



National University of Computer & Emerging Sciences,
Karachi



Computer Science Department
Spring 2023, Lab Manual – 01

Course Code: CL-1004	Course : Object Oriented Programming Lab
Instructor(s) :	Abeer Gauher, Hajra Ahmed, Shafique Rehman

LAB - 1

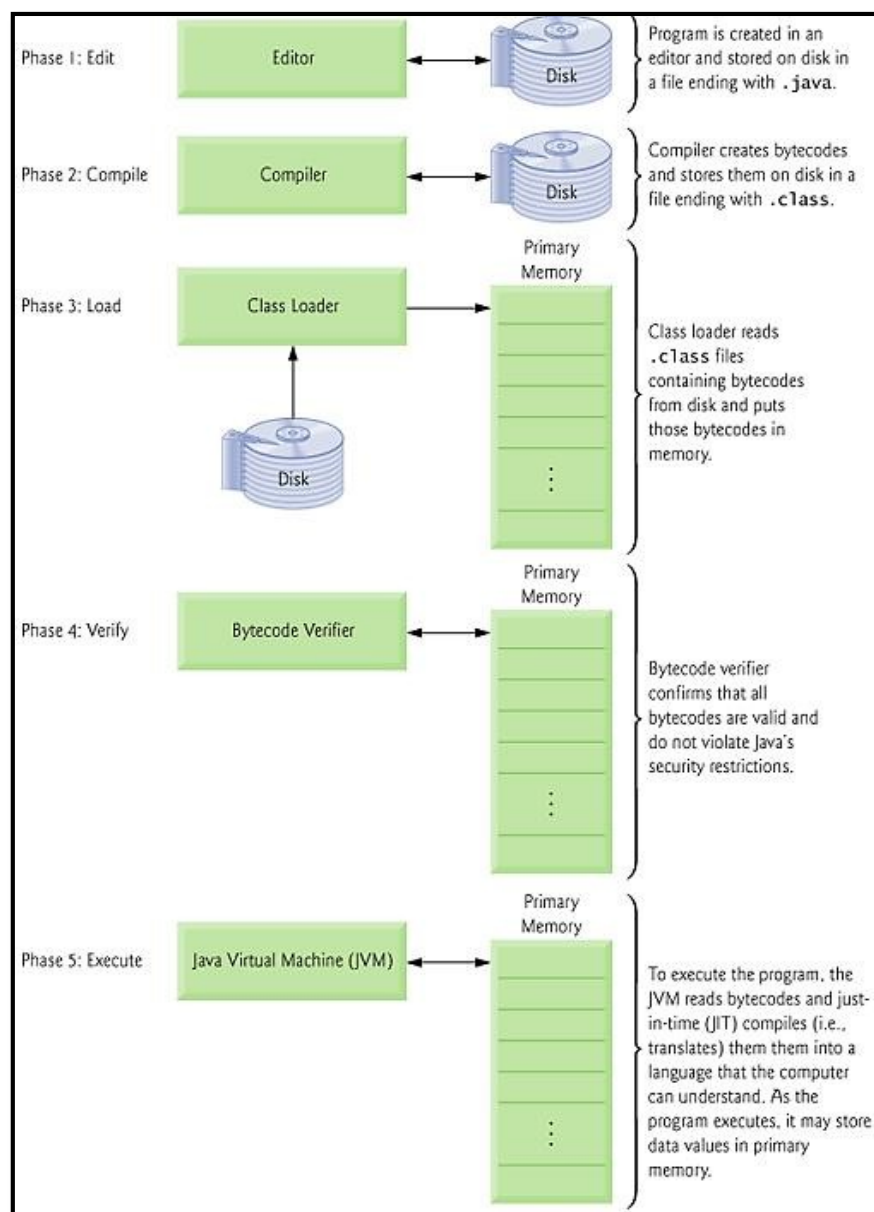
INTRODUCTION TO JAVA

JAVA

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. James Gosling initiated the Java language project in June 1991.

Java can be used to create two types of programs: applications and applets. An application is a program that runs on your computer, under the operating system of that computer. An applet is an application designed to be transmitted over the Internet and executed by a Java compatible Web browser.

TYPICAL JAVA DEVELOPMENT ENVIRONMENT



Phase 1: Creating a Program

Phase 1 consists of editing a file with an editor program. You type a Java program (typically referred to as source code) using the editor, make any necessary corrections and save the program. A file name ending with the .java extension indicates that the file contains java source code.



Phase 2: Compiling a Java Program into Bytecodes

The Java compiler compiles a program. If the program compiles successfully, the compiler produces a .class file. The Java compiler translates Java source code into bytecodes. Bytecodes are executed by the Java Virtual Machine (JVM).



Phase 3: Loading a Program into Memory

The JVM places the program in memory to execute it—this is known as loading. The JVM's class loader takes the .class files containing the program's bytecodes and transfers them to primary memory.



Phase 4: Bytecode Verification

The classes are loaded, the bytecode verifier examines their bytecodes to ensure that they're valid and do not violate Java's security restrictions.



Phase 5: Execution

The JVM executes the program's bytecodes, thus performing the actions specified by the program.

Difference between JDK, JRE, and JVM

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed.

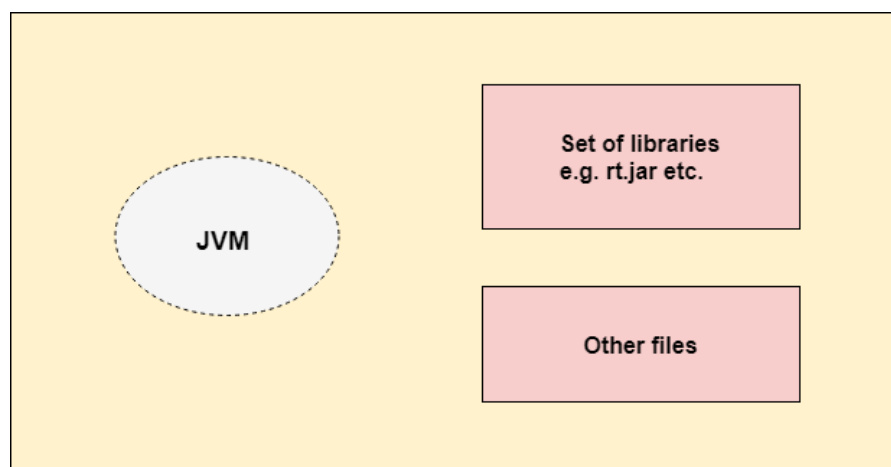
JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other. However, Java is platform independent.

The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

JRE is an acronym for Java Runtime Environment. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

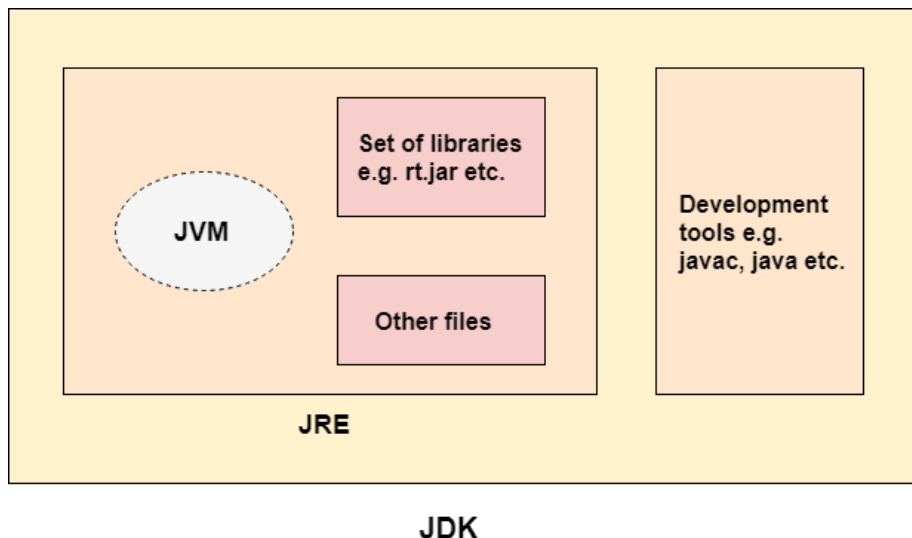


JRE

JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



IDE:

NetBeans is an integrated development environment (IDE) for Java. NetBeans allows applications to be developed from a set of modular software components called modules. NetBeans runs on Windows, macOS, and Linux. In addition to Java development, it has extensions for other languages like PHP, C, C++, HTML5, and JavaScript.

IntelliJ IDEA overview

IntelliJ IDEA is an *Integrated Development Environment (IDE)* for JVM languages designed to maximize developer productivity. It does the routine and repetitive tasks for you by providing clever code completion, static code analysis, and refactoring, and lets you focus on the bright side of software development, making it not only productive but also an enjoyable experience.

Supported languages

Development of modern applications involves using multiple languages, tools, frameworks, and technologies. IntelliJ IDEA is designed as an IDE for JVM languages but numerous plugins can extend it to provide a polyglot experience.

JVM languages

Use IntelliJ IDEA to develop applications in the following languages that can be compiled into the JVM bytecode, namely:

- Java
- Kotlin
- Scala
- Groovy

Install IntelliJ IDEA

IntelliJ IDEA is a cross-platform IDE that provides consistent experience on the Windows, macOS, and Linux operating systems.

IntelliJ IDEA is available in the following editions:

- *Community Edition* is free and open-source, licensed under Apache 2.0. It provides all the basic features for JVM and Android development.
- *IntelliJ IDEA Ultimate* is commercial, distributed with a 30-day trial period. It provides additional tools and features for web and enterprise development.
-

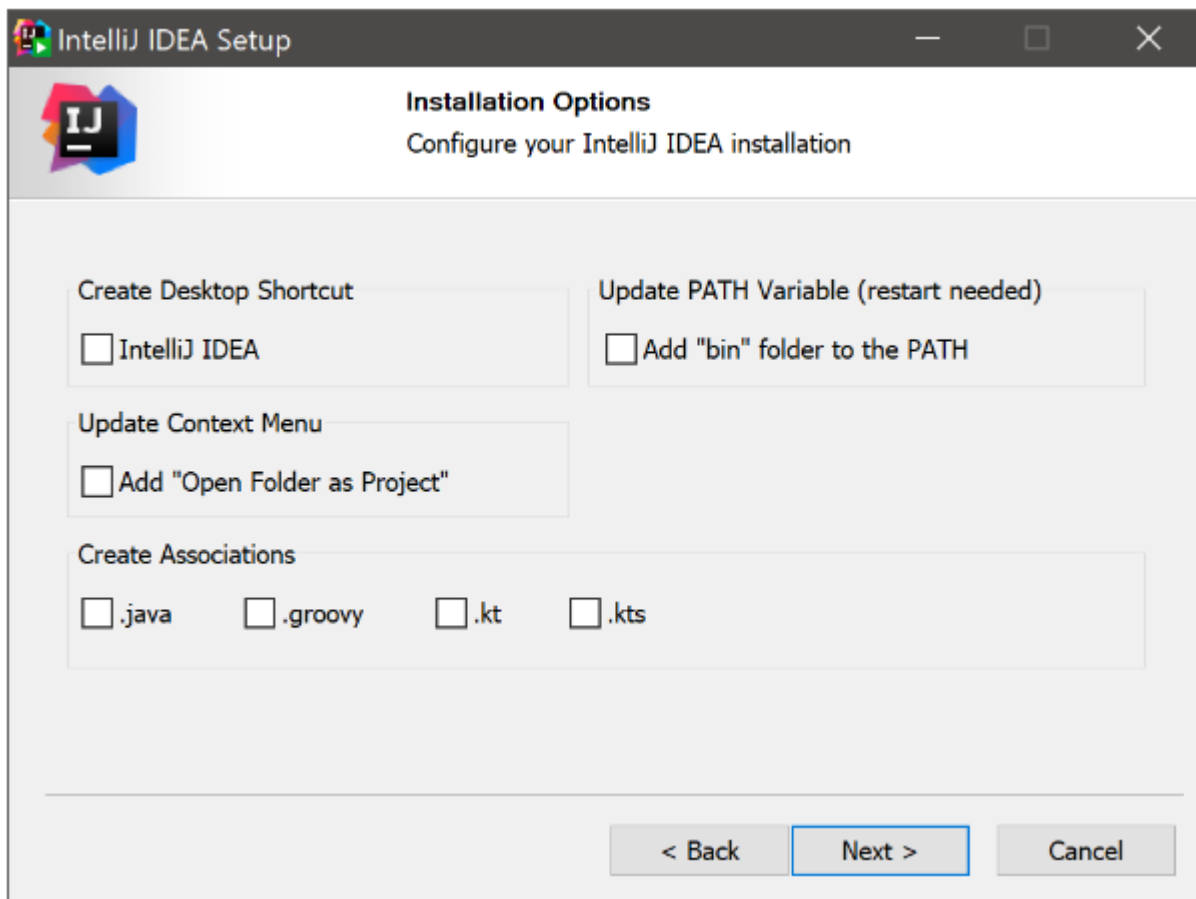
System requirements

Requirement	Minimum	Recommended
RAM	2 GB of free RAM	8 GB of total system RAM
CPU	Any modern CPU	Multi-core CPU. IntelliJ IDEA supports multithreading for different operations and processes making it faster the more CPU cores it can use.
Disk space	2.5 GB and another 1 GB for caches	SSD drive with at least 5 GB of free space
Monitor resolution	1024×768	1920×1080

1. [Download the installer](#) .exe.
2. Run the installer and follow the wizard steps.

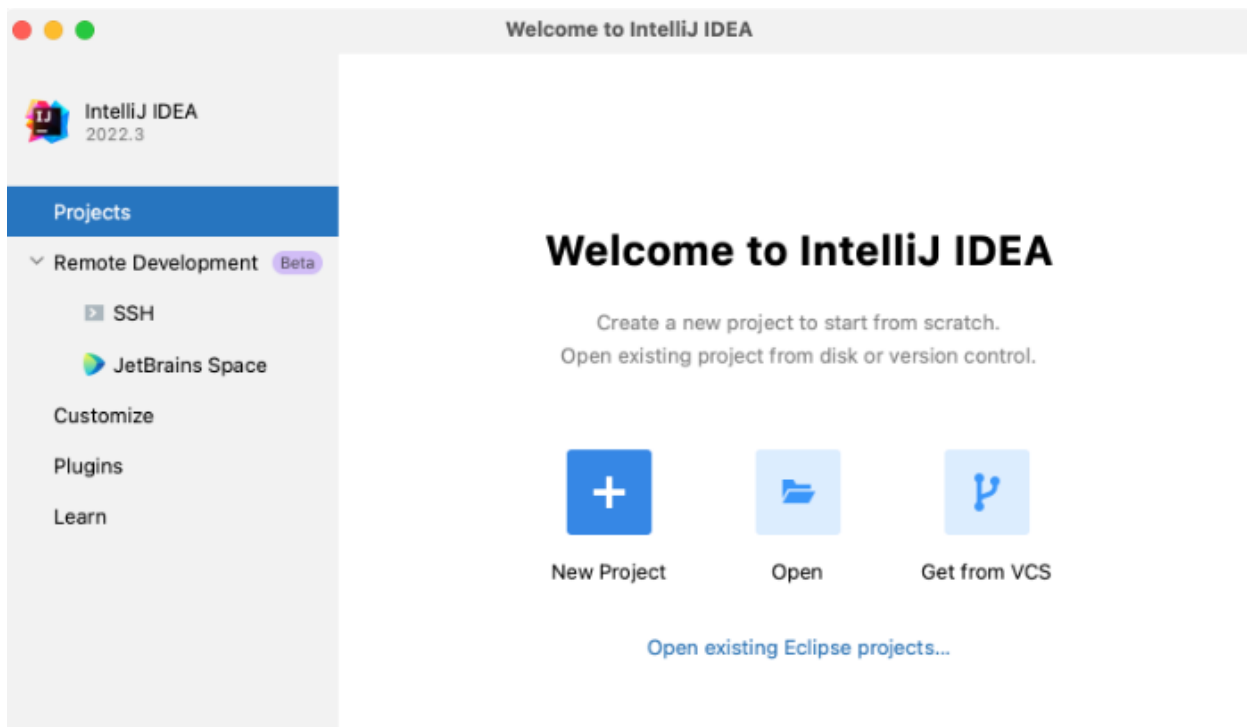
On the Installation Options step, you can configure the following:

- Create a desktop shortcut for launching IntelliJ IDEA.
- Add the directory with IntelliJ IDEA [command-line launchers](#) to the `PATH` environment variable to be able to run them from any working directory in the Command Prompt.
- Add the Open Folder as Project action to the system context menu (when you right-click a folder).
- Associate specific file extensions with IntelliJ IDEA to open them with a double-click.



To run IntelliJ IDEA, find it in the Windows Start menu or use the desktop shortcut. You can also run the launcher batch script or executable in the installation directory under **bin**.

Once you launch IntelliJ IDEA, you will see the Welcome screen, the starting point to your work with the IDE, and configuring its settings. This screen also appears when you close all opened projects. Use the tabs on the left side to switch to the specific welcome dialog.



From the Welcome to IntelliJ IDEA dialog, you can do the following:

- Create a new project


Launch IntelliJ IDEA.

If the Welcome screen opens, click New Project. Otherwise, from the main menu, select File | New | Project.

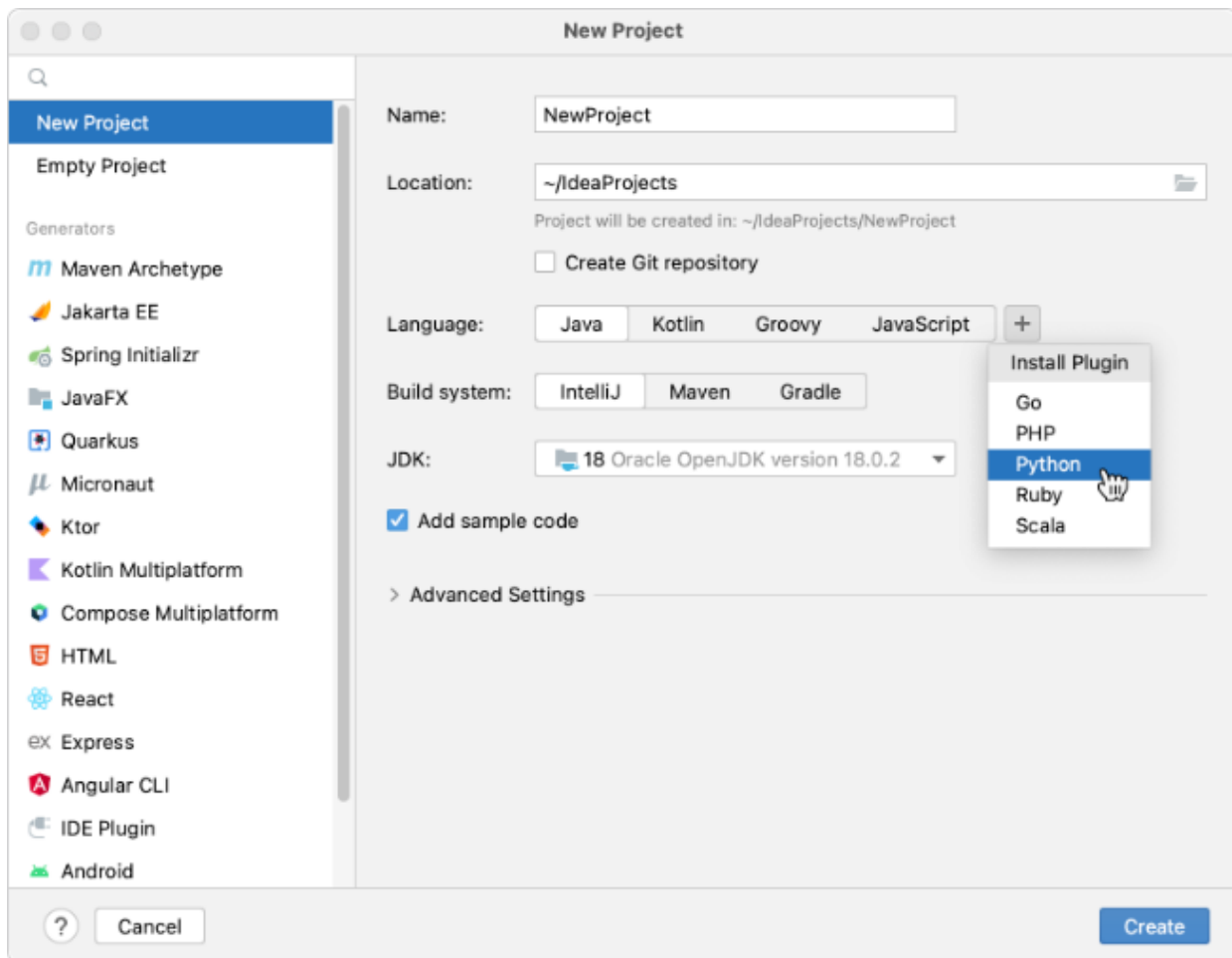
- From the list on the left, select New Project.
- Name the new project and change its location if necessary.
- Select the Create Git repository checkbox to place the new project under version control.

You will be able to do it later at any time.

- From the Language list, select the language that you want to use in your application.

If you want to use a language that is not available in IntelliJ IDEA out of the box (for example, Python or PHP), click the  button and select the necessary option.

The IDE will open a dialog in which you can select and install the necessary language plugin. After that, you can close the dialog and keep configuring the new project.



1. Select the build system that you want to use in your project: the native IntelliJ builder, Maven, or Gradle.

For Gradle, you will also need to select a language for the build script: Groovy or Kotlin.

2. From the JDK list, select the JDK that you want to use in your project.

If the JDK is installed on your computer, but not defined in the IDE, select Add JDK and specify the path to the JDK home directory.

If you don't have the necessary JDK on your computer, select Download JDK.

3. Enable the Add sample code option to create a class with a sample HelloWorld application.

Now let's create a simple program.

```
// MyFirstProgram In Java, any line starting with // is a comment.
```

`public class HelloWorld {` **In Java, every application begins with a class definition. In the program, HelloWorld is the name of the class. Every Java application has a class definition, and the name of the class should match the filename in Java.**

`public static void main(String[] args) {` **This is the main method.**

- **class keyword is used to declare a class in Java.**
- **public keyword is an access modifier that represents visibility.**
- **static is a keyword. The advantage of the static method is that there is no need to create an object to invoke the static method.**
- **The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.**
- **void is the return type of the method. It means it doesn't return any value.**
- **main represents the starting point of the program.**
- **String[] args means an array of sequence of characters (Strings) that are passed to the "main" function.**
- **When you execute a Java program via the command line: java MyProgram "This is just a test" Therefore, the array will store: ["This", "is", "just", "a", "test"]**

`System.out.println("Hello, World!");` **System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class.**

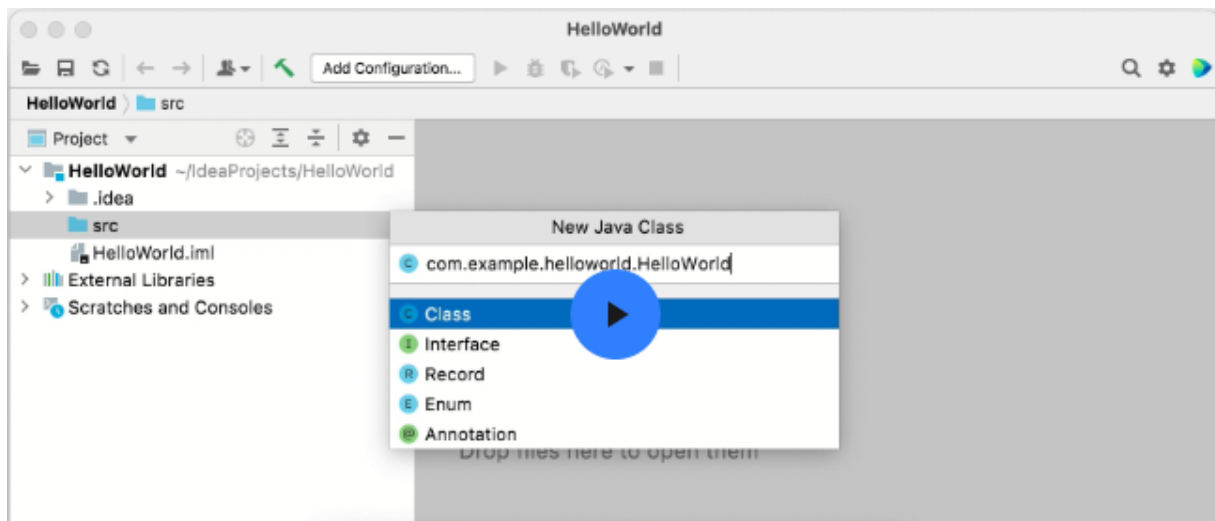
`}`

Create a package and a class

Packages are used for grouping together classes that belong to the same category or provide similar functionality, for structuring and organizing large applications with hundreds of classes.

1. In the **Project** tool window, right-click the **src** folder, select **New** (or press Alt+Insert), and then select **Java Class**.
2. In the Name field, type `com.example.helloworld.HelloWorld` and click OK.

IntelliJ IDEA creates the `com.example.helloworld` package and the `HelloWorld` class.



Together with the file, IntelliJ IDEA has automatically generated some contents for your class. In this case, the IDE has inserted the package statement and the class declaration.

This is done by means of file templates. Depending on the type of the file that you create, the IDE inserts initial code and formatting that is expected to be in all files of that type.

Write the code

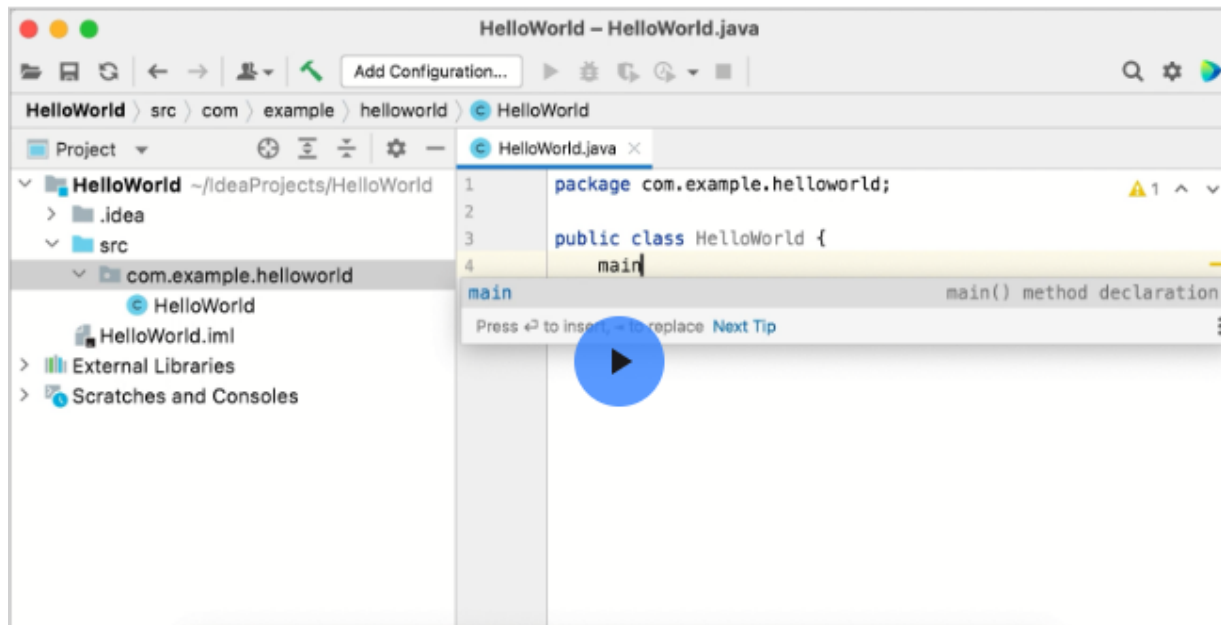
Add the `main()` method using live templates

1. Place the caret at the class declaration string after the opening bracket `{` and press Shift+Enter.

In contrast to Enter, Shift+Enter starts a new line without breaking the current one.

2. Type `main` and select the template that inserts the `main()` method declaration.

As you type, IntelliJ IDEA suggests various constructs that can be used in the current context. You can see the list of available live templates using Ctrl+J.



Live templates are code snippets that you can insert into your code. `main` is one of such snippets. Usually, live templates contain blocks of code that you use most often. Using them can save you some time as you don't have to type the same code over and over again.

Call the `println()` method using code completion

After the `main()` method declaration, IntelliJ IDEA automatically places the caret at the next line. Let's call a method that prints some text to the standard system output.

1. Type `Sy` and select the `System` class from the list of code completion suggestions (it's from the standard `java.lang` package).

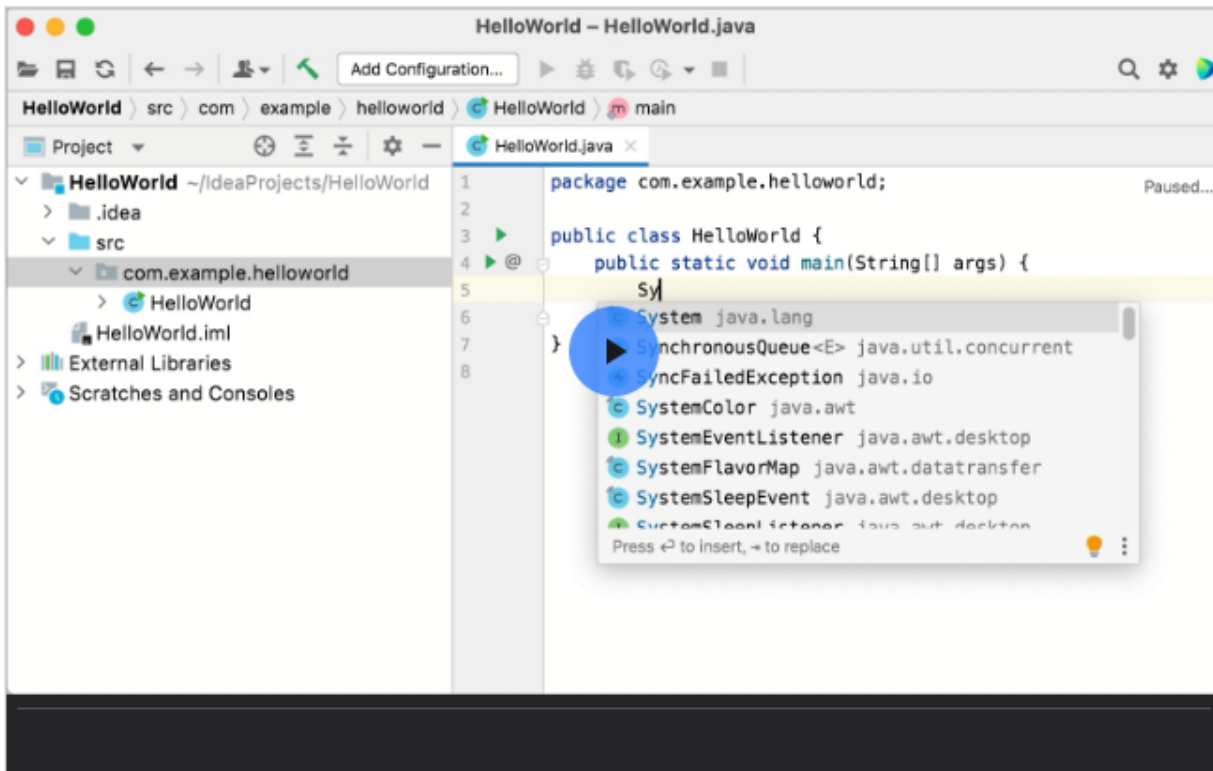
Press `Ctrl+.` to insert the selection with a trailing period.

2. Type `o`, select `out`, and press `Ctrl+.` again.
3. Type `p`, select the `println(String x)` method, and press `Enter`.

IntelliJ IDEA shows you the types of parameters that can be used in the current context. This information is for your reference.

4. Type `"`. The second quotation mark is inserted automatically, and the caret is placed between the quotation marks. Type `Hello, World!`

Basic code completion analyses the context around the current caret position and provides suggestions as you type. You can open the completion list manually by pressing `Ctrl+Space`.



Call the println() method using a live template

1.

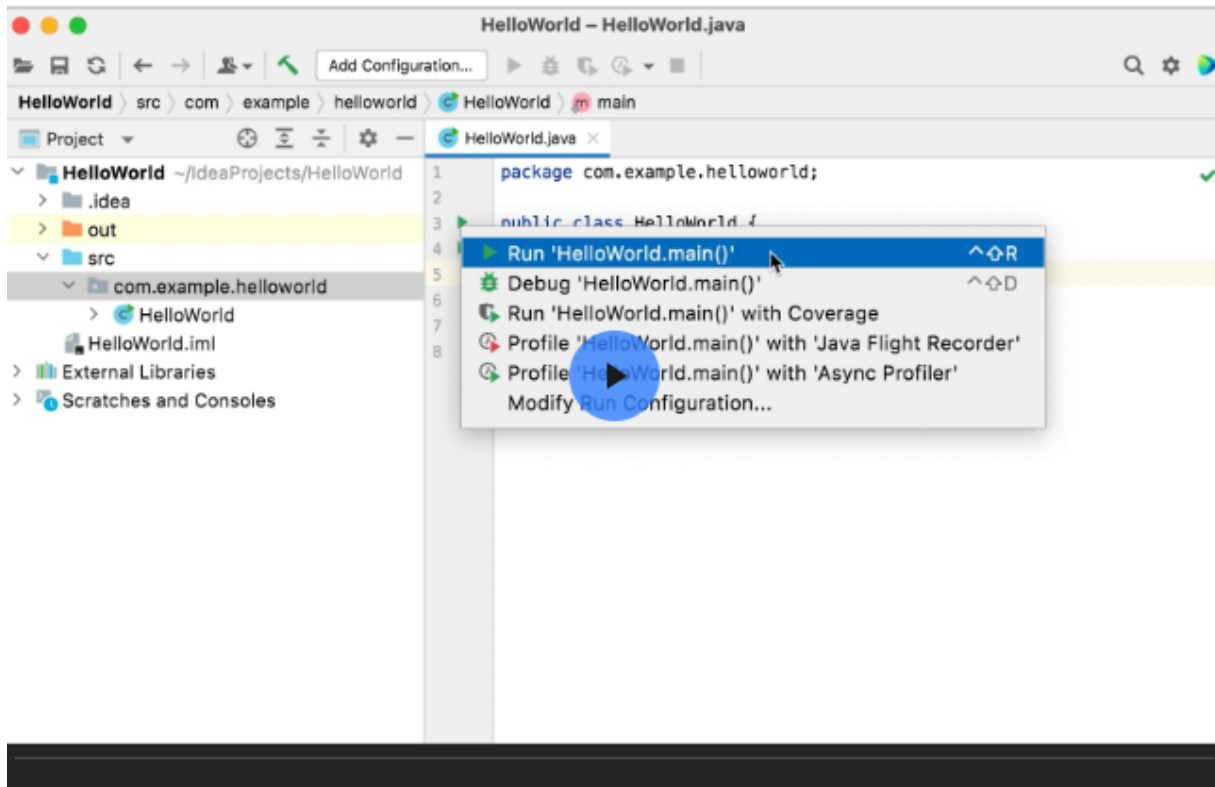
Build and run the application

Valid Java classes can be compiled into bytecode. You can compile and run classes with the `main()` method right from the editor using the green arrow icon in the gutter.

1. Click in the gutter and select Run 'HelloWorld.main()' in the popup. The IDE starts compiling your code.
2. When the compilation is complete, the Run tool window opens at the bottom of the screen.

The first line shows the command that IntelliJ IDEA used to run the compiled class. The second line shows the program output: `Hello, World!`. And the last line shows the exit code 0, which indicates that it exited successfully.

If your code is not correct, and the IDE can't compile it, the Run tool window will display the corresponding exit code.



When you click Run, IntelliJ IDEA creates a special run configuration that performs a series of actions. First, it builds your application. On this stage, javac compiles your source code into JVM bytecode.

Once javac finishes compilation, it places the compiled bytecode to the **out** directory, which is highlighted with yellow in the Project tool window.

After that, the JVM runs the bytecode.

DATA TYPES

Data types are divided into two groups:

Primitive data types - includes byte, short, int, long, float, double, boolean and char

Non-primitive data types - such as String, Arrays and Classes

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

```
public class Output {  
    public static void main(String[] args) {  
  
        int myNum = 5;           // Integer (whole number)  
        float myFloatNum = 5.99f; // Floating point number  
        char myLetter = 'D';     // Character  
        boolean myBool = true;   // Boolean  
        String myText = "Hello"; // String  
        byte bNum = 100;         // Byte  
        short sNum = 5000;       // Short  
        long lNum = 15000000000L; // Long  
  
        System.out.println("You entered integer" + " " + myNum);  
        System.out.println("You entered float" + " " + myFloatNum);  
        System.out.println("You entered character" + " " + myLetter);  
        System.out.println("You entered boolean" + " " + myBool);  
        System.out.println("You entered string" + " " + myText);  
        System.out.println("You entered Byte" + " " + bNum);  
        System.out.println("You entered short" + " " + sNum);  
        System.out.println("You entered long" + " " + lNum);  
  
    }  
}
```

Output - Run (OOLab1) %

```
--- exec-maven-plugin:3.0.0:run  
You entered integer 5  
You entered float 5.99  
You entered character D  
You entered boolean true  
You entered string Hello  
You entered Byte 100  
You entered short 5000  
You entered long 15000000000  
-----  
BUILD SUCCESS
```

INPUT AND OUTPUT IN JAVA

Java output:

In Java, you can simply use

System.out.println(); or

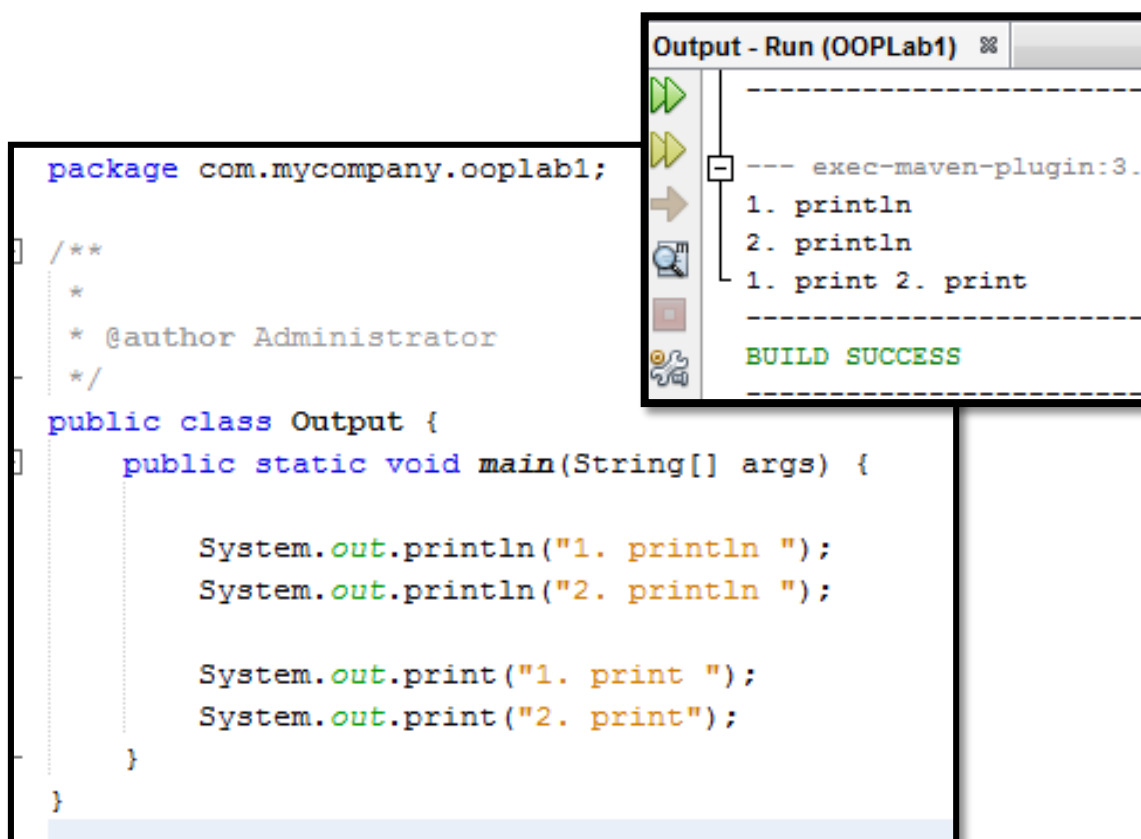
System.out.print(); or

System.out.printf();

to send output to standard output (screen). Here, System is a class out is a public static field: it accepts output data.

Difference between println(), print() and printf()

- print() - It prints string inside the quotes.
- println() - It prints string inside the quotes similar like print() method. Then the cursor moves to the beginning of the next line.
- printf() - It provides string formatting (similar to printf in C/C++ programming).



The screenshot displays an IDE with a Java source file on the left and an output window on the right. The source file, located at `package com.mycompany.ooplal1;`, contains a `public class Output` with a `main` method. The `main` method uses `System.out.println` and `System.out.print` to output specific strings. The output window, titled "Output - Run (OOPLab1)", shows the execution results, including a build success message and the printed output.

```
package com.mycompany.ooplal1;

/**
 *
 * @author Administrator
 */
public class Output {
    public static void main(String[] args) {

        System.out.println("1. println ");
        System.out.println("2. println ");

        System.out.print("1. print ");
        System.out.print("2. print");

    }
}
```

Output - Run (OOPLab1) ✖

```
--- exec-maven-plugin:3.
1. println
2. println
1. print 2. print

BUILD SUCCESS
```


FORMATTING OUTPUT WITH PRINTF() IN JAVA

Conversion characters are only valid for certain data types. Here are some common ones:

Specifier	Explanation
%c	Format characters
%d	Format decimal (integer) numbers (base 10)
%e	Format exponential floating-point numbers
%f	Format floating-point numbers
%i	Format integers (base 10)
%o	Format octal numbers (base 8)
%s	Format string of characters
%u	Format unsigned decimal (integer) numbers
%x	Format numbers in hexadecimal (base 16)
%n	add a new line character

Float and Double Formatting

To format a float number, we'll need the f format:

System.out.printf("%f%n", 5.1473); which will output: **5.147300**

To control the precision: **System.out.printf("%5.2f%n", 5.1473);** Here we define the width of our number as 5, and the length of the decimal part is 2: **'5.15'**.

Integer Formatting

The printf() method accepts all the integers available in the language — byte, short, int, long, and BigInteger if we use %d:

System.out.printf("simple integer: %d%n", 10000L);

With the help of the d character, we'll have this result: **simple integer: 10000**

String Formatting

To format a simple string, we'll use the %s combination. Additionally, we can make the string uppercase:

```
printf("%s' %n", "Java");  
printf("%S' %n", "JAVA");
```

And this is the output:

```
'Java'  
'JAVA'
```

Char Formatting

The result of %c is a Unicode character:

```
System.out.printf("%c%n", 's');
```

```
System.out.printf("%C%n", 's');
```

The capital letter C will uppercase the result:

s

S

Input from user in Java

Java Scanner Class

Java Scanner class allows the user to take input from the console. It belongs to java.util package.

Syntax

Scanner sc = new Scanner(System.in);

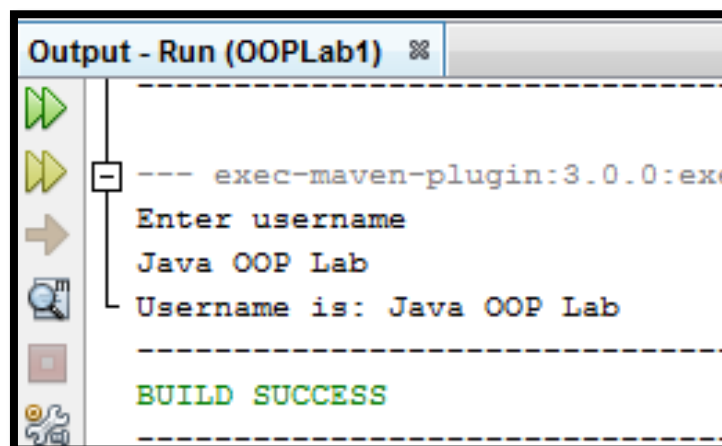
The above statement creates a constructor of the Scanner class having System.in as an argument. It means it is going to read from the standard input stream of the program. The java.util package should be imported while using Scanner class.

Methods of Java Scanner Class

Method	Description
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.
BigInteger nextBigInteger()	It is used to scan the next token of the input as a BigInteger.
BigDecimal nextBigDecimal()	It is used to scan the next token of the input as a BigDecimal.

Example: Input your name

```
package com.mycompany.ooplal1;  
import java.util.Scanner; // import the Scanner class  
  
class Input1 {  
    public static void main(String[] args) {  
        Scanner myObj = new Scanner(System.in);  
        String userName;  
  
        // Enter username and press Enter  
        System.out.println("Enter username");  
        userName = myObj.nextLine();  
  
        System.out.println("Username is: " + userName);  
    }  
}
```



```
Output - Run (OOPLab1) ✖  
--- exec-maven-plugin:3.0.0:exe  
Enter username  
Java OOP Lab  
Username is: Java OOP Lab  
---  
BUILD SUCCESS  
---
```

Example: Input integers

```
package com.mycompany.ooplal1;
import java.util.Scanner; // import the Scanner class

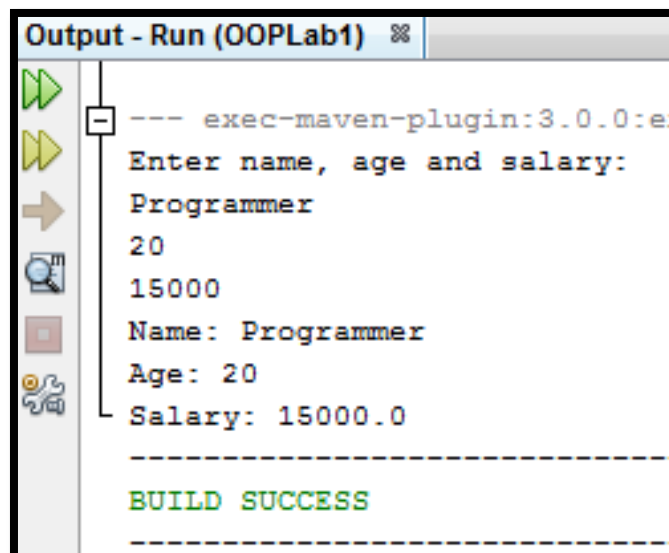
class Input1 {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);

        System.out.println("Enter name, age and salary:");

        // String input
        String name = myObj.nextLine();

        // Numerical input
        int age = myObj.nextInt();
        double salary = myObj.nextDouble();

        // Output input by user
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Salary: " + salary);
    }
}
```



```
Output - Run (OOPLab1)
--- exec-maven-plugin:3.0.0:ex
Enter name, age and salary:
Programmer
20
15000
Name: Programmer
Age: 20
Salary: 15000.0
BUILD SUCCESS
```

Example: Adding two numbers

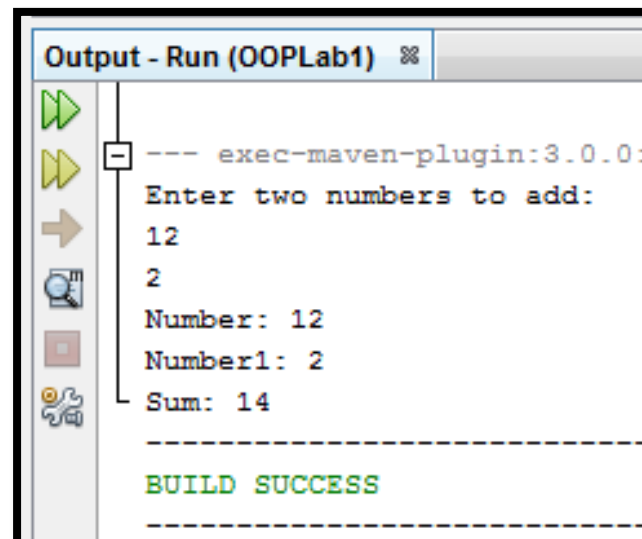
```
package com.mycompany.ooplal1;
import java.util.Scanner; // import the Scanner class

class Input1 {
    public static void main(String[] args) {
        Scanner myObj = new Scanner(System.in);

        System.out.println("Enter two numbers to add:");

        // Numerical input
        int number = myObj.nextInt();
        int number1 = myObj.nextInt();
        int sum = number + number1;

        // Output input by user
        System.out.println("Number: " + number);
        System.out.println("Number1: " + number1);
        System.out.println("Sum: " + sum);
    }
}
```



```
--- exec-maven-plugin:3.0.0:
Enter two numbers to add:
12
2
Number: 12
Number1: 2
Sum: 14
-----
BUILD SUCCESS
-----
```

LAB#01 EXERCISES

QUESTION#1

Take 3 variables x, y and z as input from user and compute the outcome of the following expression:

$$\frac{3x + y}{z + 2}$$

QUESTION#2

a) Write a program that displays the result of the following expression upto 2 decimal places:

$$\frac{9.5 \times 4.5 - 2.5 \times 3}{45.5 - 3.5}.$$

QUESTION#3

Write a Java program that inputs three integers from the user and displays the sum, average, and product of these numbers.

QUESTION#4

Write a Java program to compute the output of specified formula.

Specified Formula:

$$4.0 * (1 - (1.0/3) + (1.0/5) - (1.0/7) + (1.0/9) - (1.0/11))$$