural language that is more detailed and often used for low-level programming, such as operating systems and embedded systems. Java, on the other hand, is an object-oriented language that is easier to learn and understand because it runs on a platform using the Java Virtual Machine (JVM). Additionally, C is more machine-dependent, while Java is more flexible and not machine-dependent because it runs on the JVM.

PROGRAMS THAT USE JAVA



- **Android Apps** - Java is the primary language for Android app development before Kotlin became popular. Many famous apps like Instagram, Spotify, and Twitter were *initially* developed using Java.
- **Minecraft** - One of the world's most popular sandbox games, Minecraft, was developed using Java. The Java Edition version is still used by many players on PC.
- **LinkedIn** - Most of LinkedIn's backend uses Java to handle high scalability and performance.
- **Netflix** - Netflix's backend system uses Java to handle video streaming and user service management.

Brief About OOP (Object-Oriented Programming)

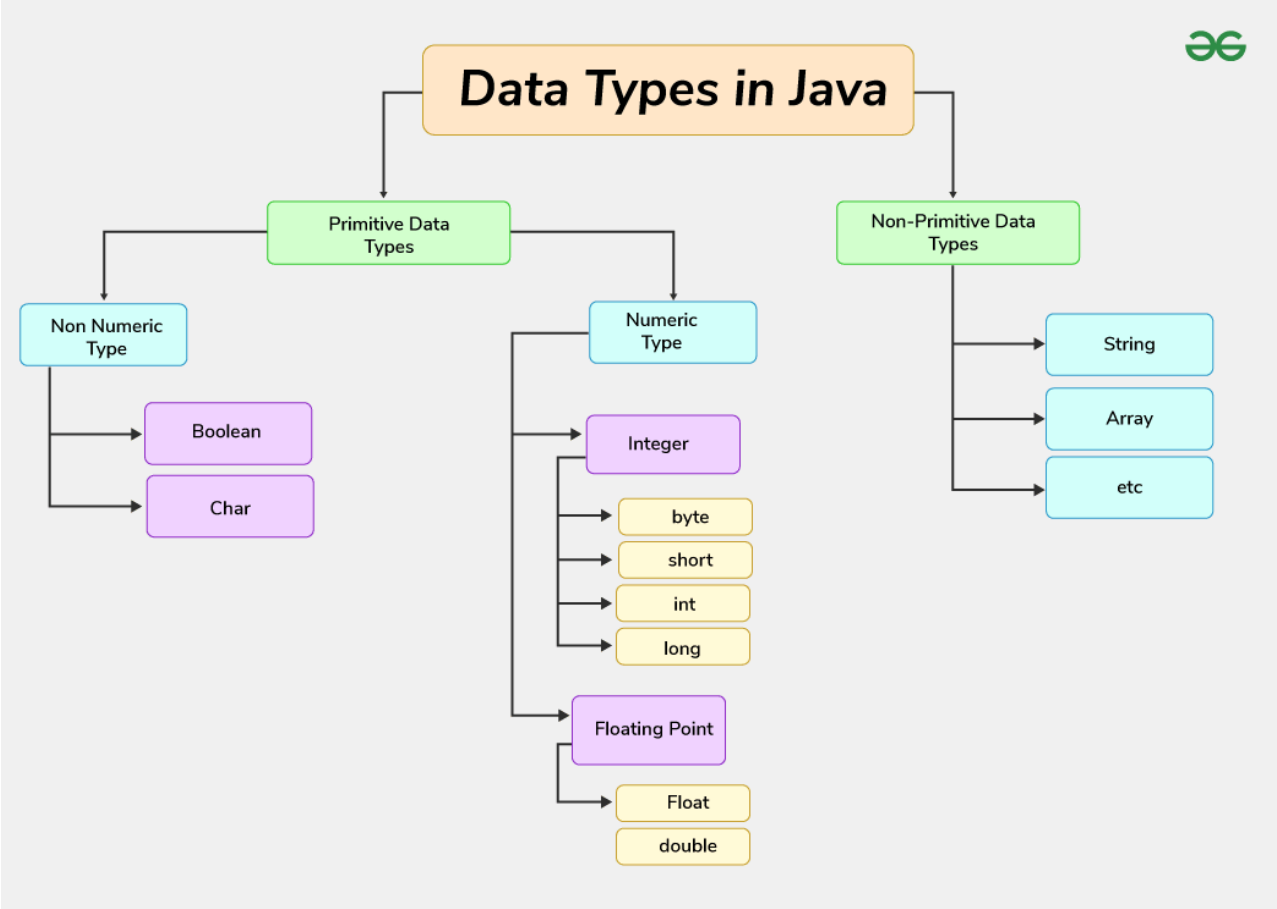


OOP is a programming concept oriented towards objects. What is an object? You will understand what an object is as the practical sessions proceed. In the OOP concept, we can create an object from a class that has been previously created. A class can be thought of as a blueprint for creating an object. Or you can think of it as a template. If you create an object from the Car class, the object will have doors and 4 wheels. If you create an object from the Motor class, the object will have 2 wheels and no doors.

You can create several objects from the same class. For example, if you create two objects from the car class, both will have doors and 4 wheels, but with different colors and speeds. That's all for now. We will learn more technical details in the next module.

MATERIALS

Data Type



Data types in Java are divided into two types: **primitive data types** and **non-primitive data types**. **Primitive data types** determine the size and type of variable values without requiring additional methods. Examples include int, char, float, and boolean. **Non-primitive data types** are also called reference types because they refer to an object. Examples include String, arrays, and classes.

The main differences between primitive and non-primitive data types are:

- **Primitive types** are predefined in Java. **Non-primitive types** are created by the programmer and are not defined by Java (except for String).
- **Non-primitive types** can be used to call methods that can perform certain operations, while **primitive types** cannot.
- **Primitive types** start with a lowercase letter, while **non-primitive types** start with an uppercase letter.
- The size of **primitive types** depends on the data type, while **non-primitive types** have the same size.

Primitive Data Types		Tipe data non primitif	
Data Type	Description	Data Type	Description
float	Stores a 32-bit decimal value with 7 digits of precision. Example: item price = 12.5f (float).	String	Stores text values. Example: nama = "John Doe" (String).
double	Stores a 64-bit decimal value with 15 digits of precision. Example: nilaiPi = 3.141592653589793d (double).	Array	Stores a collection of values of the same data type. Example: numbers = new int[]{1, 2, 3, 4, 5} (int array).
int	Stores a 32-bit integer with a range from -2,147,483,648 to 2,147,483,647. Example: population = 1000000 (int).	Object	Stores a collection of data and associated methods. Example: person = new Person("John Doe", 18) (Person object).
char	Stores a single character. Example: initial letter = 'A' (char).		
boolean	Stores a value of true or false. Example: isLoggedIn = true (boolean).		

Example:

```java
public class Module1Practice {
    public static void main(String[] args) {
        // Integer
        int population = 1000000;

        // Decimal
        float price = 12.5f;
        double weight = 75.5d;

        // Character
        char letter = 'L';

        // Boolean
        boolean isLoggedIn = true;

        // String
        String name = "Wira Yudha Aji Pratama";

        // Array
        int[] numbers1 = new int[]{1, 2, 3, 4, 5};
        int[] numbers2 = {1, 2, 3, 4, 5};
        int numbers3[] = {1, 2, 3, 4, 5};

        // Object
        Person person = new Person("Elon Musk", 53);
    }
}
```

Notes on Creating an Array: When creating an array, it's important to note that the data type stored in the array must be the same. If we define the data type at the beginning as int, then all elements in the array must also be of type int

Input Output (I/O)

Input and Output (I/O): Input and output in the Java programming language refer to how a program interacts with the outside world (outside the code). This means how the program we create can receive information from the user (input) or provide information to the user (output).

- Input

  All programming languages provide functions to perform input operations (for example, scanf() in the C language). Java itself provides three classes to receive input from the user, which are:

  1. Class Scanner
  2. Class BufferReader

3. Class Console

In this material, we will only learn about the **Scanner** class. The Scanner class is provided by Java to allow our program to receive input from the user through the keyboard. To use the Scanner class, we need to import the Scanner class into the code at the top of the code, like this:

```
Import java.util.Scanner;
```

After that, we need to create an object of the Scanner class with a variable name we want within the main method. For example, here we create a Scanner object named objInput:

```
public class modulPractice {
    public static void main(String[] args) {
        Scanner objInput = new Scanner(System.in);
    }
}
```

Let's break down the code above step by step: public class modulPractice {}: This is the name of the class we created (more details will be discussed in the next module). public static void main(String[] args) {}: This is the code for the main method. All Java programs always start with the main method. Scanner objInput = new Scanner(System.in);: Here, Scanner refers to the class we imported earlier, objInput is the name of the object we created from the Scanner class. new Scanner() means we are creating a new object from the Scanner class. Finally, System.in is used to take input from the user through the keyboard when the program runs. It might be difficult to understand just by reading. So, please try to practice it. Don't worry, you will understand it over time.

Here, we will try to create a simple program that can accept input from the user in the form of firstName and age. Here is an example:

```java
import java.util.Scanner;

public class modul1Practice {
    public static void main(String[] args){
        String firstName;
        int age;
        Scanner objInput = new Scanner(System.in);

        firstName = objInput.nextLine();
        age = objInput.nextInt();
    }
}
```

Let's Analyze the Code Above:

- The variable firstName is declared with the data type **String**.
- The variable age is declared with the data type **int**.
- firstName = objInput.nextLine(); is the code used to input a **String** value into the firstName variable. Notice that the input must go through objInput, followed by calling the nextLine() method. This method is used to receive user input in the form of a string that may include spaces.
- age = objInput.nextInt(); is the code used to input an **int** value into the age variable. The nextInt() method is specifically for the **int** data type.

What About Other Data Types?

There are other methods available for input based on different data types. Here are some of them:

- **next()** : Used to input a **String**, but it only reads up to the last space.
- **nextDouble()** : Used to input a **double** value.
- **nextFloat()** : Used to input a **float** value.
- **nextByte()** : Used to input a **byte** value.
- **nextBoolean()** : Used to input a **boolean** value.

There are many more methods available in the **Scanner** class. For a complete list, you can check the official documentation.

- Output

In Java, output is similar to printf() in C, but it uses the **System.out.println()** syntax. However, note that Java is **case-sensitive**.

Java provides several methods for displaying output:

- **System.out.print()**
- **System.out.printf()**
- **System.out.println()**
- **System.out.format()**

The **print()** and **println()** methods are both used to display text. The key difference is:

- **print()** displays the text as is.
- **println()** displays the text **with a new line at the end**.

Let's try creating a code snippet like this:

```
class modulPractice {
    public static void main(String[] args) {
        System.out.print("this is text printed with print()\n");
        System.out.println("whereas this is text printed with println()");
        System.out.printf("using printf(), %d", 50);
    }
}
```

The Output Will Be as Follows:

```
this is text printed with print()
whereas this is text printed with println()
using printf(), 50
Process finished with exit code 0
```

Sometimes, we don't just want to display plain text but also need to include text from variables and combine it with other text. For example, suppose we have a variable for **name** and **age**:

```
public static void main(String[] args) {
    String name = "Wira Yudha";
    int age = 21;
```

Then, if we want to display them using the **print()** or **println()** method, we simply include them inside the method like this:

```java
public class modulPractice {
    public static void main(String[] args) {
        String name = "Wira Yudha";
        int age = 21;
        System.out.println(name);
        System.out.print(age);
    }
}
```

This code will produce the following output:

```
Wira Yudha
21
Process finished with exit code 0
```

Actually, we don't need to use two separate print() or println() methods because we can concatenate the text using the **"+" operator**. **Example:**

```java
System.out.println(name + age);
```

If we want to add spaces properly, we can include a space within the string:

```java
System.out.println(name + " " + age);
```

The last method is **format()**, which is used to format and combine strings more efficiently. It works similarly to **printf()** in the C language.

```java
public class modulPractice {
    public static void main(String[] args) {
        String name = "Wira Yudha";
        int age = 21;
        System.out.format("My name is %s and I am %d years old %n", name, age);
    }
}
```

Explanation of the Code Above:

- The symbol **%s** is used to take the value from the variable next to it, meaning it represents a **String**.
- **%d** is used for **int** data types.
- **%n** is used for a **new line**, but we can also use **\n** for the same purpose.

For a complete list of format specifiers, you can check the official documentation here.

When the code above is executed, the program output will be:

```
My name is Wira Yudha and I am 21 years old

Process finished with exit code 0
```

Here is an example of combining **input and output** in Java:

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        String firstName;
        int age;
        Scanner objInput = new Scanner(System.in);

        System.out.print("Enter Your Name: ");
        firstName = objInput.nextLine();
        System.out.print("Enter Your Age: ");
        age = objInput.nextInt();

        System.out.println("Name: " + firstName);
        System.out.println("Age: " + age);
    }
}
```

Output:

```
Enter Your Name: Wira Yudha
Enter Your Age: 21
Name: Wira Yudha
Age: 21

Process finished with exit code 0
```

Condition

Conditions or branches in Java are used to determine whether a block of code will be executed or not. This is based on the Boolean value (true or false) of the expression being evaluated.

- Switch/Case

  This branching is another form of if/else/if branching, the difference is that this branching uses the switch and case keywords. The procedure for writing switch/case branches is the same as in C. Here is an example:

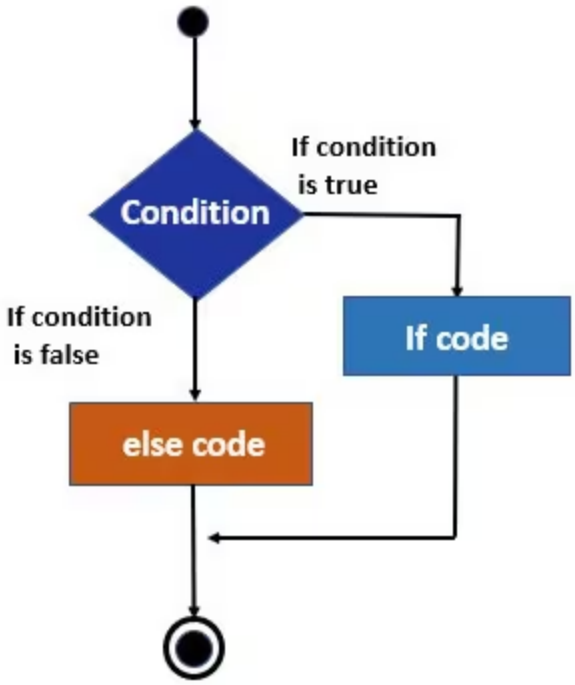
```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        String color = "red";

        switch (color) {
            case "red":
                System.out.println("Color is red");
                break;
            case "blue":
                System.out.println("Color is blue");
                break;
            default:
                System.out.println("Unknown color");
        }
    }
}
```

Output:

```
Color is red

Process finished with exit code 0
```

- If

This branching is used to execute a block of code if the expression produced by the condition is true. This means that the choices in if will only be executed if the condition is true.



But if it is wrong, then it will not do anything. Example:

```
public class Main {
    public static void main(String[] args) {
        int number = 10;

        if (number > 5) {
            System.out.println("Number is greater than 5");
        }
    }
}
```

Output:

```
Number is greater than 5

Process finished with exit code 0
```

- If/Else

  This branch is used to execute a block of code that if the result of the expression in if is not met then the else code will be executed. In the sense that if IF is wrong, then there is another alternative to run the code. Here is an example:

```java
public class Main {
    public static void main(String[] args) {
        int number = 5;

        if (number > 5) {
            System.out.println("Number is greater than 5");
        } else {
            System.out.println("Number is not greater than 5");
        }
    }
}
```

Output:

```
Number is not greater than 5

Process finished with exit code 0
```

- If/Else/If

  If the IF/ELSE branch only has two options. Then the IF/ELSE/IF branch has more than two options. Here is an example:

```java
import java.util.Scanner;

public class modulPractice {
    public static void main(String[] args) {
        int score;
        String grade;
        Scanner scan = new Scanner(System.in);

        System.out.print("Enter your score: ");
        score = scan.nextInt();

        if (score >= 90) {
            grade = "A";
        } else if (score >= 80) {
            grade = "B+";
        } else if (score >= 70) {
            grade = "B";
        } else if (score >= 60) {
            grade = "C+";
        } else if (score >= 50) {
            grade = "C";
        } else if (score >= 40) {
            grade = "D";
        } else {
            grade = "E";
        }

        System.out.println("Grade: " + grade);
    }
}
```

When we try to input the value 66, the output will be like this:

```
Enter your score: 66
Grade: C+

Process finished with exit code 0
```

- Nested If

  In the previous semester, we also learned about what nested if is in C language. In Java, the form of nested if is not much different, let's go straight to the code example:

```java
import java.util.Scanner;

public class modulPractice {
    public static void main(String[] args) {
        int totalPurchase, discount, payment;
        String card;
        Scanner scan = new Scanner(System.in);

        System.out.print("Do you have a membership card: ");
        card = scan.nextLine();
        System.out.print("Total purchase: ");
        totalPurchase = scan.nextInt();

        if (card.equalsIgnoreCase("yes")) {
            if (totalPurchase > 500000) {
                discount = 50000;
            } else if (totalPurchase > 100000) {
                discount = 15000;
            } else {
                discount = 0;
            }
        } else {
            if (totalPurchase > 100000) {
                discount = 5000;
            } else {
                discount = 0;
            }
        }

        payment = totalPurchase - discount;
        System.out.println("Total Payment: Rp " + payment);
    }
}
```

Loop

Loops in Java allow us to run a block of code repeatedly as long as certain conditions are met. This is very useful when we want to perform a task automatically continuously and sequentially. Because in the previous semester this material has been discussed, this discussion is only an introduction to how to write in Java.

- For

  The format for writing a For loop in Java is like this:

```java
for( int count = 0; count <= 10; count++ ){
    // block of code that will be repeated
}
```

The count variable is a variable used to store repeated values, so as long as the count value is less than 10, the loop will continue to be performed and the count ++ has a function to add +1 to the count value in each loop. The For code block begins with the '{' sign and ends with '}'. Here is an example:

```
public class Main {
    public static void main(String[] args){
        for (int i = 0; i < 10; i++) {
            System.out.println("Iteration to-" + i);
        }
    }
}
```

Output:

```
Iteration to-0
Iteration to-1
Iteration to-2
Iteration to-3
Iteration to-4
Iteration to-5
Iteration to-6
Iteration to-7
Iteration to-8
Iteration to-9

Process finished with exit code 0
```

- For Each

This loop is actually used to display the contents of an array. In short, an array is a variable that stores more than one value and has an index. For more details, it will be studied in module 5. The for each loop is done with the For keyword. Here is an example:

```
for (int item : dataArray) {
    // block of code that is repeated
}
```

The item variable will store the value of the array and we can read it like this: "For each item in dataArray, then do the loop". Example program with For Each:

```java
public class ModulePractice {
    public static void main(String[] args) {
        int numbers[] = {3, 1, 42, 24, 12};

        for (int x : numbers) {
            System.out.print(x + " ");
        }
    }
}
```

Output:

```
3 1 42 24 12
Process finished with exit code 0
```

- While

While can be interpreted as long as, the way this loop works is almost the same as branching. It will loop as long as the condition in while is true. The structure for writing a while loop is as follows:

```java
while (condition) {
    // code that will be executed
}
```

we can fill the condition with a comparison or boolean value, the condition only has true and false values. This loop will stop when the condition is false. Example code for a while loop:

```java
public class Main {
    public static void main(String[] args){
        int i = 0;
        while (i < 5) {
            System.out.println("Iteration to-" + i);
            i++;
        }
    }
}
```

```
Iteration to-0
Iteration to-1
Iteration to-2
Iteration to-3
Iteration to-4

Process finished with exit code 0
```

- Do While

  This loop actually works the same way as a while-loop, but do-while will loop once first and then check the condition. The writing structure is like this:

```
do {
    // code that will be repeated
} while (condition);
```

For an example code for using do-while:

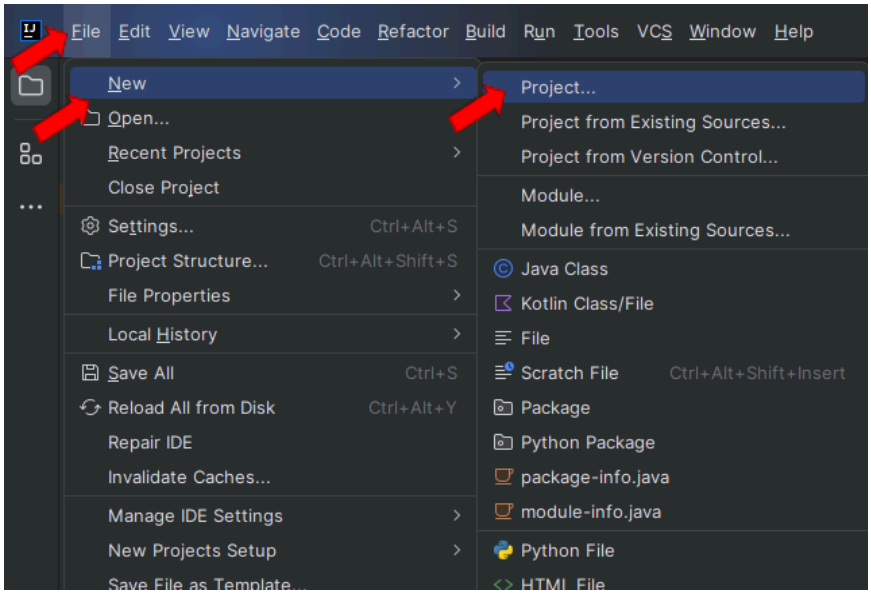
```
public class Main {
    public static void main(String[] args){
        int i = 0;
        do {
            System.out.println("Iteration to-" + i);
            i++;
        } while (i < 3);
    }
}
```

Output:

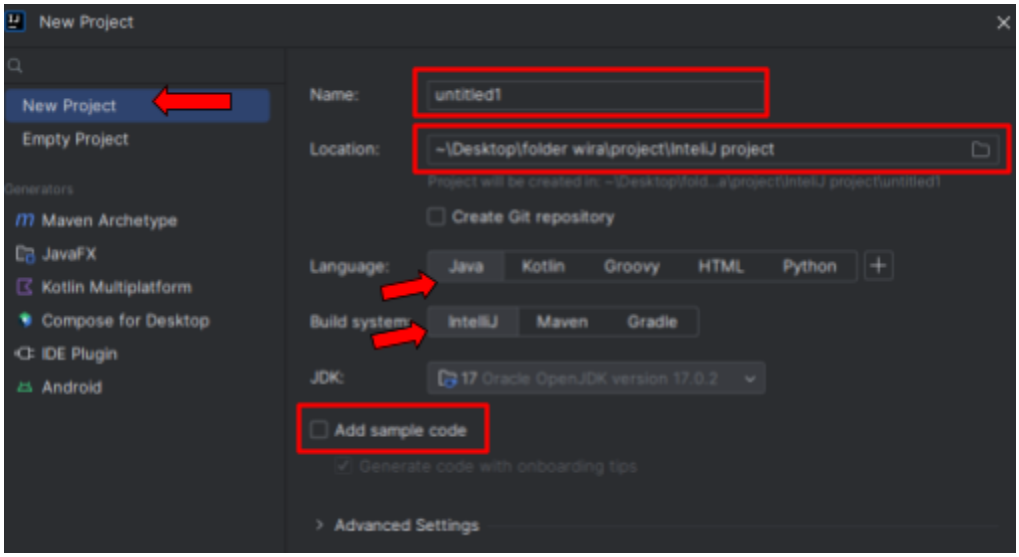
```
Iteration to-0
Iteration to-1
Iteration to-2

Process finished with exit code 0
```

PRACTICE

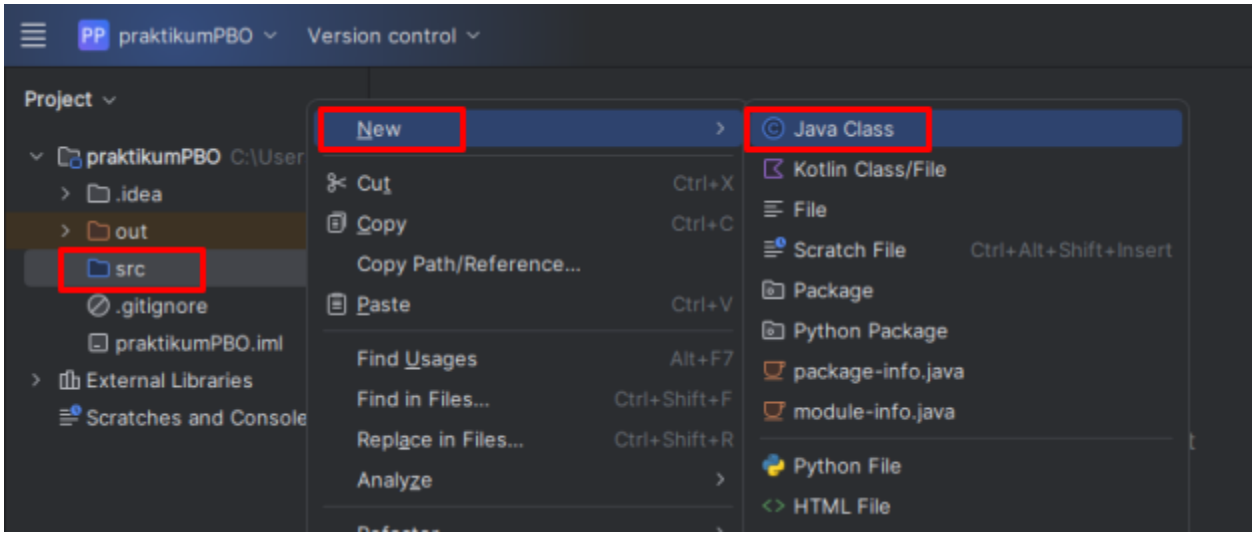
Let's experiment and create a new project! Open IntelliJ and follow the red arrows that will accompany you during this practice.



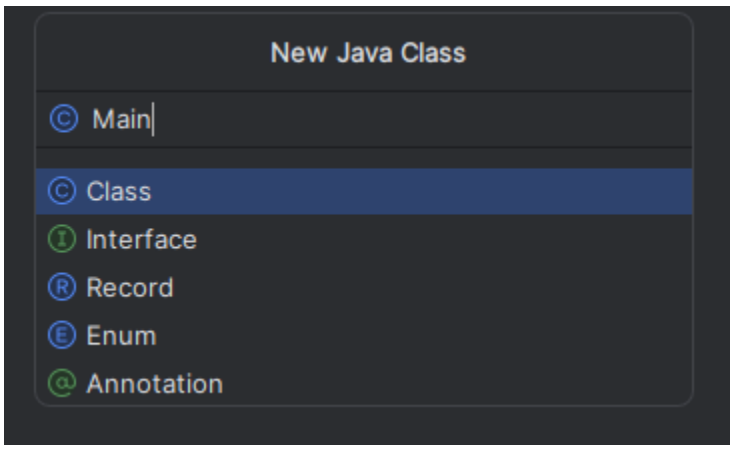
You can change the name and directory of the project you created. You can also check the 'Add sample code' section to start with existing coding (but this time we just uncheck it and start from the beginning). Once done, click create below.



Next, right-click on the 'src' folder and follow the steps to add a new class to start creating your program:

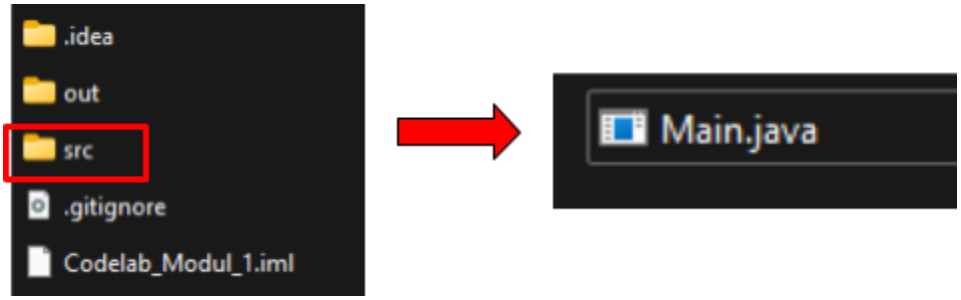


You are free to name your file, but this time let's just name it 'Main', guys.

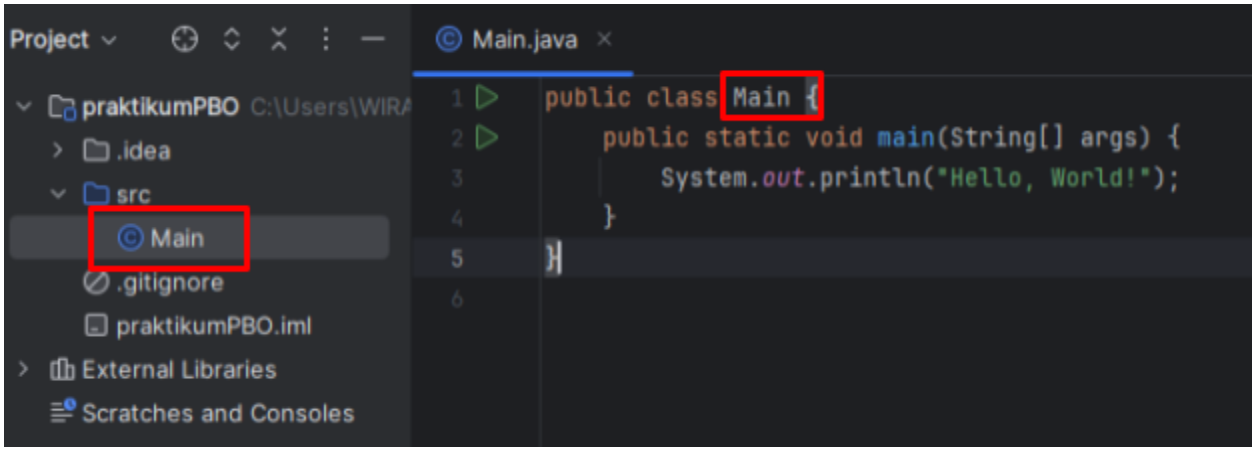


When you create a new project and class in IntelliJ, the project will be created in the form of a folder and its contents. This is different from the C language yesterday which was only a single file '.c'

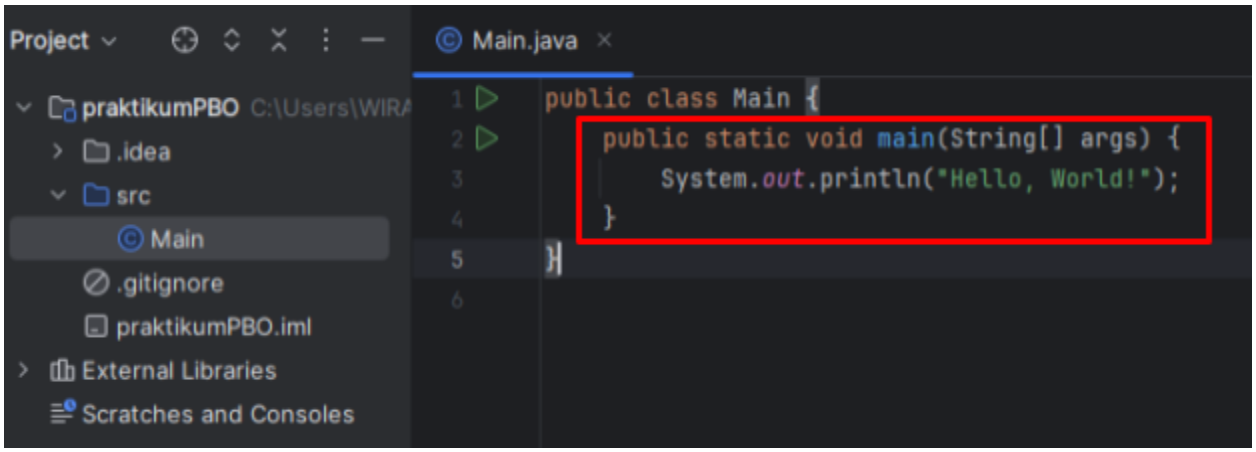
All the classes you create will be placed in the 'src' folder



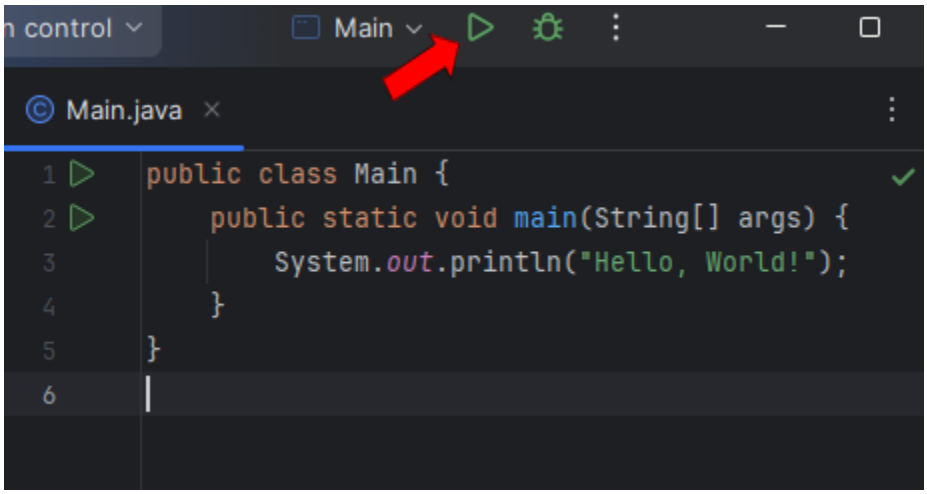
Here is an example of the contents of a simple class in Java. Please rewrite it. Pay attention to what I marked in the picture. The class name in the source code and folder must be in sync.



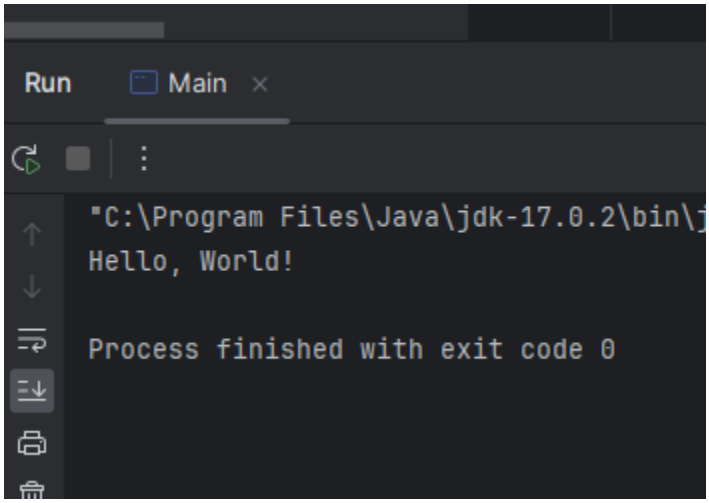
Then in the next image, what is marked is the main function. The main function itself is the same as in the C language. Where it is used to run the program. Functions and classes themselves have different uses. We will learn this in the next module. Fill the main function with System.out.println("Hello, World!"); to display the output as you have learned in the previous sub-chapter.



If so, you can run your program by pressing the button marked in the next image:



Then the output can be seen as follows:



Congratulations. That's your first Java program (I guess?). Feel free to develop it further ;). If you have any difficulties, please ask your respective assistants. Good luck!

TIPS

Displaying Hello World Text:

Video tutorial

Data Types and Variables:

Video tutorial

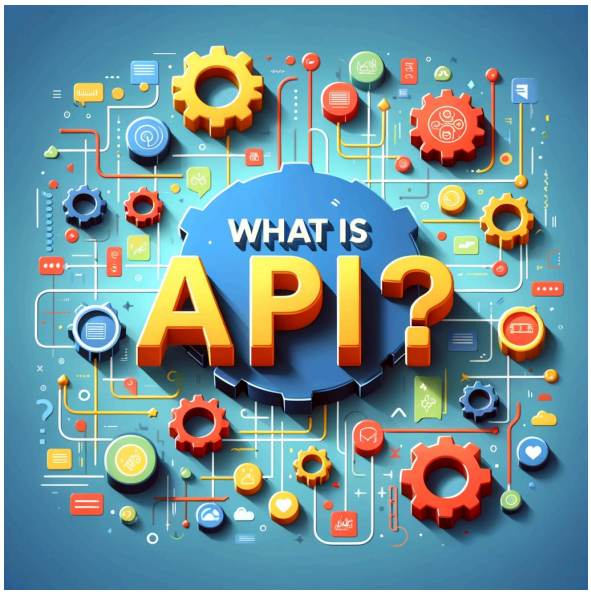
Exploring Variables and Data Types:

Video tutorial

JAVA API

THEORY

What is Java API?



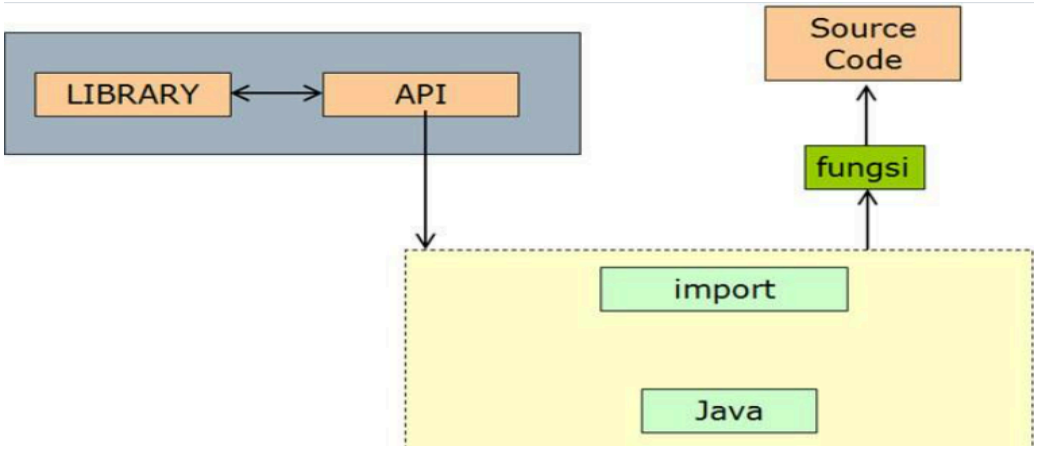
Java Application Programming Interface (Java API) is a Java component and class that has been provided by Java and has various capabilities. The capabilities of the Java API vary, ranging from handling objects, strings, numbers, and so on. Java API is a set of methods provided by the JDK. JDK provides many API libraries that can perform basic programming tasks such as displaying GUI, math functions, and many others. Java API classes are wrapped in a package written in the structured Java programming language and run on the JVM.

MATERIALS

API parts in Java

- Standard API (Java SE) → used for applications and applets with basic language services
- Enterprise API (Java EE) → for server applications with database services and server-side applications (servlets)
- API for micro devices (Java ME) → such as mobile phones

Java API Scheme:



Here is an example of using the Java API:

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Main {
    public static void main(String[] args){
        LocalDateTime timeNow = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

        System.out.println("Current Time Is : " + timeNow.format(formatter));
        System.out.println("Current Month Is : " + timeNow.getMonth());
    }
}
```

Output:

```
Current Time Is : 2025-02-19
Current Month Is : FEBRUARY

Process finished with exit code 0
```

PRACTICE

Let's experiment using a simple Java API! We will continue the practice in the previous chapter. Open the project, then please import the API that we will use (we will use java.util.Date) in the following way:menggunakan API Java sederhana! Kita akan melanjutkan praktek pada bab sebelumnya. Buka project tersebut, lalu silahkan import API yang akan kita gunakan (kita akan menggunakan java.util.Date) dengan cara seperti berikut:

```java
import java.util.Date;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

java.util.Date is part of the Java API. Java API is a collection of classes and interfaces provided by Java to help developers build applications. So, in this program, we use the Date class from the Java API to get and display the current time. Follow these steps:

```java
import java.util.Date;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.out.println("Current date and time: " + new Date());
    }
}
```

TIPS: you can type “sout” then press tab to create the System.out.println(); function instantly.

Here is the output (the output will adjust according to the time when you run the program):

```
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-jav
Hello, World!
Current date and time: Fri Feb 14 20:02:28 WIB 2025

Process finished with exit code 0
```

Very cool, isn't it? Feel free to experiment with other classes. If you have trouble, please ask your respective assistants. Good luck!

TIPS

Short video to learn API:

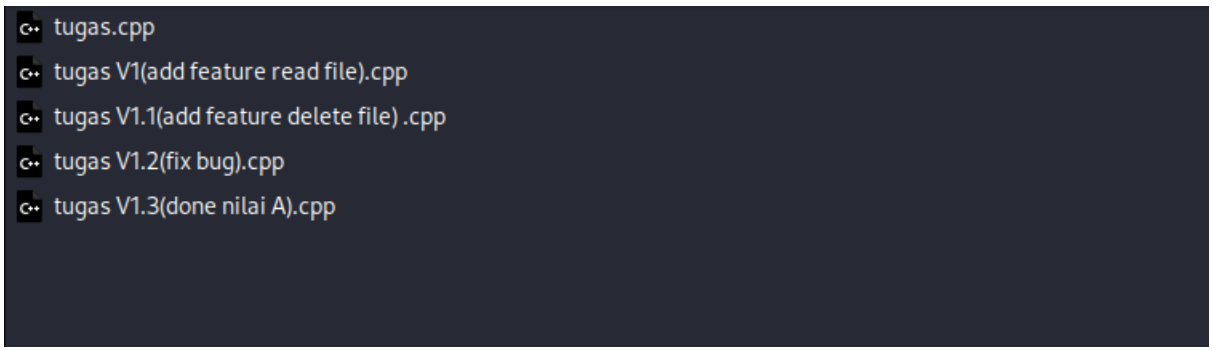
Video tutorial

GIT & GITHUB

THEORY

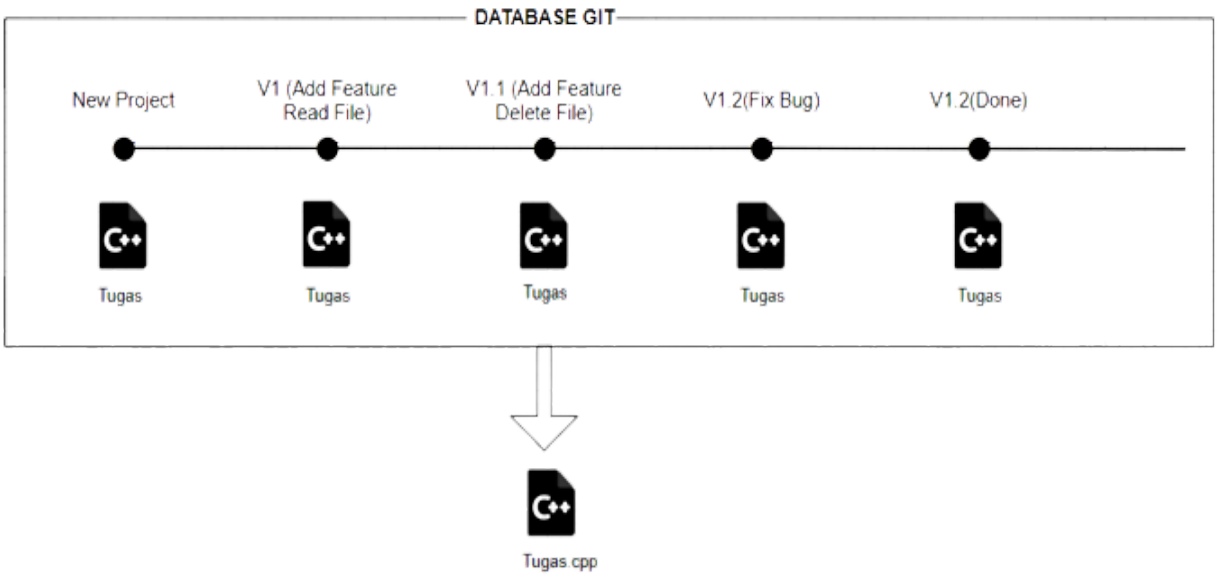
GIT

Git is a software based on Version Control System (VCS) that is used to record changes to all files or repositories of a project that we create. Software developers usually use Git for distributed revision (distributed VCS), this aims to store the database not only in one place, but everyone can be involved in creating the code that can store this database.



For more details, see the example case above, which is usually done by students in working on an assignment. Every time a revision is made, the old file is not deleted and saved with a different name. While the latest one is saved with a name, for example: "Task V1 (add feature)".

The concept of working in this way is considered very inefficient by many developers because the storage capacity will increase because the old file is not deleted. This is where VCS functions to help store history without creating a new file, only data changes in the file are stored. So that the file storage capacity becomes lighter.



As in the image above, every manual data change will produce many files. While VCS carries a concept to store the history of changes in one file only. This procedure can help if a division in a project monitors and connects (merges) between different extensions easily. So that the application created by a project team can function without manual naming.

GITHUB

Github is a cloud service that functions to store and manage a project called a repository (git repo). How it works on Github must be connected to the internet, so there is no need to install additional software on our computer. This can provide relief on the computer storage that we use because the project files are stored in the Github cloud.

The concept of Github's work is basically the same as Git, which is writing a source code individually or time. The user interface available on Github is more attractive and easy to understand for new users. If working as a team, one user can see who wrote the code and the date the code was created.

Difference between GIT and GitHub

Git	GitHub
Install software in local storage	Host via cloud service
Managed by The Linux Foundation	Acquired by Microsoft in 2018
Focuses on version control and code sharing	Focuses on centralized source code hosting
Offline access	Online access
No user management features	Uses user management
Provides a desktop interface called "Git GUI"	Uses a desktop interface called "GitHub Desktop"
Competes with Mercurial, Subversion, IBM, Rational Team, Concert, and ClearCase	Competes with GitLab and Atlassian BitBucket
Open sourced licensed	Options for free and paid users

MATERIALS

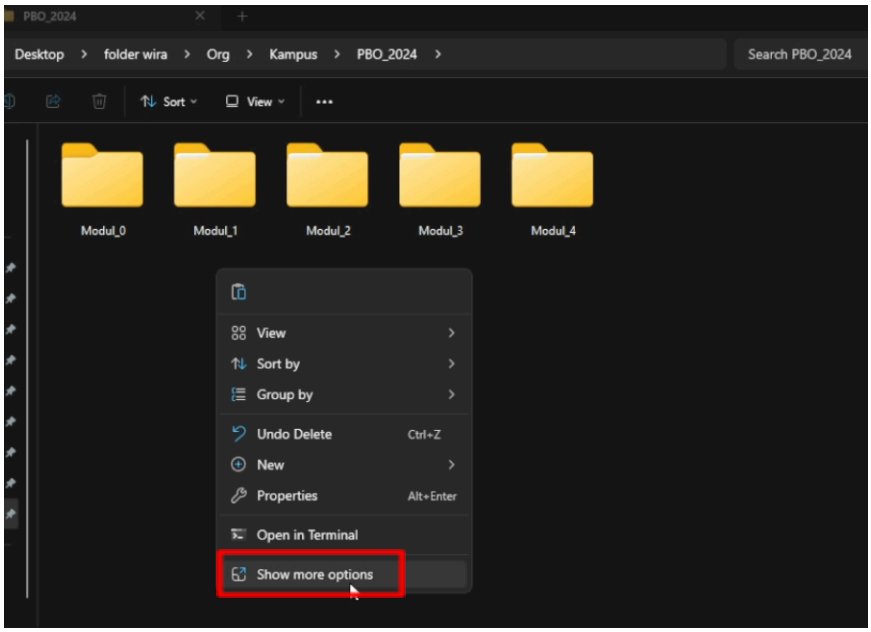
How to upload source code to a Github repository with Git

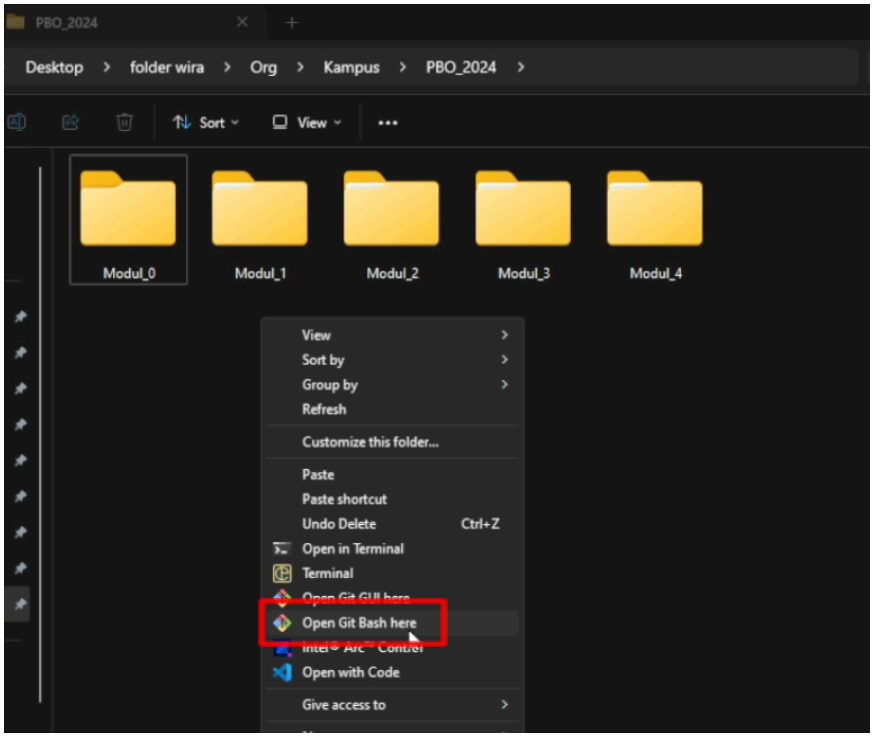
Requirement:

- Create an account at https://github.com
- For windows: Download and install git https://git-scm.com/downloads
- For Linux: apt-get install git
- For MacOS: *sorry, we are constrained by the lack of Mac devices T-T*

Go to the steps:

1. Open the source code directory/folder that will be uploaded to github. If you are already in the directory/folder, right-click and select open git bash here as shown in the example in the picture:



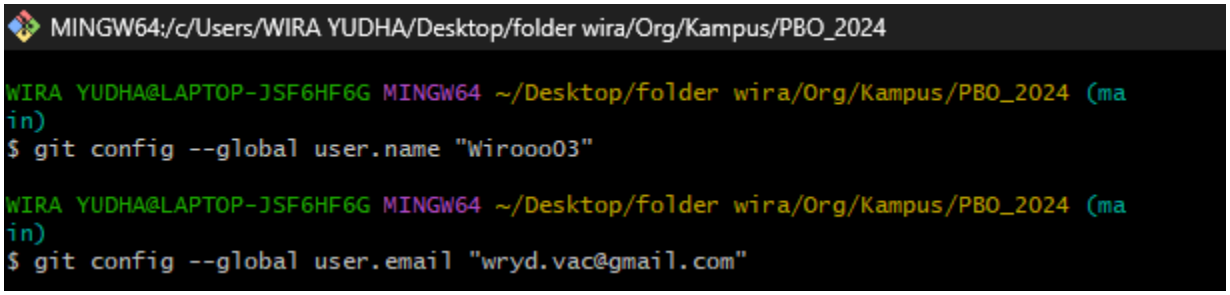


2. Enter the user and password using the following command:

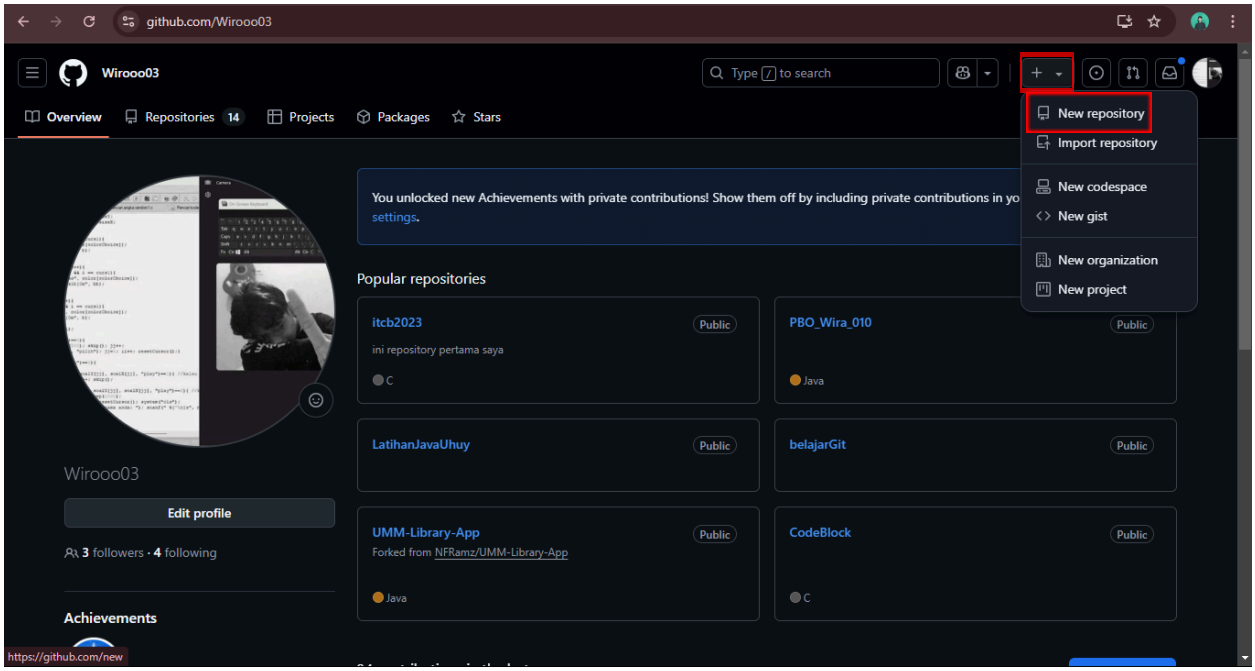
```
git config --global user.name "username "
git config --global user.email "email@gmail.com"
```

Note: Replace the words in double quotes with the username and email that match the username and email you created on github. This second step is only necessary for those of you who are installing Git for the first time. If you have used Git before, you can go directly to step 3

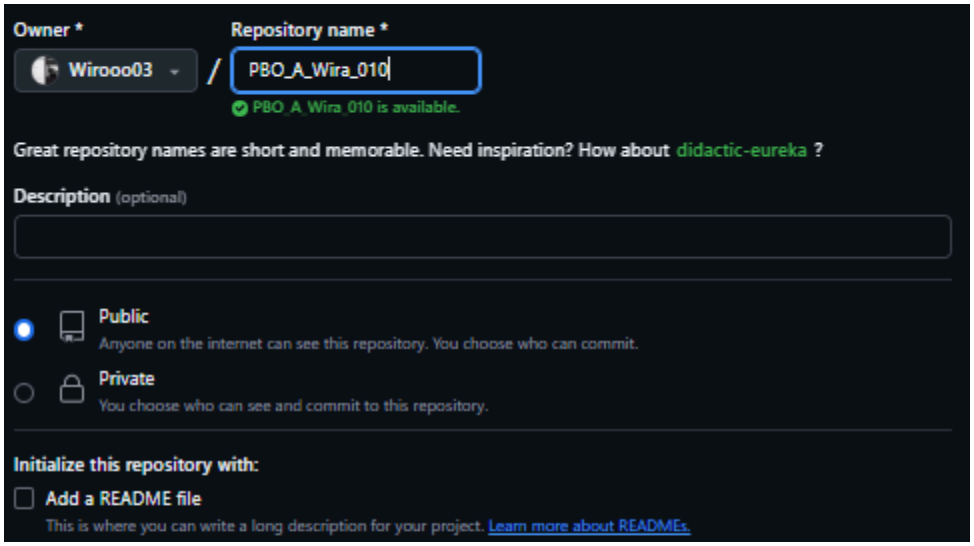
Example:

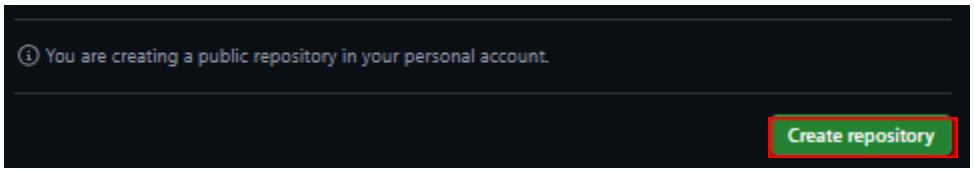


3. Login to the github account that was created earlier. After successfully logging in, click the plus icon on the top right of the active GitHub page. Next, click New Repository



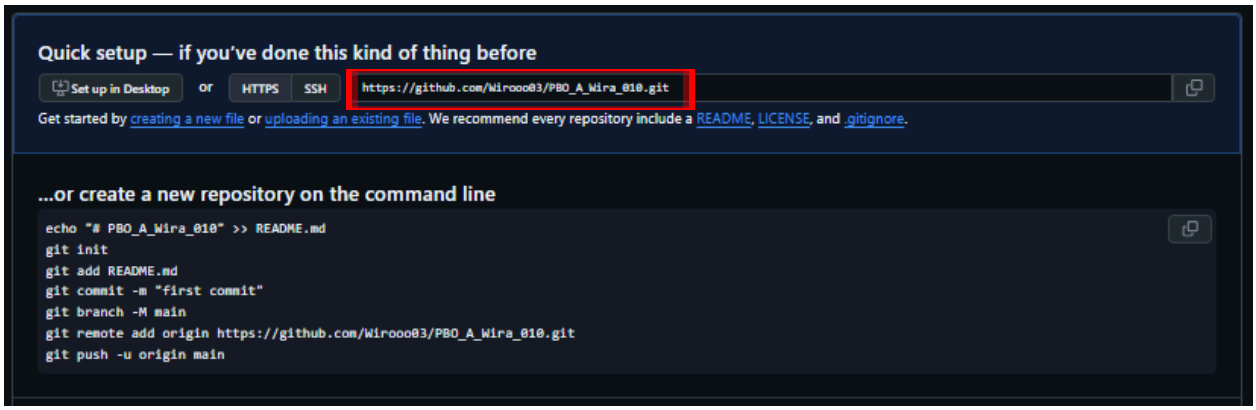
4. Fill in the Repository Name column. Make sure the other menus are filled in according to the example image. If they are the same, click Create Repository.





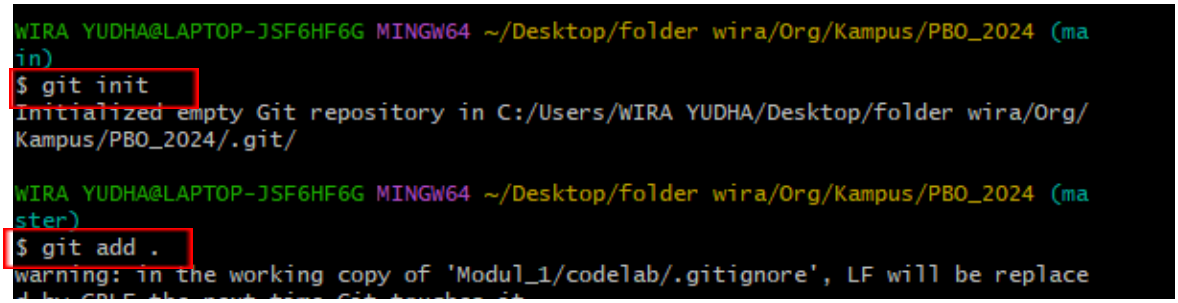
Important note: Repository is a central folder that functions to store our source code later. The owner (creator) can limit repository access rights to anyone who can access it. Only certain people you give permission to can read, write, and delete files in the repository. This permission access setting is in the collaborator section. For now, uncheck Add a README file.

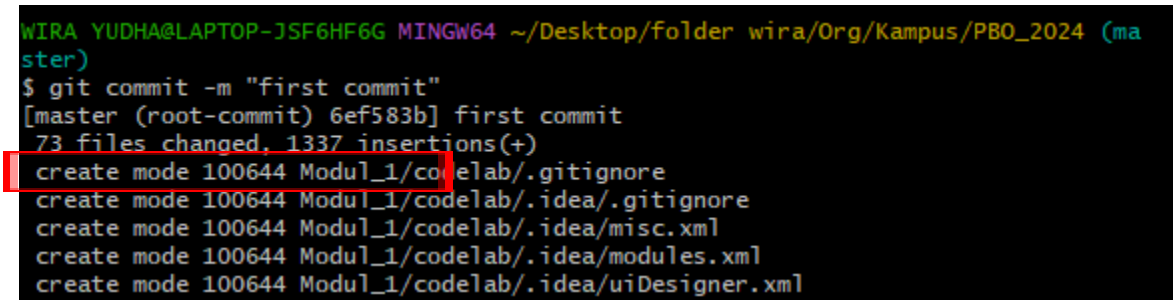
5. Copy the link in the quick setup form

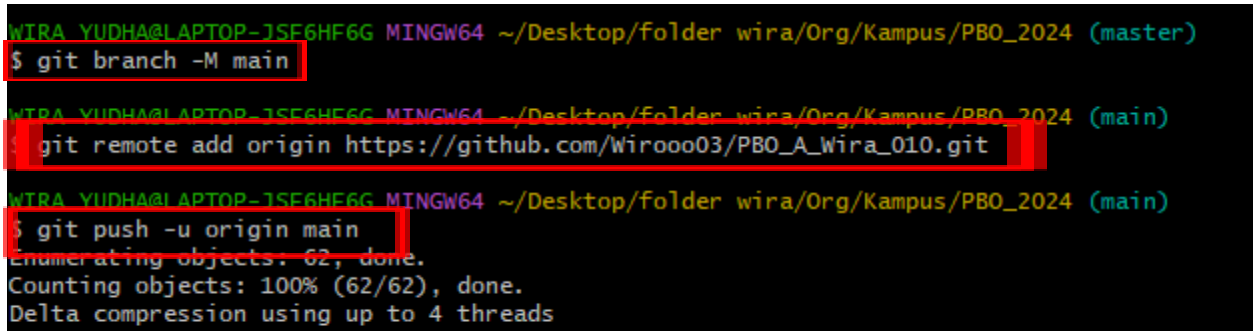


6. To upload the source code that has been created, follow the command according to the following example:
    - Type the command in gitbash as in step 2

- Before typing the command git push -u origin main type the command git remote add origin <url copied in step 5>







Notes:

- The "." sign in "git add ." means adding anything new to the folder. Or adding anything that has been changed from the previous file. You can add only certain files by "git add tugas.java"
- **git commit –m is a command to provide a message for the uploaded file. You can replace the message inside the double quote symbol ("....") with another message.**
- The main command is the name of the main branch in your repository. The term is branch.

7. Once you have successfully uploaded the file to GitHub, please go to the repository.

8. If the uploaded file is correct, it means that you have successfully added the file to GitHub.

Intellij IDEA Built-In Version Control

In this material we will try to configure Git to be connected to intellijIDEA. Follow the steps below to configure Git on intellijIDEA:

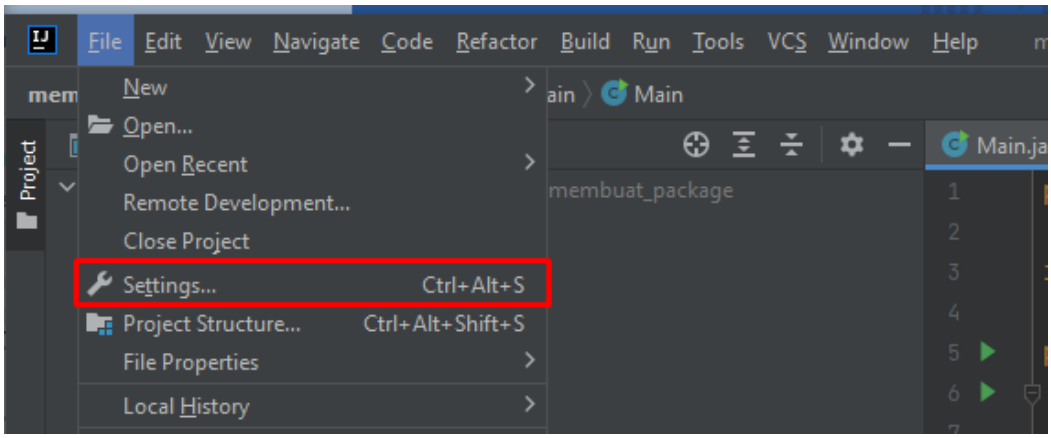
1. Make sure git is installed on the OS we are using, if not installed can follow the steps of the previous material. To make sure git is installed please open cmd or terminal for linux, then type the command git -v.

   Note: Create a repository first as in steps 3 and 4

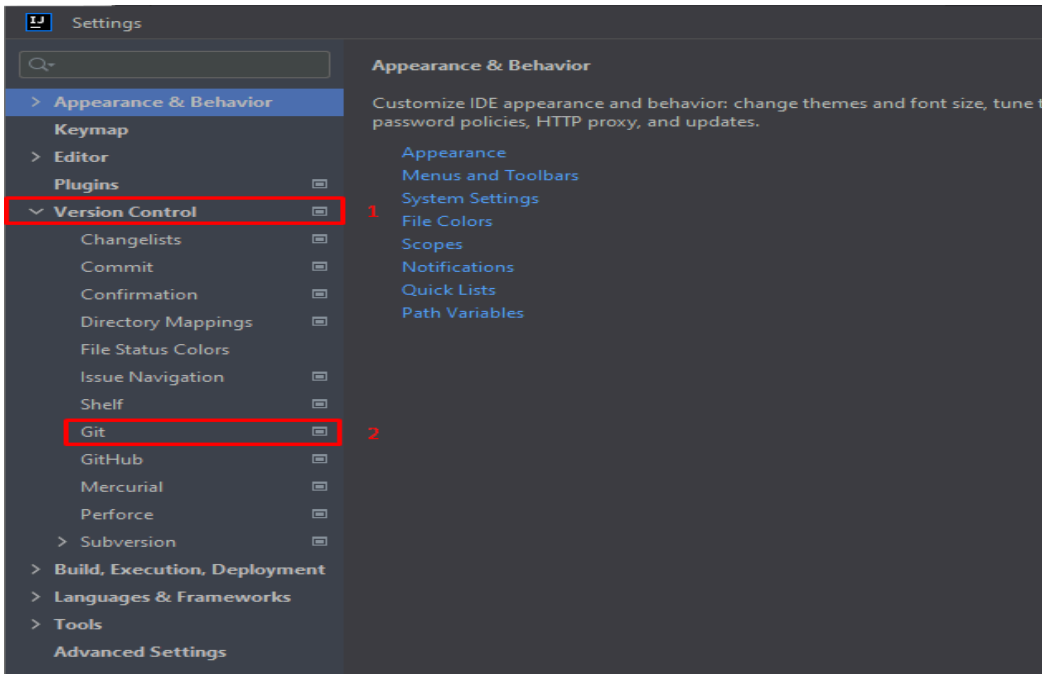
   ```
   C:\Users\WIRA YUDHA>git -v
   git version 2.44.0.windows.1

   C:\Users\WIRA YUDHA>
   ```

   If the output appears as in the image above, it means that Git has been installed with version 2.37.2.windows.2

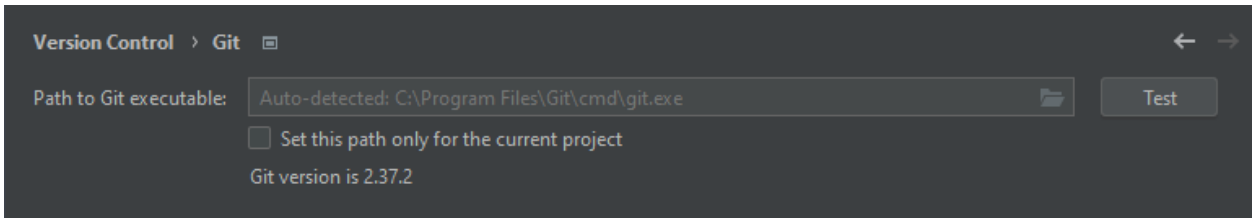
2. Open intellijIDEA and go to File in the top left corner > Settings.



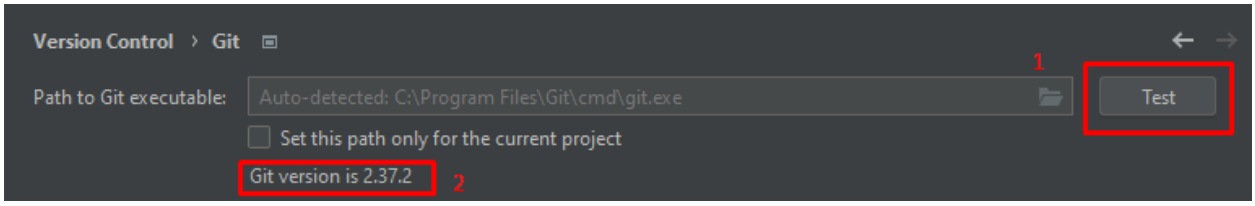
3. Select the Version Control > Git option.



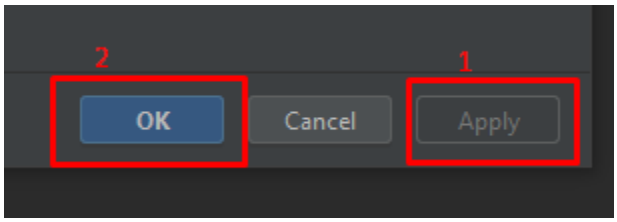
4. In the Path to Git executable section, specify the path to the Git executable installation (usually git or git.exe for windows). If it says autodetected like this, it means our Git installation has been automatically detected.



5. Click Test to make sure the path entered is correct. A git version message will appear if it is correct.



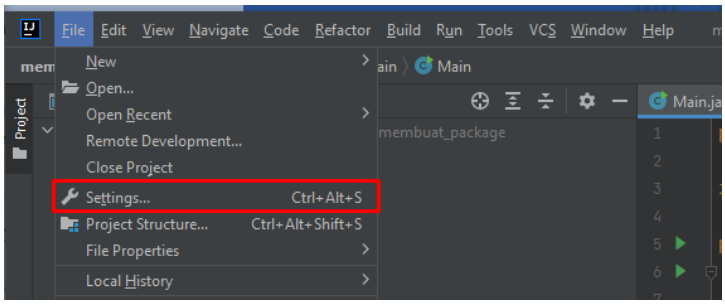
6. Click apply and OK to save the configuration.



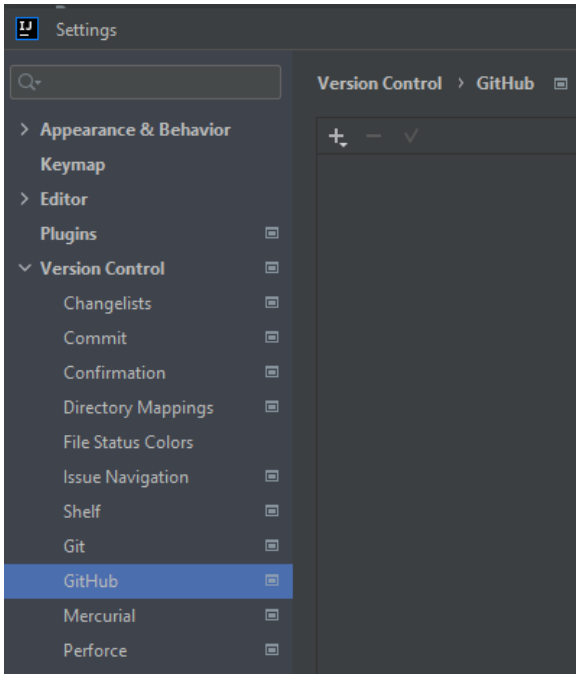
Github Integration with IntellijIDEA

For the Github integration process with intellijIDEA, you can follow the steps below:

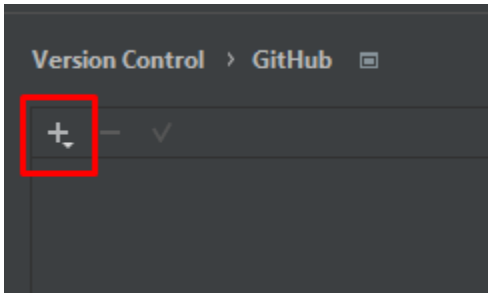
1. Go to File > Settings



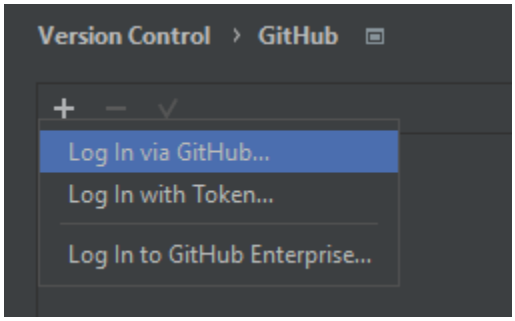
2. Navigate to Version Control > Github



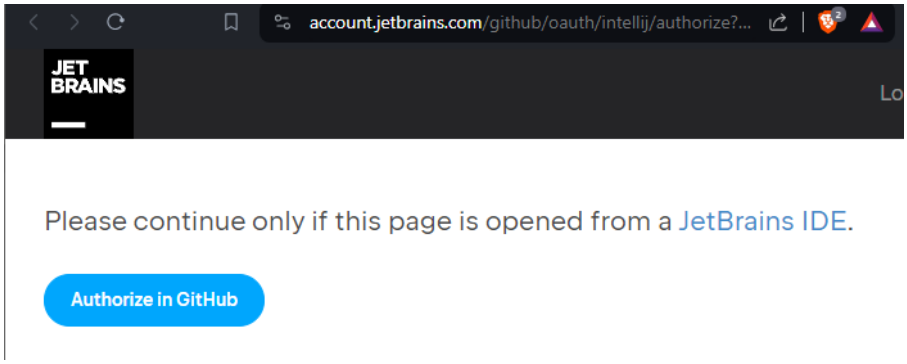
3. Press the + icon to add a Github account.



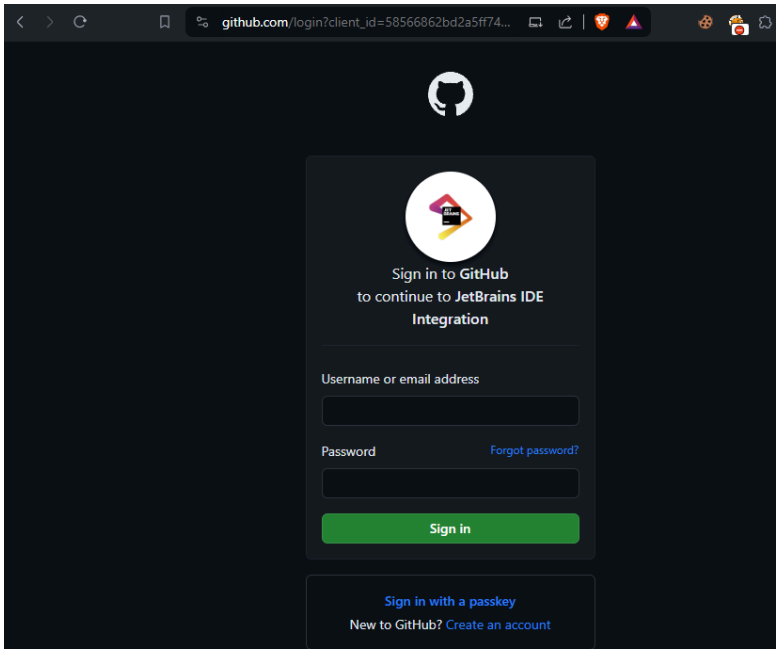
4. Select login access via Github or Token, in this example we will use Github



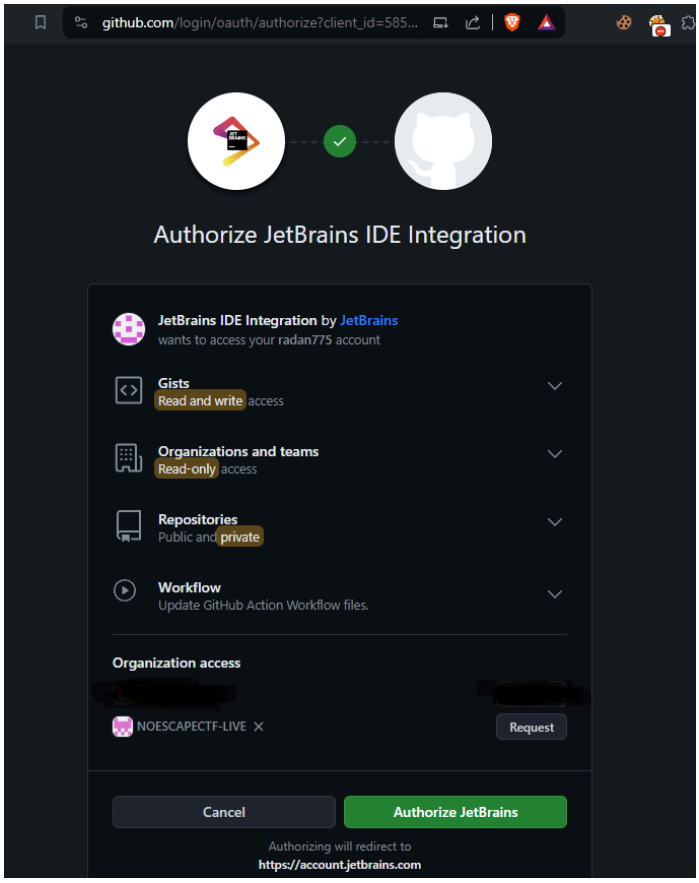
5. When you press Log In via Github, a new browser tab will appear and click Authorize in Github.



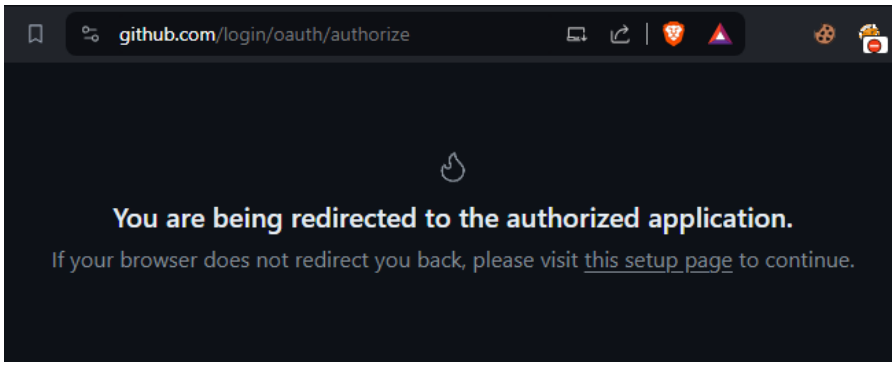
6. Fill in the account credentials according to the username and password, then click Sign In.



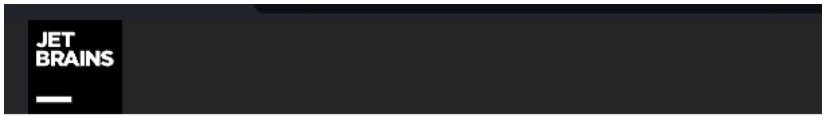
7. Click Authorize JetBrains



8. When a display like this appears, please wait.

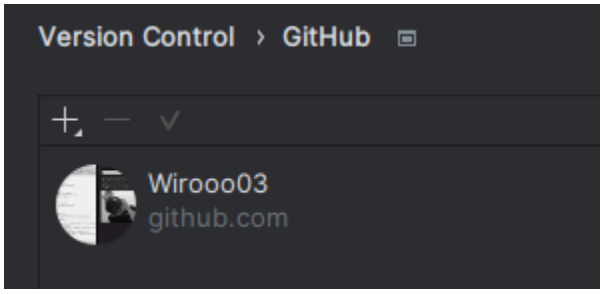


9. And a website will appear with the following display

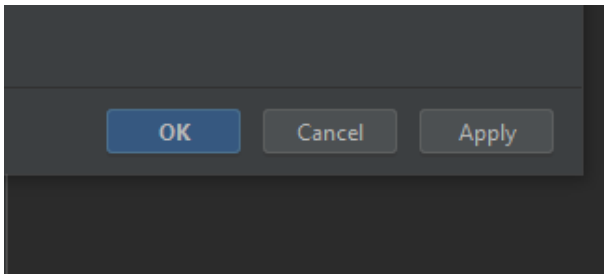


You have been successfully authorized in GitHub. You can close the page.

10. Go back to IntellijIDEA, the added account will appear.



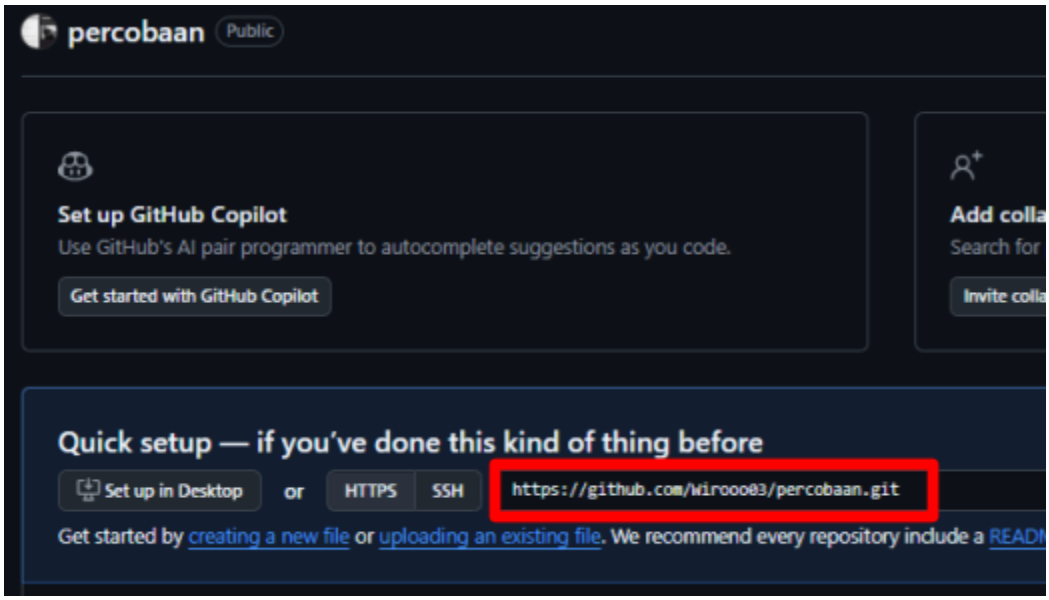
11. Finally click Apply and OK



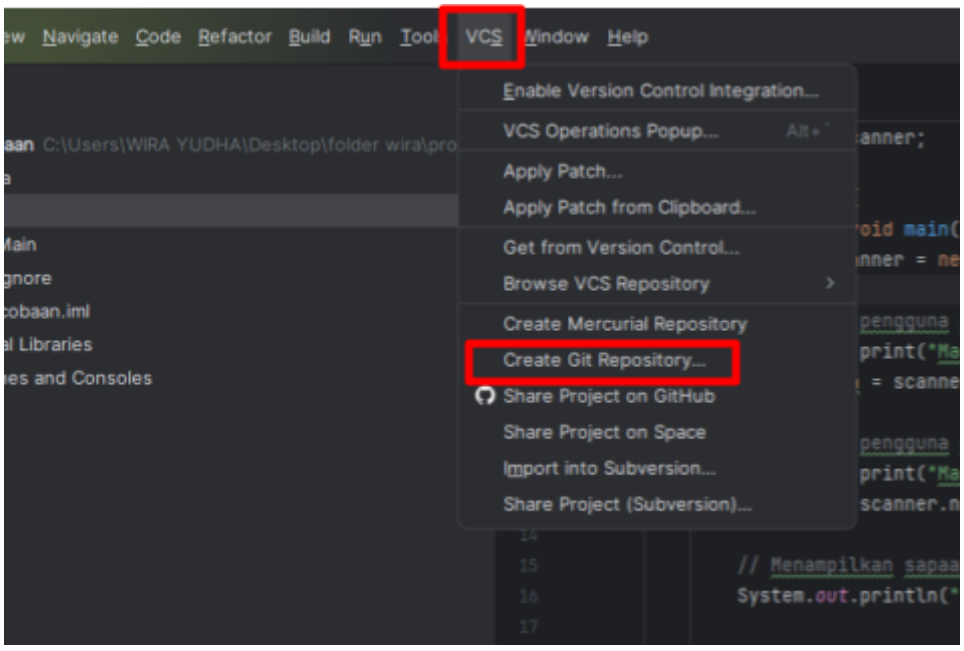
Working with Git Repository

To create or clone a Git repository, follow these steps:

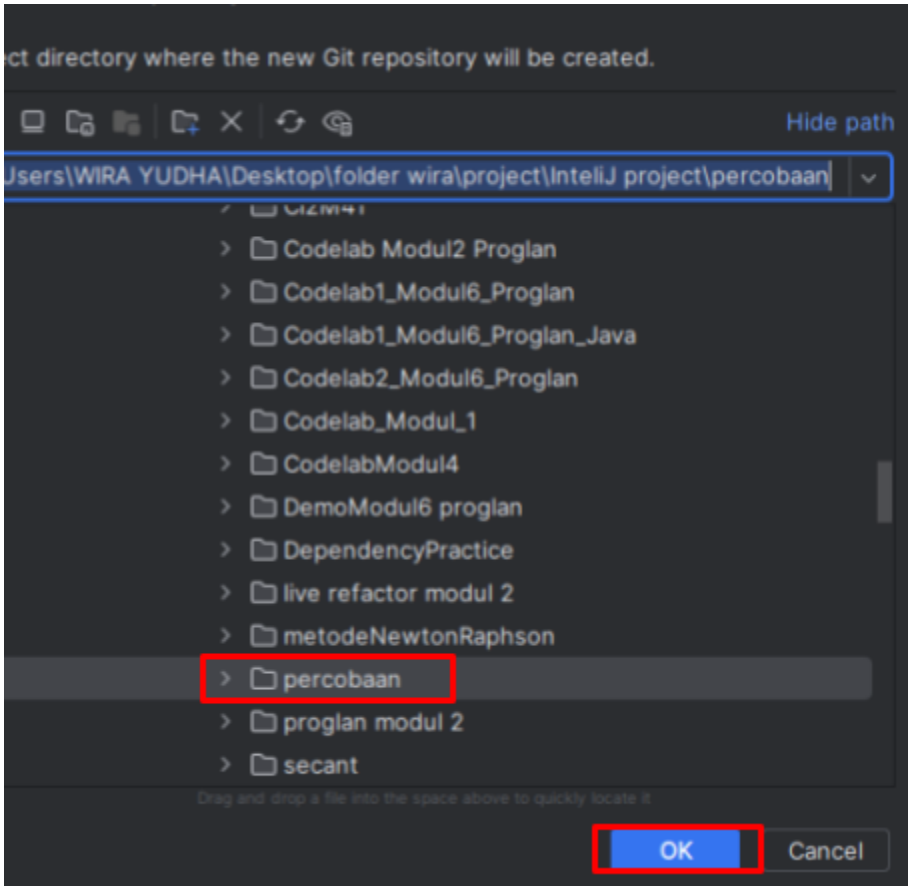
1. Create a repository on github like Steps 3 and 4 in the git and github tutorial and copy the repository url.



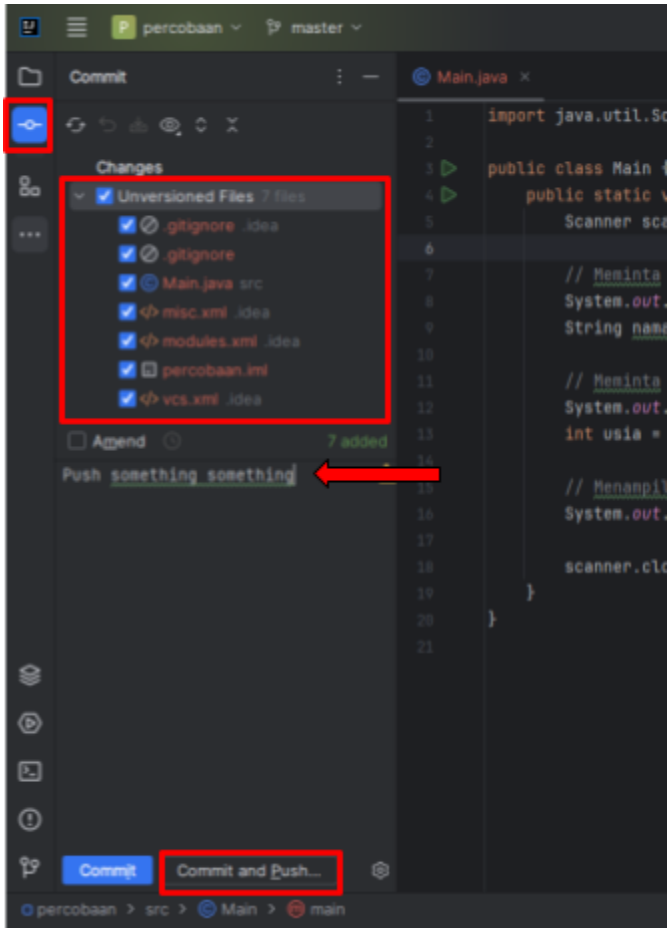
2. To create a new repository, go to VCS > Get From Version Control



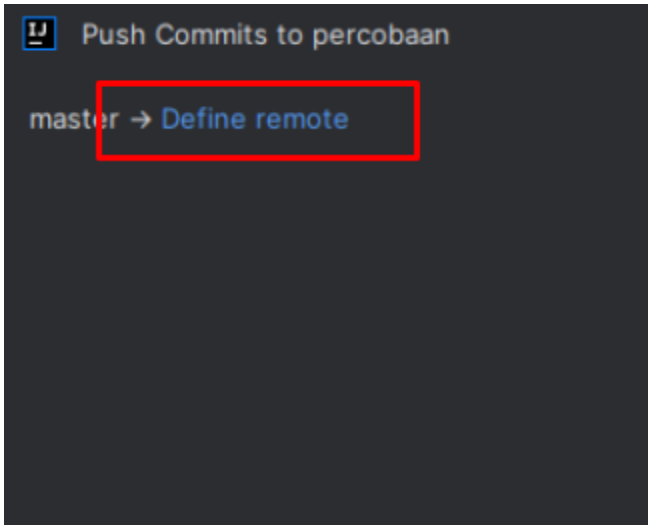
3. Select the file directory that will be pushed to GitHub then click "ok"



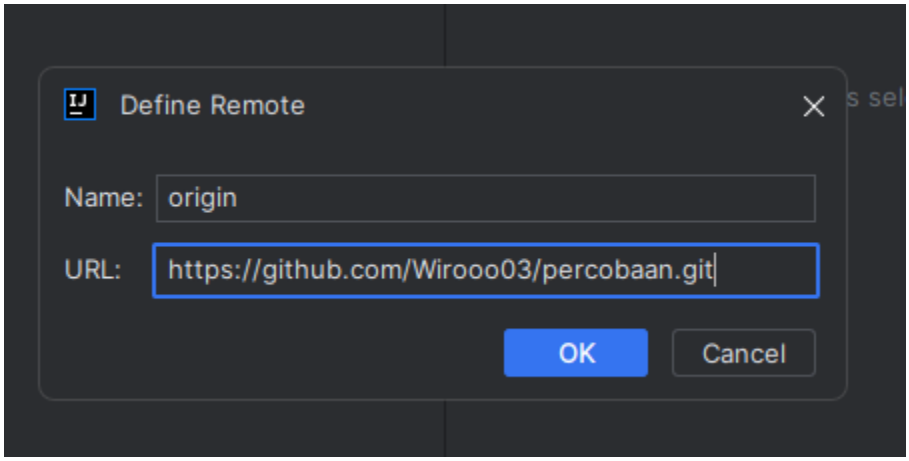
4. Select the commit sidebar then check all the files that will be pushed to GitHub > fill in the column below as a commit description, for example "first commit" select commit / commit and push



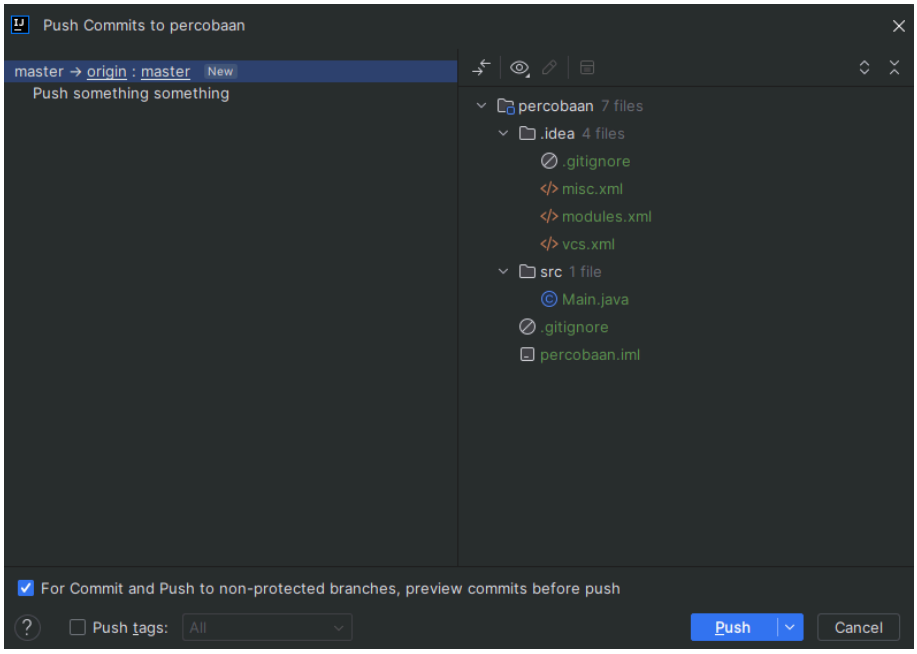
5. After completing step 3, a new window will appear if we are doing development for the first time / pushing for the first time and select define remote



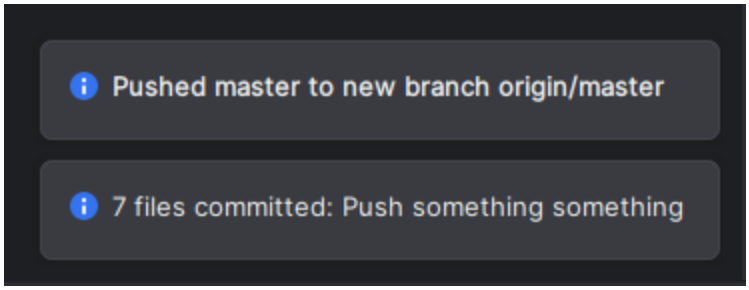
6. For the URL form, fill in the repository URL that was created in step 1 above and click OK.

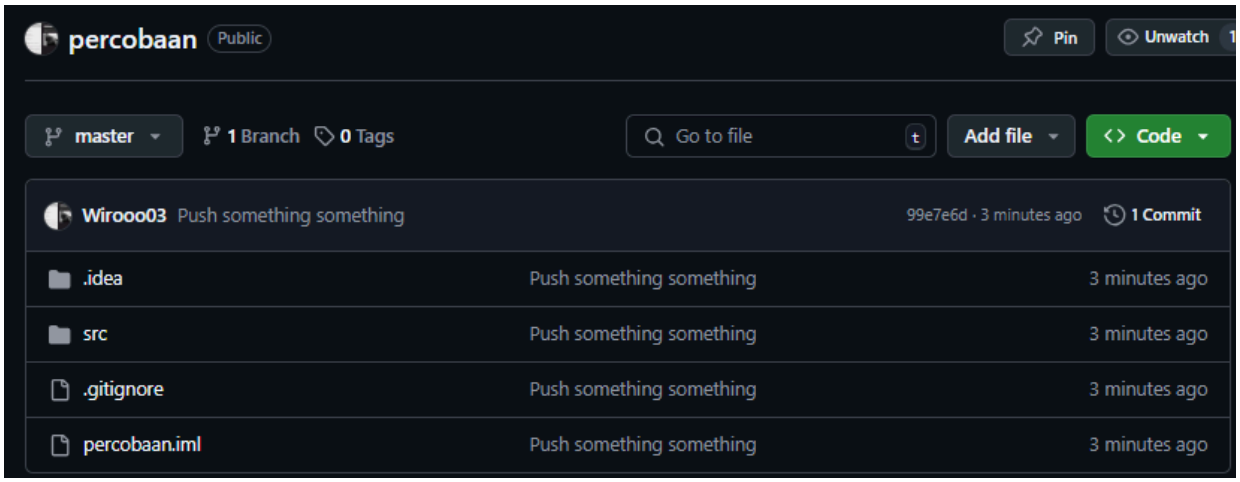


7. then a preview of the folder we have selected appears and click push



8. If there is a message like the image below in the bottom right corner, then our source code has been successfully uploaded to GitHub.





TIPS

Quick git bash tutorial:

Video tutorial

CODELAB & TASK

CODELAB

Create a simple Java program that prompts the user to enter their personal details, with the following specifications:

1. Use Scanner to read input from the user and make sure the program closes the Scanner when finished.
2. Required input:
    - Name (String)
    - Gender (only in the form of letters 'F' or 'M')
    - Year of Birth (Integer)
3. Calculate the user's age based on the year of birth entered. The current year should be obtained using the Java API java.time.LocalDate.
4. Rules for gender:
    - If the user enters 'M' or 'm', then the output displayed is "Male"
    - If the user enters 'F' or 'f', then the output displayed is "Female"
5. Display User Name, Gender, and Age.
6. Example of expected output:

```
Enter name: Wira Yudha
Enter gender (M/F): M
Enter year of birth: 2004

Personal Data:
Name            : Wira Yudha
Gender          : Male
Age             : 21 years

Process finished with exit code 0
```

TASK 1

Create a Java program that implements the concept of branching (if-else) and Scanner to create a simple login system. This program must have the following features:

1. Selecting Login Type

- The program must display the login options:
    1. Admin
    2. Student

2. Invalid Choice
    - If the user enters a choice that is not 1 or 2, display the message "Invalid choice.".
3. Admin Login
    - If the user chooses to login as Admin, the program must ask for a username and password.
    - A valid username is "Admin + (the last 3 digits of your student ID)" and a valid password is "Password + (the last 3 digits of your student ID)".
    - Example:
        1. **Username** = Admin010
        2. **Password** = Password010
    - If the username and password are correct, display the message "Admin login successful!".
    - If incorrect, display the message "Login failed! Wrong username or password.".
4. Student Login
    - If the user chooses to login as Student, the program must ask for a Name and Student ID.
    - Valid names are "Your Name" and valid student IDs are "Your Student ID".
    - Example:
        1. **Name** = Ken Aryo Bimantoro
        2. **Student ID** = 202310370311006
    - If the name and student ID are correct, display the message "Student Login Successful!" and print the name and student ID.
    - If incorrect, display the message "Login Failed! Wrong name or student ID."
5. Example of expected output:

```
Select Login Type:
1. Admin
2. Student
Enter your choice: 3
Invalid choice.

Process finished with exit code 0
```

```
Select Login Type:
1. Admin
2. Student
Enter your choice: 1
Enter Admin Username: Admin069
Enter Admin Password: Password069
Admin login successful!

Process finished with exit code 0
```

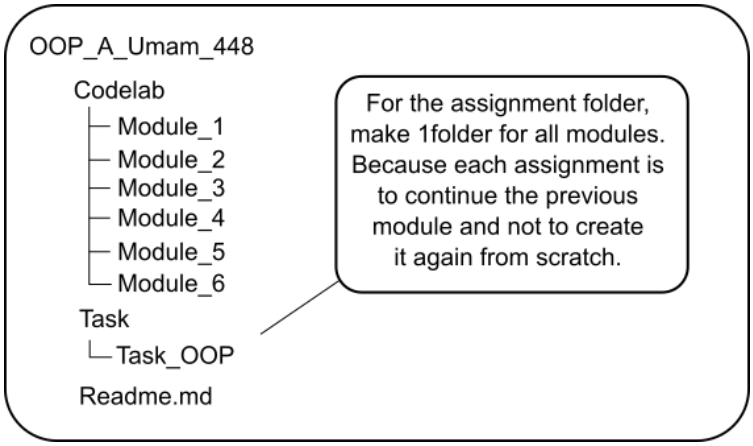
```
Select Login Type:
1. Admin
2. Student
Enter your choice: 1
Enter Admin Username: admin
Enter Admin Password: password
Login failed! Wrong username or password.

Process finished with exit code 0
```

```
Select Login Type:
1. Admin
2. Student
Enter your choice: 2
Enter Name: Azril Kucai
Enter Student ID: 202310370311069
Student login successful!
Name: Azril Kucai
Student ID: 202310370311069

Process finished with exit code 0
```

```
Select Login Type:
1. Admin
2. Student
Enter your choice: 2
Enter Name: Andre Tahapok
Enter Student ID: 34i2893257875287 5487592
Login failed! Wrong name or student ID.

Process finished with exit code 0
```

TASK 2

Push the assignment 1 that you have created to your GIT account using Git Bash (demonstrated to each assistant during the practicum) with the Repo name: PBO_(Class)_(Name)_(last 3 digits of NIM). Example: PBO_A_Umam_448. Here is an example of a file structure for your repo:



After that, list your repository link in the following spreadsheet: **https://docs.google.com/spreadsheets/d/1Y2DsBETV9JWLPKKl-UVVlNhgrWvOUTgJee-AvqWyQHs/edit?usp=sharing**

Note:

- Pay attention to your class; do not list it under a different class.
- You are not allowed to replace the repository you initially listed with a new one. The initial repository will be used until the end of the module.

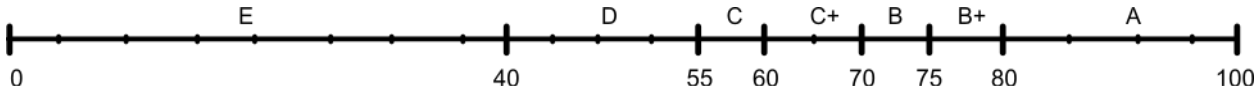
EVALUATION

ASSESSMENT RUBRIC

Assessment Aspects	Poin
CODELAB	**Total 25%**
Original code (not copied)	5%
Code accuracy	10%
Code neatness	5%
Code creativity	5%
TASK 1	**Total 50%**
Original code (not copied)	5%

Code accuracy	10%
Code neatness	5%
Ability to explain	15%
Ability to answer question	15%
TASK 2	**Total 25%**
Successfully Pushed to GITHUB	15%
Ability to explain	10%
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

Congratulations to all of you who have completed this module with dedication and hard work! Your consistency and honesty in completing the tasks without cheating demonstrate an outstanding character. Learning is not just about finding the answers but understanding the process and growing from every challenge.

Easy, right? Well, this is just Module 1, so keep up the spirit until Module 6! wkwkw Stay motivated because, in the real world, it's not just the final result that matters but also the effort you put in to achieve it.

If you encounter any difficulties, don’t hesitate to ask and discuss. We are all on the same learning journey.

Keep up the spirit, and see you in the next module!