**1. Install and Setup java environment**

The Java Development Kit (**JDK**) is software used for Java programming, along with

the Java Virtual Machine (**JVM**) and the Java Runtime Environment (**JRE**). The JDK

includes the compiler and class libraries, allowing developers to create Java programs

executable by the JVM and JRE.

**In this tutorial, you will learn to install the Java Development Kit on Windows.**

**Prerequisites**







A system running Windows 10.

A network connection.

Administrator privileges.

**Check if Java Is Installed**

Before installing the Java Development Kit, [check if a Java version is already installed on](https://phoenixnap.com/kb/check-java-version-on-mac-windows)

[Windows. F](https://phoenixnap.com/kb/check-java-version-on-mac-windows)ollow the steps below:

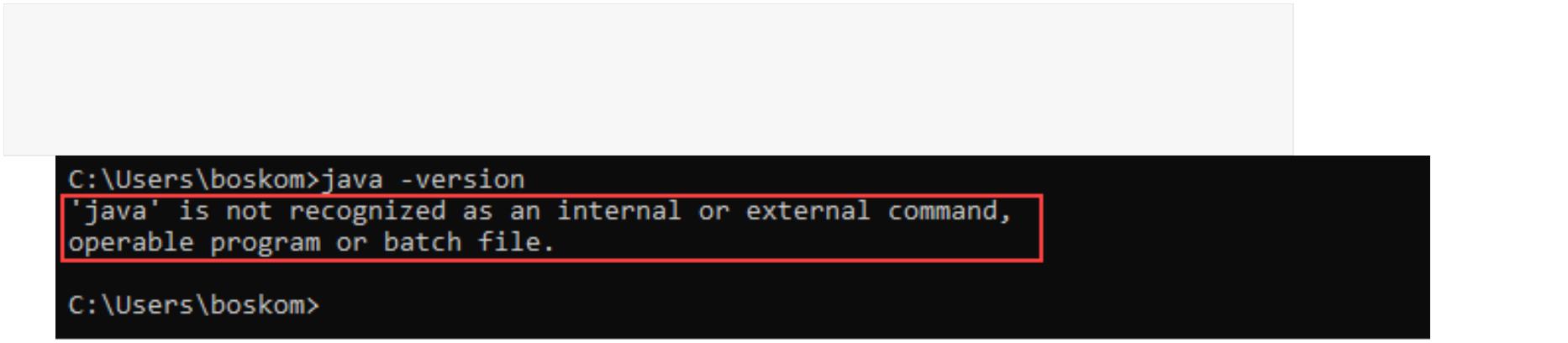
1. Open a command prompt by typing *cmd* in the search bar and press **Enter**.

2. Run the following command:

java -version

The command outputs the Java version on your system. If Java isn't installed, the output is a

message stating that Java isn't recognized as an internal or external command.

image1

**Download Java for Windows 10**

Download the latest Java Development Kit installation file for Windows 10 to have the latest

features and bug fixes.

1. Using your preferred web browser, navigate to the [Oracle Java Downloads page.](https://www.oracle.com/java/technologies/downloads/#jdk17-windows)

2. On the *Downloads* page, click the **x64 Installer** download link under

the **Windows** category. At the time of writing this article, Java version 17 is the latest

long-term support Java version.

Wait for the download to complete.

**Install Java on Windows 10**

After downloading the installation file, proceed with installing Java on your Windows

system.

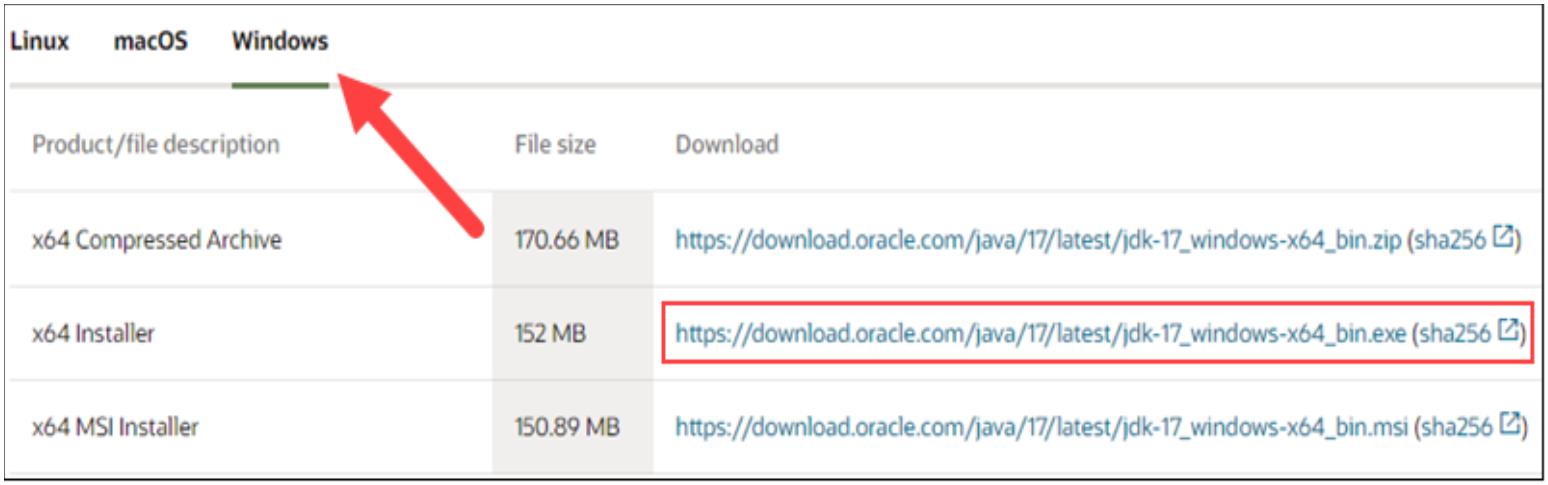
Follow the steps below:

**Step 1: Run the Downloaded File**

Double-click the **downloaded file** to start the installation.

**Step 2: Configure the Installation Wizard**

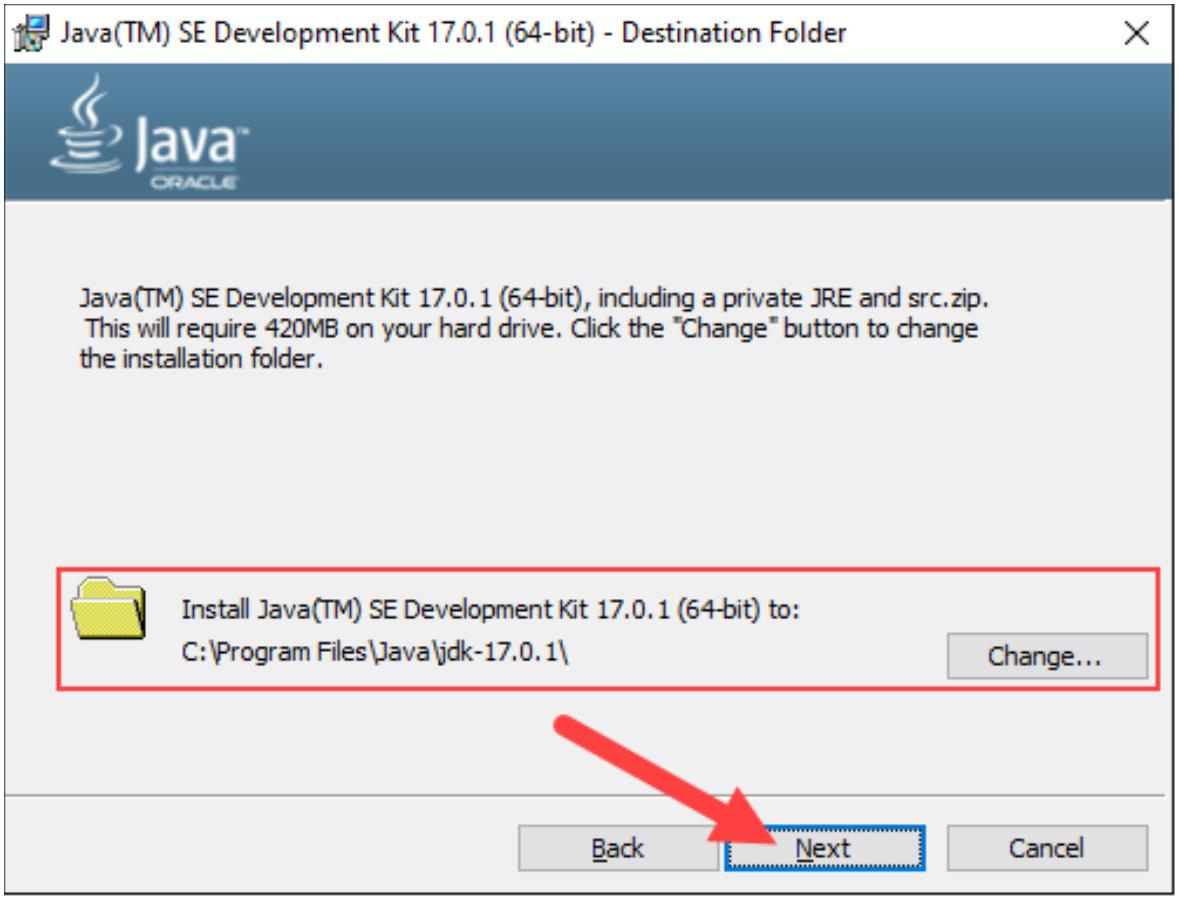
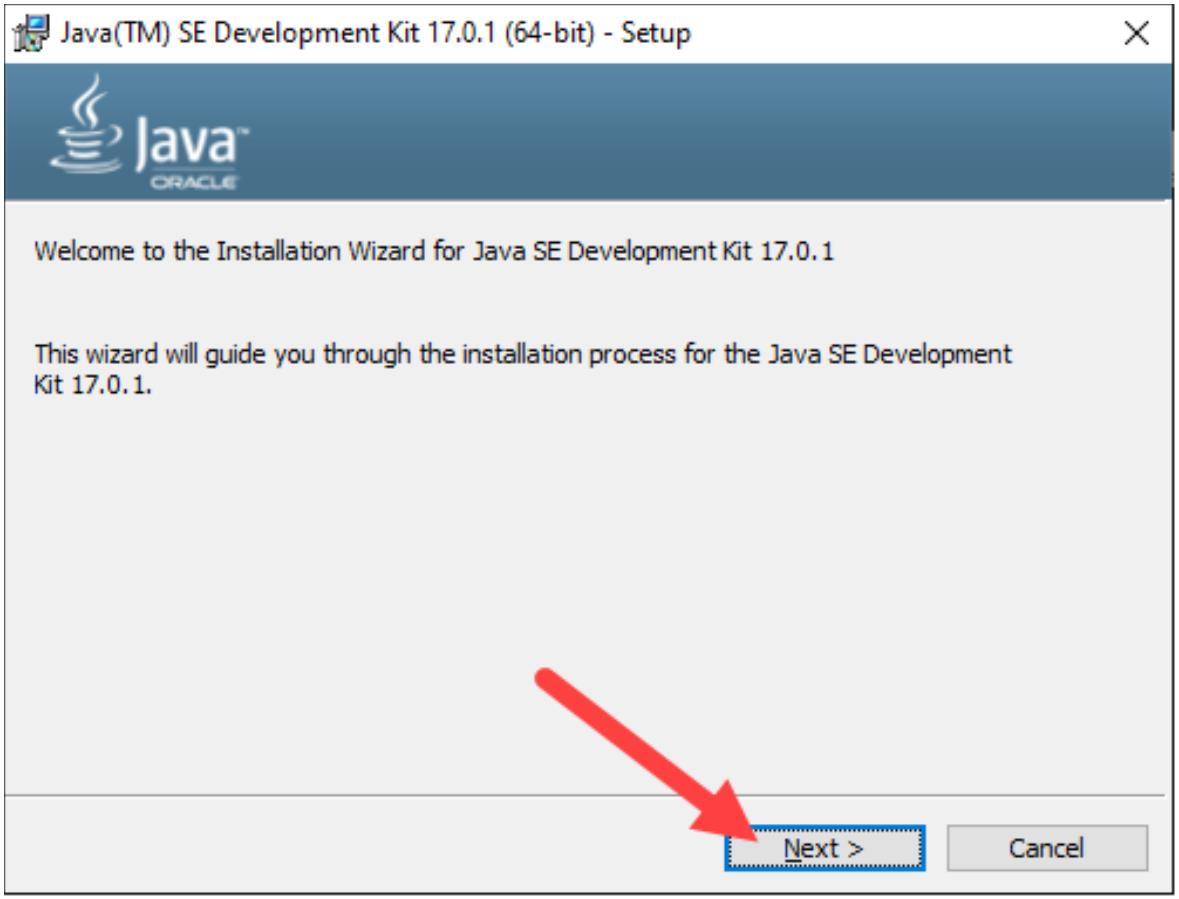
After running the installation file, the installation wizard welcome screen appears.

image3

1. Click **Next** to proceed to the next step.

2. Choose the destination folder for the Java installation files or stick to the default path.

Click **Next** to proceed.



3. Wait for the wizard to finish the installation process until the *Successfully*

*Installed* message appears. Click **Close** to exit the wizard.

**Set Environmental Variables in Java**

Set Java [environment variables](https://phoenixnap.com/kb/windows-set-environment-variable) to enable program compiling from any directory. To do so,

follow the steps below:

**Step 1: Add Java to System Variables**

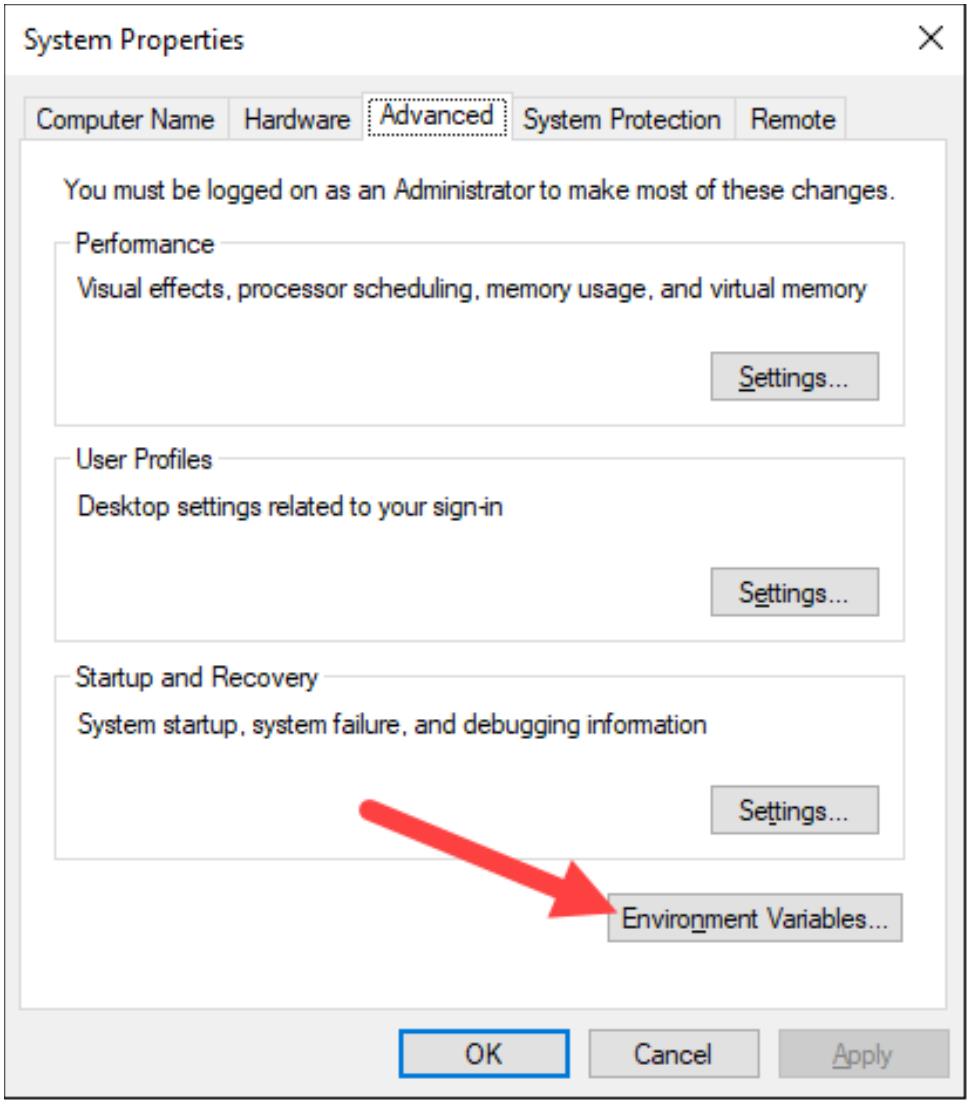
1. Open the **Start** menu and search for *environment variables*.

2. Select the **Edit the system environment variables** result.

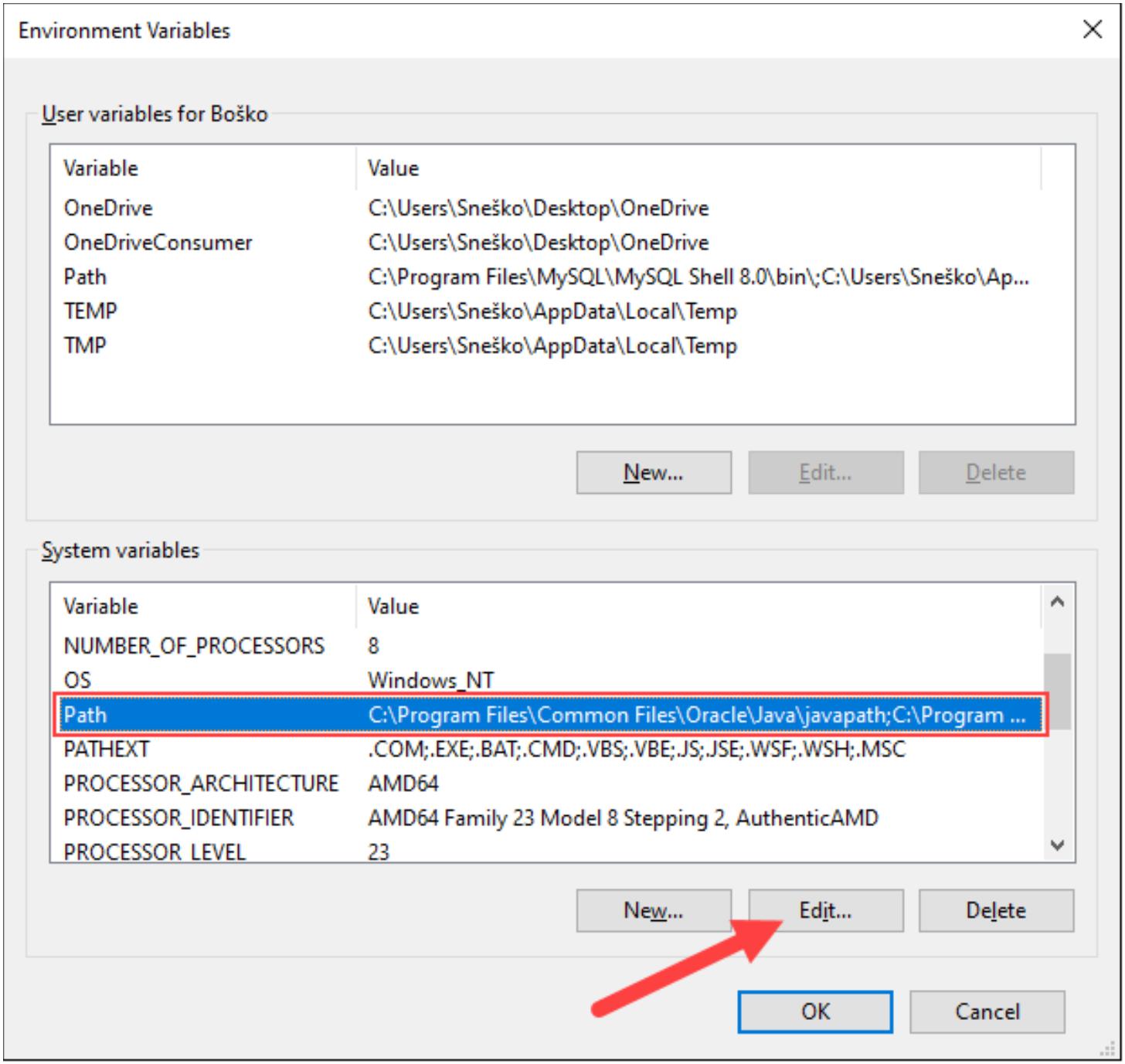
image7

3. In the *System Properties* window, under the *Advanced* tab, click **Environment**

**Variables…**



4. Under the *System variables* category, select the **Path** variable and click **Edit**:



5. Click the **New** button and enter the path to the Java bin directory:

**Note:** The default path is usually *C:\Program Files\Java\jdk-17.0.1\bin*.

6. Click **OK** to save the changes and exit the variable editing window.

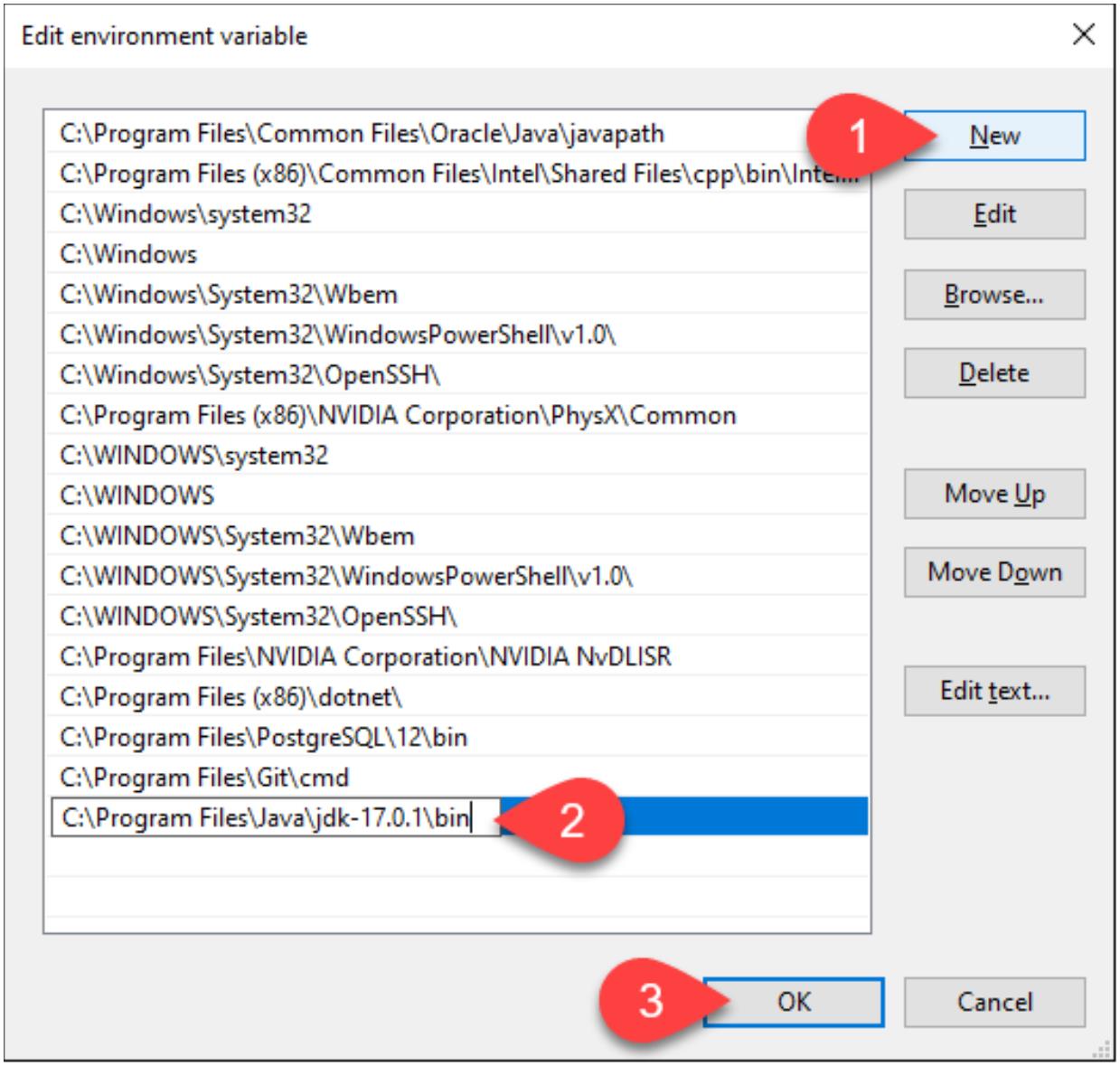
**Step 2: Add JAVA\_HOME Variable**

Some applications require the **JAVA\_HOME** variable. Follow the steps below to create the

variable:

1. In the *Environment Variables* window, under the *System variables* category, click

the **New…** button to create a new variable.

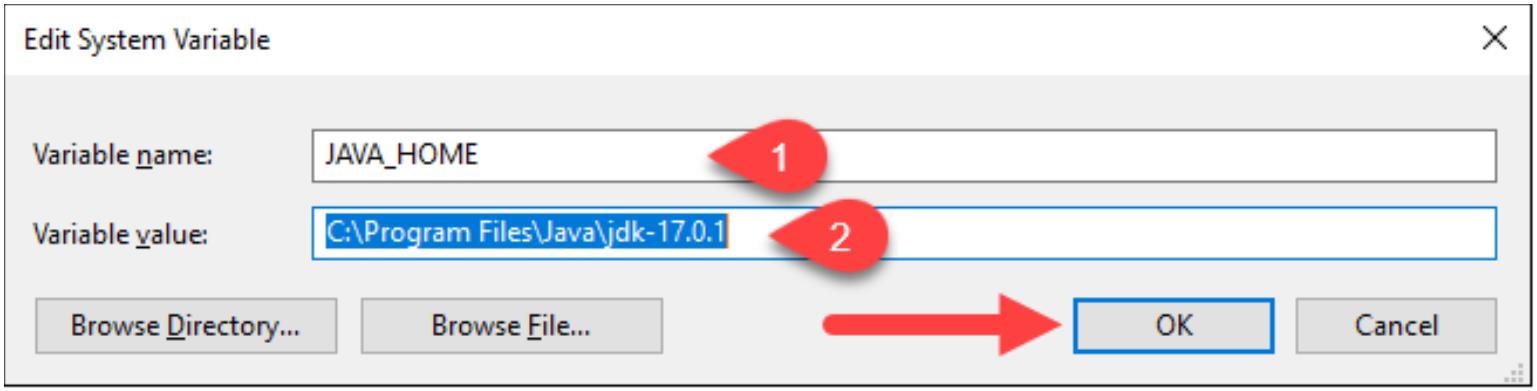
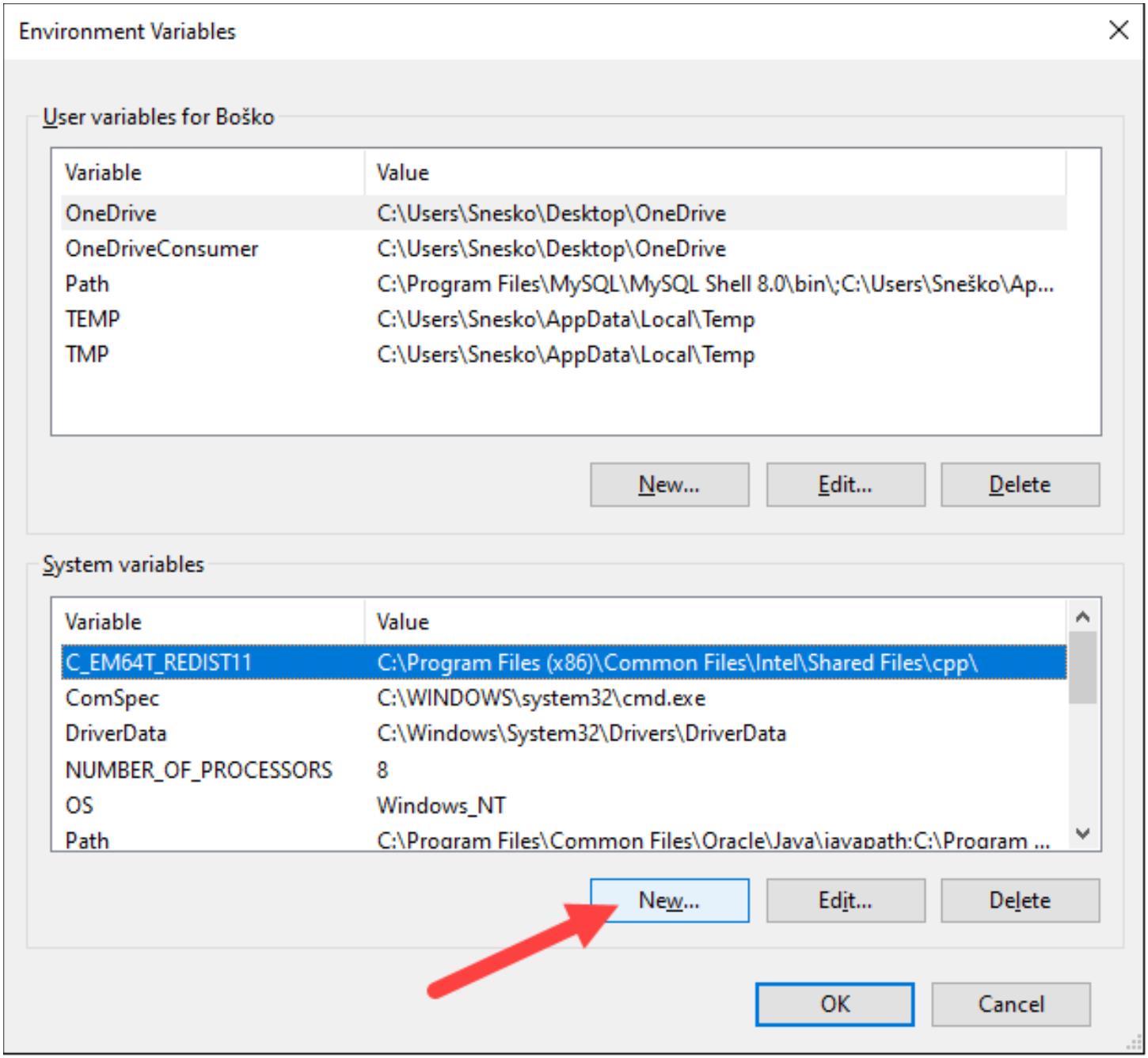


2. Name the variable as ***JAVA\_HOME***.

3. In the variable value field, paste the path to your Java jdk directory and click **OK**.

4. Confirm the changes by clicking **OK** in the *Environment Variables* and *System*

*properties* windows.



**Test the Java Installation**

Run the **java -version** command in the command prompt to make sure Java installed correctly:

If installed correctly, the command outputs the Java version. Make sure everything works by

writing a simple program and compiling it. Follow the steps below:

**Step 1: Write a Test Java Script**

1. Open a text editor such as Notepad++ and create a new file.

2. Enter the following lines of code and click **Save**:

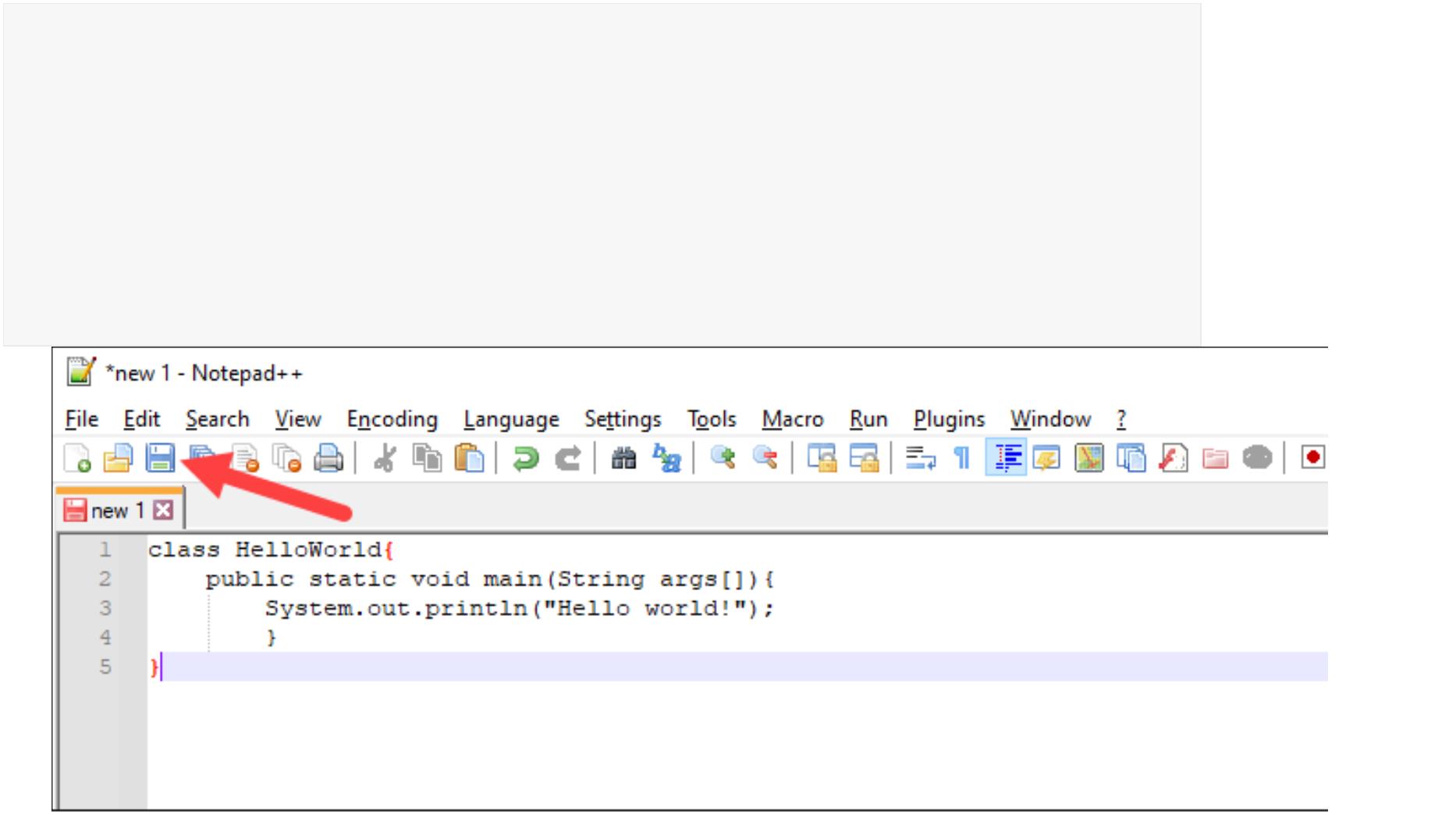
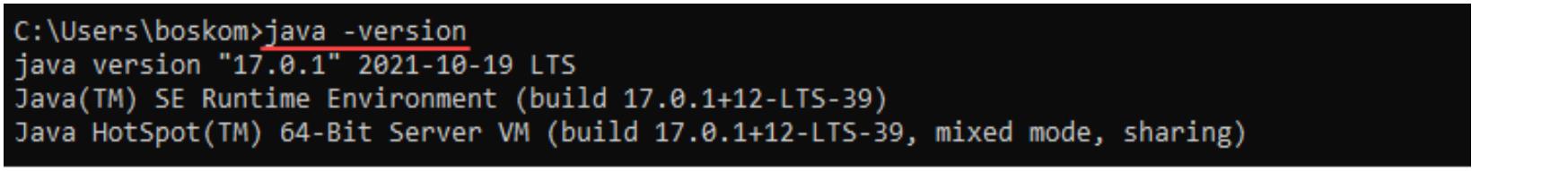
class HelloWorld{

public static void main(String args[]){

System.out.println("Hello world!");

}

}

image15

3. Name the file and save it as a **Java source file (\*.java)**.

**Note:** When using Notepad, select **All files** for the Save as type option and add

the *.java* extension to the file name.

**Step 2: Compile the Test Java Script**

1. In the command prompt, change the directory to the file's location and use the following

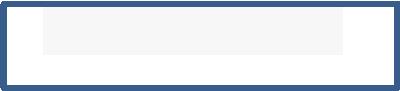
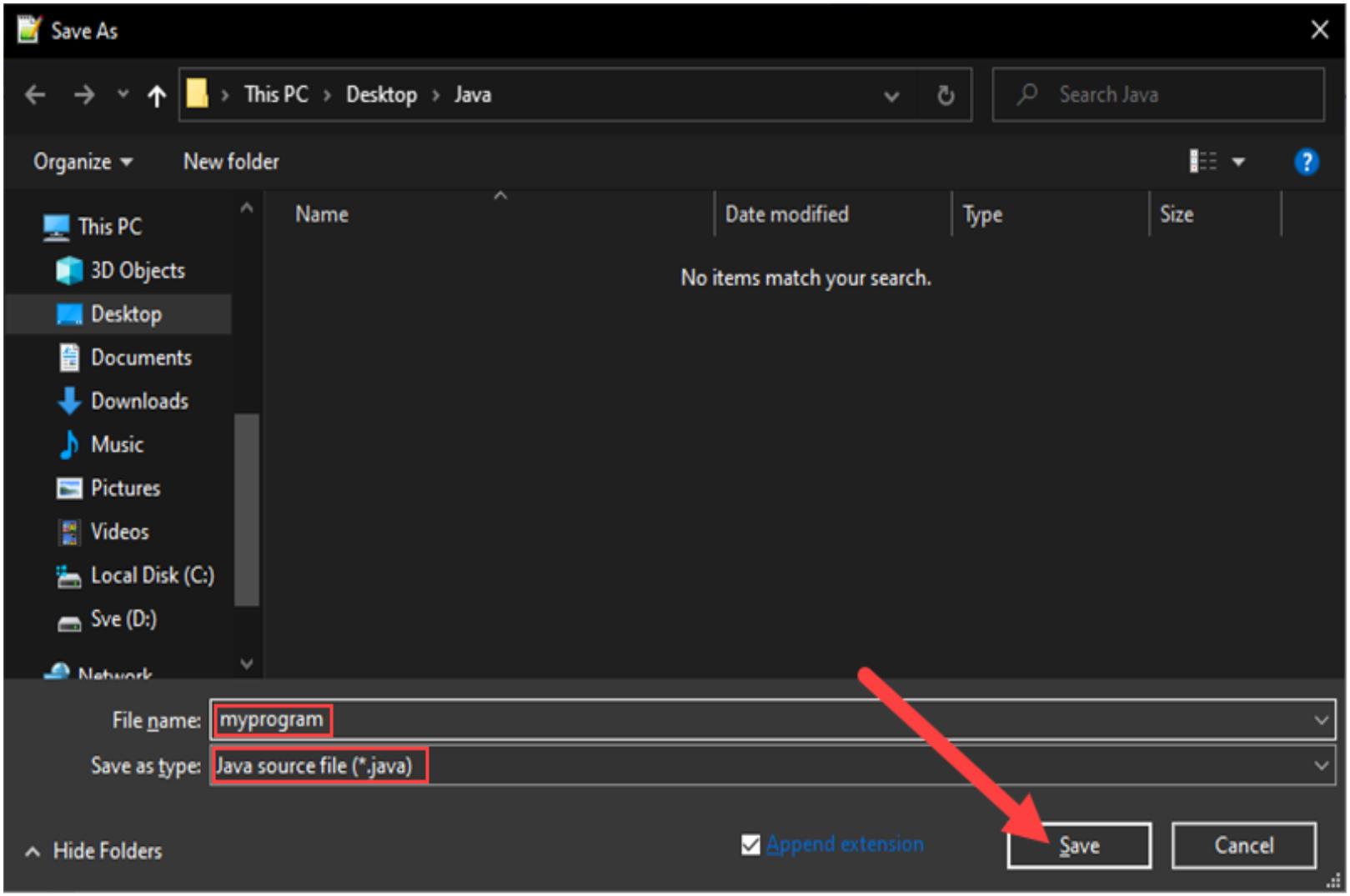
syntax to compile the program:

**javac [filename]**

For example:

After a successful compilation, the program generates a *.class* file in the file directory.

2. Run the program with the following syntax:



java [filename]

The output shows that the program runs correctly, displaying the *Hello world!* message.



**2. Install java editor (Eclipse for Enterprise Java) and configure**

**workspace**

1. Go to the [Oracle Java downloads page](http://www.oracle.com/technetwork/java/javase/downloads/index.html) for Eclipse Java download.

2. Select the JDK that you want to download and install. As you scroll down on the page,

you will see multiple versions of JDK available, choose the one that fits your needs based

on the Eclipse that you want to download.

3. Follow the installation instructions and install the Eclipse JDK.

Once you have JDK on your system, you are set to install Eclipse.

**To install and configure Eclipse, you can do the following:**

Go to the <https://www.eclipse.org/downloads/>and you will see something like below.

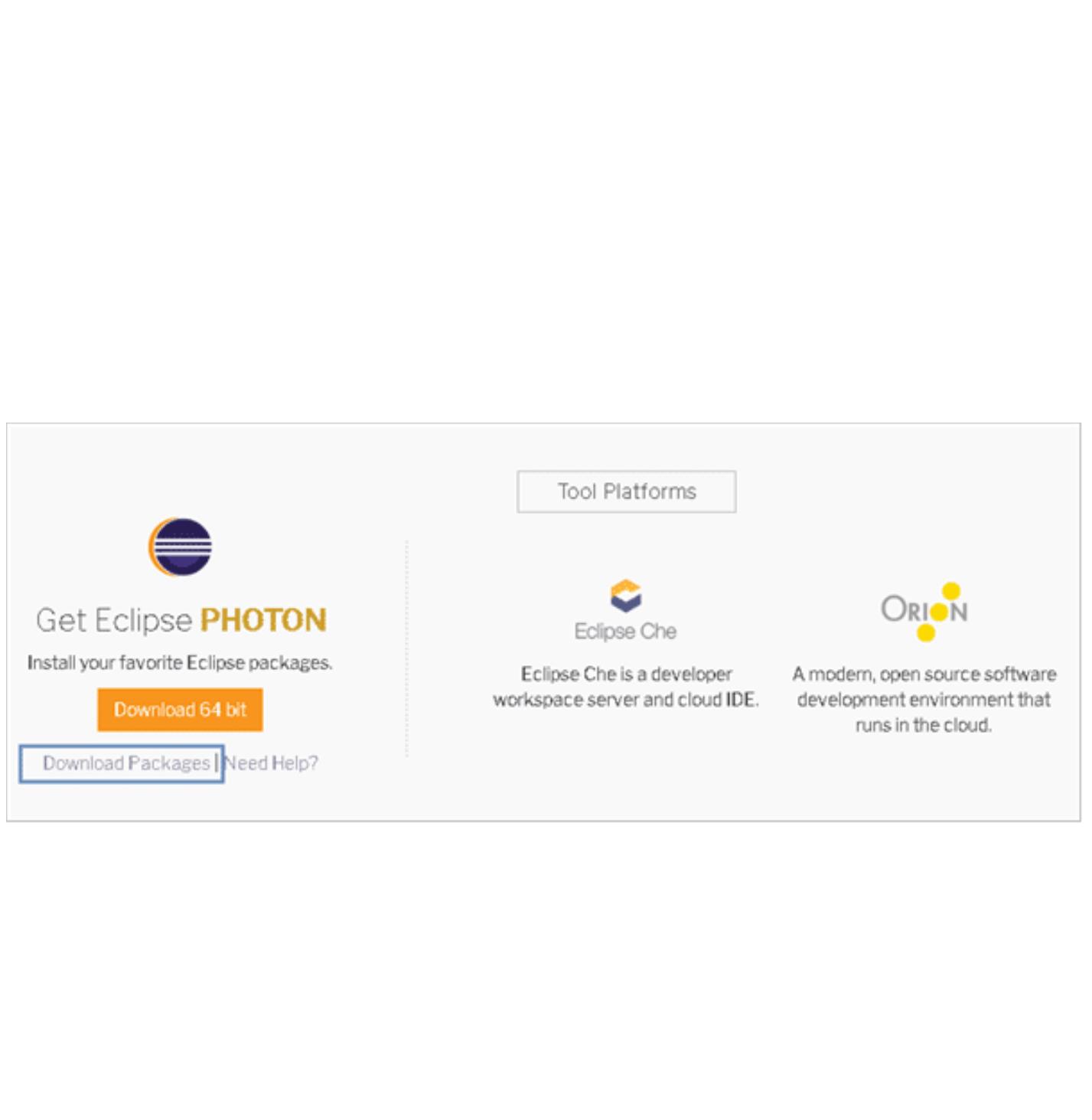
On that page the latest version will be displayed for download, you can directly click on

download 64 bit (this depends on the OS you have on your machine) or click on download

packages to view more packages available.

**On clicking download packages, the following page will be displayed with multiple Eclipse**

**versions.**

image23

You can choose to go with the latest version, or if you want to go with a previous version, click

on the one you want from the highlighted other versions. Once you click on that a similar list of

available Eclipse IDE download links will be displayed for that version.

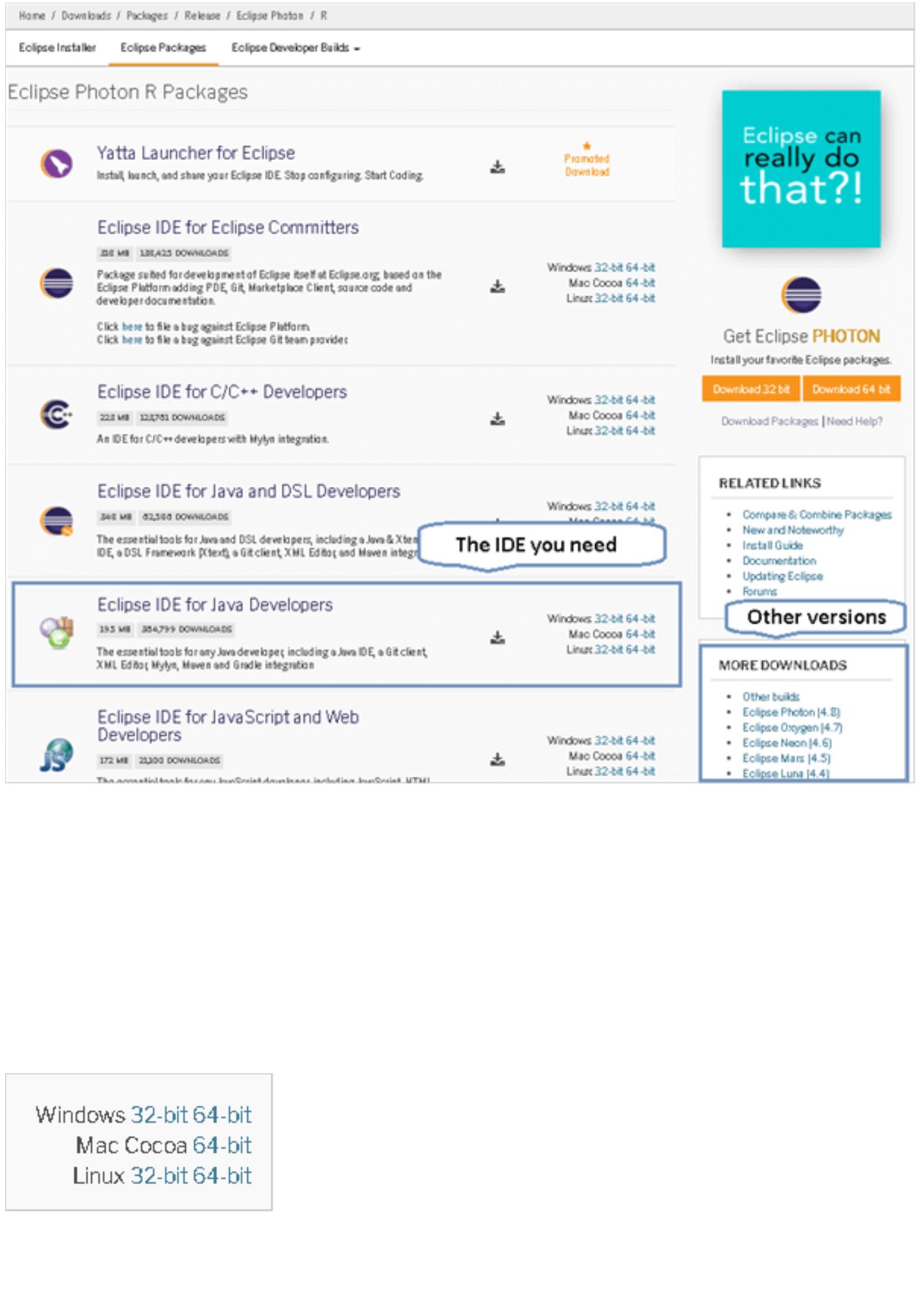
**For Example,** if you click on Eclipse Neon, you will get a list of Neon IDE downloads.

**Select Eclipse IDE for Java developers and select the download link from one of the below**

**depending on the operating system of your machine.**

**For a windows system, you can check your OS type from your system information, by going**

**through the control panel.**

image25

**Once you select the OS type download link, you will be redirected to the following page.**

As soon as you click the highlighted link, the download begins and after some time the

downloaded zipped file will be saved in your machines downloads folder.



Extract the contents of this Zip folder, and you will get a folder by the name of “Eclipse” and

place this in the C drive.

**You should see the below folder structure once you are done with the above steps.**

Eclipse IDE does not require installation, once you have this folder structure look for the

Eclipse executable file highlighted above. Launching this executable file opens the IDE for your

use.

**Setting Up Your Eclipse Workspace**

Once you have the Eclipse application on your machine, you can launch the IDE simply

by clicking on the executable file. However, the first time you open Eclipse, you will be asked to

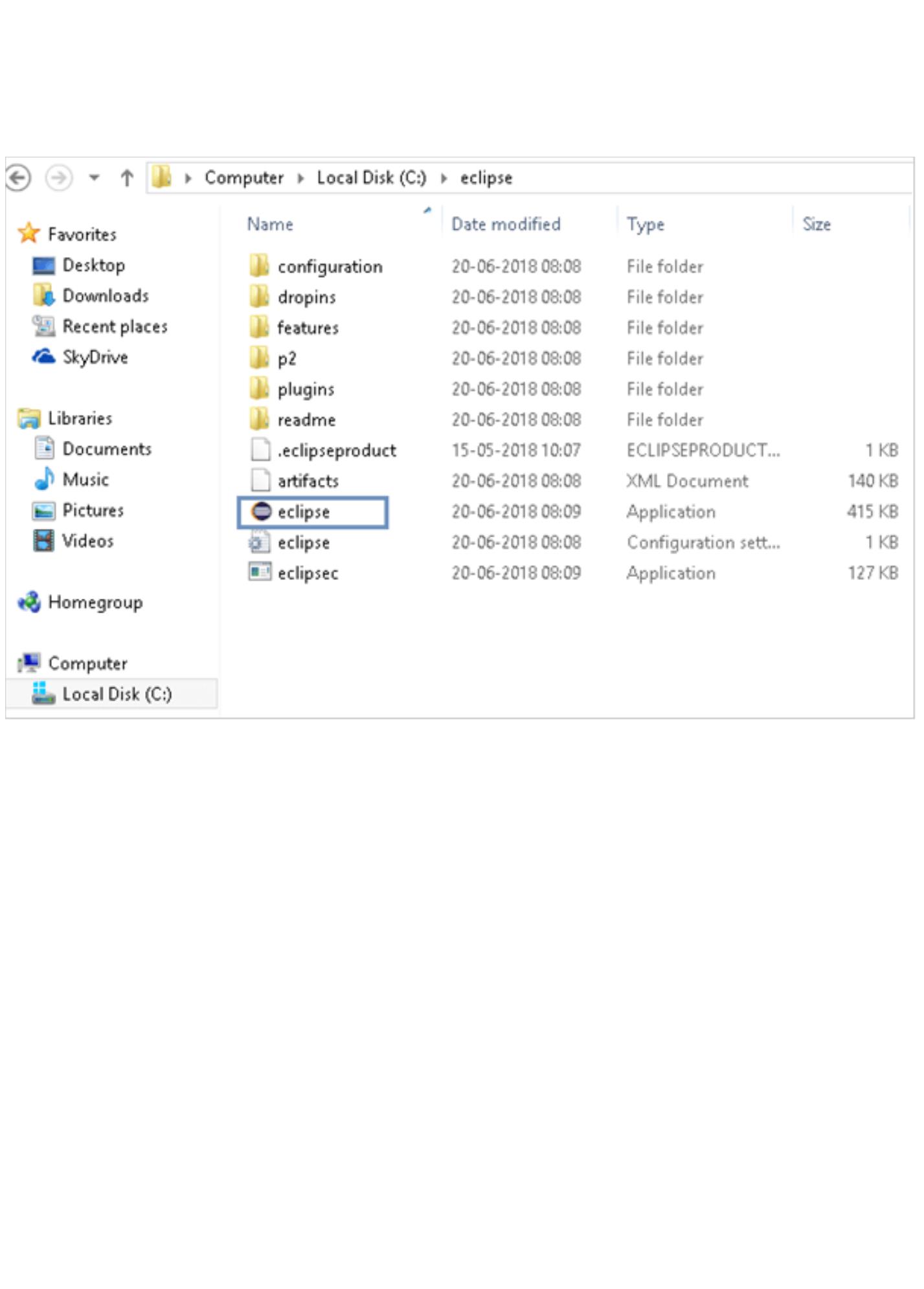
create a workspace.

A workspace is a location on your machine where all the work you do through Eclipse

will be stored as files. It is easier to create one workspace and save it as default so that your

application remembers your past work.

**Once you open Eclipse, you will see the below image.**



**The following window will show up after a few seconds.**

Here you can see the highlighted default workspace location. If you are fine with this,

then simply click the launch button. If you want to create a workspace in a specific location on

your machine, then browse to that folder and then click launch.

You can see the checkbox asking to use the location as default checked, this is to ensure

that the next time when you launch Eclipse it automatically opens the same workspace, so that

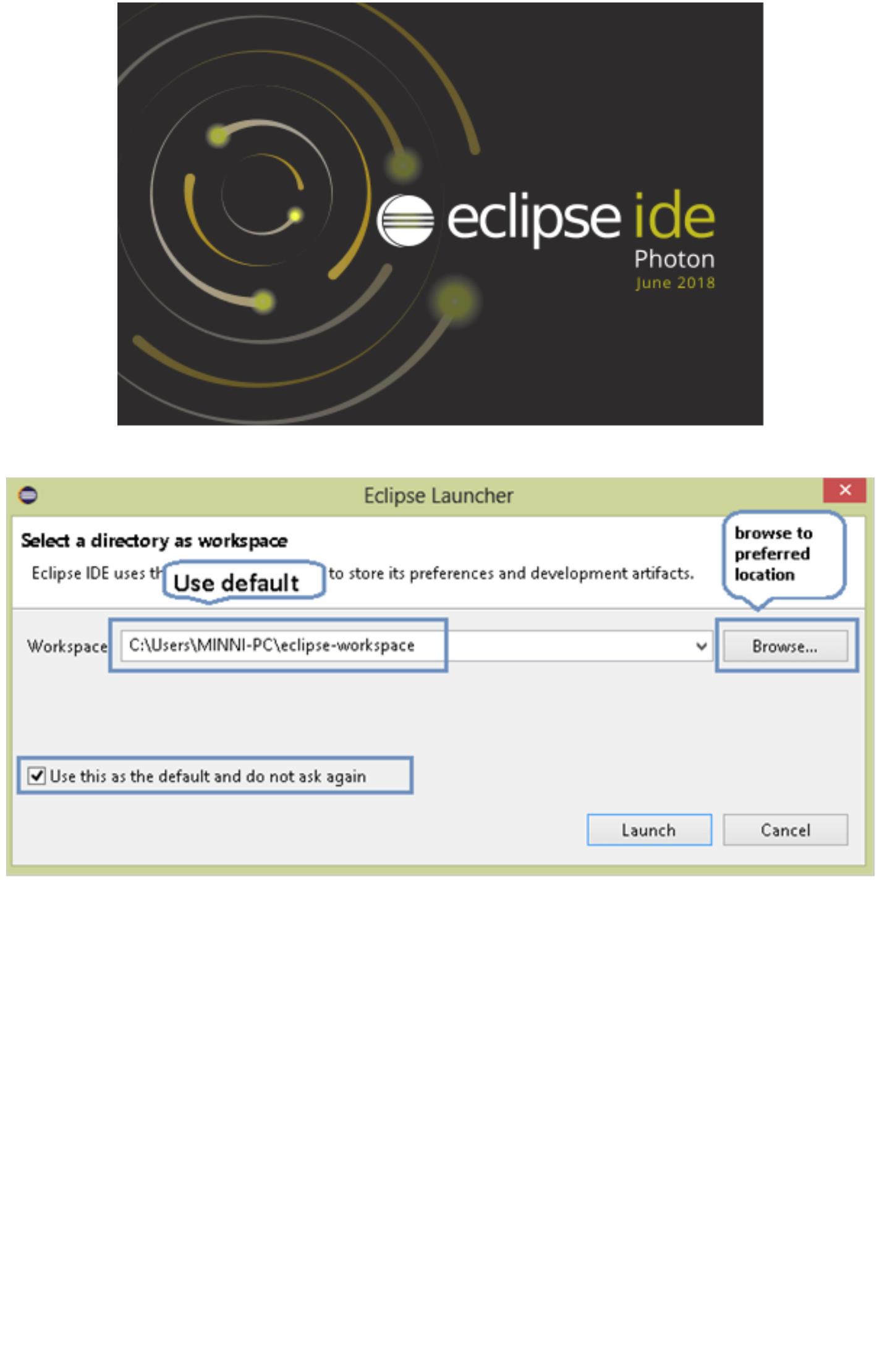
you have access to all the work saved in this location.

After clicking the launch button, it may take some time for the Eclipse workbench to

open. This is because the workspace is being created or in case of an existing workspace, the

projects, etc. maybe fetched. Hence, do not worry if it takes a few minutes.

**Once the workbench is launched, you will see the following default view.**



This default view is the welcome page of Eclipse. As a beginner, you may want to read

through some of the links given regarding the tutorials or samples to gain a better or detailed

understanding. You can also read through the “What’s new” section to see what this latest version

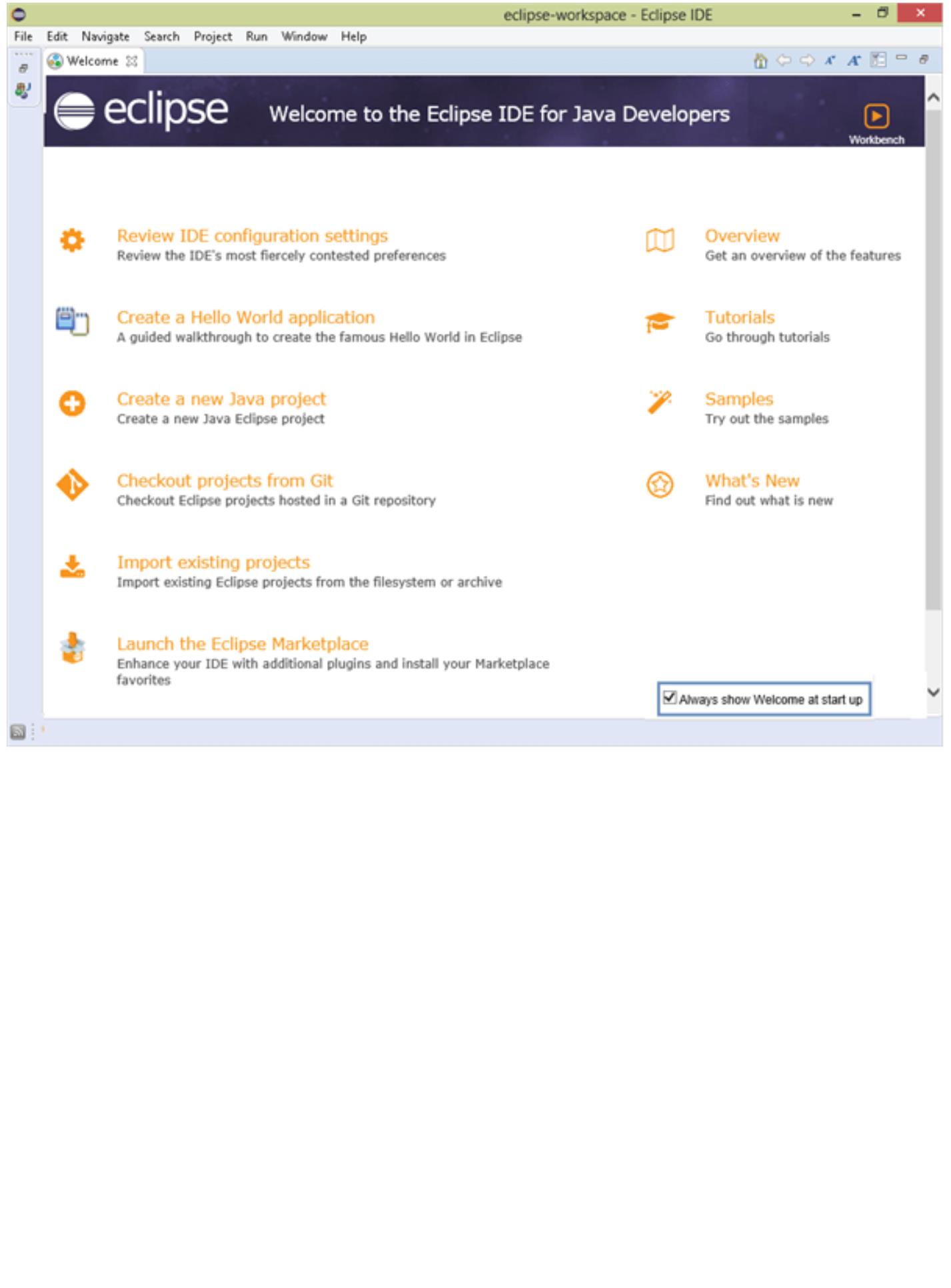
of Eclipse has.

If you do not want to see the welcome page when you launch Eclipse in future, then you

can turn it off by unchecking the checkbox highlighted at the bottom of the page or you can

simply close the welcome page by clicking on the closed sign on the title of the welcome page.

**After closing the welcome page, you will see the following view.**



At this point, you have successfully launched your Eclipse IDE workbench and are ready to start

creating your first code.

Before we get into creating projects and writing actual codes, there are few more setup

related things that you must know. These are environment variables. Eclipse automatically

configures itself to the available JRE in the system, if you have set the environment variables at

the time of setting up Java.

**The primary environment variables include:**

1. **JAVA\_HOME:** Point to JDK folder.

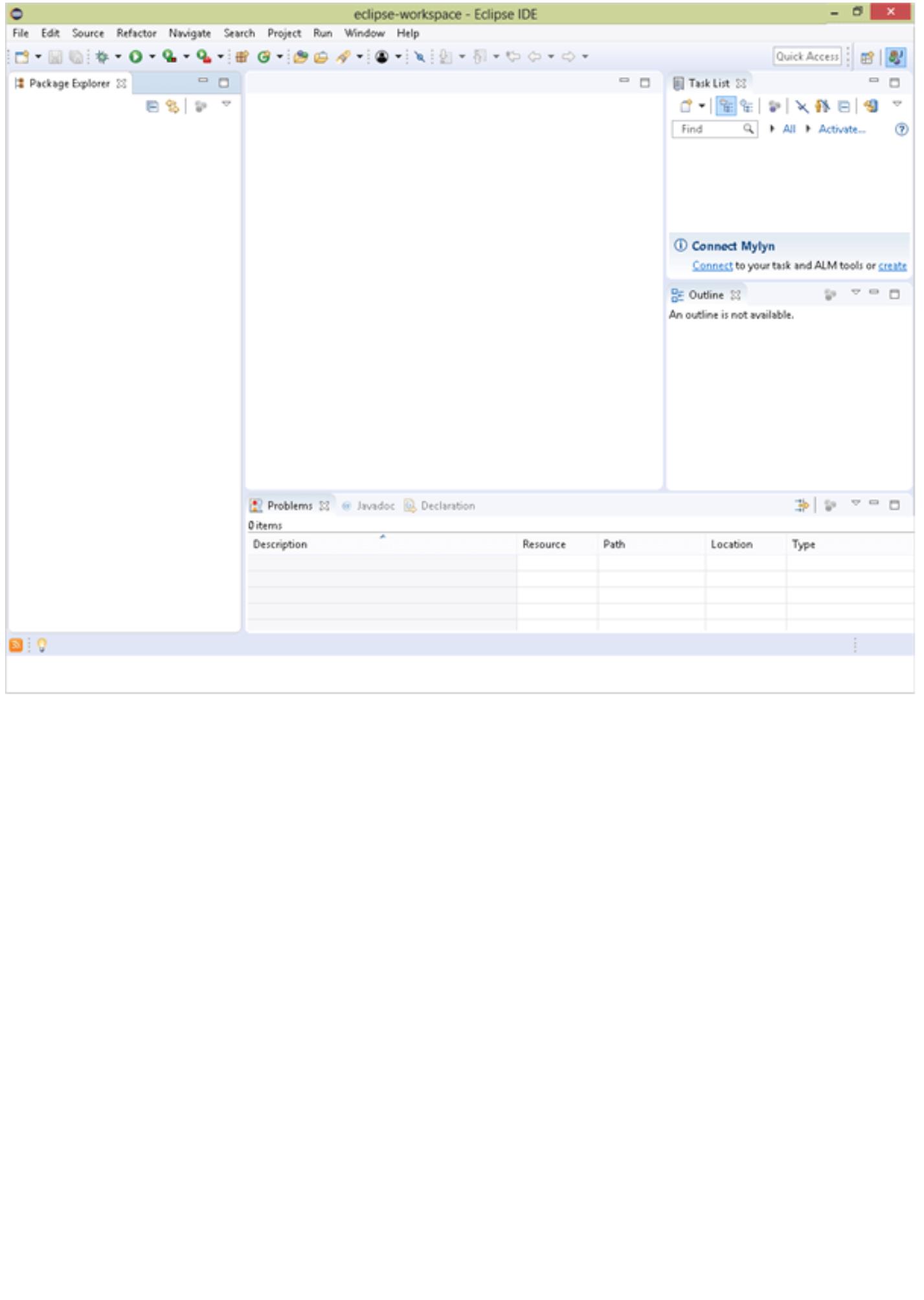
2. **JRE\_HOME:** Point to JRE folder

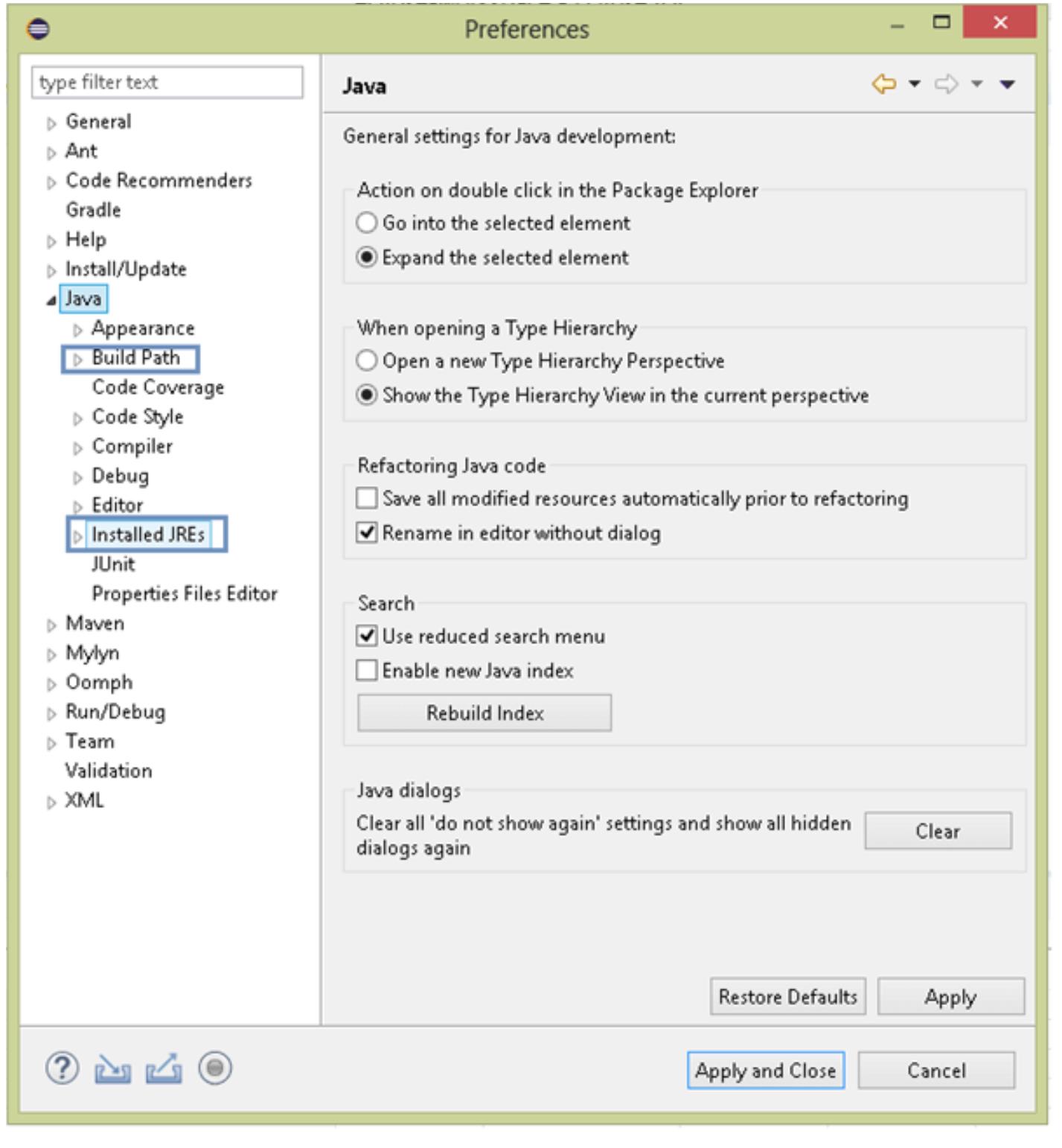
3. **Path:** It is added above the locations to an existing variable. Please ensure that you do not

delete anything that is already present in this variable, just add your new folder locations.

In Eclipse, if you want to update any path related to Java files, then you can do this by going to

preferences (Go to top menu item Window-> Preferences) and the below window will open up.



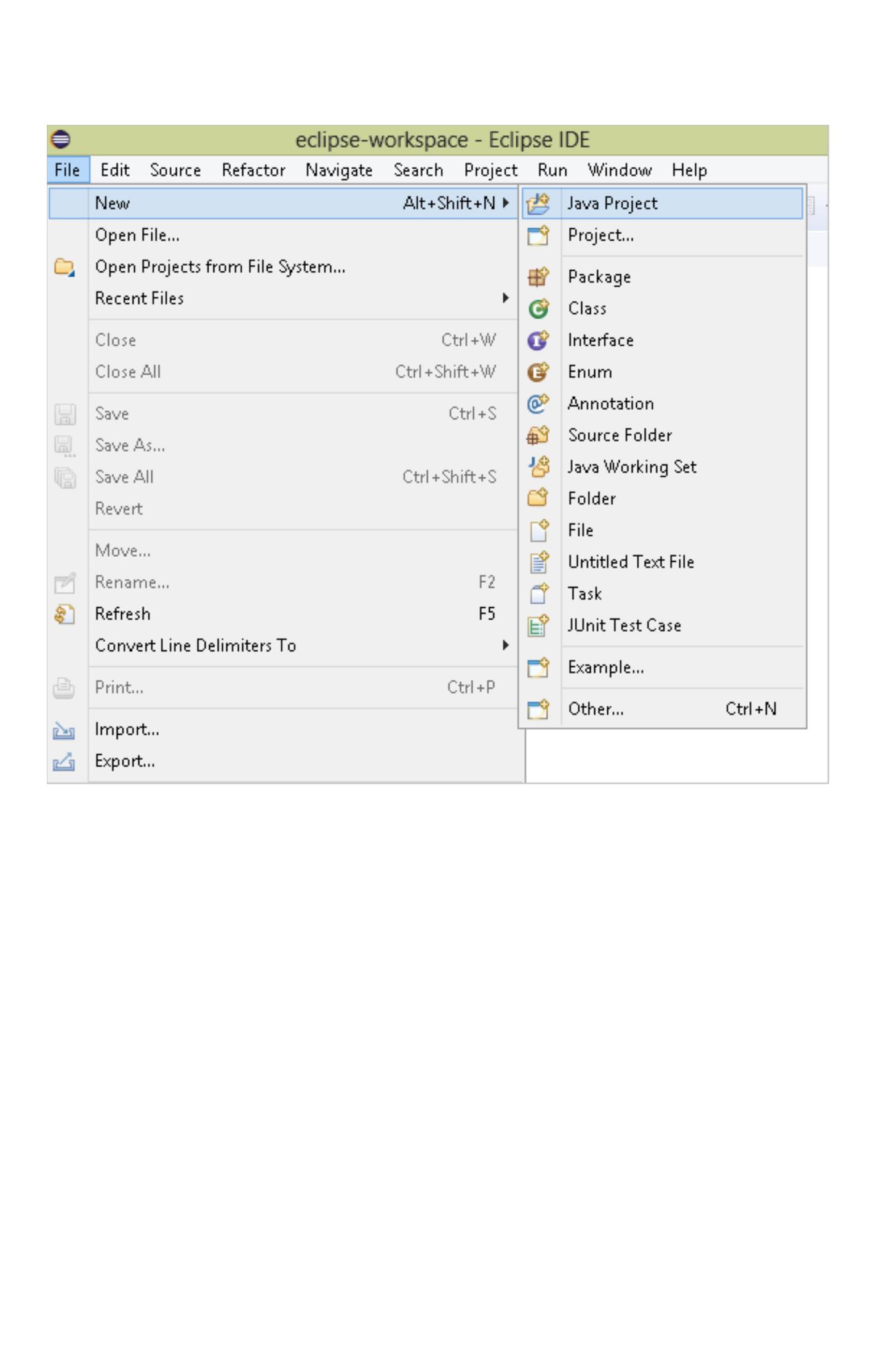


**Creating and Running Your First Java Project in Eclipse**

**To create a simple first Java project follow the steps:**

**#1) Click on File -> New -> Java project.**

**#2) The following window will open:**

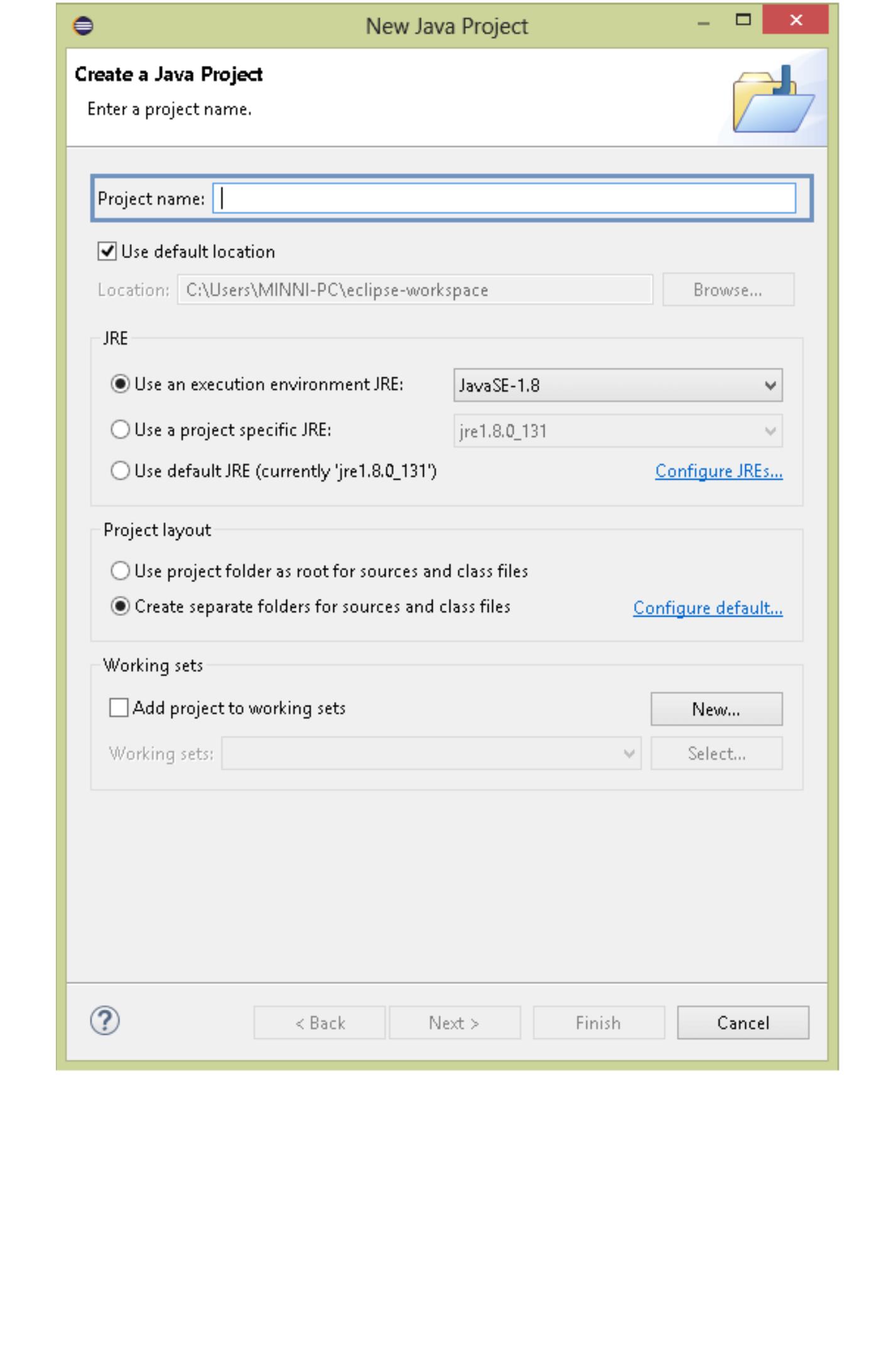


Give a name to your project in the highlighted text field. While creating a formal project the

name should have a logical sense, however as we are in the learning process, for now, you can

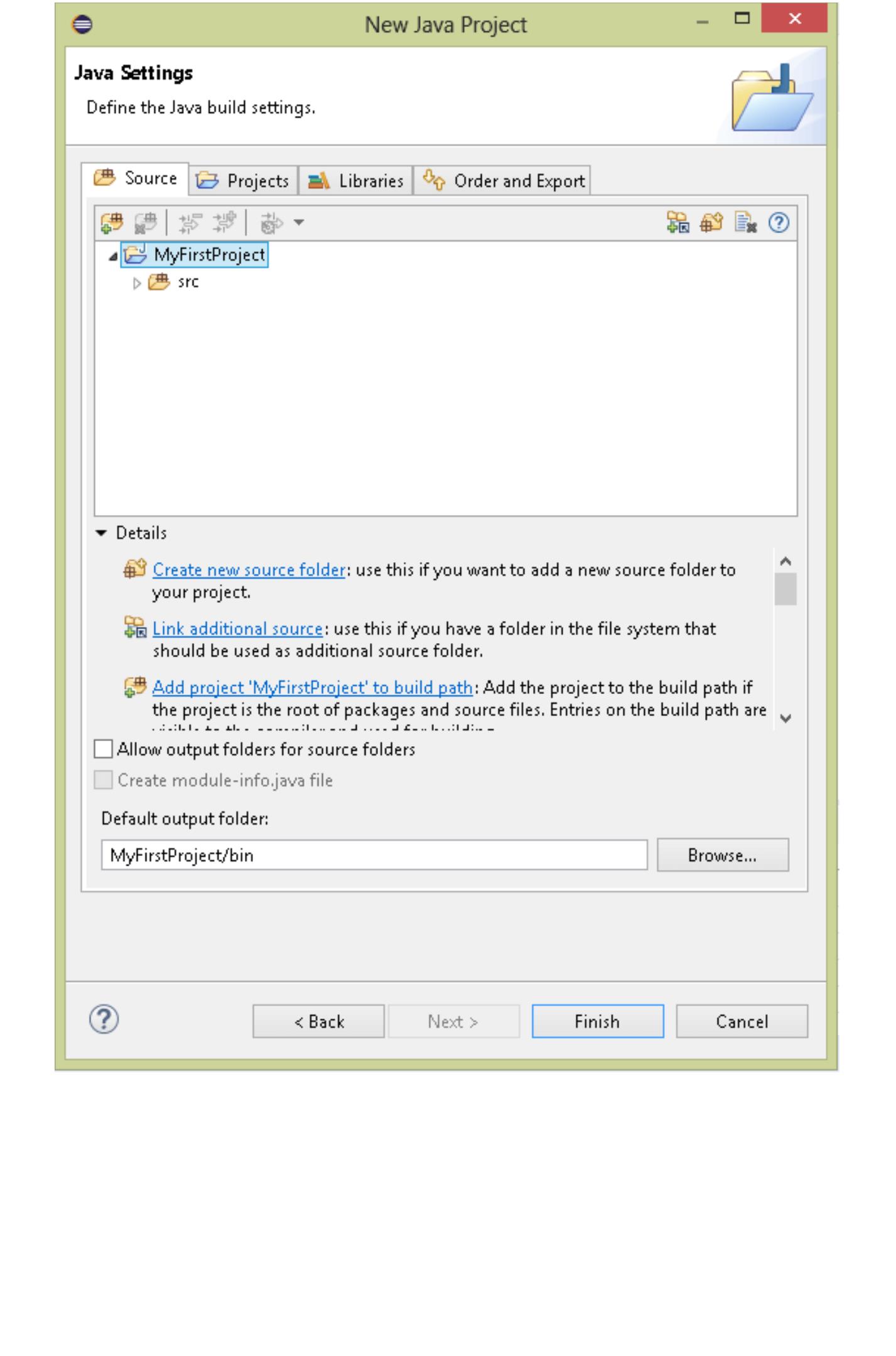
name your project as “MyFirstProject” and leave the rest of the fields as default and click next.

**#3) The following screen opens up:**



Leave all the options as it is and click on finish.

**You will see the following structure created in your project explorer.**



Here you can see two sub-items under your project name. The JRE system library is the

default Java library that Eclipse adds to your project. This library provides Java support and

without this, you cannot proceed with creating a Java project. The other folder is the “src” folder

or the source folder. Your project structure will get created inside this.

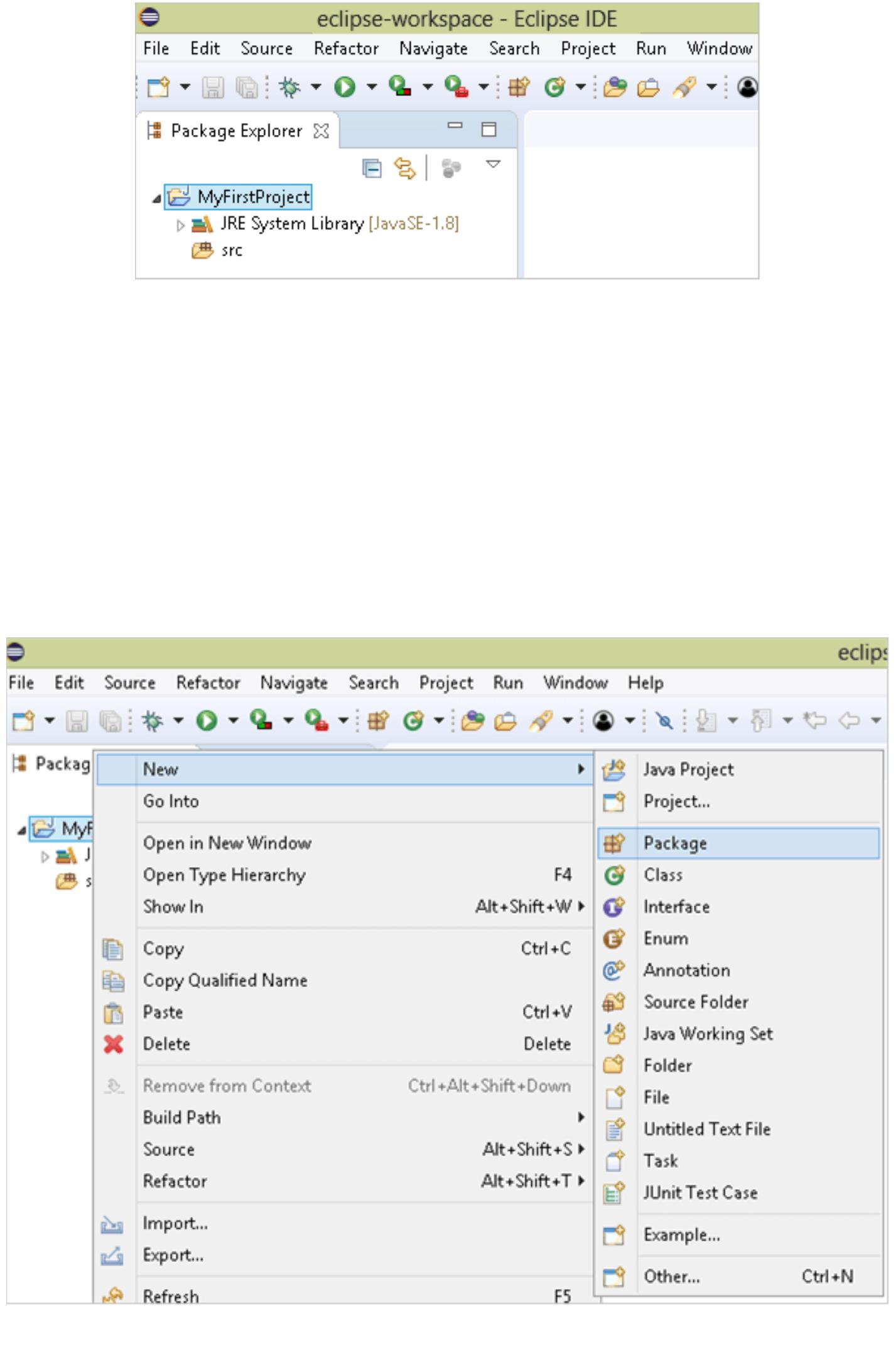
**#4)** You now have your base project structure, let’s add more to this. In Java, we have packages

and classes. Packages provide logical separation for classes. Packages are just folders, whereas

classes are where you write your actual code.

**To create a package right-click on your project name as shown below.**

**Click on the package:**



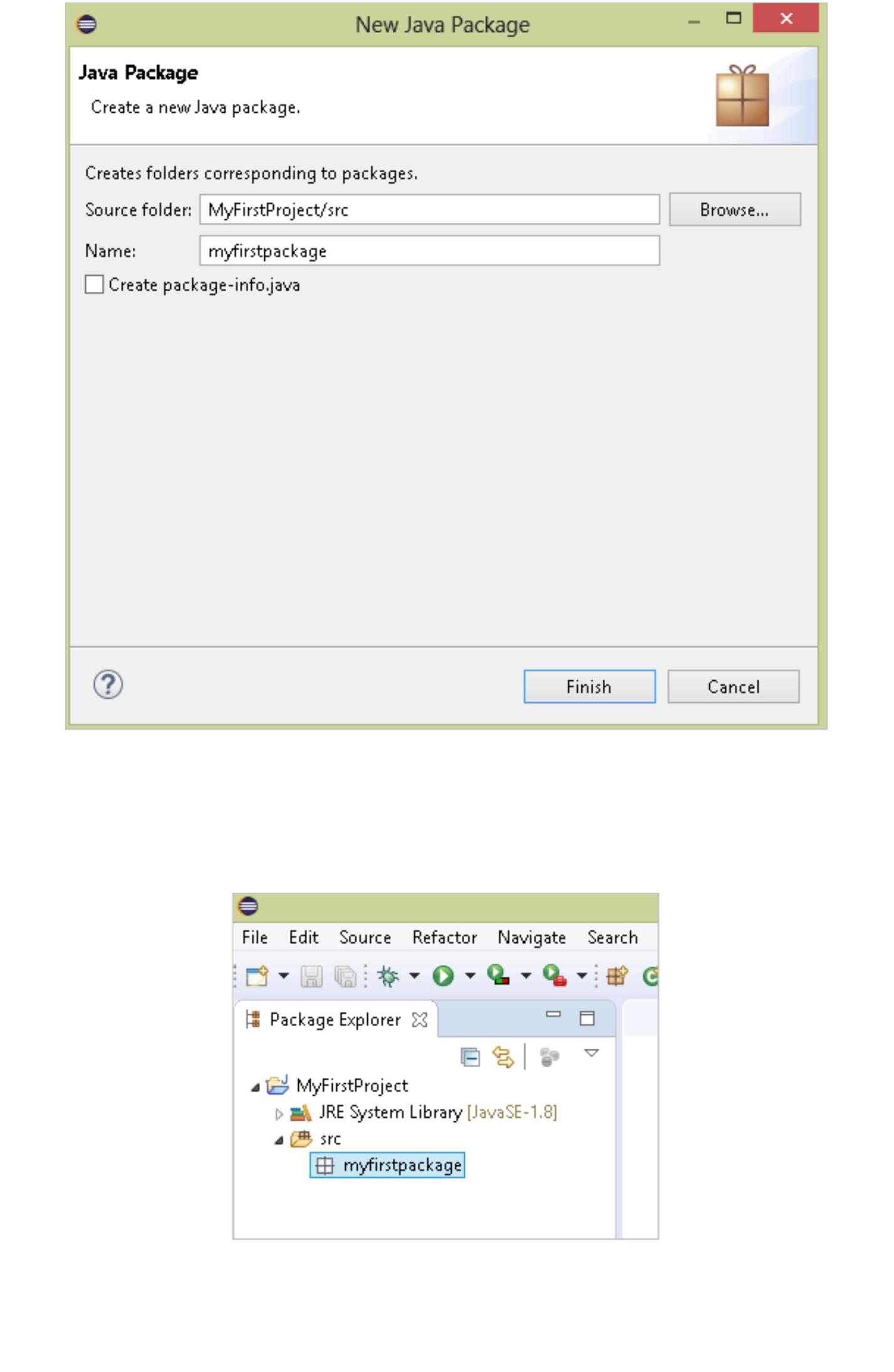
Java says package names should be in a small case, so give any name such as “myfirstpackage”

and click on finish.

**Check your Project Structure.**

**#5)** Next, you need to create a class, and this is where your actual code will go. Right-click on the

package name as shown below.

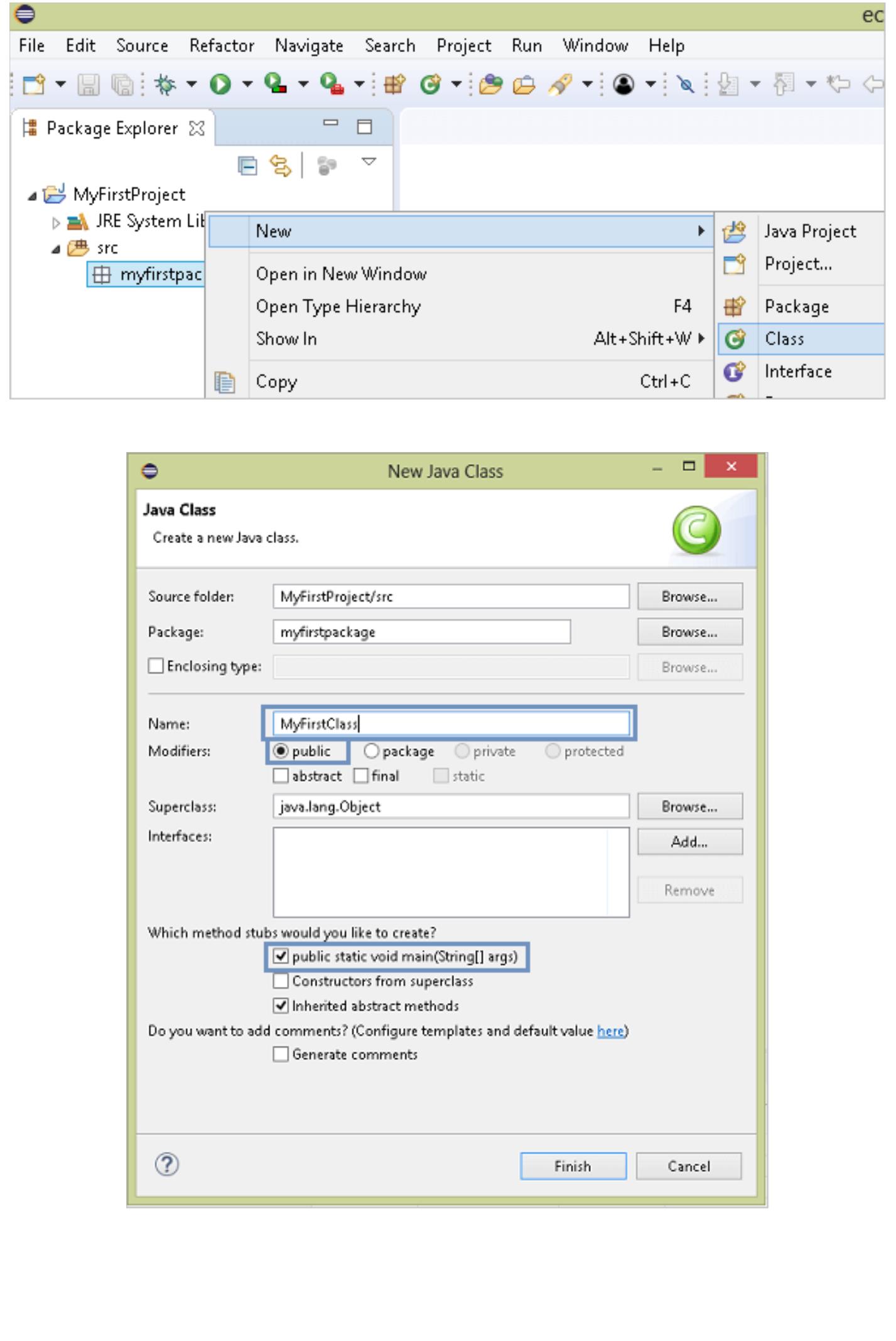


**Click on the class:**

Give a name to your class and keep the modifier selected as public. While creating a simple Java

project, ensure to check the checkbox for creating the method “public static void main(String[]

args)”.



**Click on finish, and now your project will look as shown below.**

You can see that the class is created and as we selected the main method to be added, the method

signature is added to the class. Similar to a class you can also create an interface by following the

same steps and by selecting “New Interface”.

**#6)** As we have successfully created a project and class, let’s put a simple print statement in it and

run the same to check the output.

**Write the below code and save:**

**Code:**

**Code generated:**

package myfirstpackage;

public class MyFirstClass

{

public static void main(String[] args)

{

// TODO Auto-generated method stub

System.out.println("This is my first code");

}

}



Once you have saved the file there are multiple places from where you can run it as a Java

application.

**a)** Keep your cursor on the class that you want to run and click on this icon:

**b)** Right-click on the class name in the project explorer and select Run as and click as shown

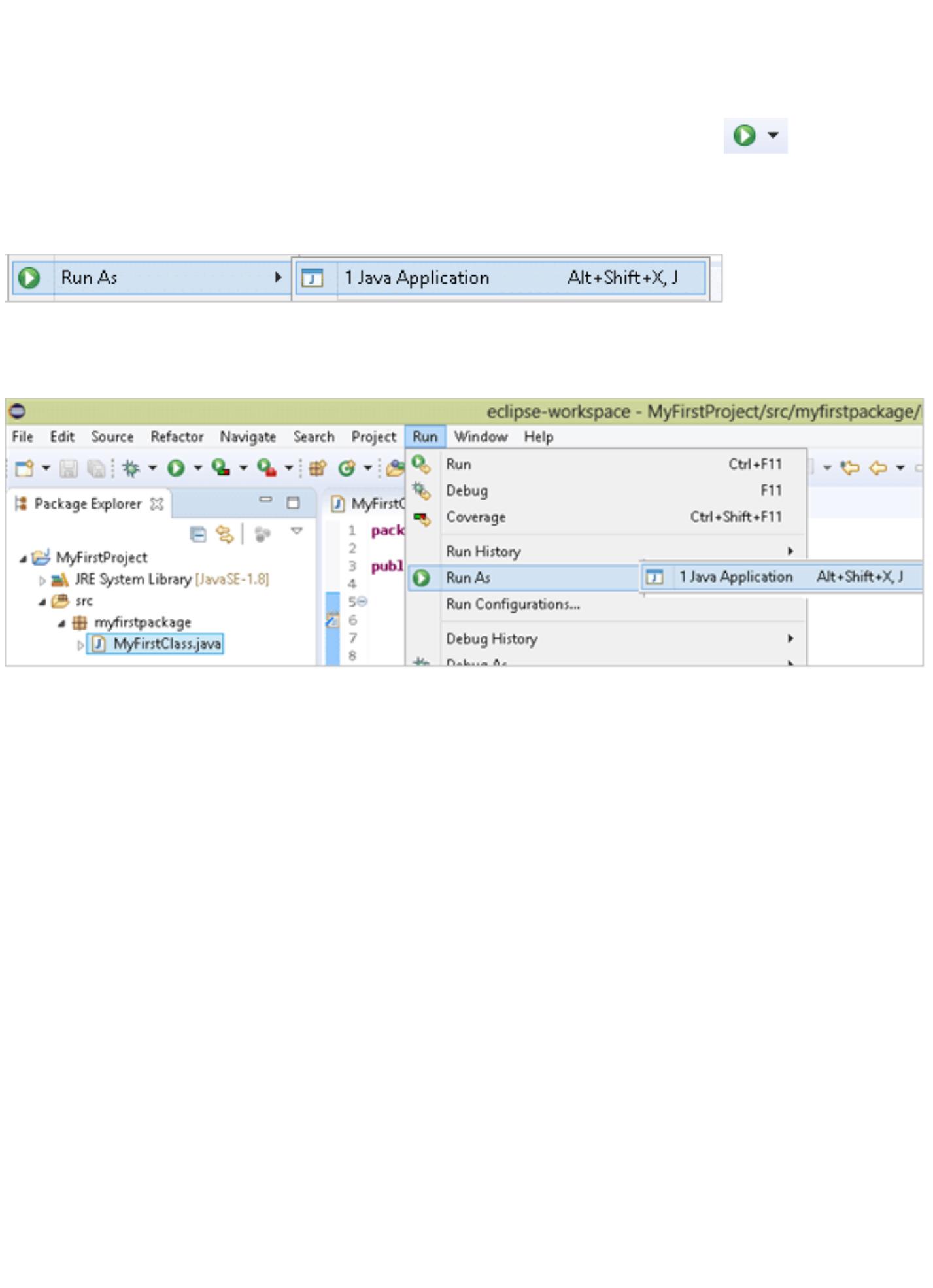
below.

**c)** Select the class name in the project explorer and then select the **Run option from the top**

**menu -> select Run As and then click on Java application.**

#7) As soon as you run your class from any of the above places, a console window opens up in

the bottom pane of your workbench and displays the output of your program.



The console in Eclipse displays the runtime logs while your application is running. In a simple

application like ours, it is simply the output of the print statement, however, in complex projects,

there may be a lot more information displayed.

The console also displays error information if your program encounters any exceptions while

running. So in this section, we have created a simple project and seen how we can run it and see

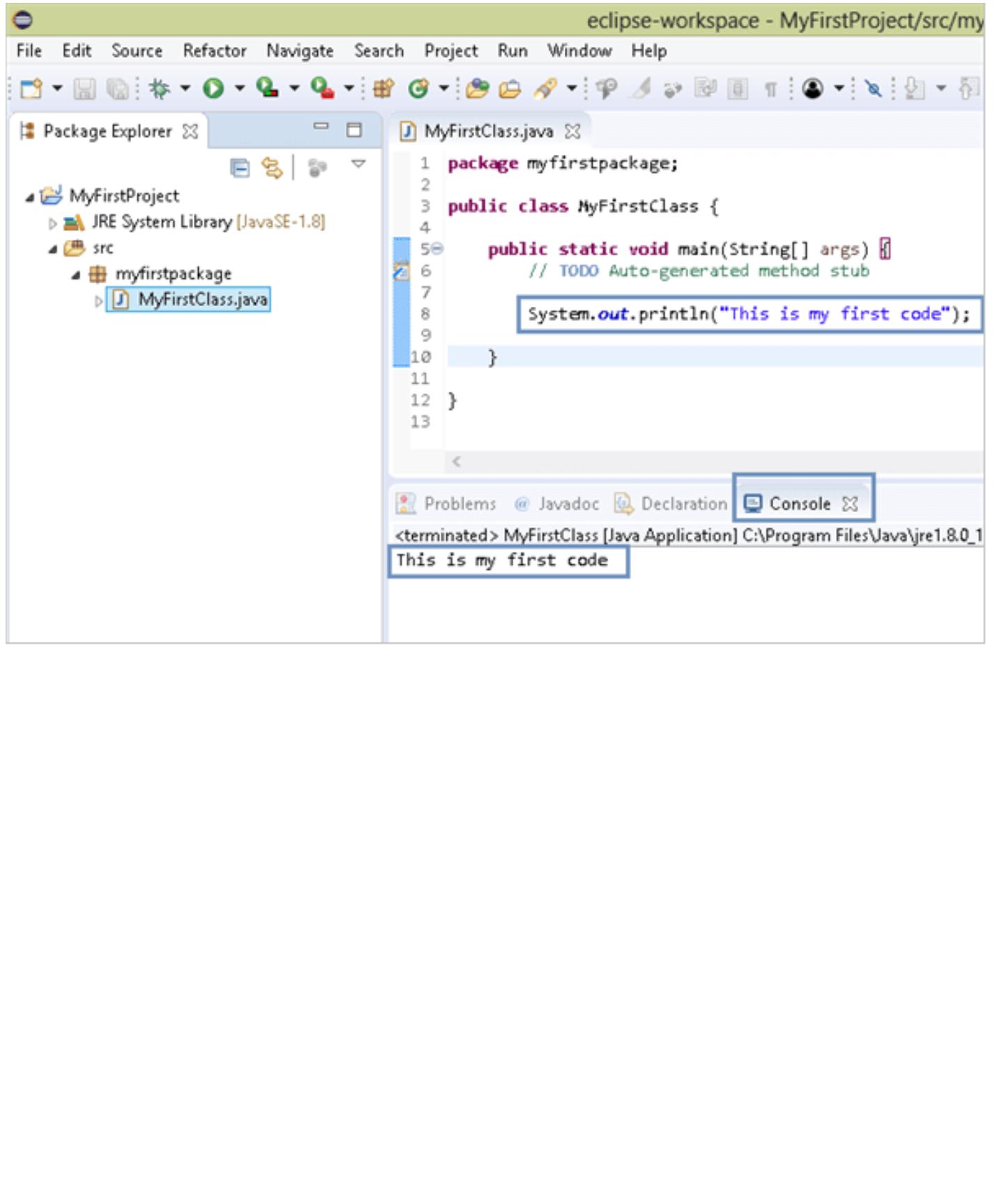
the output generated by your code.

Apart from classes, there are many other types of files that you can create in Eclipse which can be

used in a Java project such as XML files, interfaces, or even simple files.

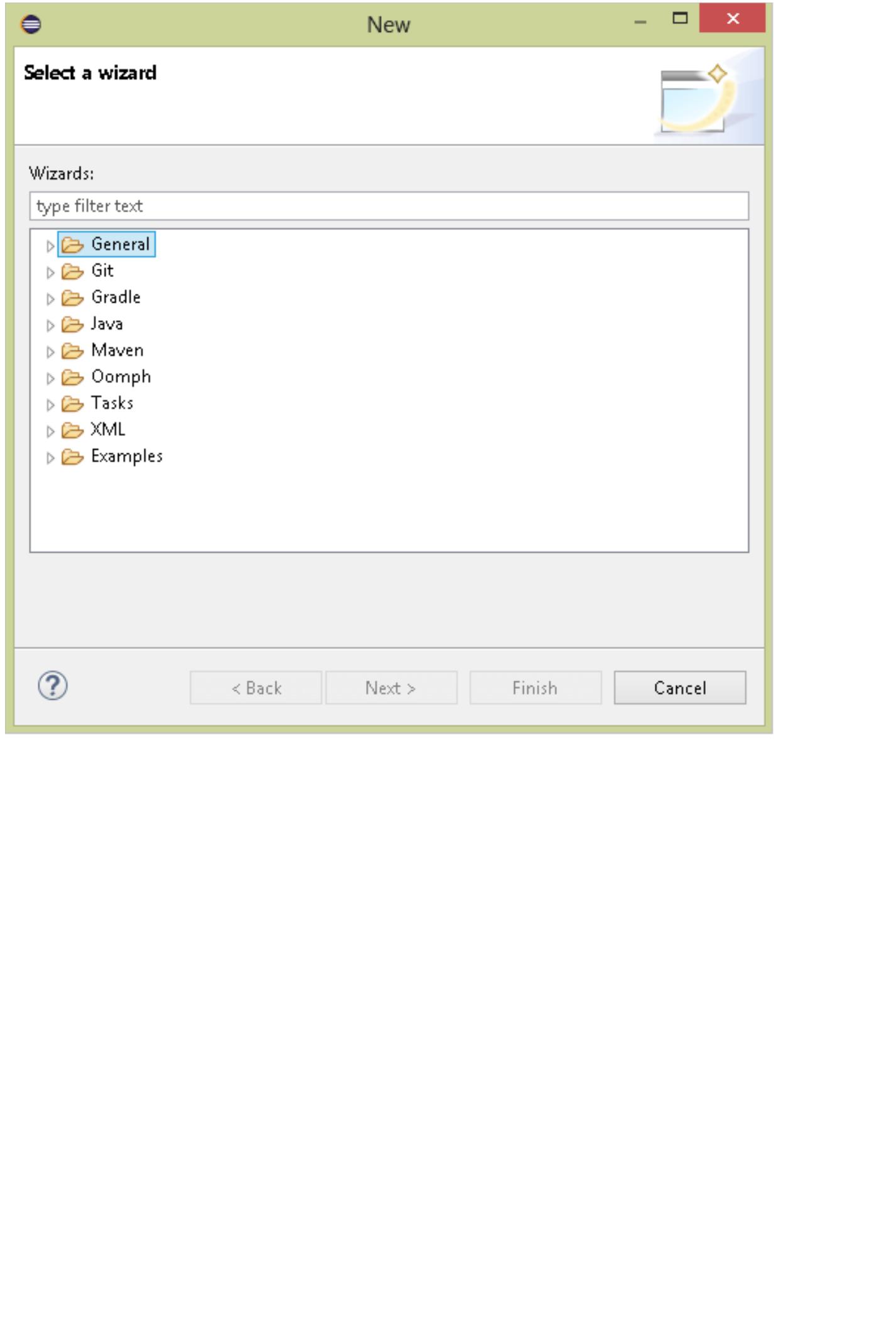
You can explore the types of files or types of projects that you can create by clicking on **File->**

**New-> Others,** and the following wizard will open up.



However, as a beginner, you may leave these for a while and work with the simple Java project

by adding more packages and classes as you need.



**Week 1( Lab Exercise Program)**

**Write and Execute java program to swap values of two variables.**

**class** Swap

{

**public static void** main(String[] args)

{

**int** x=10, y=20;// x and y are to swap

**int temp;**

System.out.println("before swapping numbers: x = "+x +" and y = "+ y);

/\*swapping \*/

temp = x;

x = y;

y = t;

System.out.println("After swapping numbers: x = "+x +" and y = "+ y);

}

}

**Output:**

Before swapping numbers: x = 10 and y = 20

After swapping numbers: x = 20 and y = 10



**Week 2 (Lab Program Exercise)**

**Design and Develop java code to demonstrate Class, Object, Methods, Data types and**

**Type casting.**

**public class** Datacast

{

**void** datatype()

{

**byte** a=10;

**short** b=100;

**int** c=1000;

**double** d=10.1200;

**char** f='a';

**boolean** g=**true**;

**boolean** h=**false**;

System.***out***.println("The value of byte a is:"+a);

System.***out***.println("The value of short b is:"+b);

System.***out***.println("The value of int c is:"+c);

System.***out***.println("The value of double d is:"+d);

System.***out***.println("The value of char f is:"+f);

System.***out***.println("The value of boolean g is:"+g);

System.***out***.println("The value of boolean h is:"+h);

}

**void** typecast()

{

**int** y;

**double** x ;

x=23.56;

System.***out***.println("The value of double x is:"+x);

y=(**int**)x;

//conversion from double to int

System.***out***.println("The value of integer y is:"+y);

}

}

**public class** demo

{

**public static void** main(String[] args)

{

Datacast ob=**new** Datacast();

ob.datatype();

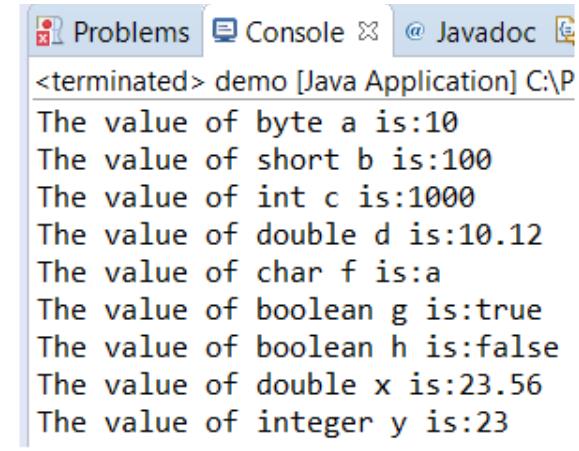
Datacast ob1=**new** Datacast();

ob1.typecast();

}

}

**Output:**

image46

**Week 3 (Lab Program Exercise)**

**Design and Develop Java code to demonstrate types of Constructors.**

**public class** rectangle1

{

**int** length;

**int** breadth;

rectangle1()

{

System.***out***.println("I am constructor");

}

rectangle1(**int** a,**int** b)

{

length=a;

breadth=b;

**int** area=length\*breadth;

System.***out***.println("The area is:"+area);

}

}

**public class** cons

{

**public static void** main(String[] args)

{

rectangle1 ob=**new** rectangle1();

rectangle1 ob1=**new** rectangle1(23,45);

}

}

**OUTPUT:**

**I am constructor**

**The area is:1035**

image49Demonstration of output without use of **“this”** operator

**class** Student

{

**int** rollno;

String name;

**float** fee;

Student(**int** rollno,String name,**float** fee)

{

rollno=rollno;

name=name;

fee=fee;

}

**void** display()

{

System.*out*.println(rollno+" "+name+" "+fee);

}

}

**class** TestThis

{

**public** **static** **void** main(String args[])

{

Student s1=**new** Student(111,"ankit",5000f);

Student s2=**new** Student(112,"sumit",6000f);

s1.display();

s2.display();

}

}

Output

0 null 0.0

0 null 0.0

Use of **“this“** operator.

**class** Student

{

**int** rollno;

String name;

**float** fee;

Student(**int** rollno,String name,**float** fee)

{

**this.rollno=rollno;**

**this.name=name;**

**this.fee=fee;**

}

**void** display()

{

System.*out*.println(rollno+" "+name+" "+fee);

}

}

**class** TestThis

{

**public** **static** **void** main(String args[])

{

Student s1=**new** Student(111,"ankit",5000f);

Student s2=**new** Student(112,"sumit",6000f);

s1.display();

s2.display();

}

}

**Output**

111 ankit 5000.0

112 sumit 6000.0

**If local variables(formal arguments) and instance variables are different, there is no need to use this keyword like in the following program:**

Program where this keyword is not required

**class** Student1

{

**int** rollno;

String name;

**float** fee;

Student1(**int** r,String n,**float** f)

{

rollno=r;

name=n;

fee=f;

}

**void** display()

{

System.*out*.println(rollno+" "+name+" "+fee);

}

}

**class** TestThis3

{

**public** **static** **void** main(String args[])

{

Student1 s1=**new** Student1(111,"ankit",5000f);

Student1 s2=**new** Student1(112,"sumit",6000f);

s1.display();

s2.display();

}

}

**OUTPUT**

111 ankit 5000.0

112 sumit 6000.0

# Autoboxing and Unboxing:

The automatic conversion of primitive data types into its equivalent Wrapper type is known as boxing and opposite operation is known as unboxing.

**public** **class** Auto\_Unbox {

**public** **static** **void** main(String[] args) {

**int** a=50;

Integer a2=**new** Integer(a);//Boxing

Integer a3=5;//Boxing

System.*out*.println("a2= "+a2+" a3= "+a3);

System.*out*.println("Unboxing");

Integer i=**new** Integer(50);

**int** a1=i; //unboxing

System.*out*.println("a1="+a1);

}

}

**OUTPUT**

a2= 50 a3= 5

Unboxing

a1=50

***Ternary* Operator and *instanceof* operator**

**public** **class** OperatorDemo

{

**public** **static** **void** main(String[] args)

{

**int** a, b;

a =10;

b =(a ==1)?20:30;

System.*out*.println("Value of b is : "+ b );

b =(a ==10)?20:30;

System.*out*.println("Value of b is : "+ b );

String name ="James";

// following will return true since name is type of String

System.*out*.println("Use of instanceof operator");

**boolean** result = name **instanceof** String;

System.*out*.println( "Object name is instanceof String " +result );

}

}

**OUTPUT**

Value of b is : 30

Value of b is : 20

Use of instanceof operator

Object name is instanceof String true

**Week 5**

Loop and Conditional Statements

**public** **class** If\_Loop

{

**public** **static** **void** main(String args[])

{

**int** num = 1;

//Demonstration of while

System.*out*.println("Statement while Loop!");

**while**(num <= 10)

{

System.*out*.println(num);

num++;

}

System.*out*.println("Statement Do while Loop!");

num=1;

**do** {

System.*out*.println(num);

num++;

}**while**(num <= 10);

}

}

**OUTPUT**

Statement while Loop!

1

2

3

4

5

6

7

8

9

10

Statement Do while Loop!

1

2

3

4

5

6

7

8

9

10

**For loop and for each loop**

**public** **class** for\_forEach

{

**public** **static** **void** main(String args[])

{

System.*out*.println("Statement for loop!");

**for**(**int** i = 0; i < 10; i++) {

System.*out*.println("i = " + i);

}

System.*out*.println("Statement foreach!");

**int**[] arrayList = {10, 20, 30, 40, 50};

**for**(**int** i : arrayList) {

System.*out*.println("i = " + i);

}

System.*out*.println("Statement after for-each!");

}

}

Statement for loop!

i = 0

i = 1

i = 2

i = 3

i = 4

i = 5

i = 6

i = 7

i = 8

i = 9

Statement foreach!

i = 10

i = 20

i = 30

i = 40

i = 50

Statement after for-each!

**WEEK 6**

// Java program to demonstrate encapsulation

**class** Encapsulate

{

// private variables declared these can only be accessed by public methods of class

**private** String geekName;

**private** **int** geekRoll;

**private** **int** geekAge;

// get method for age to access private variable geekAge

**public** **int** getAge() { **return** geekAge; }

// get method for name to accessprivate variable geekName

**public** String getName()

{

**return** geekName;

}

// get method for roll to accessprivate variable geekRoll

**public** **int** getRoll()

{

**return** geekRoll;

}

// set method for age to access private variable geekage

**public** **void** setAge(**int** newAge)

{

geekAge = newAge;

}

// set method for name to access private variable geekName

**public** **void** setName(String newName)

{geekName = newName;}

// set method for roll to access private variable geekRoll

**public** **void** setRoll(**int** newRoll)

{

geekRoll = newRoll;

}

}

**public** **class** TestEncapsulation {

**public** **static** **void** main(String[] args)

{

Encapsulate obj = **new** Encapsulate(); // Creating object

// setting values of the variables using set methods

obj.setName("Harsh");

obj.setAge(19);

obj.setRoll(51);

// Displaying values of the variables using get methods

System.*out*.println("Geek's name: " + obj.getName());

System.*out*.println("Geek's age: " + obj.getAge());

System.*out*.println("Geek's roll: " + obj.getRoll());

// Direct access of geekRoll is not possible/ due to encapsulation

// System.out.println("Geek's roll: " + obj.geekName); // Not possible

}

}

**OUTPUT:**

Geek's name: Harsh

Geek's age: 19

Geek's roll: 51