COURSEWARE

Hello World Example

Contents

Overview

In this module, we will be going over how to implement "Hello World" in Java.

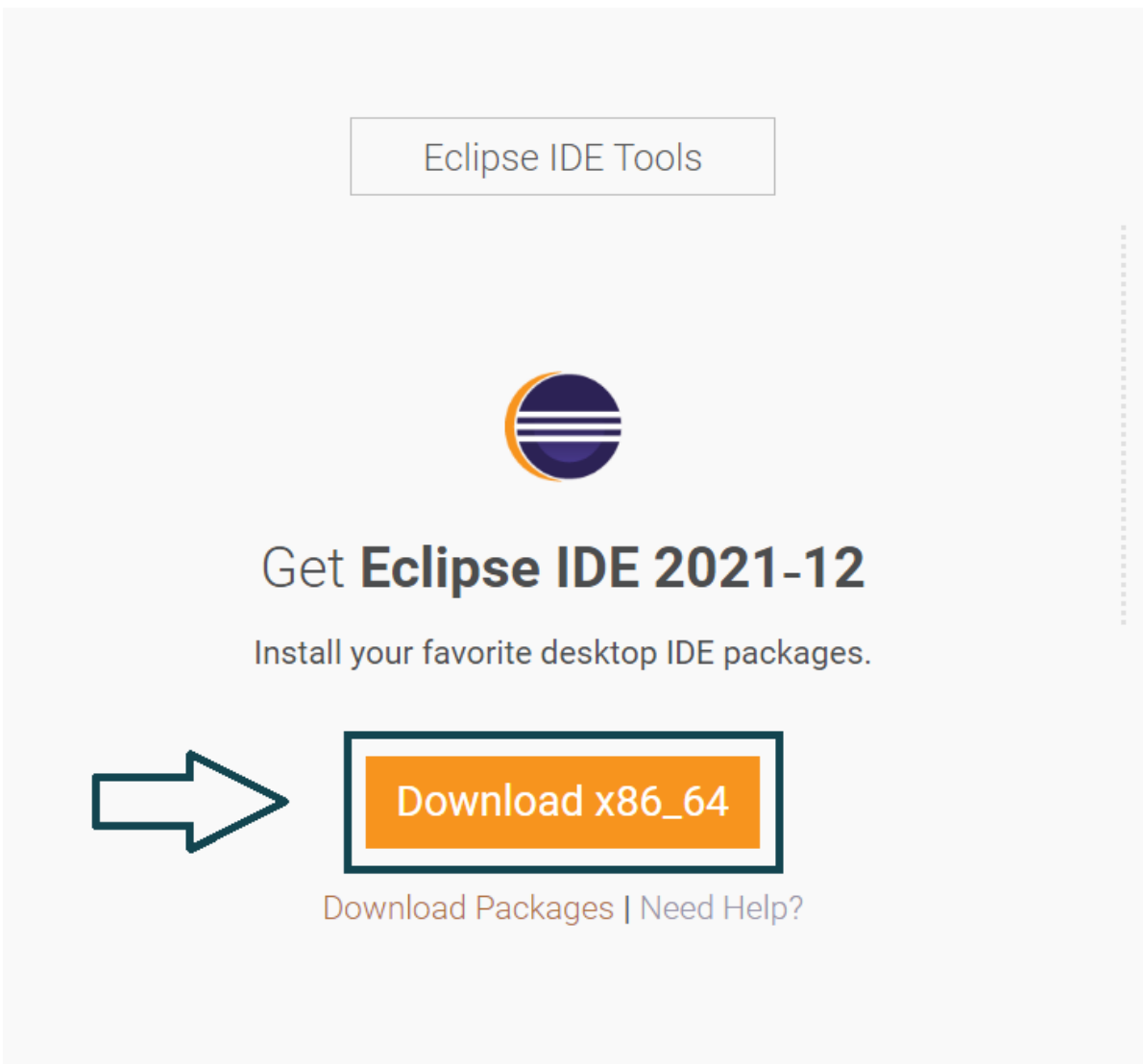
Tutorial

First, we need to make sure that we have installed Java, if you haven't done this check out the Java Installation module.
Then we will need to install an Integrated Development Environment (IDE), we will be using Eclipse.
Then we can create a Java project in our IDE, we will be going over how to do this in Eclipse.

Installing Eclipse

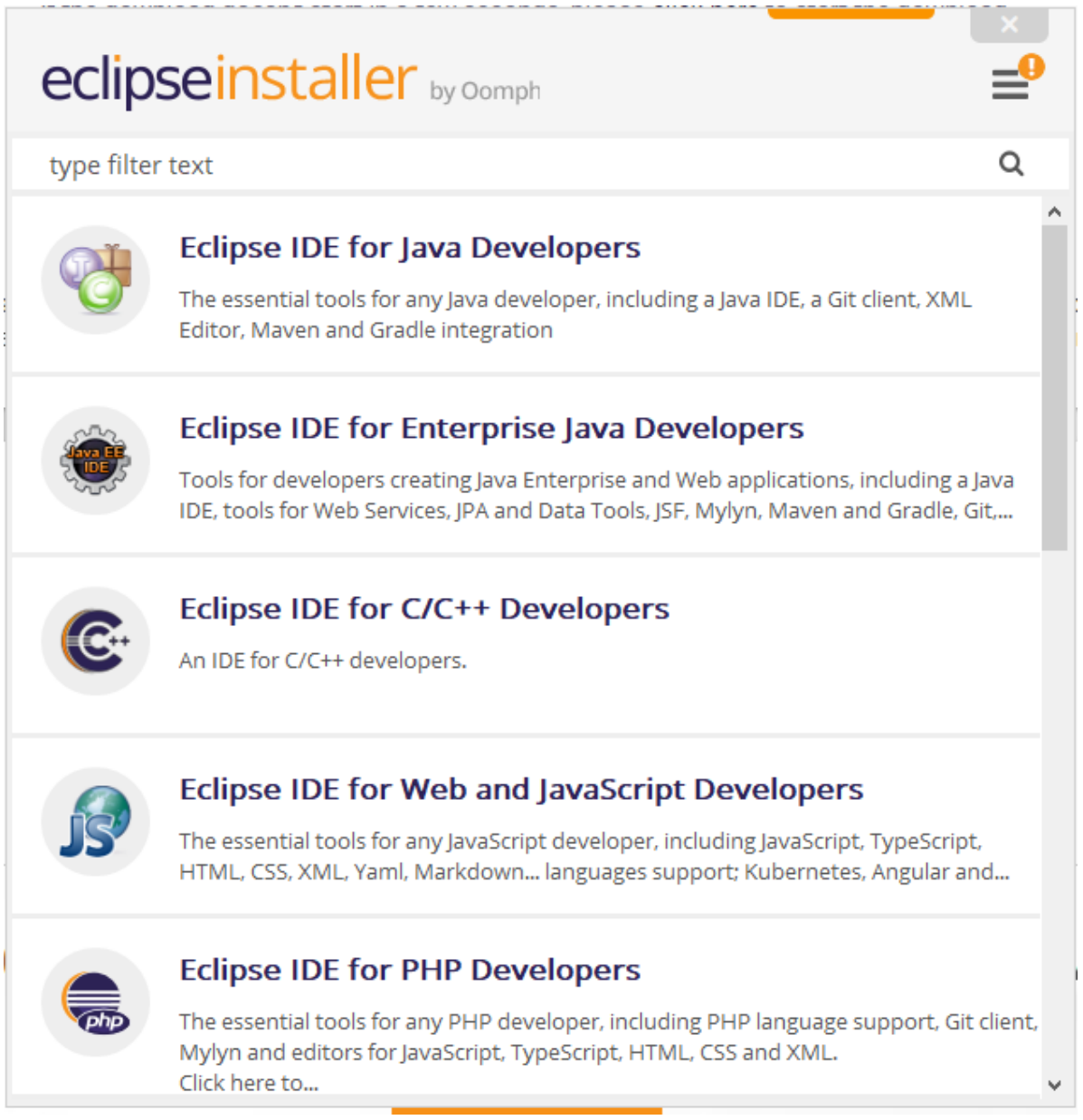
To install Eclipse

1. Download Eclipse Installer from http://www.eclipse.org/downloads

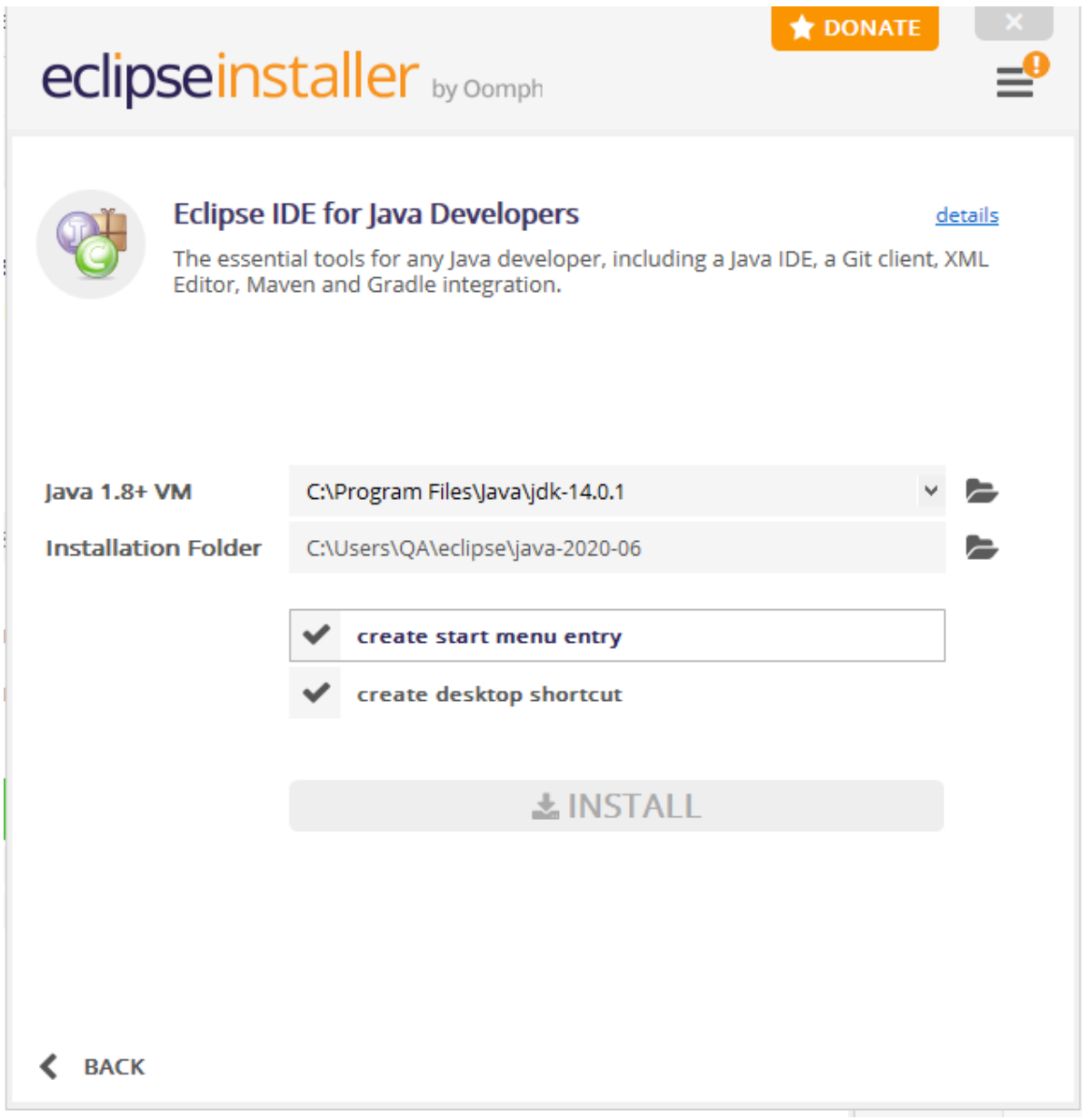


2. Run the `.exe` installer

3. Select, 'Eclipse IDE for Java Developers'



4. Now, 'Install'



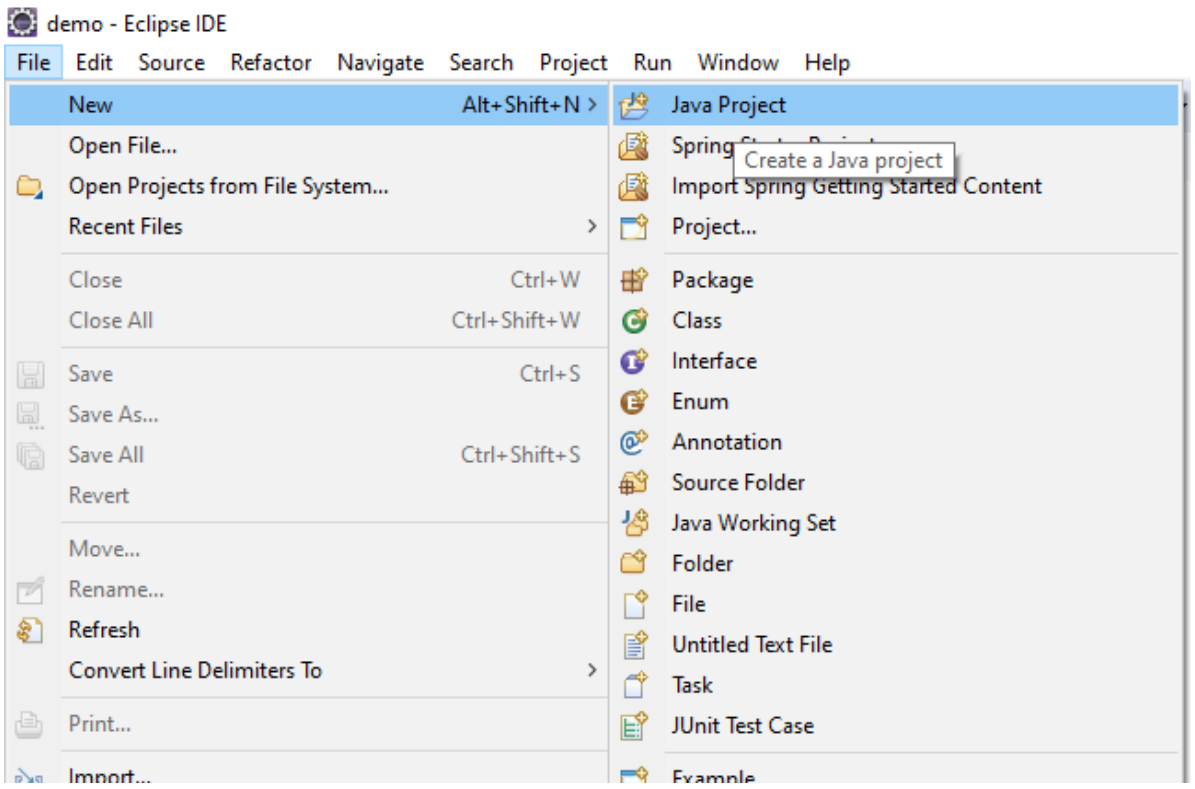
5. Next, you can simply 'Launch' and Eclipse will be fully functional.

Creating a Project in Eclipse

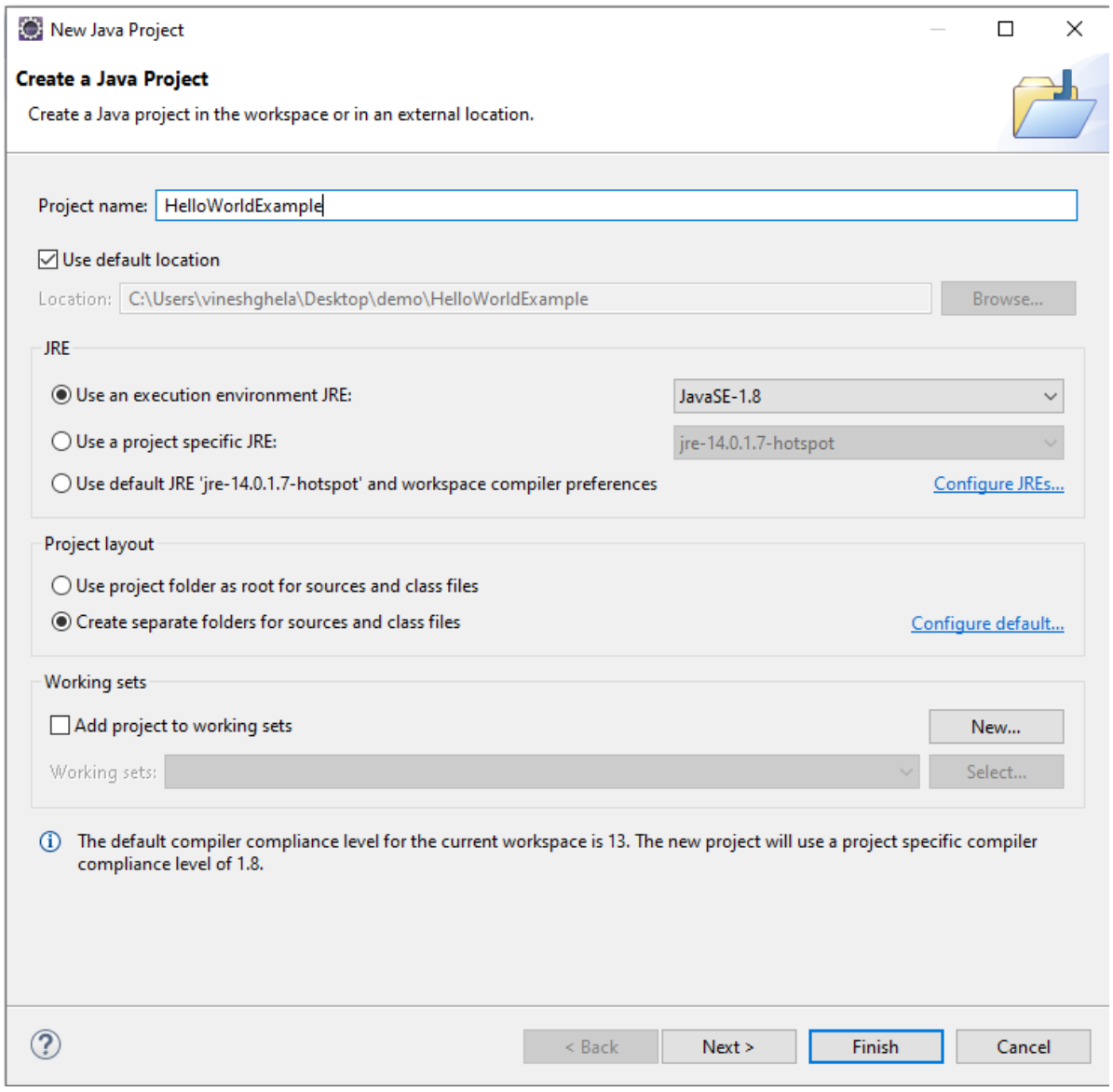
To create a project in Eclipse, follow these steps:

1. Open Eclipse and define a directory for your workspace, this can be located anywhere you want, then click "Launch".

2. Now that we have Eclipse open, click "File", then "New", then "Java Project".



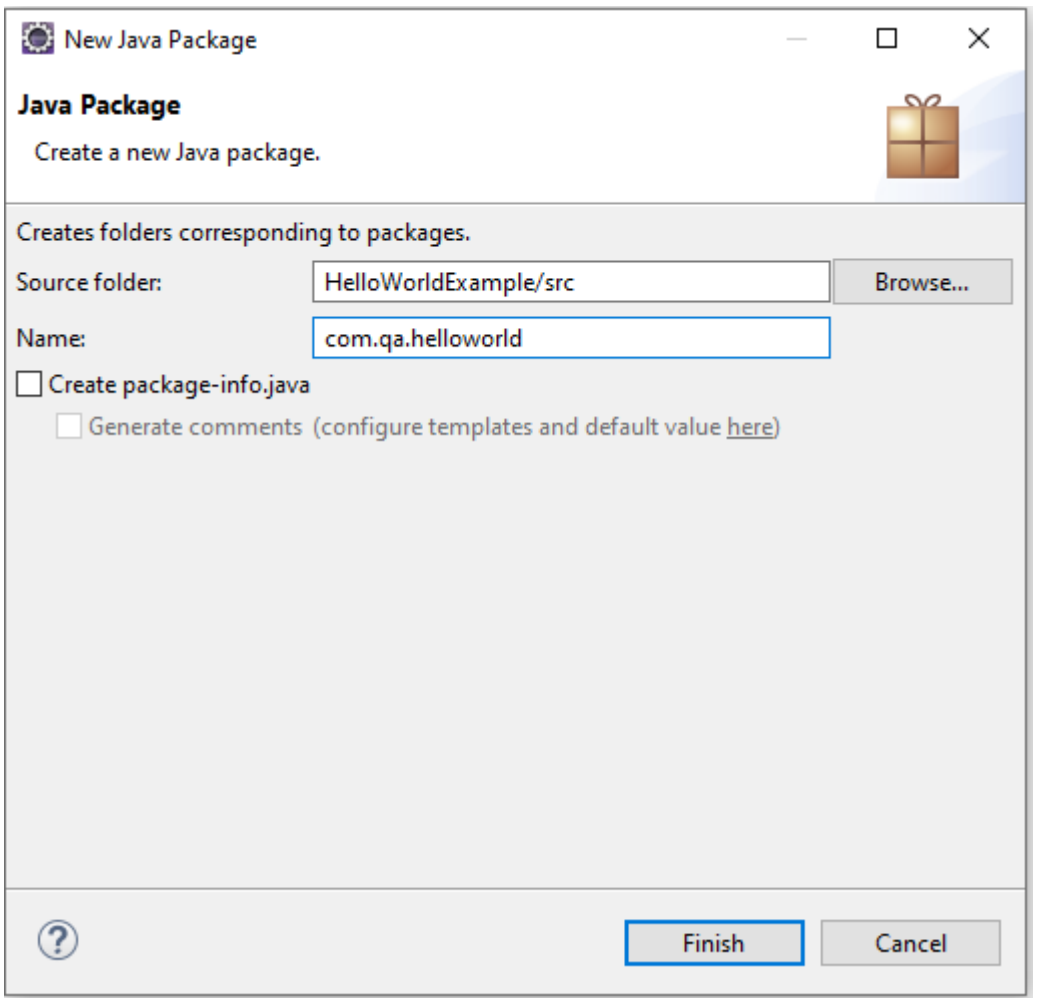
3. Once you have clicked "Java Project" you will be presented with the below screen.



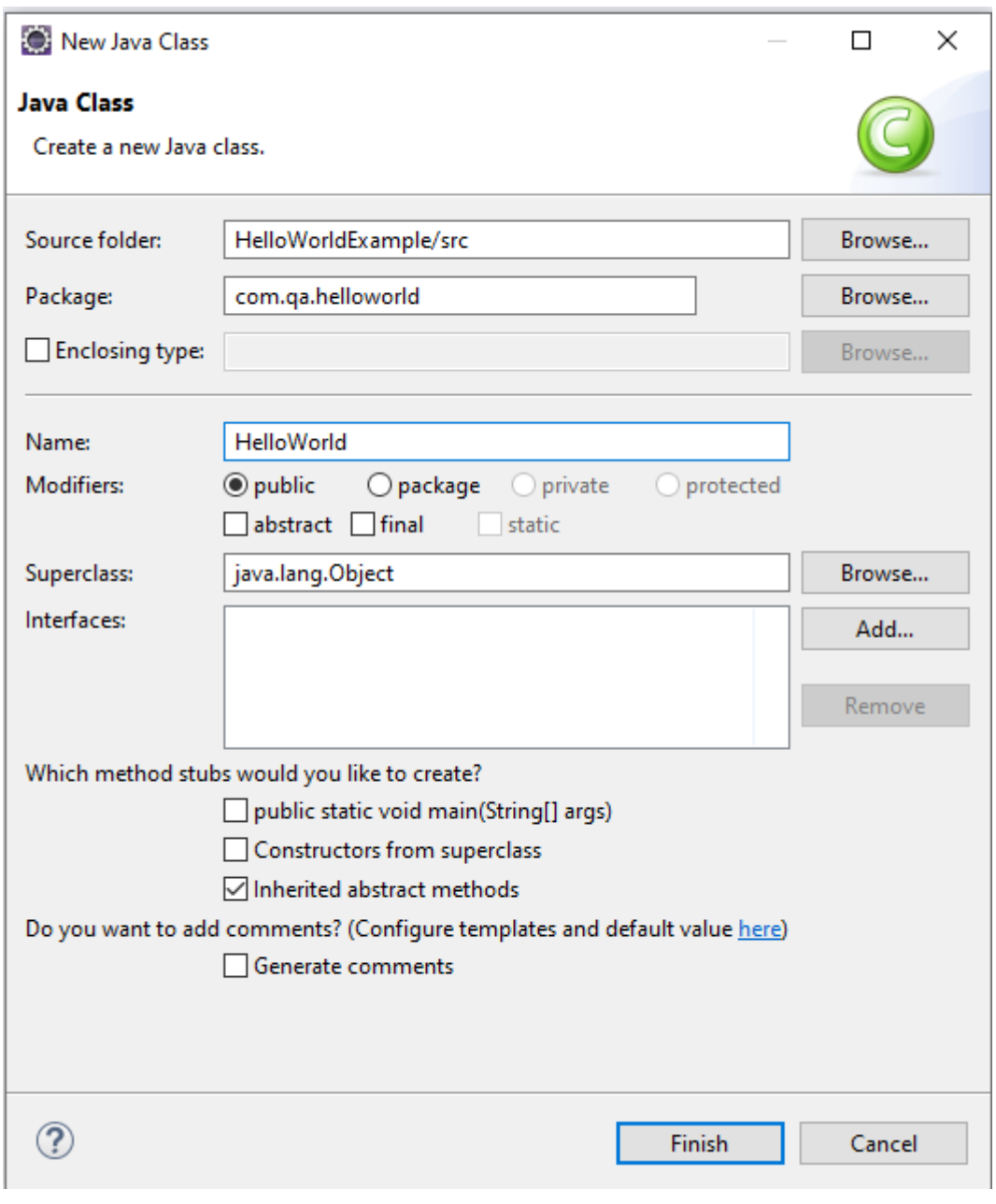
Enter a name for your project in the top text box, and make sure you have a Java Runtime Environment (JRE) selected.
Once you have entered a name for your project and ensured you have a JRE click "Finish".

4. The project will now be created and appear in the left-hand side of the screen under "Package Explorer".
Now we need to create a package e.g. `com.qa.helloworld`.
To create a new package, expand the project by clicking the arrow to the left of the project name, then right-click on the folder named "src", then hover over "New", then click "package".

A package is used to group related classes. *Package names are written in lower case to avoid conflict with names of classes or interfaces.*

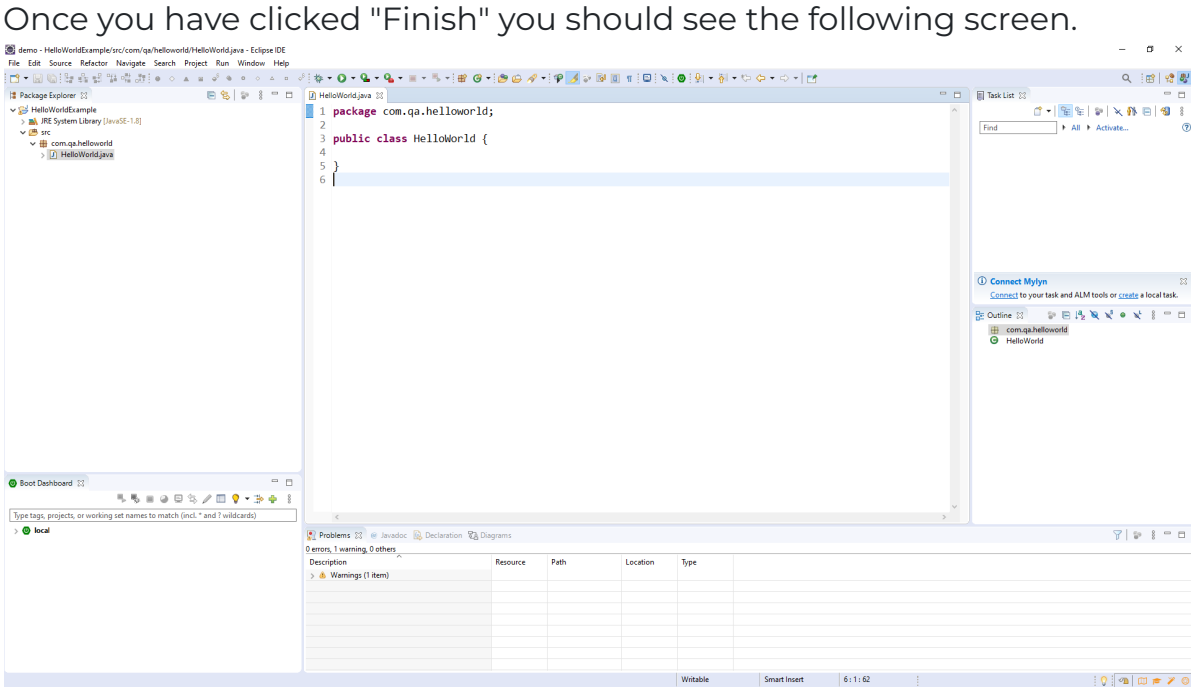


5. Now we need to create a class for the project so that we have somewhere to begin writing our code.
   To create a new class, expand the project by clicking the arrow to the left of the project name, then right-click on the newly created package name `com.qa.helloworld`, then hover over "New", then click "Class".
   You should now see the below window.



6. Enter a name for your class and click "Finish".
   The naming convention for classes is PascalCase, which means every new word has a capital letter at the beginning, with no spaces.

Once you have clicked "Finish" you should see the following screen.



Coding Hello World

Now that we have our project and class setup we can begin coding our application.
For our class to know what to run, we need to declare the main method in our class.

1. To declare the main method we need to add the following method within the scope of our class, the scope of the class is whatever is within the { }.

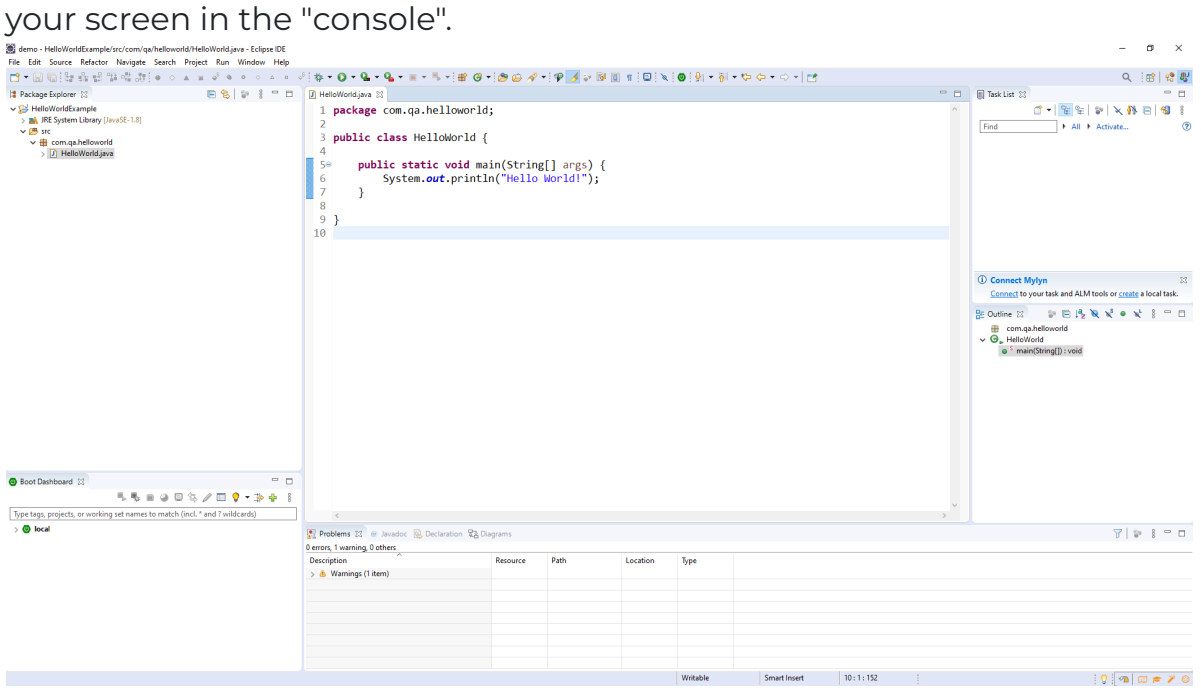
```
public static void main(String[] args) {

}
```

   Now when we run our project, anything code within the main method will be executed.

2. To tell Java to print "Hello World!" to the console we will need to use the following line, which makes Java print to the console.

```
System.out.println("Hello World!");
```

   If you run your code you will get the following output at the bottom of your screen in the "console".



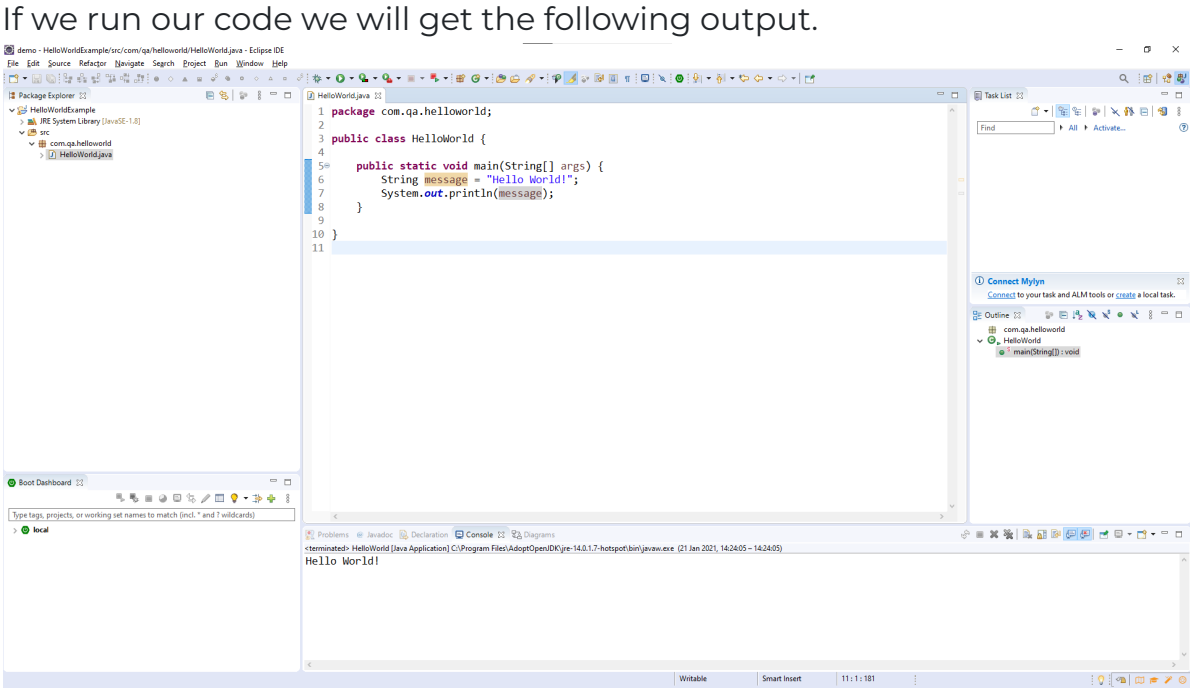
3. We can improve on this application by taking the text out of the System.out.println call and instead pass it as a variable.
   To pass the text as a variable we must first define and initialise a variable with the text "Hello World!".
   The variable we will be using needs to be initialised before we can use it in the System.out.println call, so it will need to go above that call in the method.
   To initialise the text as the variable use the following line above the System.out.println call.

```
String message = "Hello World!";
```

Now that we have our text initialised in a variable, we can use the variable in the System.out.println call like this:

```
System.out.println(message);
```

If we run our code we will get the following output.



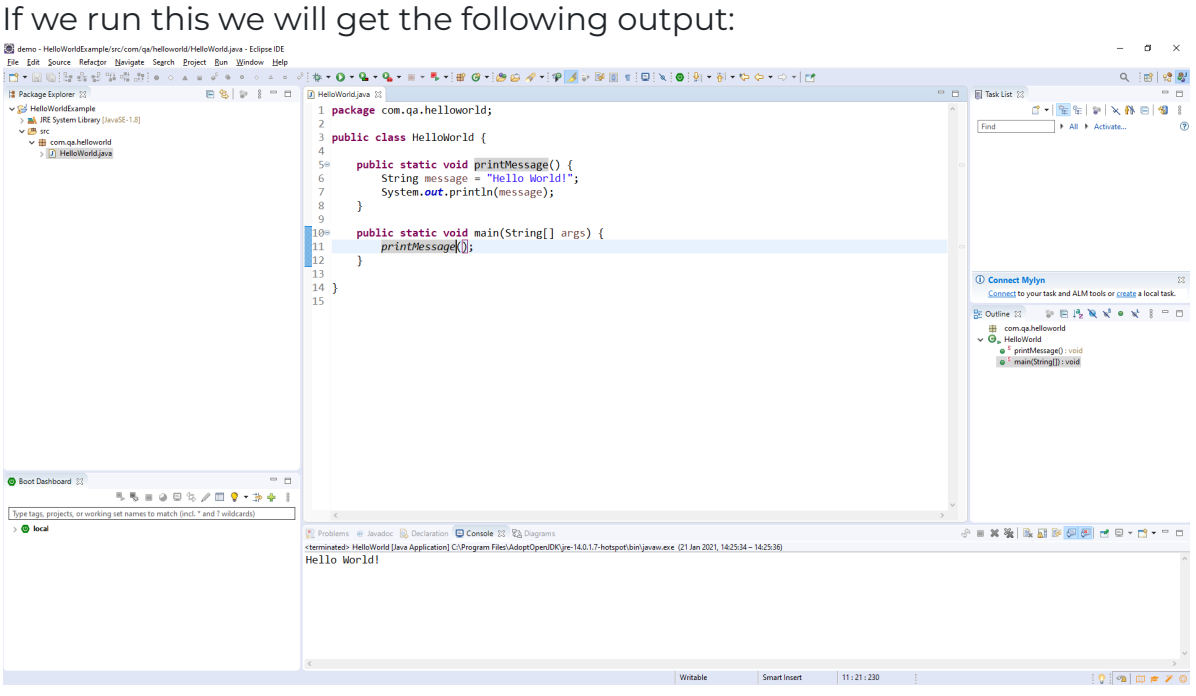
4. We can improve on this further by moving the variable initialisation and System.out.println call into its method so that it is more readable and reusable.
   To do this, we will need to declare a new method, which we can name whatever we want.
   The naming convention for methods is camelCase which means there are no spaces between words, the first word has no capital letters and each subsequent word has a capital letter at the beginning.
   For our new method we will use the following:

```
public static void printMessage() {
    String message = "Hello World!";
    System.out.println(message);
}
```

   Then we simply need to call our new method from the main method, to do this we just need to type the name of our method followed by parentheses and a semicolon like this:

```
public static void main(String[] args) {
    printMessage();
}
```

   If we run this we will get the following output:



5. We can once again improve on this further by having our new method take in a parameter so that the printMessage() method is even more reusable.
   A parameter is a variable local to the method only and is passed in when

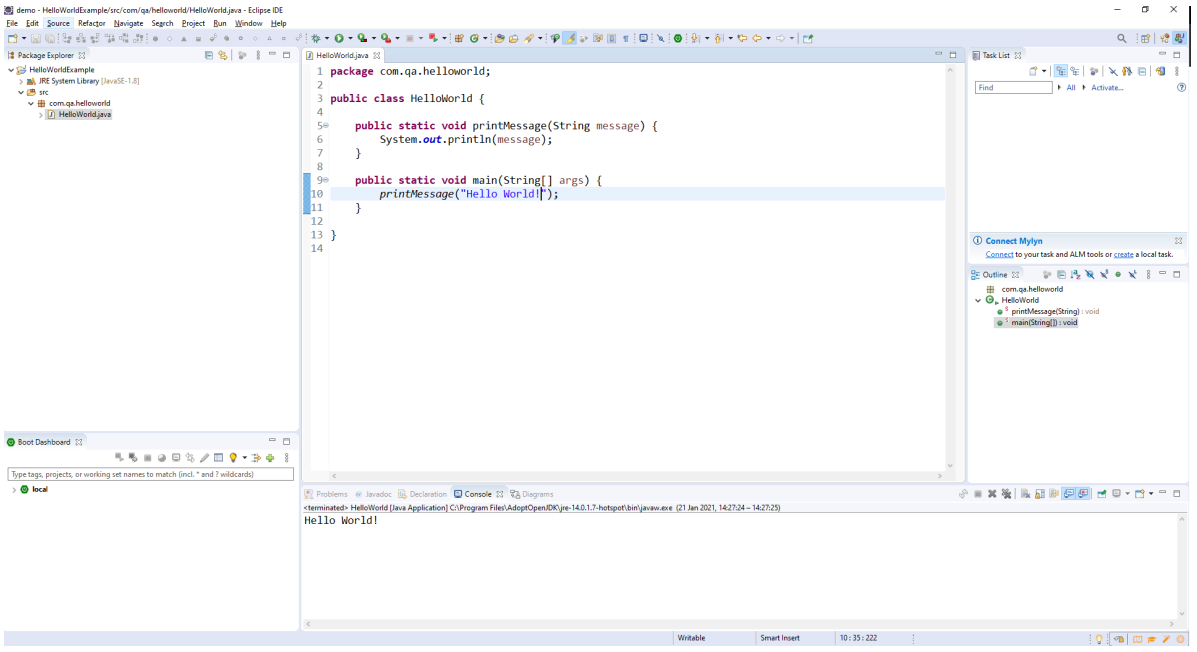
the method is called.
To have our new method take a parameter we will have to adjust the method declaration slightly by having a variable declared within the parentheses, like this:

```
public static void printMessage(String message) {
    System.out.println(message);
}
```

Since we are now passing our message in as a parameter there is no need to initialise it inside the method as this would interfere with the message we want to be printed.
To call our method from the main method we would do the following:

```
public static void main(String[] args) {
    printMessage("Hello World!");
}
```

We have to pass text into the method call so that Java knows what we want printing out to the console.
If we run this we will get the following output:



Exercises

1. Output "Hello World!" to the console.
2. Store "Hello World!" in a variable and then output it to the console.
3. Create a method that takes a String as a parameter, and then outputs it to the console.
4. Modify your method to **return** a String once called, which you then use to output to the console.