ECLIPSE IDE HANDBOOK

Hot Recipes for the Eclipse IDE

eclipse

JAVA CODE GEEKS

Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Eclipse IDE Handbook

Contents

Preface

Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages through the use of plugins (Source: https://en.wikipedia.org/wiki/Eclipse_(software)).

Eclipse provides IDEs and platforms for nearly every language and architecture. They are famous for their Java IDE, C/C++, JavaScript and PHP IDEs built on extensible platforms for creating desktop, Web and cloud IDEs. These platforms deliver the most extensive collection of add-on tools available for software developers (Source: https://www.eclipse.org/).

In this ebook, we provide a compilation of Eclipse tutorials that will help you kick-start your own programming projects. We cover a wide range of topics, from setup and configuration, to plugins installation and UI creation. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

About the Author

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike.

JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

You can find them online at https://www.javacodegeeks.com/

Chapter 1

Eclipse Tutorial for Beginners

This is an article about the Eclipse Integrated Development Environment (IDE) for Java Developers, more specifically, the Mars release of the Eclipse IDE for Java Developers.

You will get a brief introduction about how to download, install, and use the software.

The following table shows an overview of the entire article:

1.1 Introduction

Eclipse is a well-known and respected Integrated Development Environment (IDE) developed by the Eclipse Foundation. Eclipse is beneficial to programmers because it aids in the development process by providing the following key features:

- An easy to use graphical user interface that navigates through your code hierarchy.
- Syntax highlighting that displays source code in a color code format to improve readability.
- Code completion that makes recommendations on methods and parameters as you type.
- Recommendations on how to fix errors and automatic error correction.
- A graphical debugger that allows for line-by-line code inspection.
- Single key compilation and execution of a program.
- Automatic code generation for commonly used patterns.
- Integration with source code version control repositories.

There are several benefits to the experienced programmer; however, novice programmers should use IDEs cautiously. Oftentimes, novice programmers become dependent on IDEs without really understanding what is going on behind the scenes, especially as it relates to code generation. Once a programmer understands the basics of writing code from scratch, an IDE is a powerful tool to speed up application development.

1.2 Requirements

In order to use the Mars release of Eclipse IDE for Java Developers, at a minimum, Java Development Kit (JDK) 7 is needed. The JDK includes the Java Runtime Environment (JRE), Java Virtual Machine (JVM), and all other tools needed to write, compile, and execute Java programs. If you already have JDK 7 or higher, you do not need to re-install it. However, if you do not have a JDK or an outdated version, go to Oracle and download the JDK.

1.3 Download

There are several versions of Eclipse that cover several different programming languages. Specifically, for this tutorial, we will cover the Mars release of Eclipse for Java Developers. To download Eclipse, go to the Eclipse IDE for Java Developers website. When you go to the website, you should see a page similar to what's shown below.

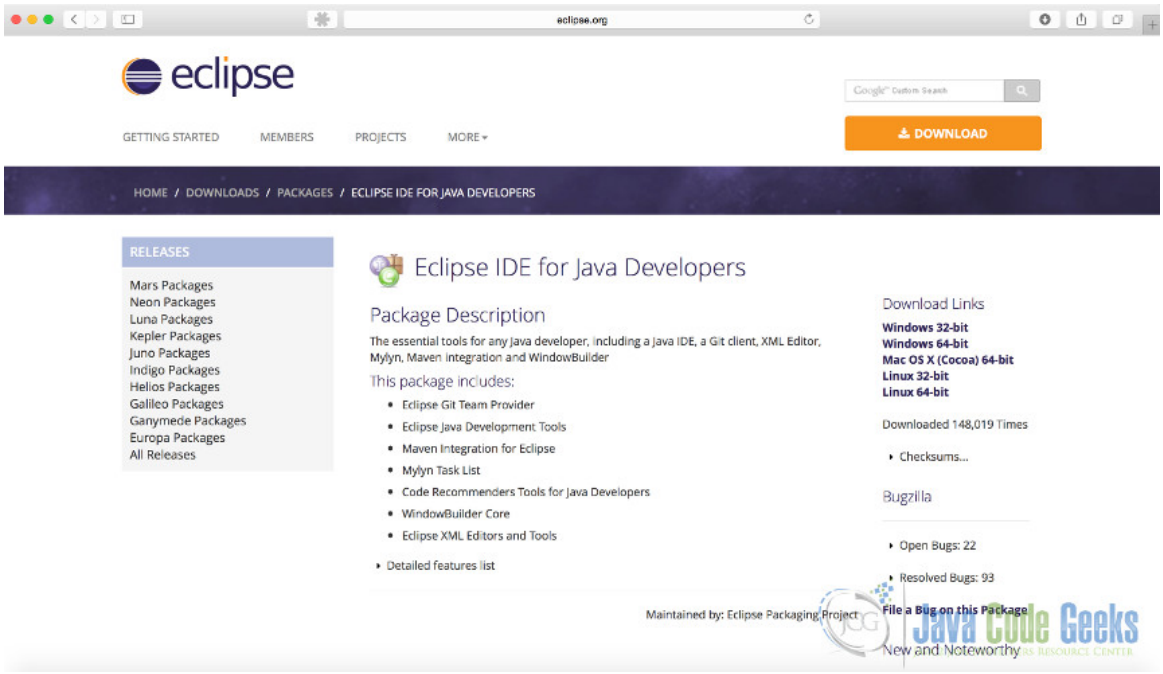

Figure 1.1: Eclipse IDE for Java Developers website

Click on "Mars Packages" in the upper right hand corner, which will bring you to a screen similar to what's shown below.

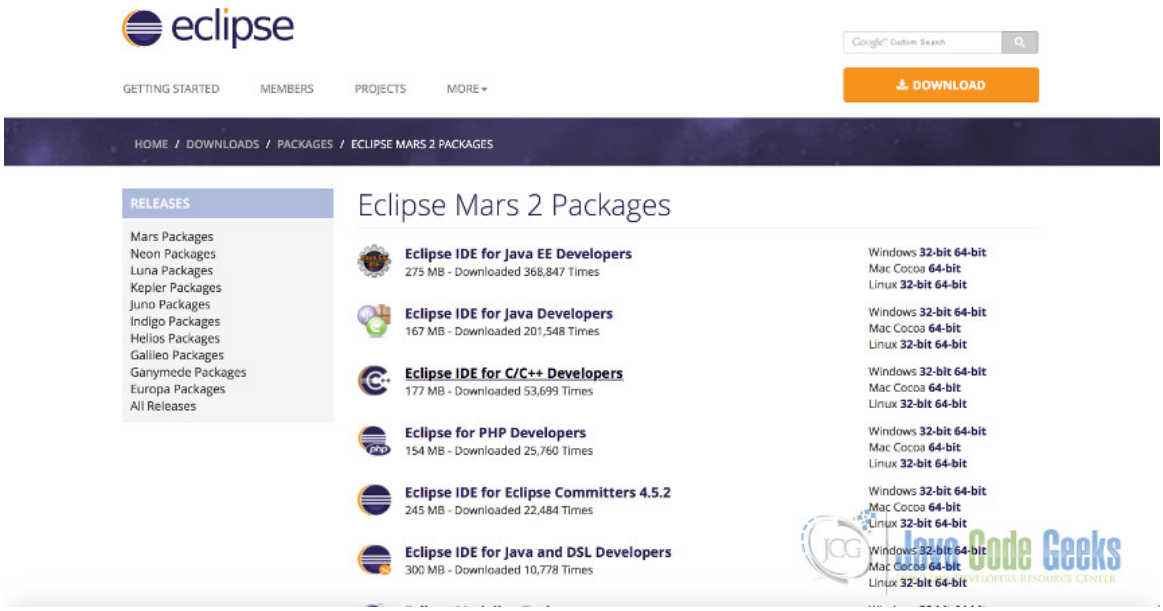

Figure 1.2: Eclipse Mars

As mentioned, there are various versions of Eclipse IDE for several languages. This tutorial covers the Eclipse IDE for Java Developers. The version to download is operating system and JDK version dependent, so make sure you download the version appropriate for your operating system and Java installation. For example, I am using a Mac with a 64-bit installation of Java, so I selected the “64-bit” link next to “Mac Cocoa” from the right-hand side of the screen.

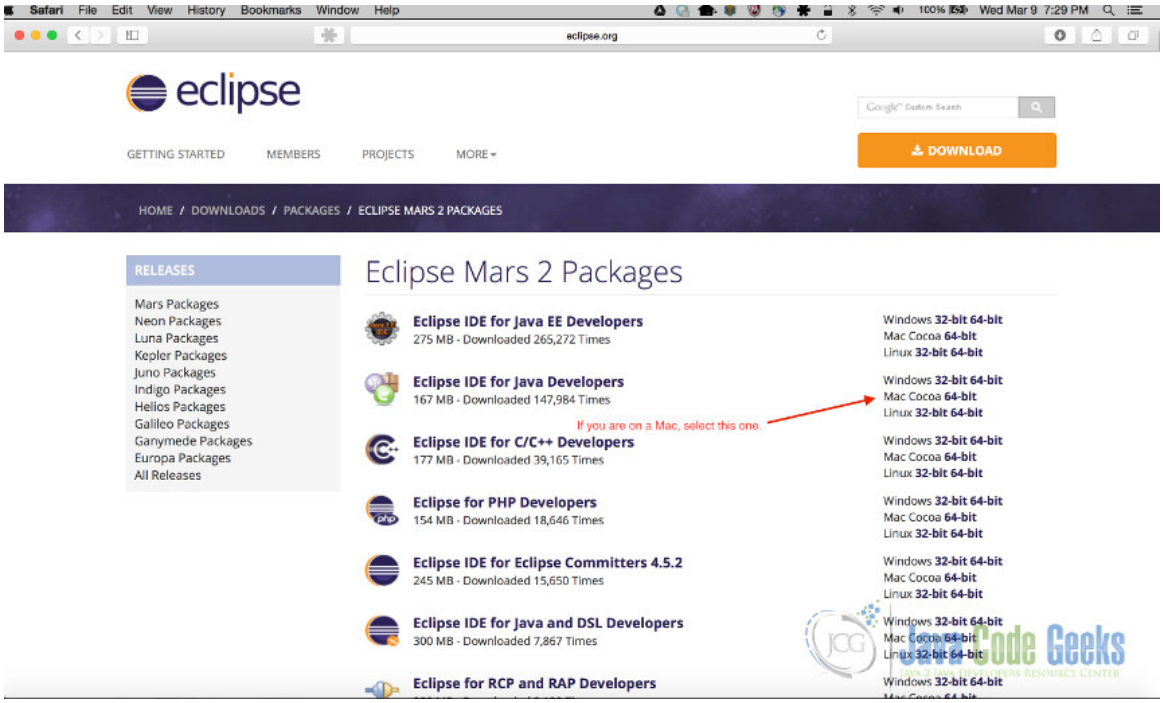

Figure 1.3: Eclipse Download

After clicking the link, you should see a screen that is similar to what’s shown below:

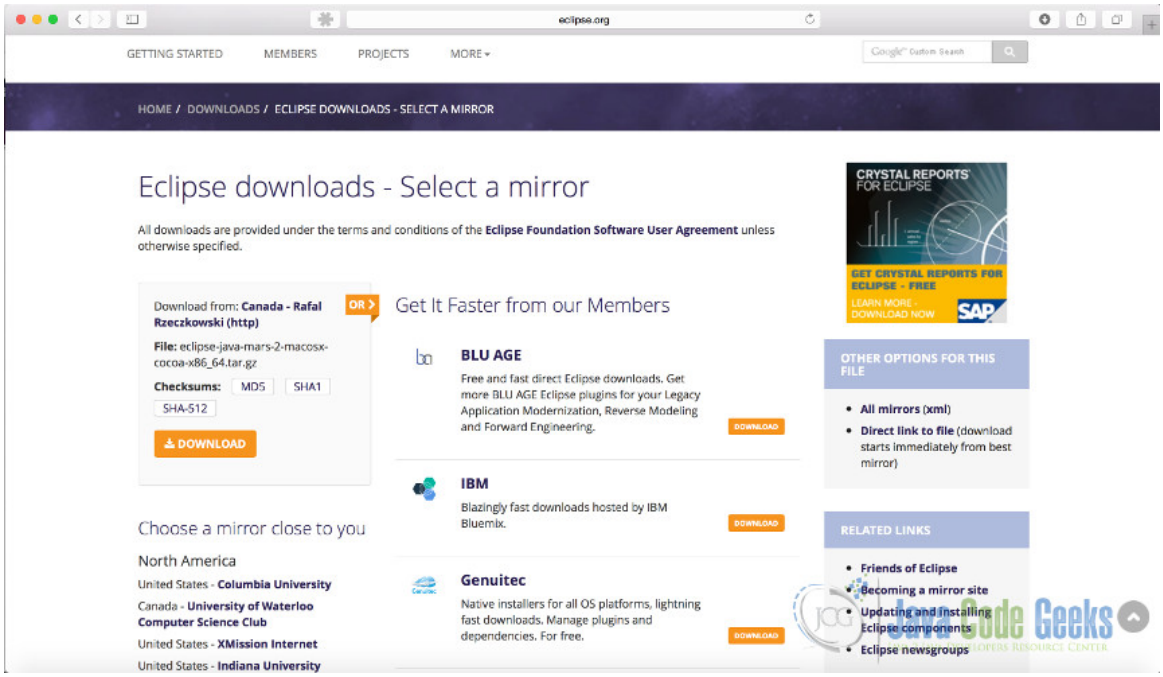

Figure 1.4: Mirror Sites

You should scroll down to view the mirror sites that are available in the “Choose a mirror close to you” section. Select the mirror that is closest to you in the list in order to speed up your download of the tool. Columbia University is the mirror closest to my location; therefore, I clicked on the Columbia University link, which displayed the screen below and started the download process.

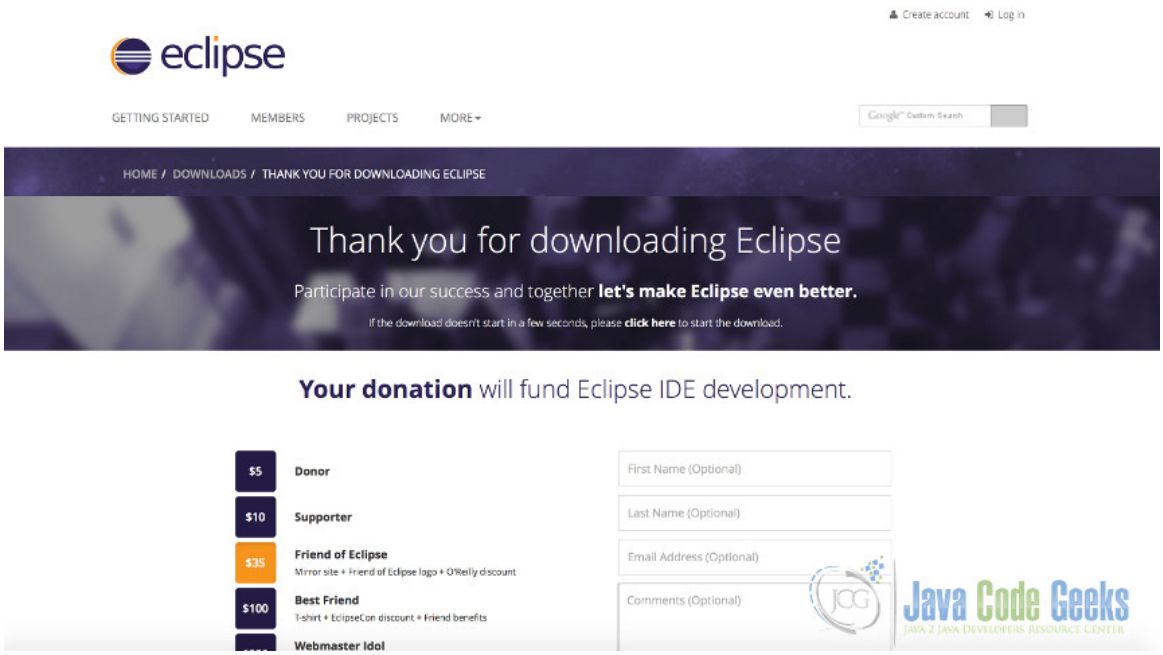

Figure 1.5: Download Success

1.4 Installation

Once your download completes, double click the file you downloaded to extract it to your chosen directory. After the compressed file is extracted, no further work is required to install Eclipse apart from making sure you have installed a JDK, which we covered in a previous step. If you are on a Mac, click on the “eclipse” icon file located in the folder where you extracted the file or if you are on Windows, double-click “eclipse.exe” to launch the application.

You may see startup errors if you have downloaded a version that doesn’t coincide with your operating system and/or Java installation version.

The Workspace Launcher dialog displays:

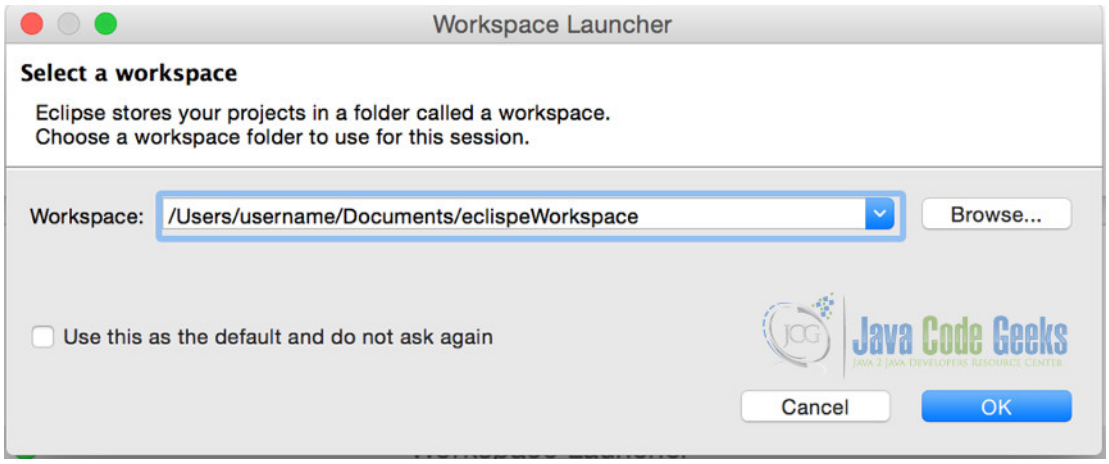

Figure 1.6: Launcher Dialog

An Eclipse workspace stores your Eclipse configuration and workspace data. Determine a directory where you would like the workspace to reside, enter a name for the workspace, and click "OK". For your information, a “workspace” just represents the physical location on your computer where your files will be stored. After clicking "OK", the “Welcome to the Eclipse IDE for Java Developers” screen displays. In the upper right hand corner, click on "Workbench".

Figure 1.7: Workbench

1.5 Tool Overview

After clicking on Workbench, the following screen displays:

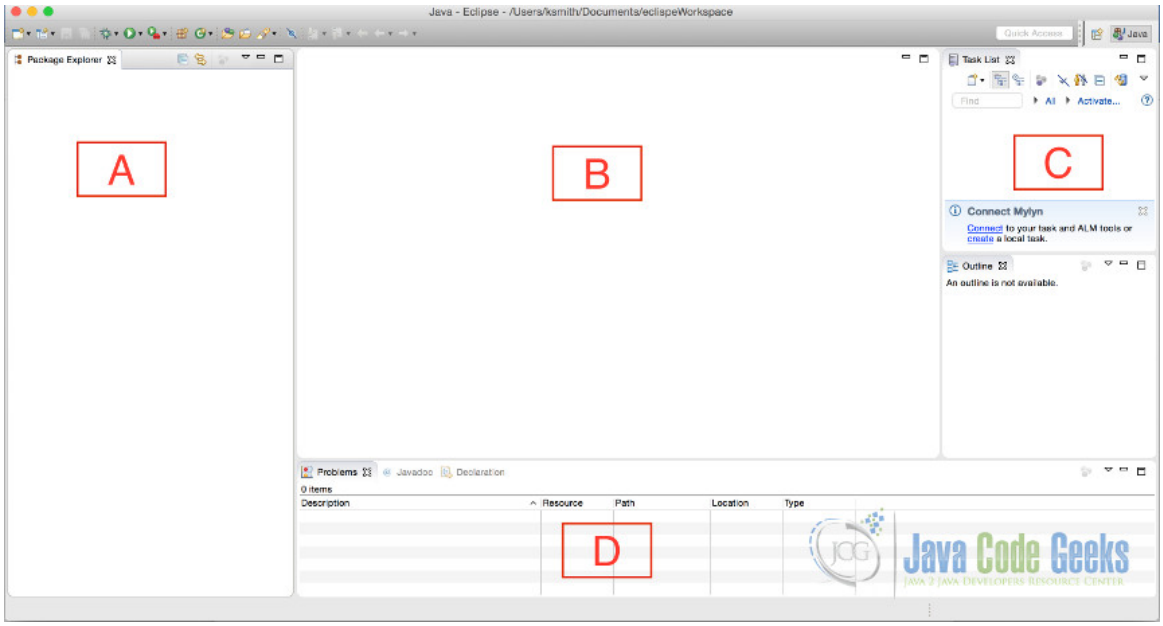

Figure 1.8: Tool Overview

When Eclipse initially launches, it defaults to a view of the “Java Perspective”, which appears in the upper right-hand corner.

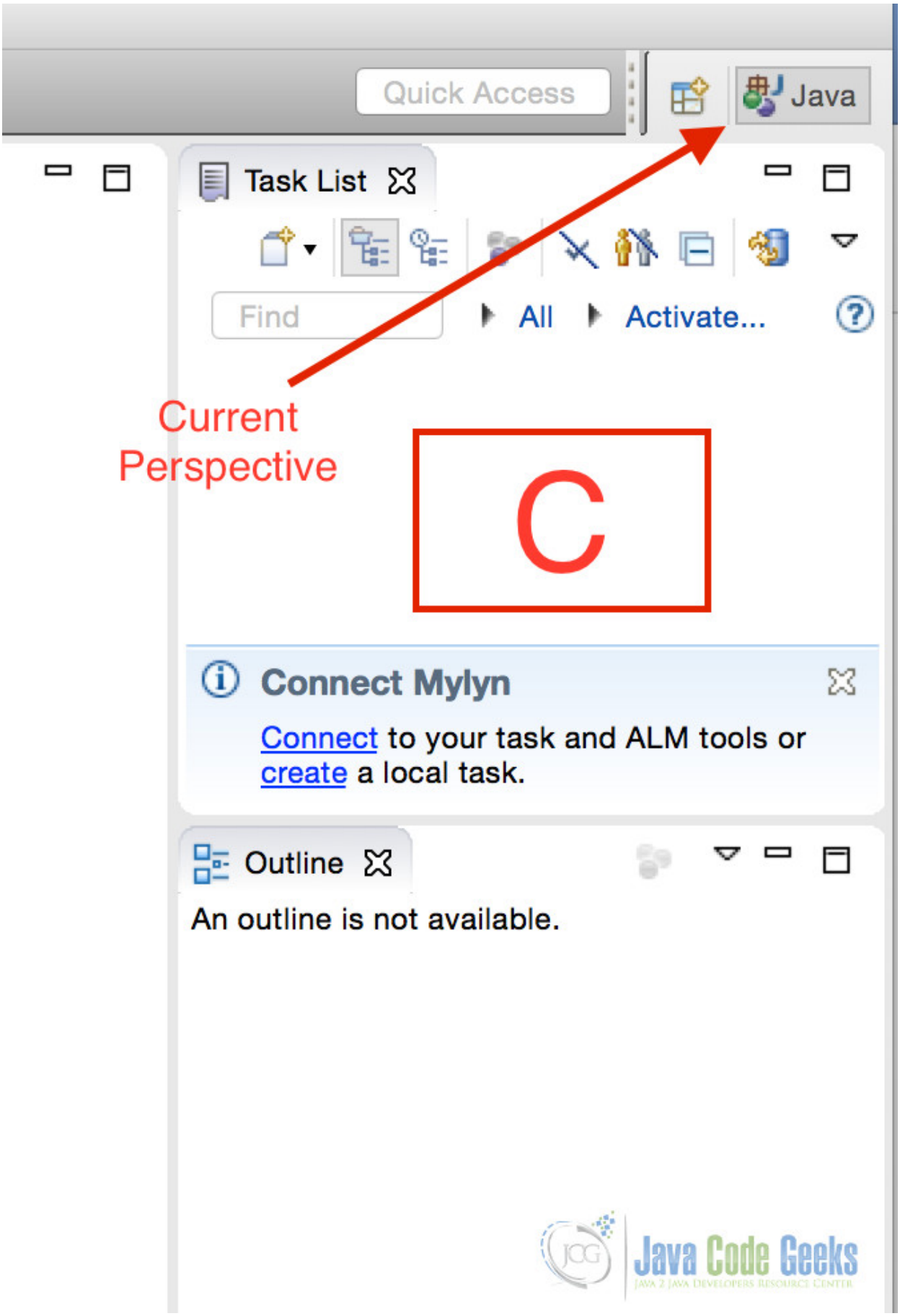

Figure 1.9: Java Perspective

In Eclipse, a Perspective is a grouping of related windows and features that allows for a developer to perform a specific set of tasks. The Java Perspective offers views and editors for creating and executing Java applications.

By default, the main window of the Eclipse IDE includes the following sections:

- Section A - Package Explorer
- Section B - Editor
- Section C - Task List
- Section D - Tabbed Views Pane

The Package Explorer allows you to navigate all of the files associated within a project.

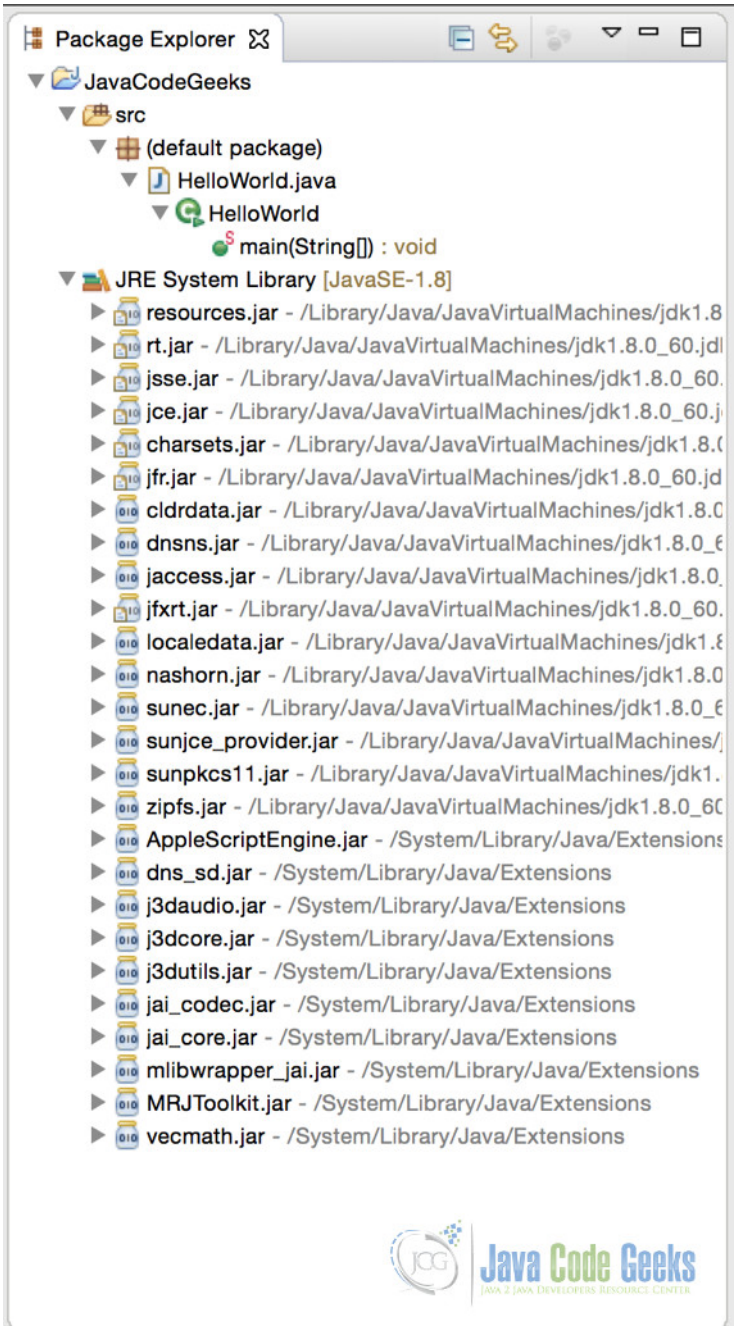

Figure 1.10: Package Explorer

You may open a file by double-clicking on it; the opened file appears in the Editor window.

The Editor window allows you to modify Java source code or text-based files. You may have more than one Editor window opened at once, each displaying a different file. The example below shows one file called, “HelloWorld.java”.

Figure 1.11: Editor Window

The Task List links to external bug tracking systems and displays assigned tasks. To learn more about the Task List, read about Eclipse Mylyn.

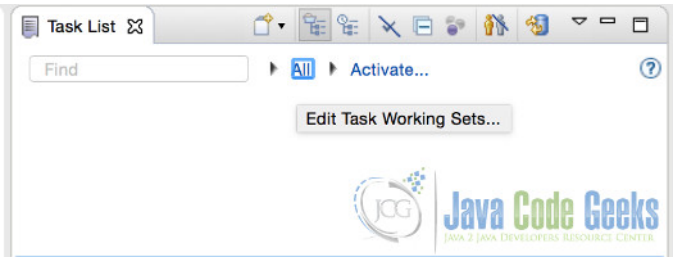

Figure 1.12: Task List

The Task List view is not to be confused with the “Tasks” view. The “Tasks” view is discussed in section the Tabbed Views Pane below.

The Outline window displays the structure of the file currently selected in the Editor window.

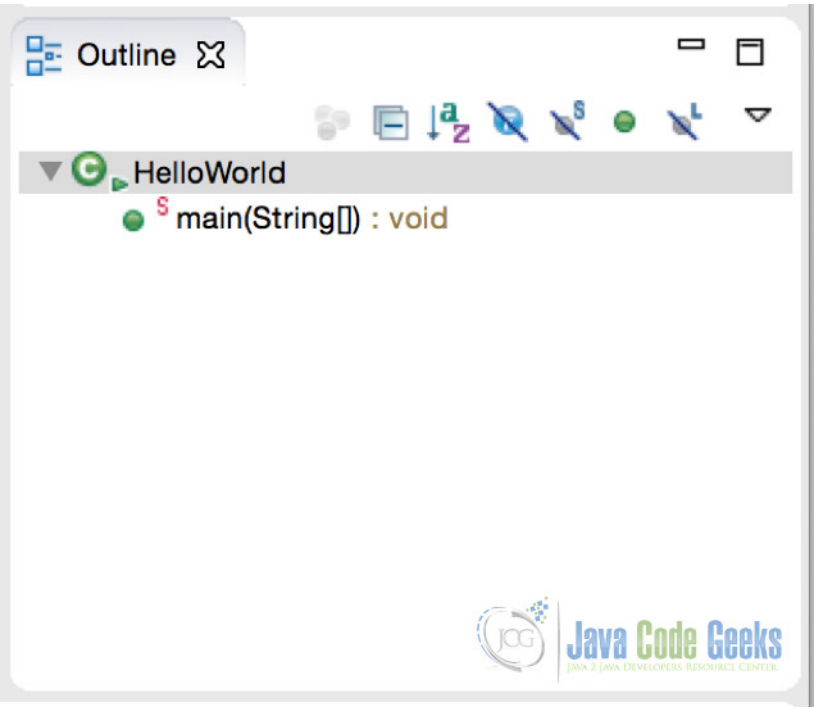

Figure 1.13: Outline View

The Tabbed Views Pane is located at the bottom of the screen and houses various views that can be hidden or shown based on developer preference. The default views that display within the tabbed pane are Problems, JavaDoc, and Declarations.

The Problems view shows any error messages or warnings associated with source code found in your project.

Figure 1.14: Problems View

The JavaDoc window shows the documentation for an item selected in the Editor window.

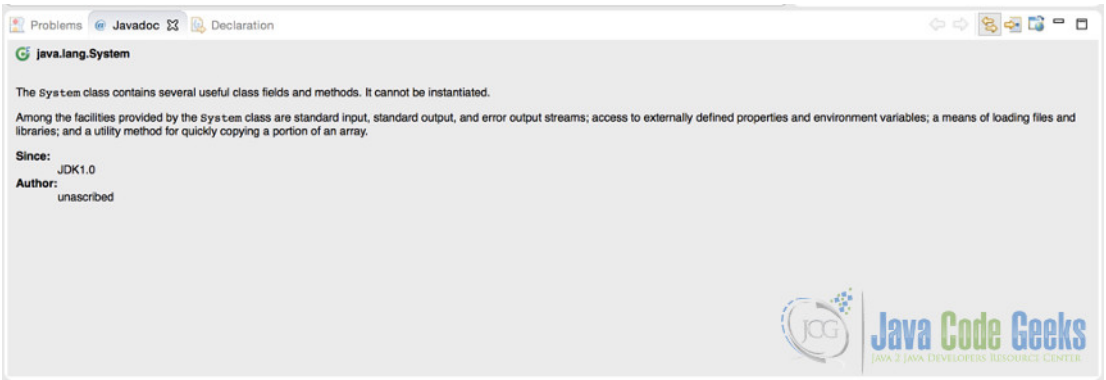

Figure 1.15: JavaDoc View

The Declarations window tells you about the declaration of the Java object currently selected in the Editor.

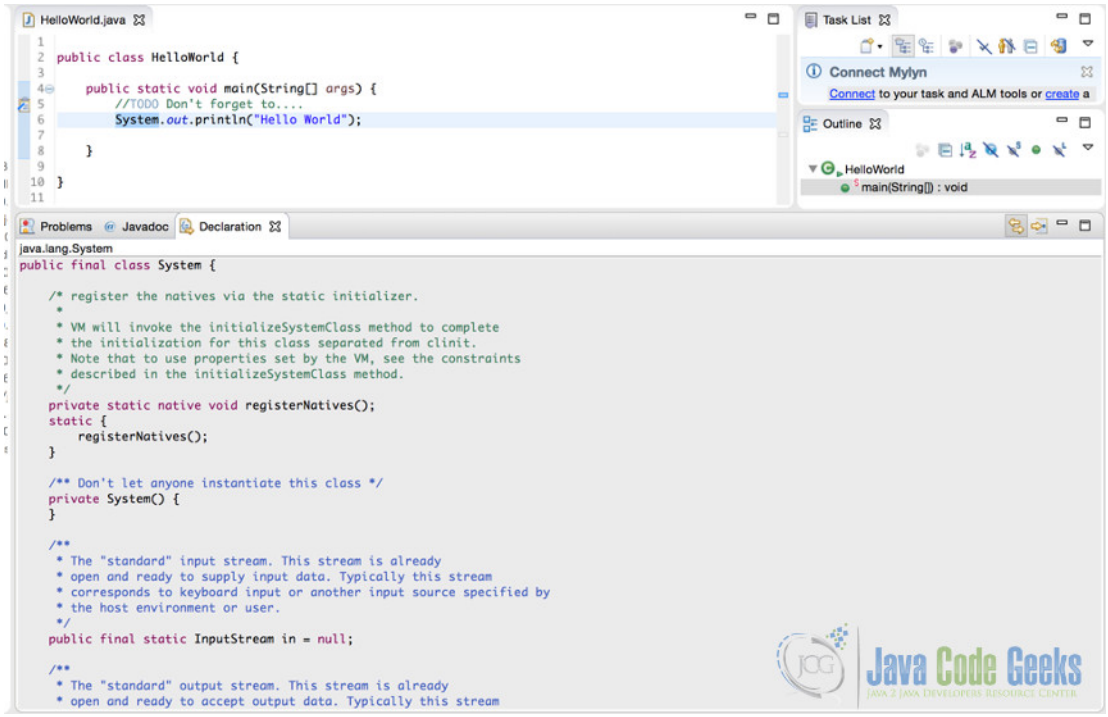

Figure 1.16: Declaration View

You may add additional views to the bottom tabbed pane by clicking on "Window→Show View" and selecting what you want to see.

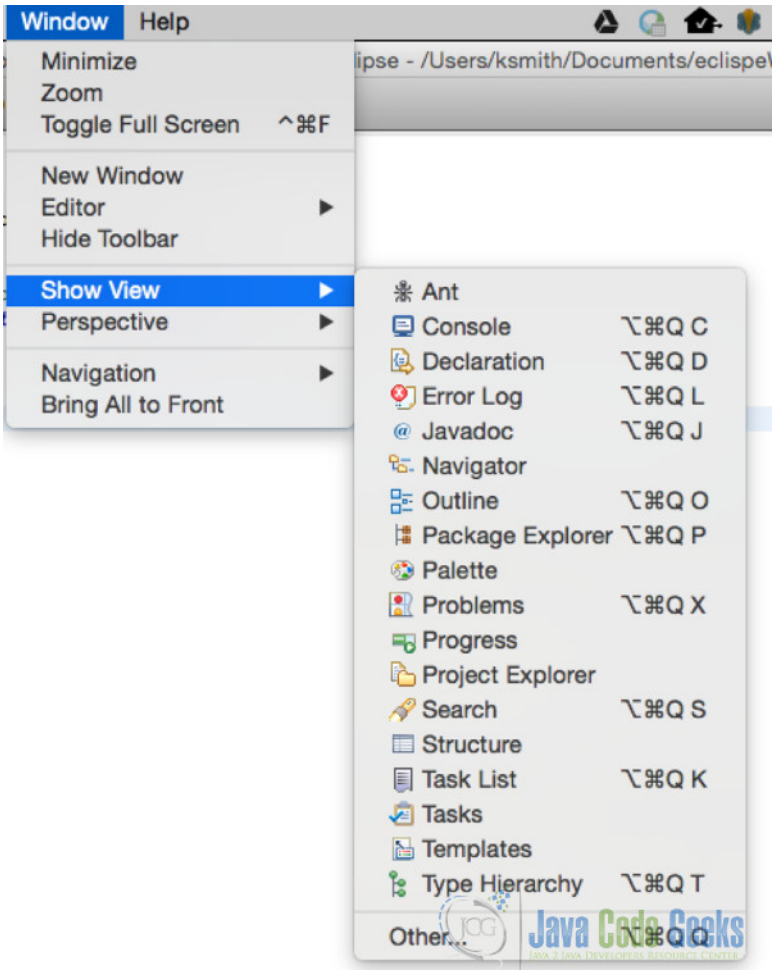

Figure 1.17: Show View

At a minimum, add the “Console” and “Task” tabs as they are very useful during development.

The “Console” view displays your program’s output or any runtime exceptions produced by your code.

Figure 1.18: Console View

The “Tasks” view displays markers, such as, “//TODO” that you have placed in your source code as a reminder to yourself to do something.

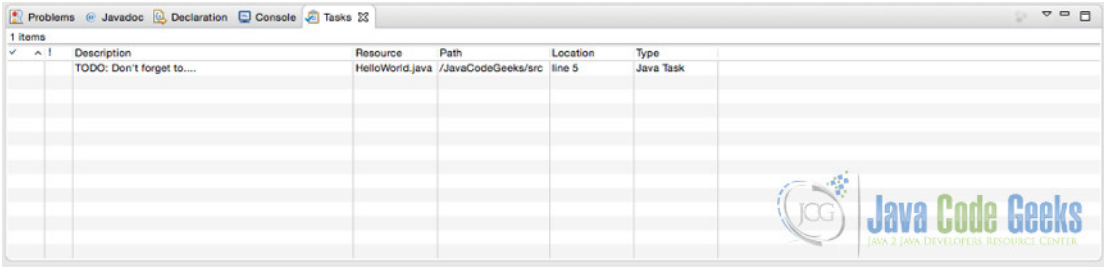

Figure 1.19: Tasks View

1.6 Tool Configuration

Before starting the first example, ensure that Eclipse is properly configured to your development preferences. From the Eclipse menu bar, select Eclipse→Preferences to open the Preferences dialog box.

Figure 1.20: Preferences

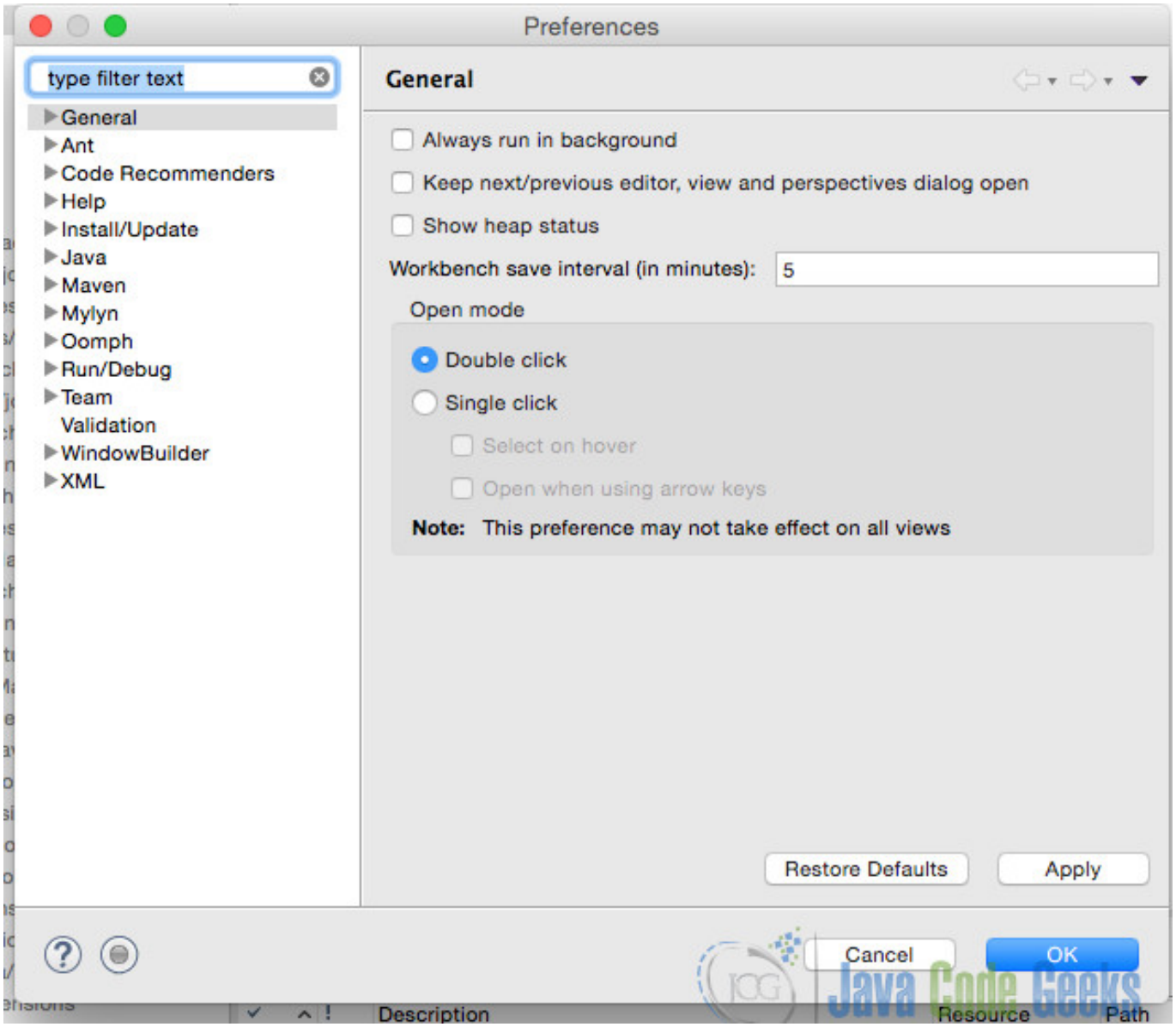

Figure 1.21: Preferences

In the Preferences dialog, you can setup the configuration information for your workspace and set your own development preferences. Expand the “Java” category and select the "Installed JREs" option.

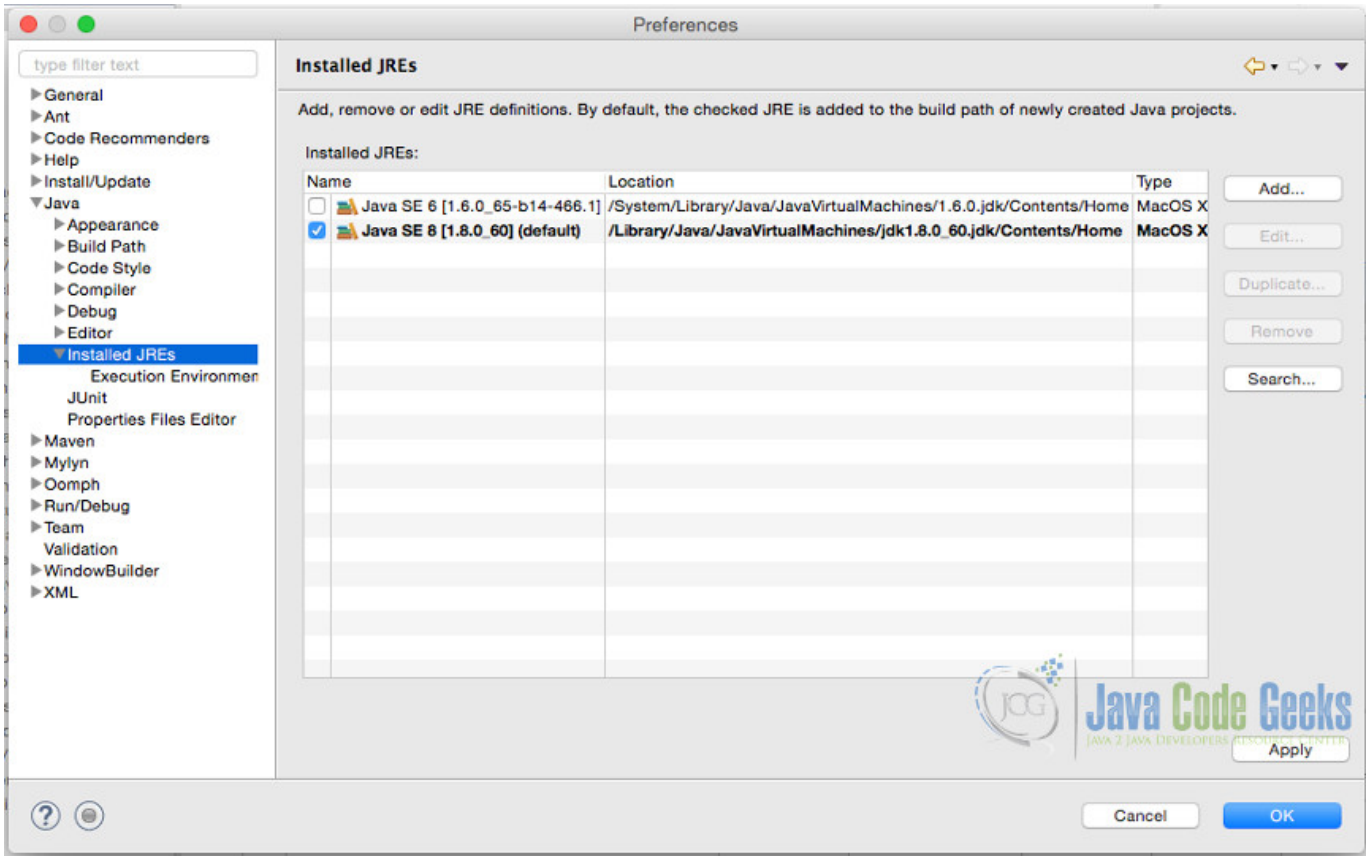

Figure 1.22: Installed JREs

Eclipse is smart enough to locate the JREs already installed on your computer. If you have more than one, select the appropriate JRE based on your preferences. If your JRE does not appear in the list, you may add it by clicking “Add”. Next, under the "Java" category, select the "Compiler" option and set the "Compiler Compliance" Level to the corresponding version of the JDK version you are using.

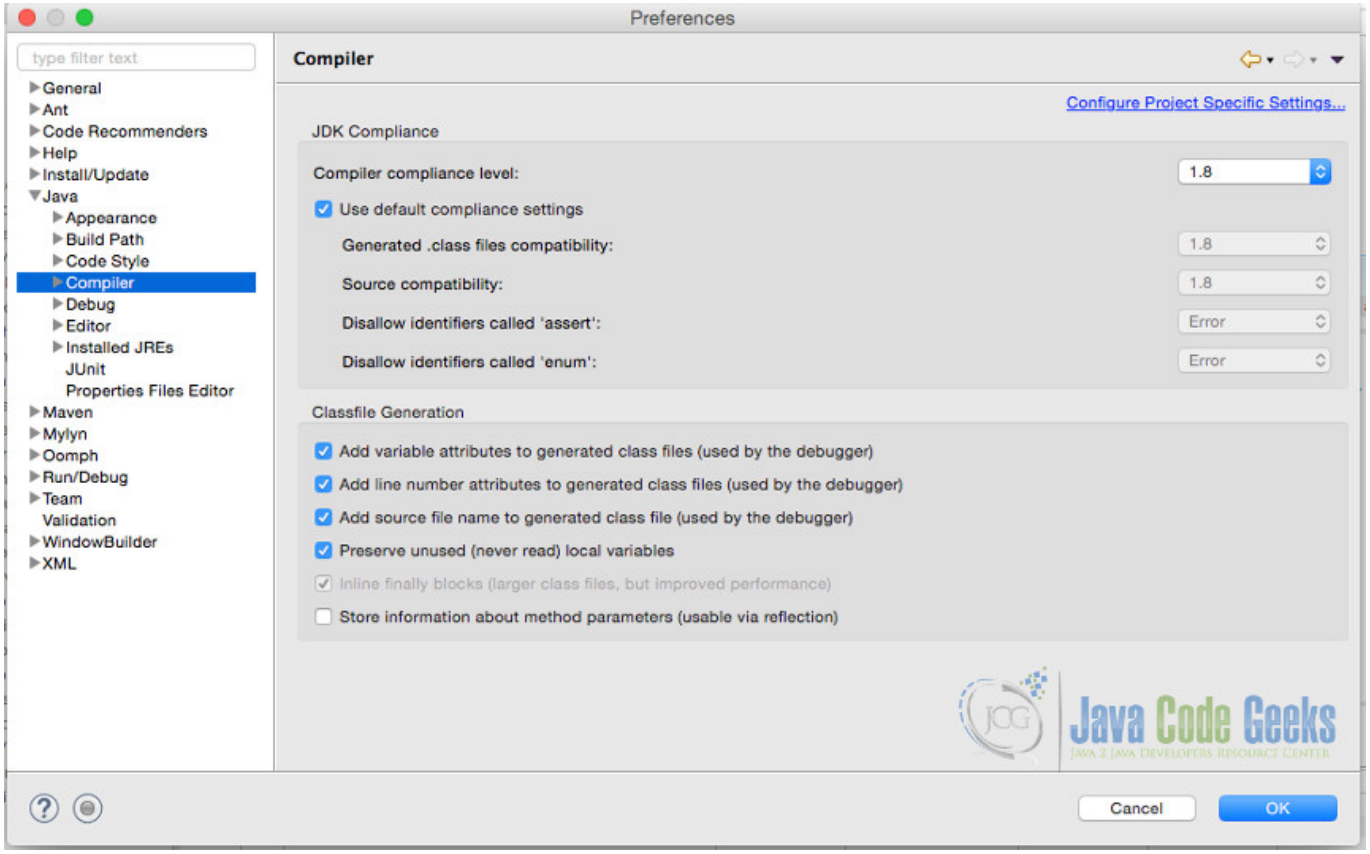

Figure 1.23: Compiler Options

Next, set your preferences for source code formatting by selecting "Java→Code Style→Formatter".

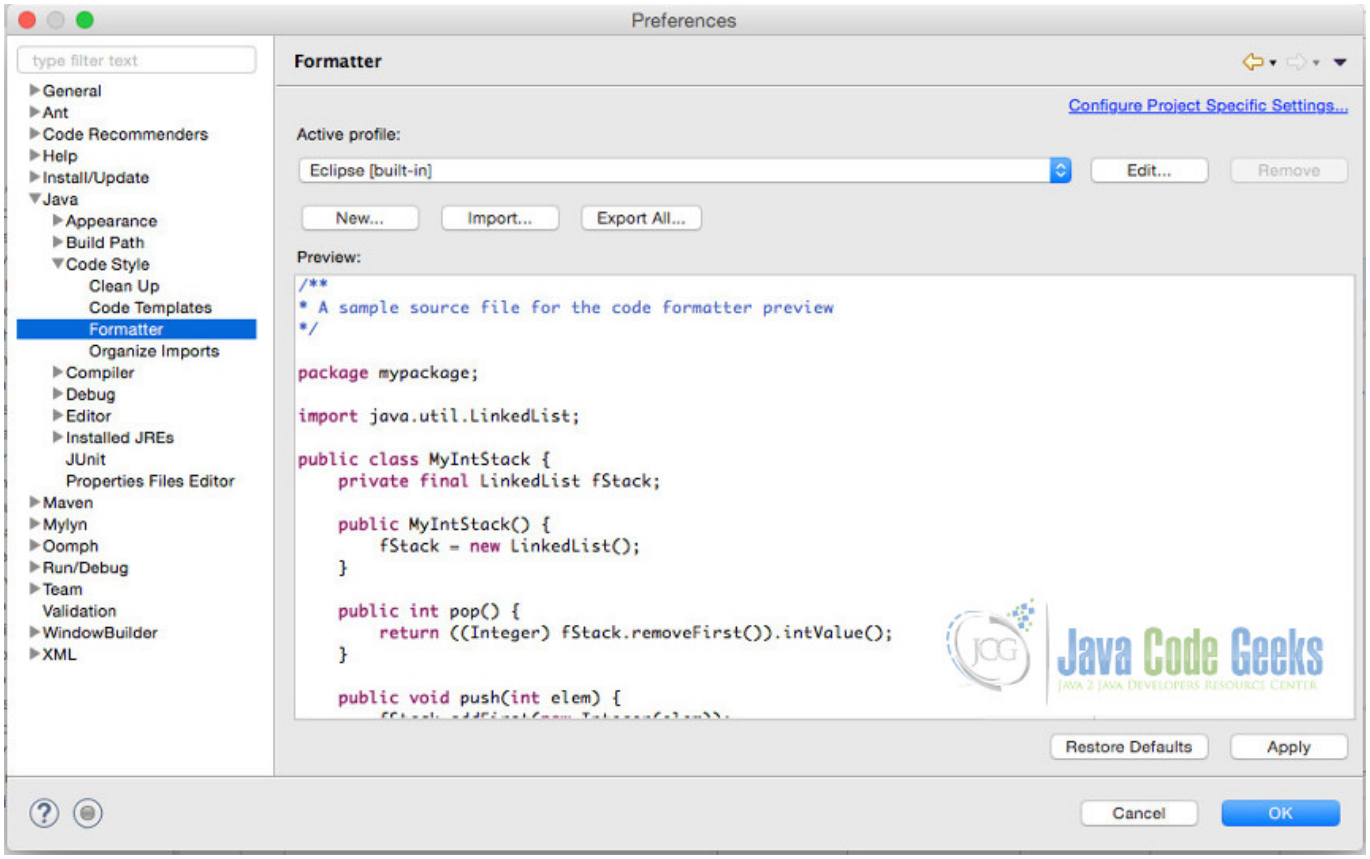

Figure 1.24: Code Formatter

The “Formatter” section contains workspace preferences for formatting source code. Under the "Active Profile Option", select the “Java Conventions [built-in]” option for the profile and click "Apply" and then "OK".

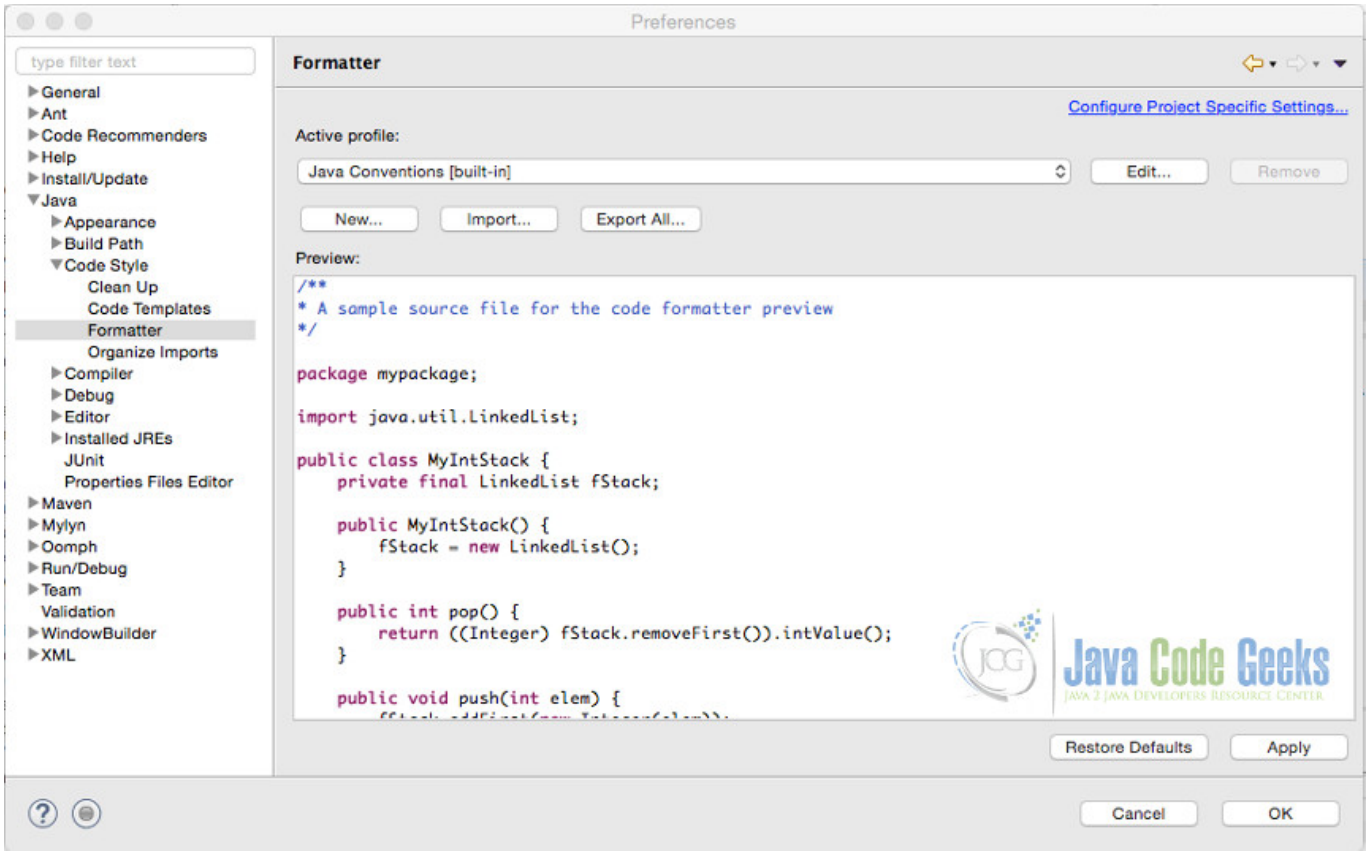

Figure 1.25: Java Conventions

Now, you are ready to start coding in the Eclipse IDE!

1.7 Hello World Example

1.7.1 Writing Your First Program

To begin developing a Java program using Eclipse, create a new project. A project groups source code, configuration settings, and other files into a deployable unit. From the "File" menu, select "New→Java" Project.

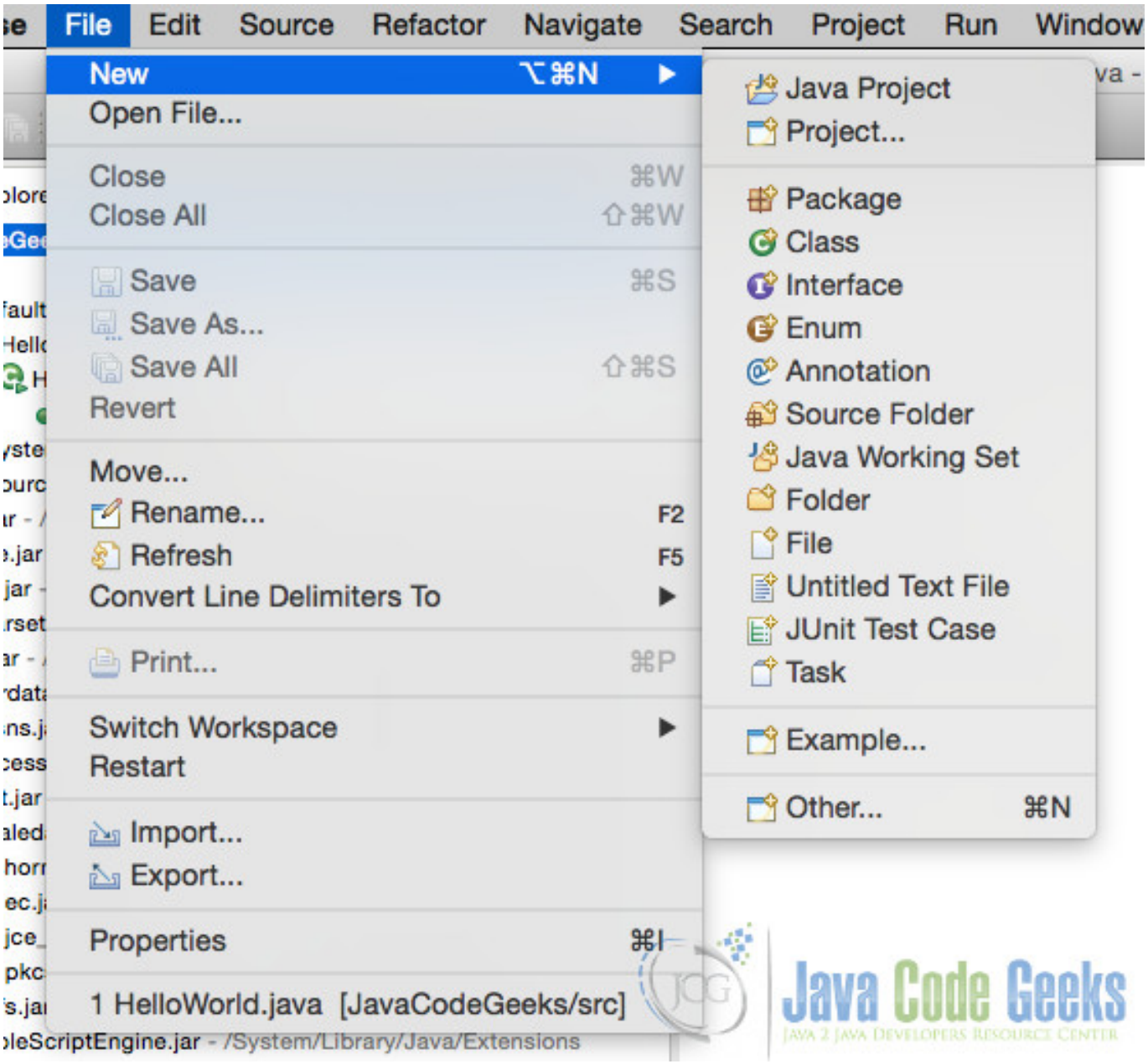

Figure 1.26: New Java Project

When the “New Java Project” wizard is displayed, enter a name for your new project; accept all defaults when stepping through the rest of the wizard and click ‘Finish’.

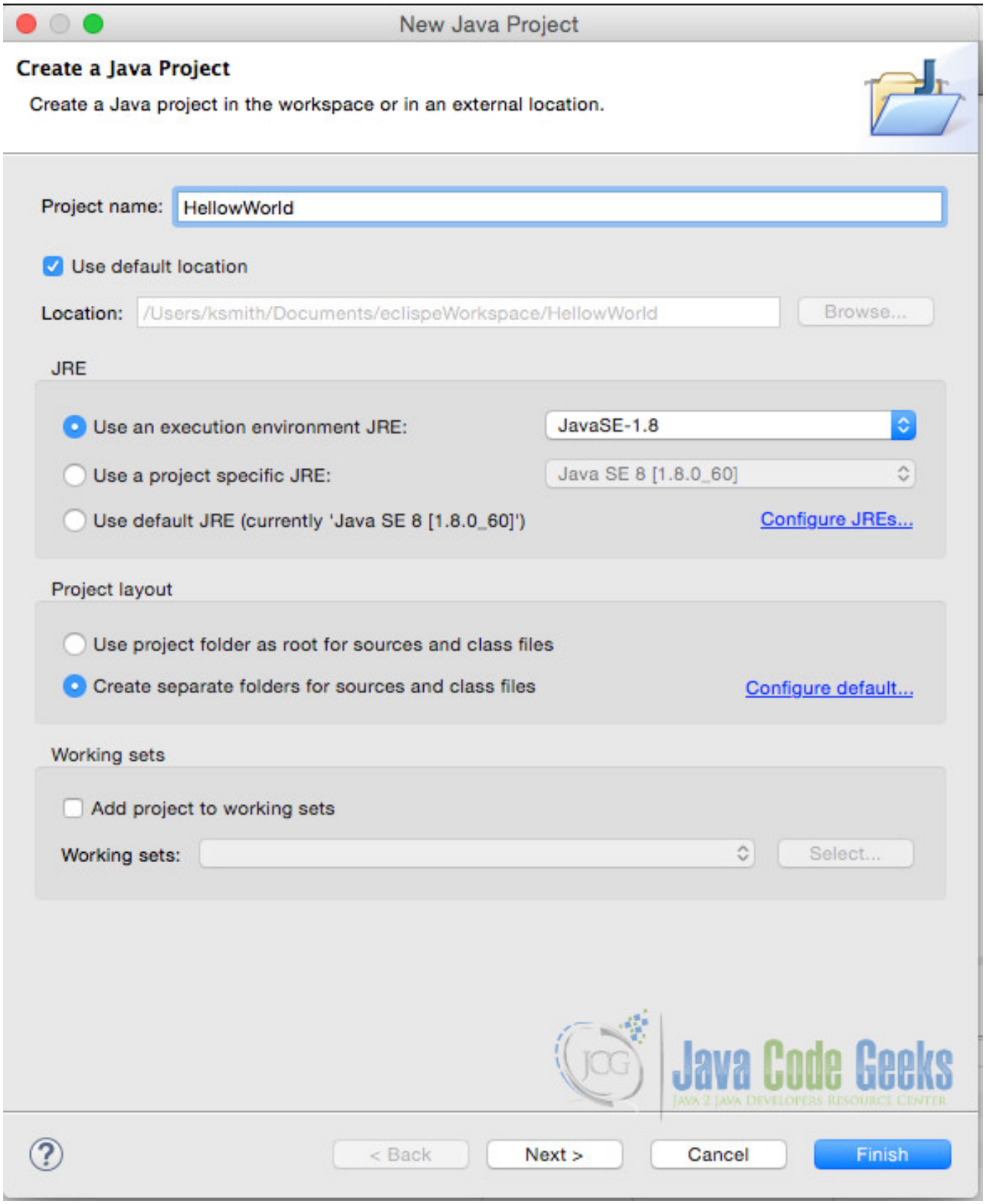

Figure 1.27: Create A Java Project

An empty project displays, as shown below.

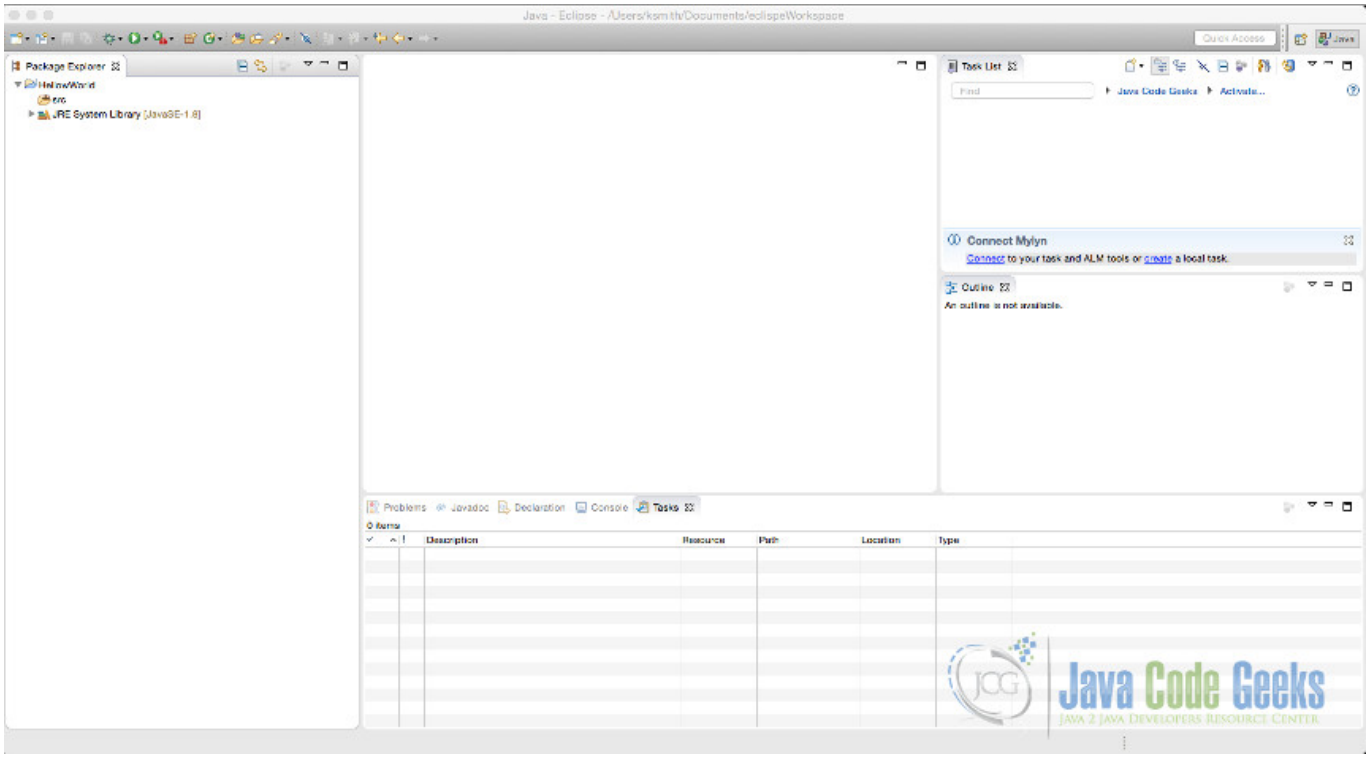

Figure 1.28: Empty Project

To add a class to your workspace, right click on your class name and select "New→Class". In my example, my project name is "HelloWorld" as shown below:

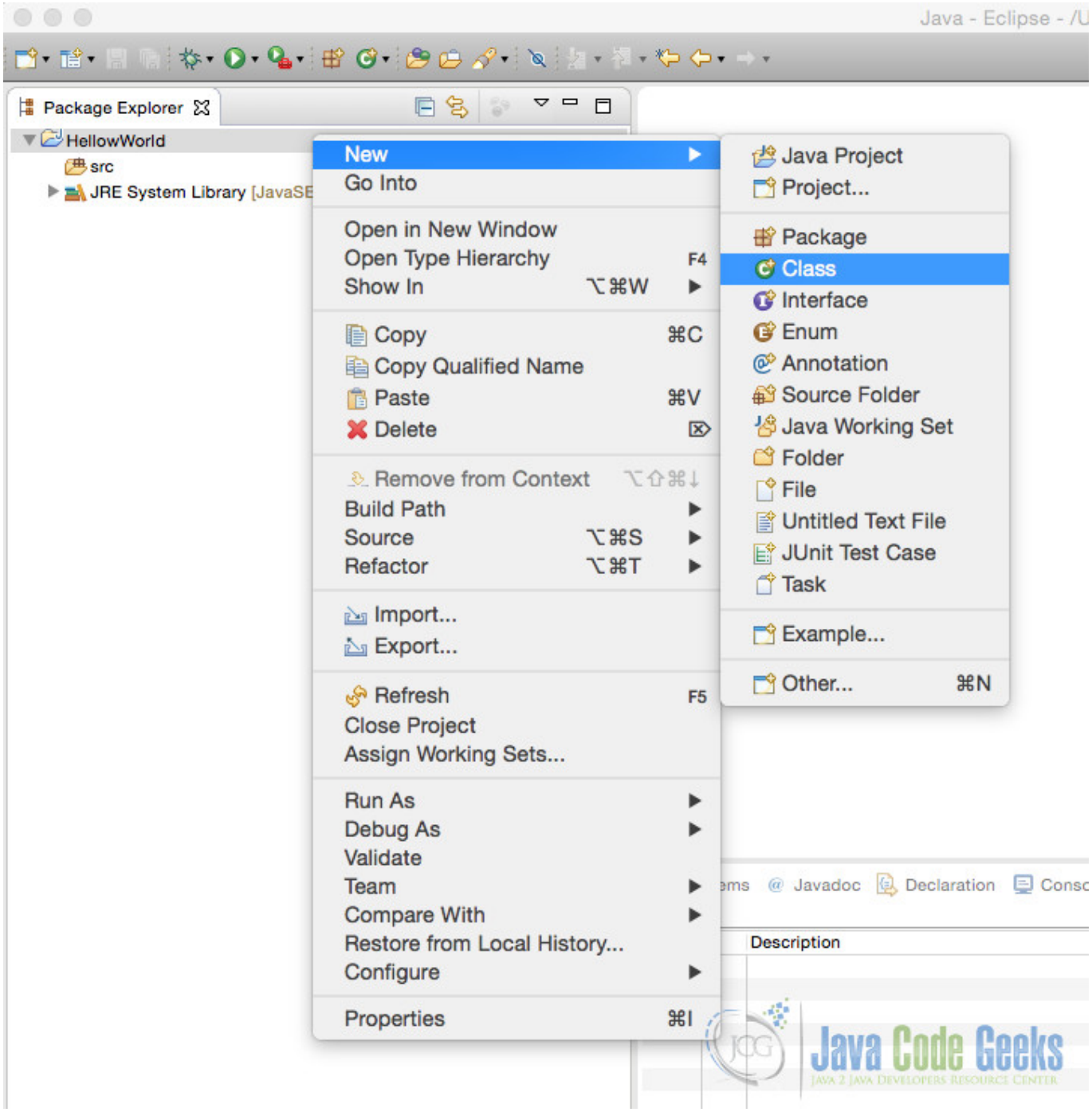

Figure 1.29: HelloWorld Class

Right click the name of your newly created package in the Package Explorer and click "New→Class". The New Java Class dialog displays.

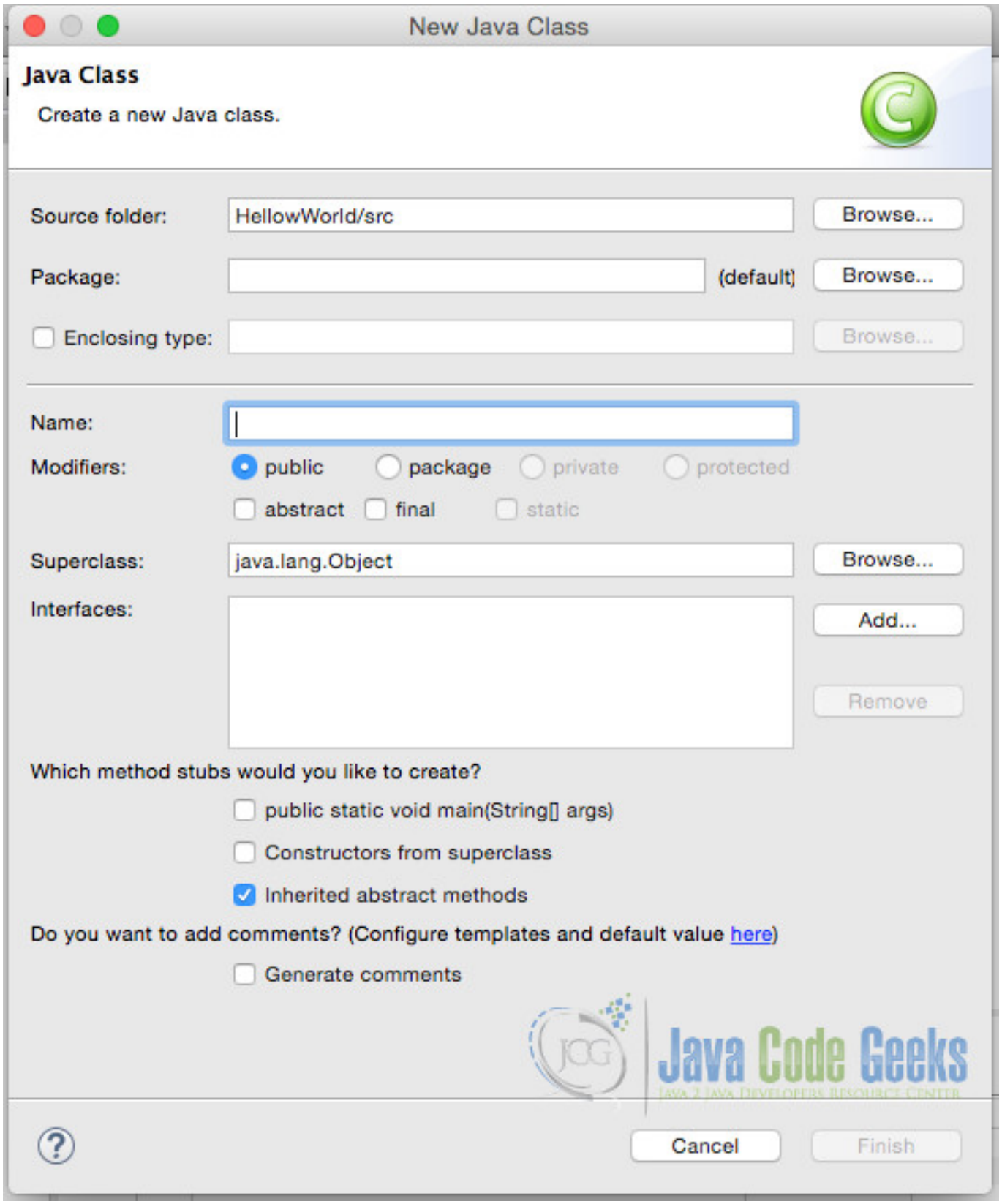

Figure 1.30: Class Dialog

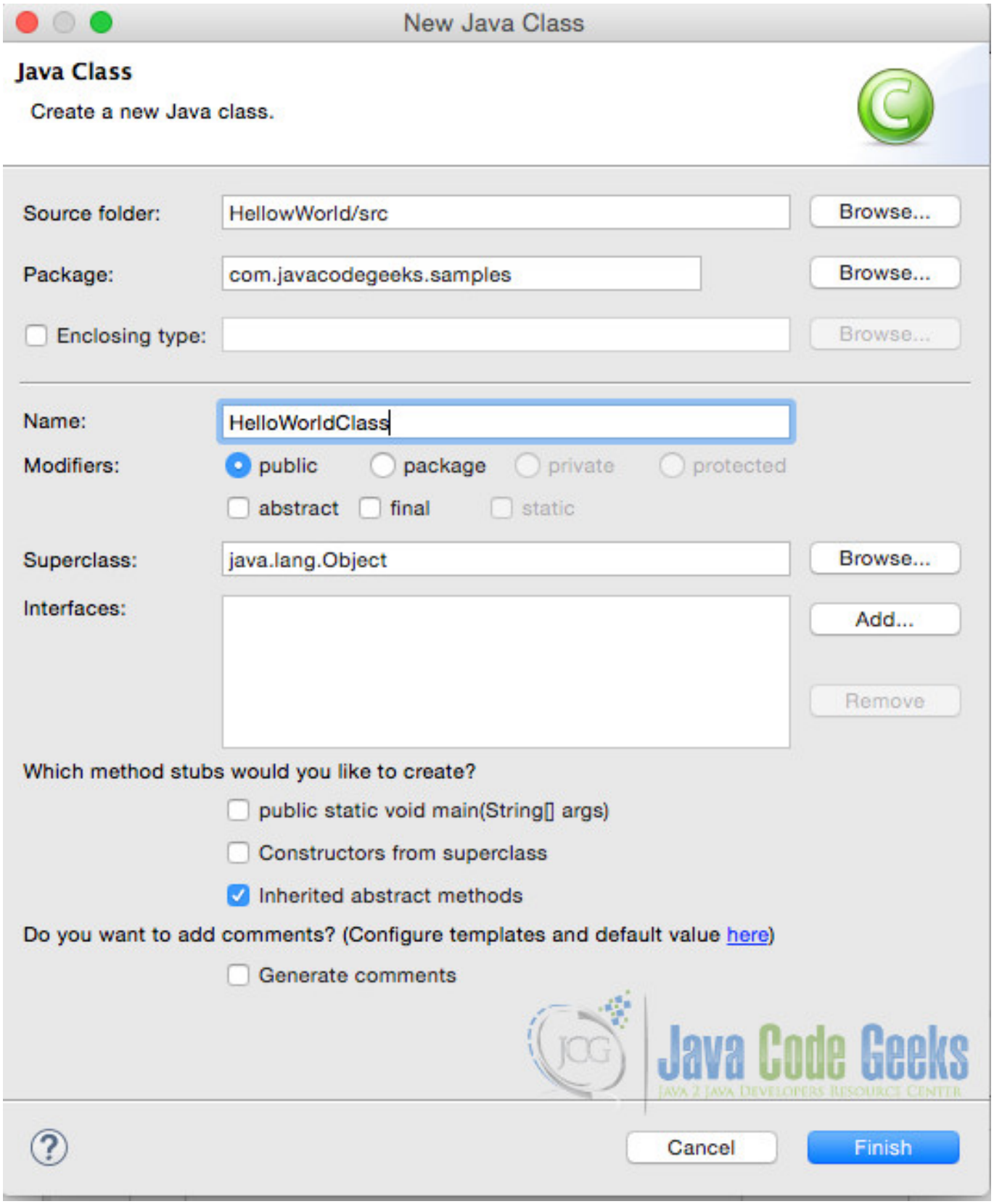

Figure 1.31: Hello World Class

Enter a “Package” and “Name” for your class following the example below.

Under the “Which method stubs would you like to create?”, leave the default of “Inherited abstract methods” and also select “public static void main(String[] args)” and click “Finish”.

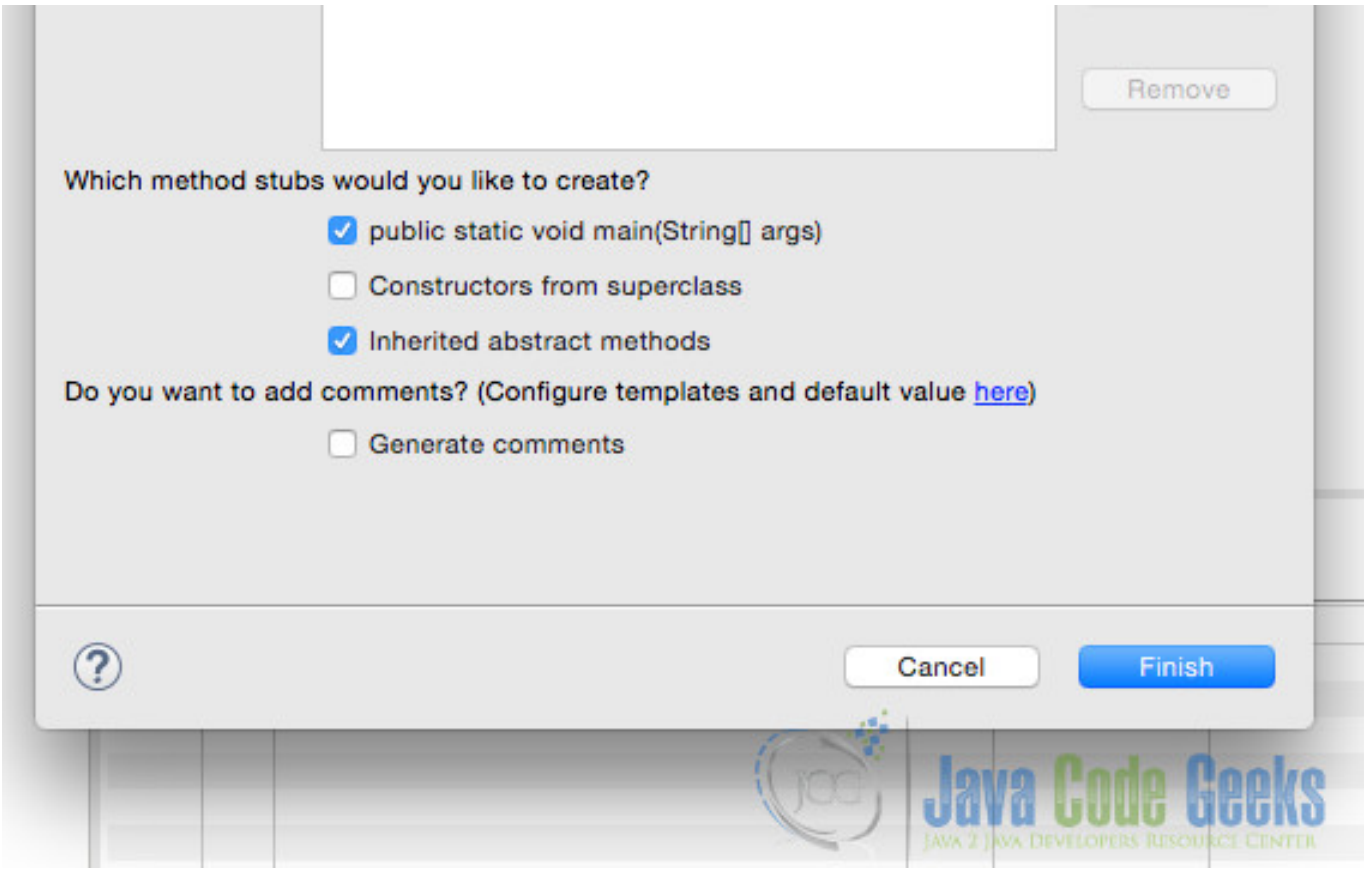

Figure 1.32: Method Stubs

Eclipse generates a class stub that contains several necessary items:

- package line
- class name
- default main method
- default TODO statements

The next step is to add code to your main method.

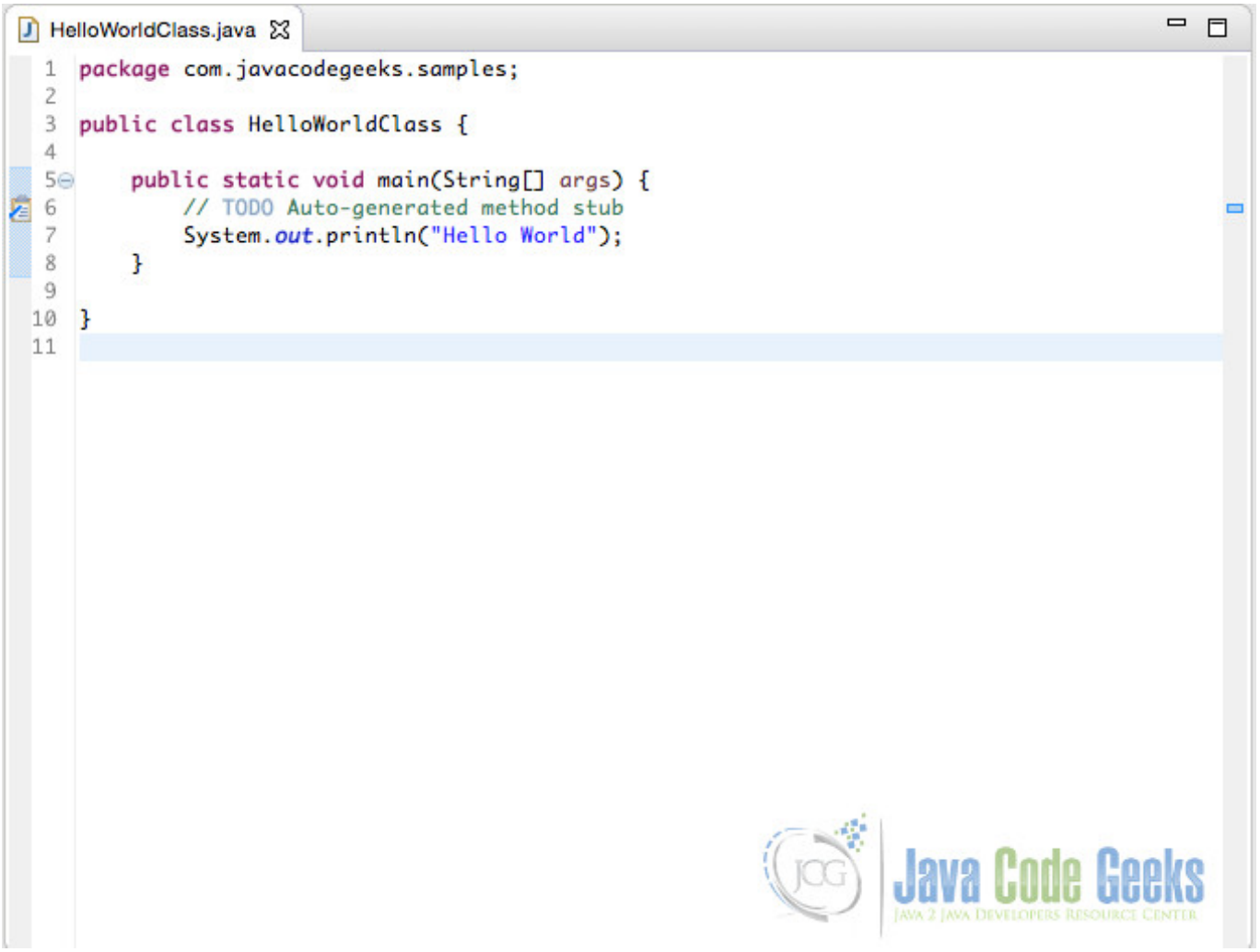

Figure 1.33: HelloWorldClass.java

```
package com.javacodegeeks.samples;

public class HelloWorldClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Hello World");
    }

}
```

1.7.2 Executing Your First Program

After adding code to the main method, the program can be executed within the Eclipse IDE environment. If you've added the Console view to the bottom tabbed pane of your Eclipse IDE, the execution output of your program will display there. To execute your program, right click on your project name and select "Run As→Java" Application.

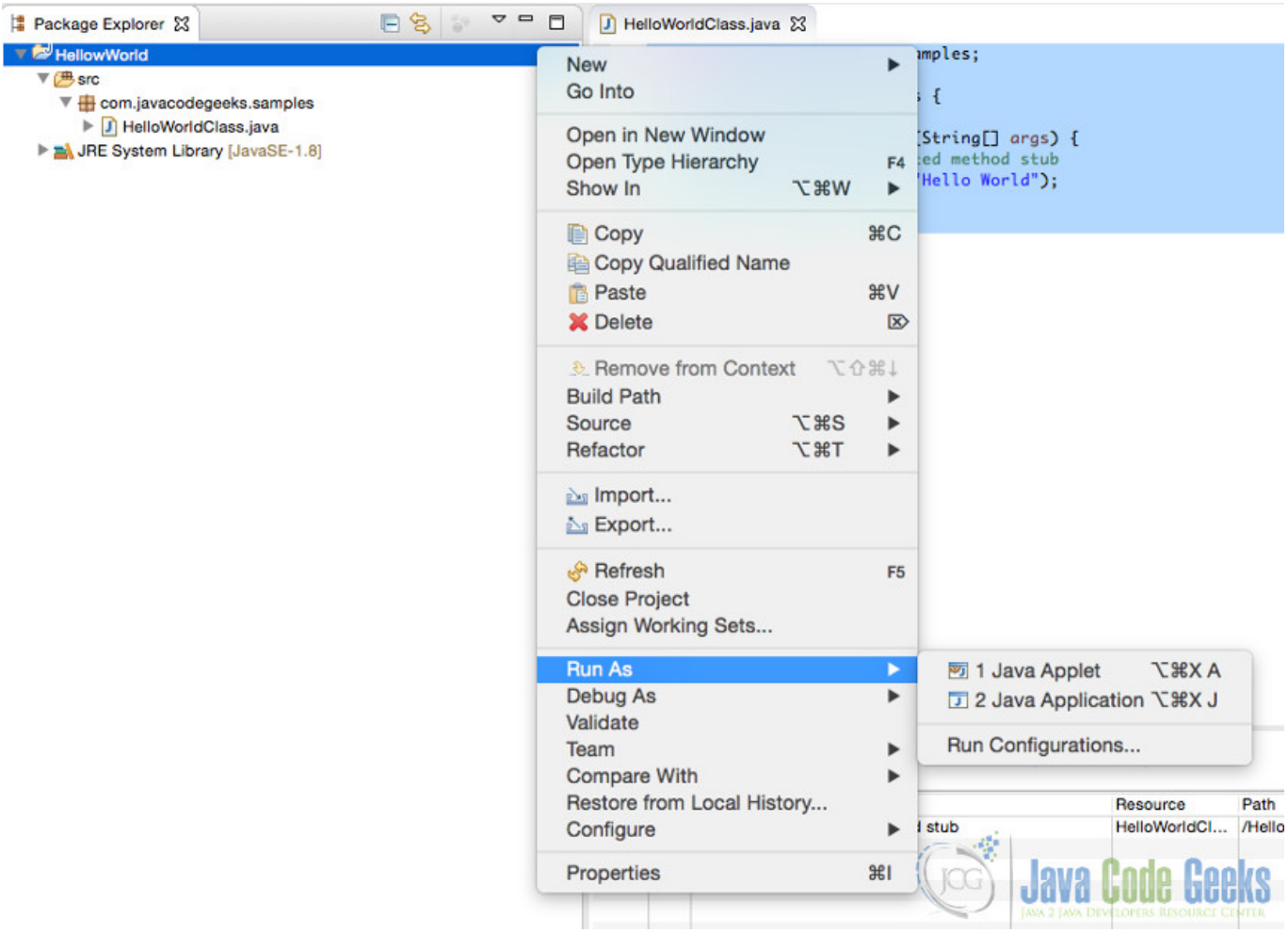

Figure 1.34: Executing Your Program

The output will display in the Console tab.

Figure 1.35: Console Tab Output

Congratulations! You have written and executed your first Java program in the Eclipse IDE.

1.7.3 Debugging Your First Program

Even before compiling a program in Eclipse, the editor will display problems in your program using Eclipse's auto-correction feature. Notice the red X in the left-most column as it indicates a problem in your code. Upon double-clicking on the red X, a description of the problem and some helpful options to fix the error display.

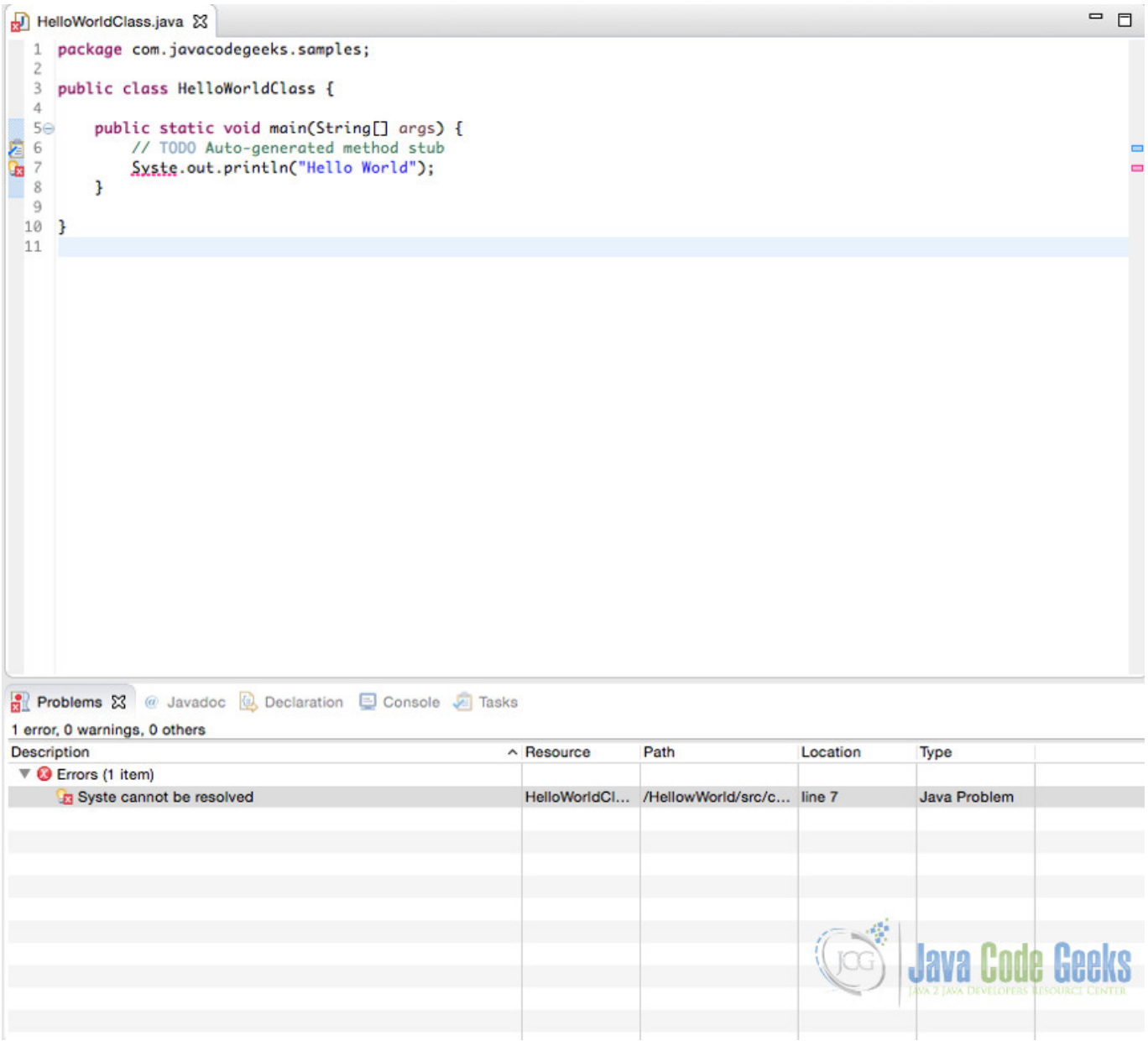

Figure 1.36: Auto-correction

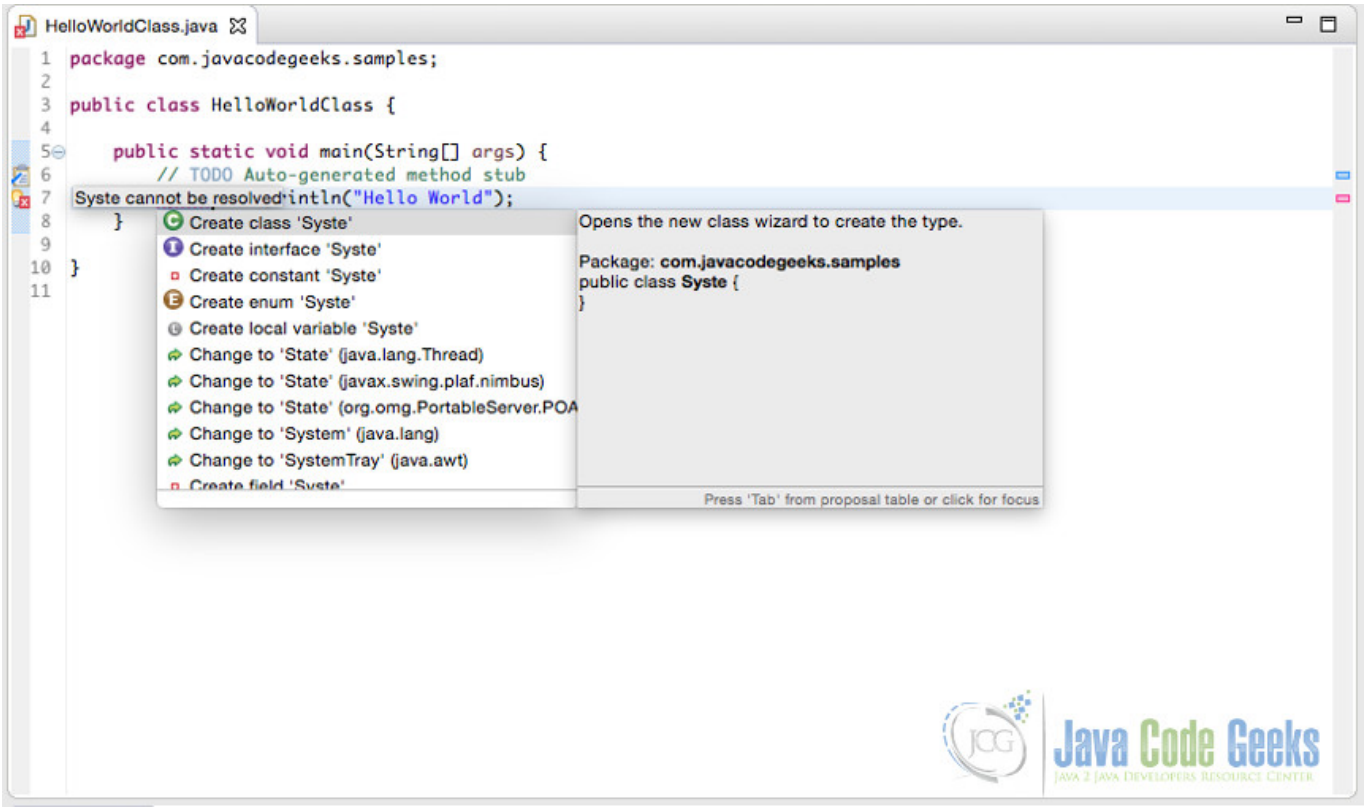

Figure 1.37: Auto-correction

To fix the issue, click “Change to ‘System’ (java.lang)”; Eclipse automatically corrects the code for you.

This auto-correction feature is useful for correcting compilation errors; however, there are times when errors can only be found during the execution of your program. A debugger is a tool used to capture runtime errors that occur during the execution of your program. Luckily, the Eclipse IDE has a built-in debugger that helps you find the root cause of errors (i.e. bugs) in the code. The Eclipse IDE debugger allows you to examine and step through Java code line by line.

Now, let’s look at breakpoint debugging. To fully understand the power of the Eclipse IDE’s debugger, add a few more lines of code to your Hello World sample as shown below. The additional lines will be useful to highlight the flow of control during debugging. Ensure that the newly added code builds and executes correctly before continuing on.

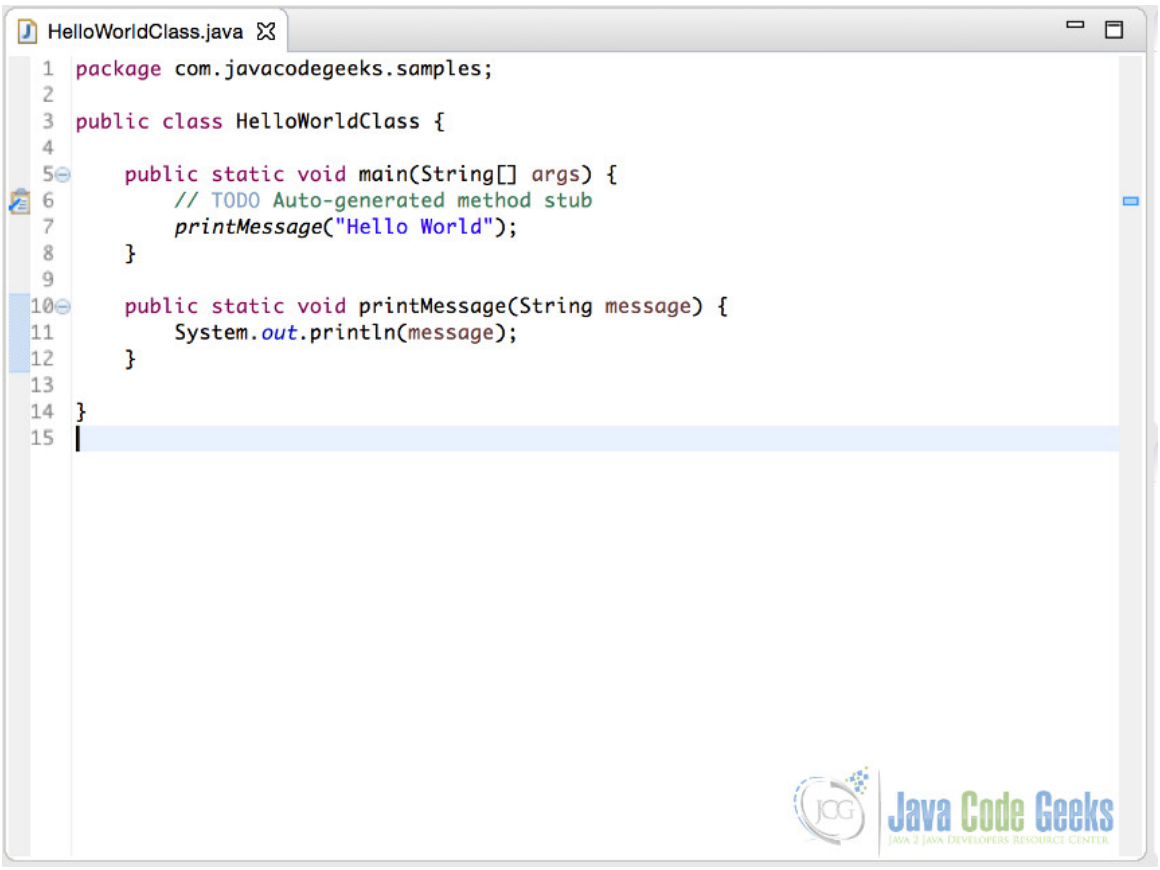

Figure 1.38: Additional Lines

```
package com.javacodegeeks.samples;

public class HelloWorldClass {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        printMessage("Hello World");
    }

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

}
```

The first step to debugging your program is setting a breakpoint, which is a place in the code at which program execution suspends. If you do not set a breakpoint, your program will run to completion without letting you do any debugging. A breakpoint can be set by double-clicking in the gray margin on the left side of the editor, next to the line where the execution should stop. A blue dot appearing in the margin, indicates an active breakpoint.

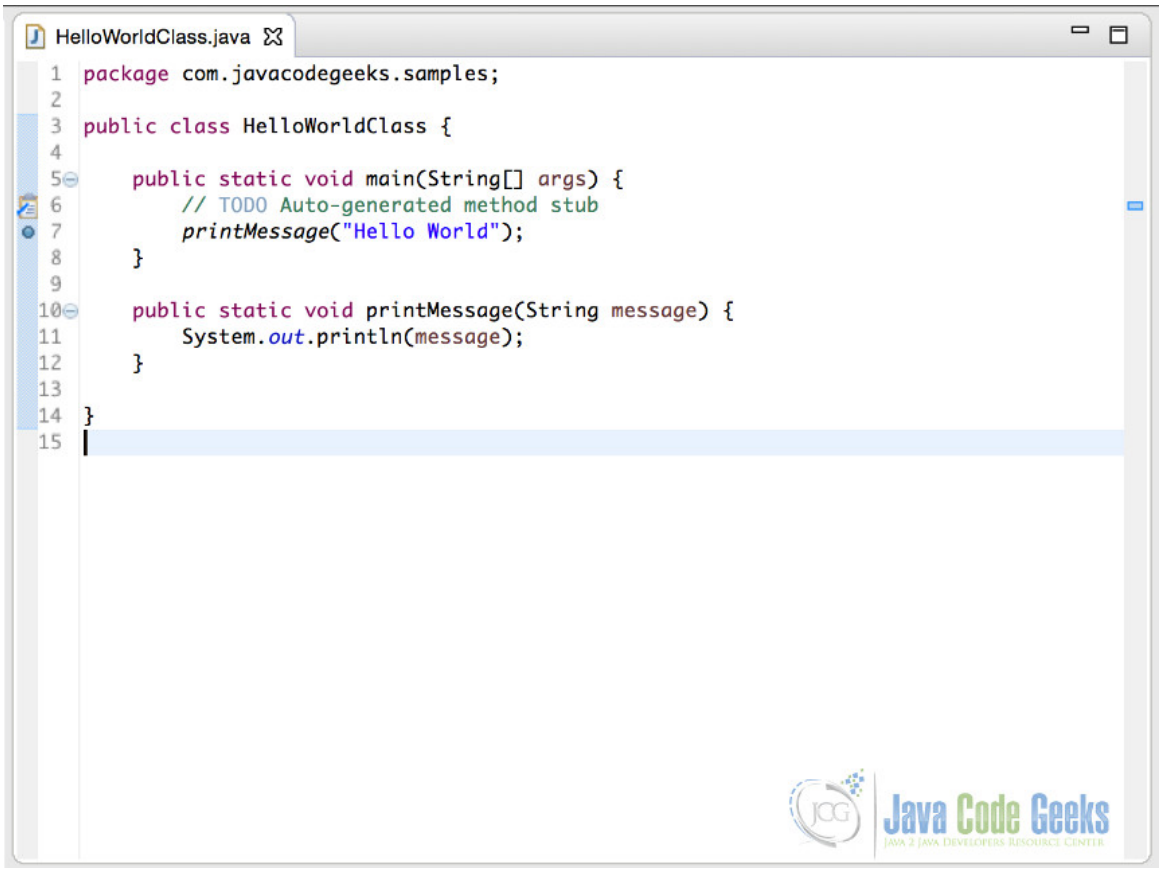

Figure 1.39: Breakpoint

After setting the breakpoint, select the menu option Run→Debug As→Java Application to start the debugger. Upon starting the debugger, Eclipse switches the display to the debugger perspective. The flow of execution pauses at the breakpoint that was set. Before switching to the Debug Perspective, the Confirm Perspective Switch dialog may appear.

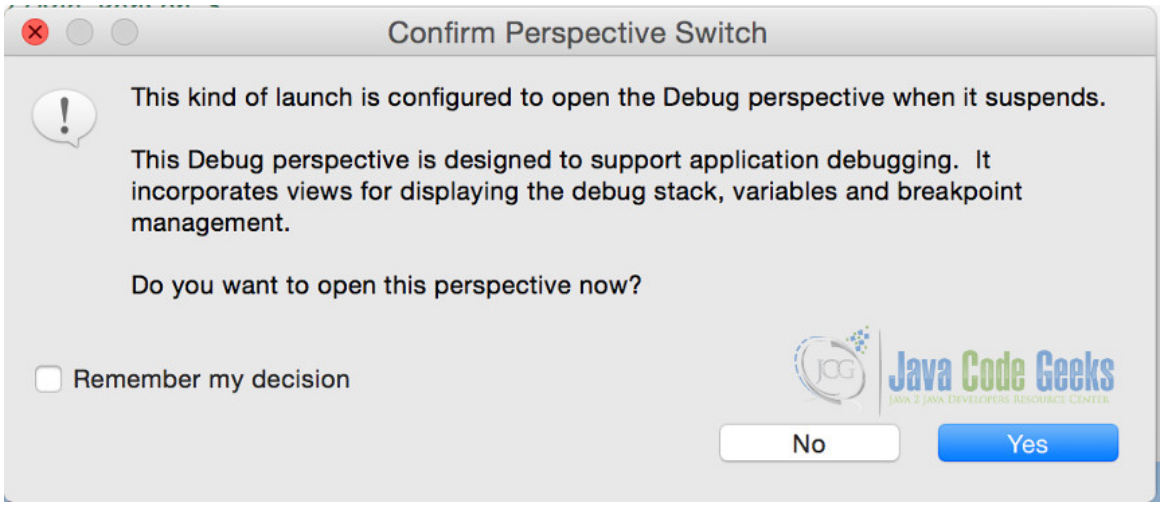

Figure 1.40: Confirm Perspective Switch

If it does appear, click Yes. Notice that in the Debug Perspective, execution suspends at the breakpoint.

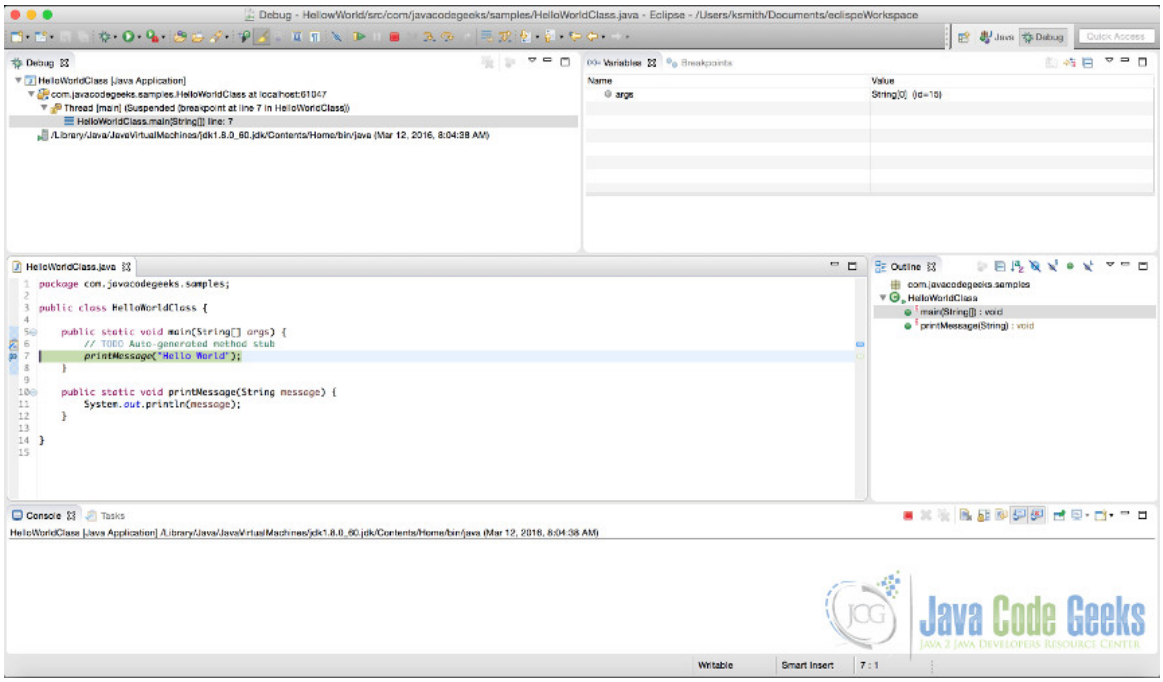

Figure 1.41: Debug View

The perspective includes several new views useful for Debugging. One important view at the top left is the Debug view (not to be confused with the Debug Perspective), which shows the call stack, and the status of all current threads.

Stepping through the code is easy. The debug toolbar allows you to control the flow of the program execution.

Figure 1.42: Debug Toolbar

The most useful menu options are:

- Resume
- Suspend
- Terminate
- Step Into
- Step Over

Resume - Starts debugging after suspending debugger.

Figure 1.43: Resume

Suspend - Pauses the debugger.

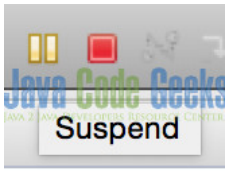

Figure 1.44: Suspend

Terminate - Stops the debugger.

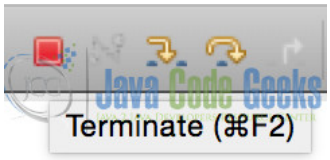

Figure 1.45: Terminate

Step Into - This takes you into a method that is called.

Figure 1.46: Step Into

Step Over - This allows you to call a method without stepping into it line by line.

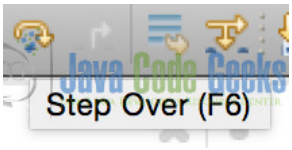

Figure 1.47: Step Over

Now, let's evaluate variables and expressions. Click the "Step-Into" button to bring the flow of control to the "printMessage" method. To see the value of a variable, simply hold the mouse pointer over the variable. The content of the variable is displayed in a small window next to the variable name, as shown below.

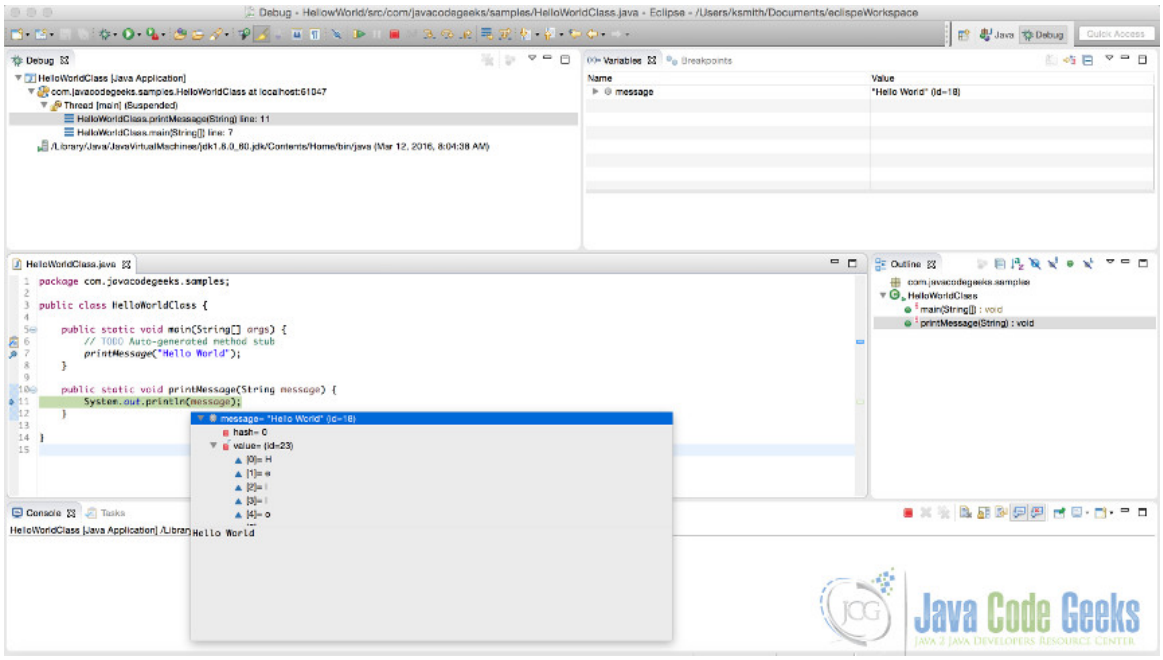

Figure 1.48: Step Into

The other window views are useful to examine the state of the variables in your program. The Variables view shows the state of all the local variables active in the current method. The data can be examined by expanding the variable. This allows us to view recursively down the tree.

Figure 1.49: Other Window Views

This additional information can prove useful when trying to track down errors in your code.

1.8 Useful Features

1.8.1 Code Formatting

Sometimes source code may be hard to read if you've copied the code from another source or opened source code written by someone else. The Eclipse IDE provides an easy way to reformat code with just one-click. For example, if we had code similar to what's shown below and wanted to update the format, we would make use of the source code formatting.

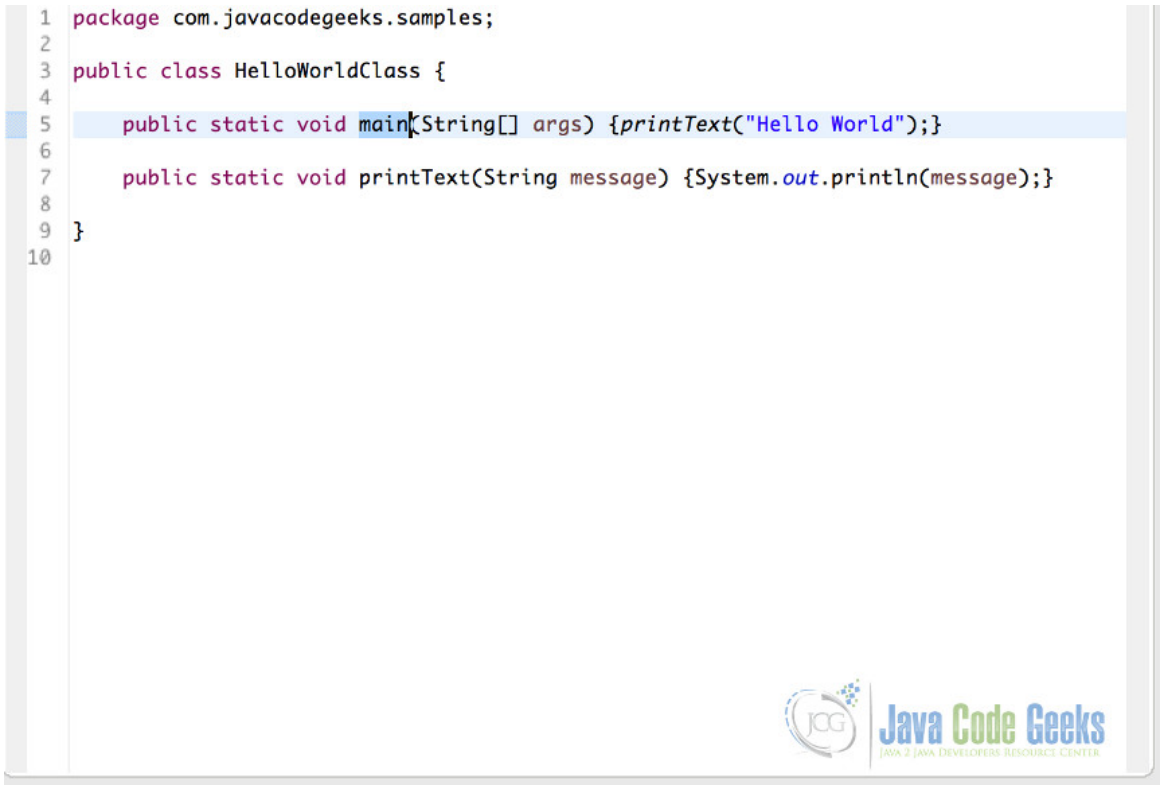

Figure 1.50: Poorly formatted code

```
package com.javacodegeeks.samples;

public class HelloWorldClass {

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

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

}
```

To format code, from the toolbar, select “Source→Format”:

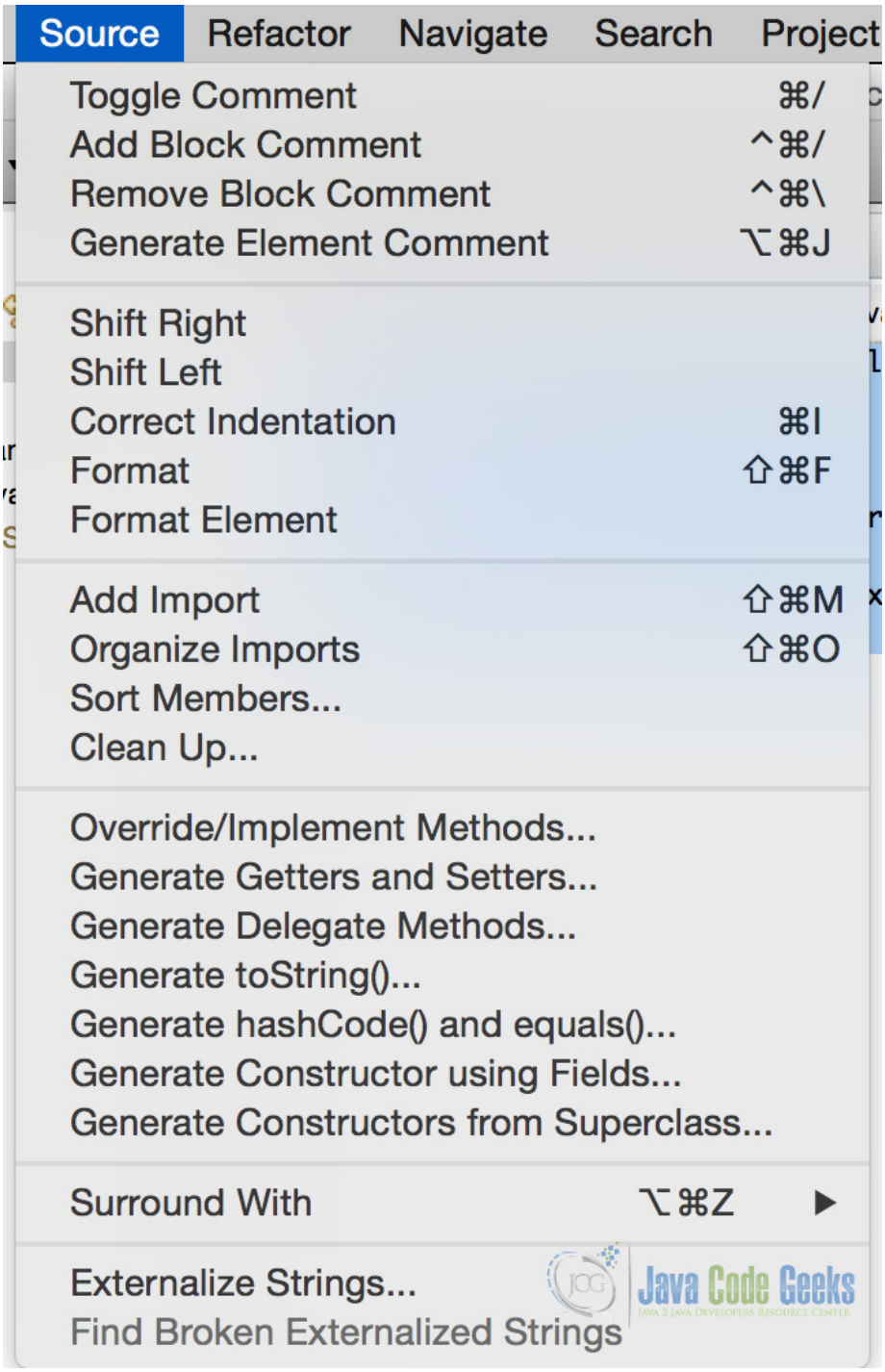

Figure 1.51: Format Option

The code is automatically changed to what is shown below based on the preferences for source code formatting that were set earlier in the tutorial.

Figure 1.52: Formatted Code

1.8.2 Refactoring

After a program is developed, renaming an object, class, or method requires a considerable amount of work. Typically, every place where that object, class, or method is used would have to be changed. Eclipse offers a way to make changes and ripple those changes across the entire application with just one-click using refactoring.

In this example, refactor the code by changing the name of the “printMessage” method to “printText”.

Highlight the name of the method:

Figure 1.53: Selected Method to Refactor

Select Refactor from the menu

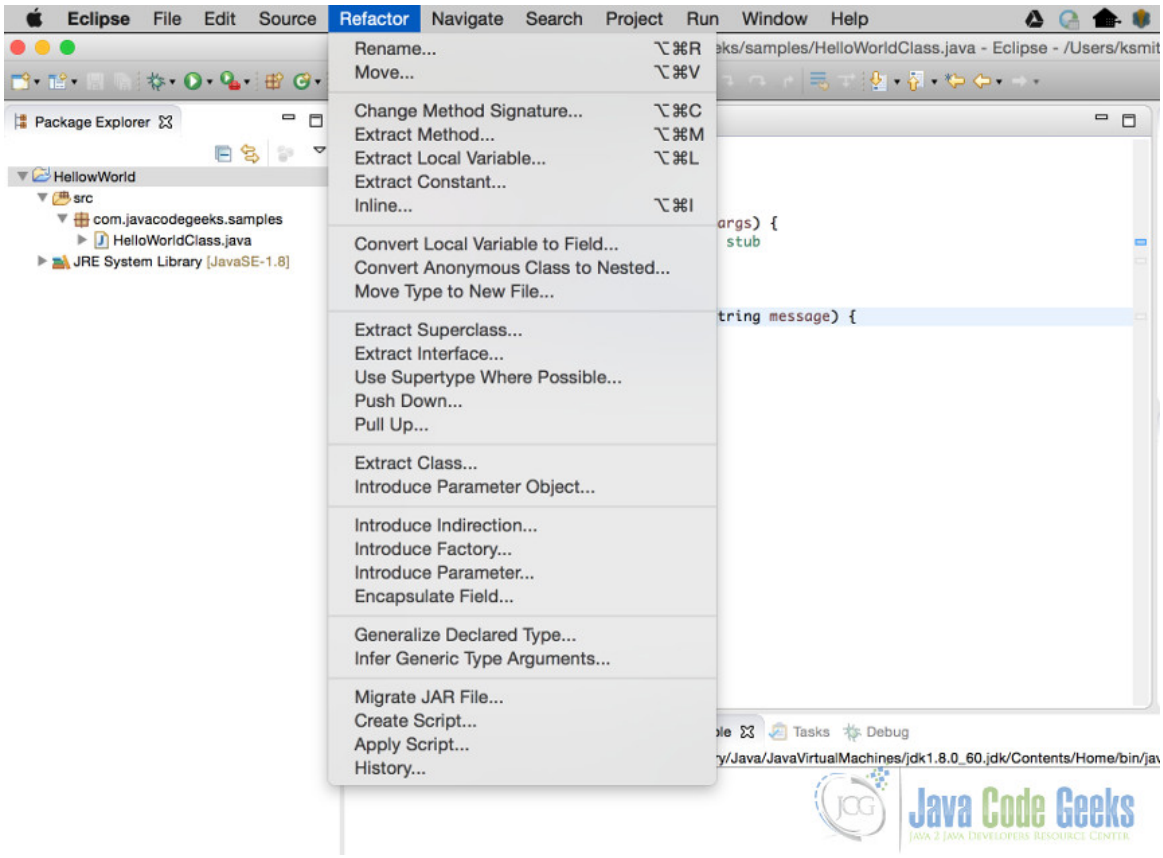

Figure 1.54: Refactor Menu Item

Select Rename, which displays a screen like below:

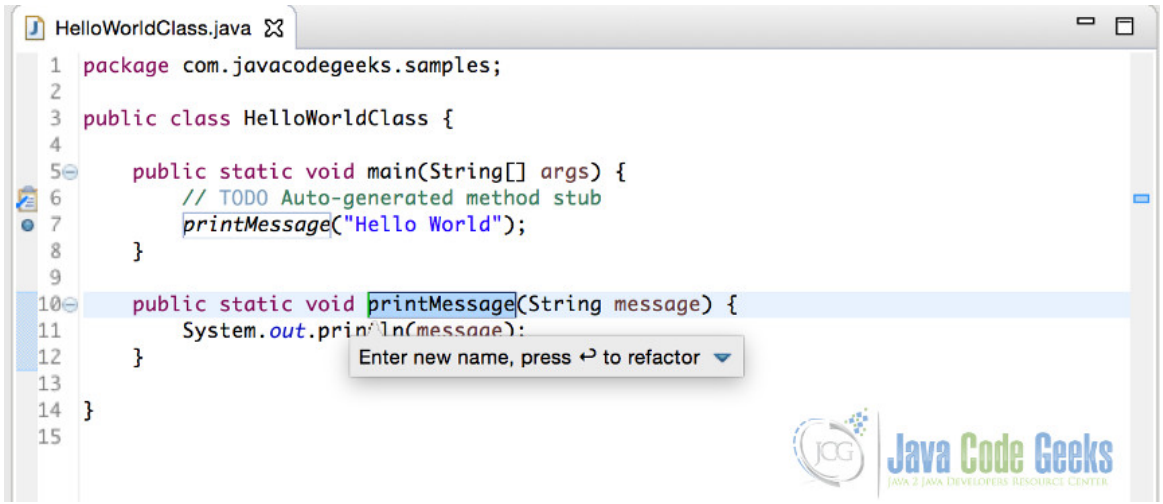

Figure 1.55: Rename

Enter the new name “printText” and press Enter. All occurrences of the method name are updated.

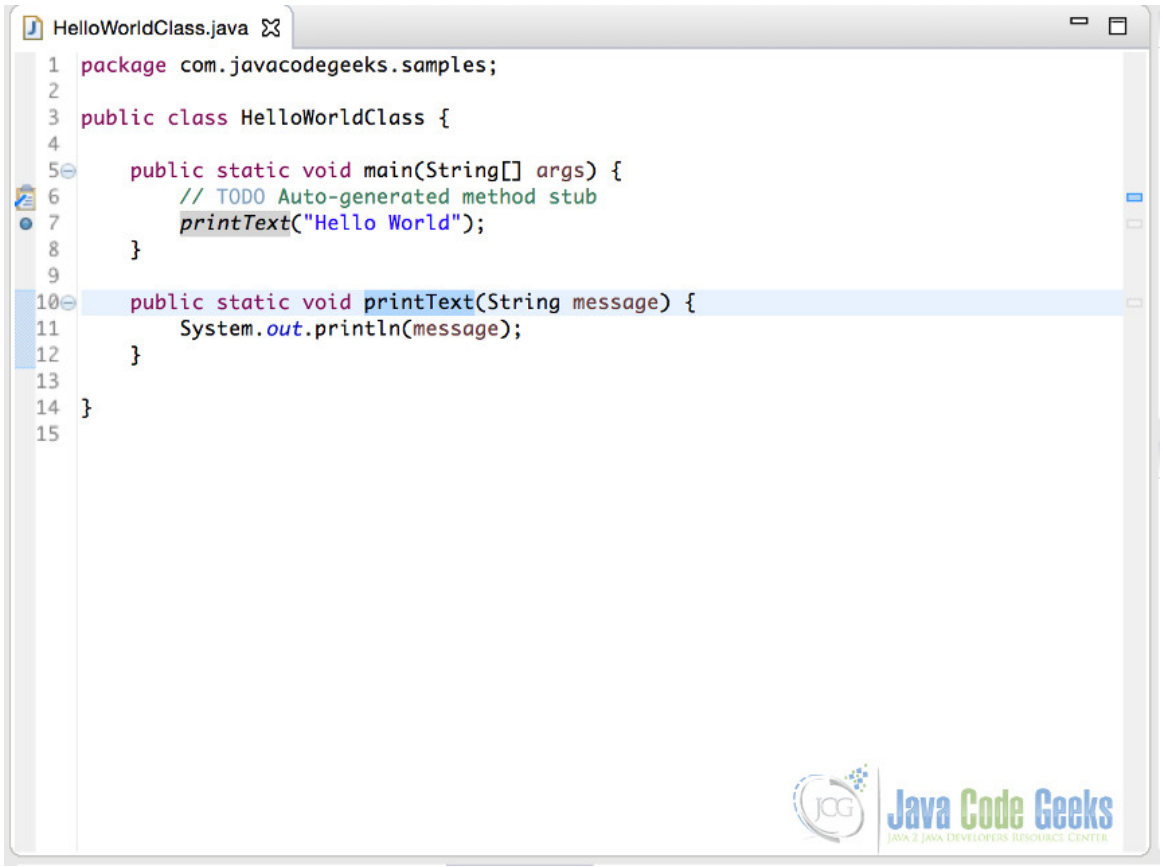

Figure 1.56: Method Rename

1.8.3 Call Hierarchy

When maintaining an application with a lot of classes and dependencies, it is sometimes hard to track other methods that call a particular method. The Call Hierarchy functionality lists all methods that call a given method.

In this example, select the “PrintText” method and right click:

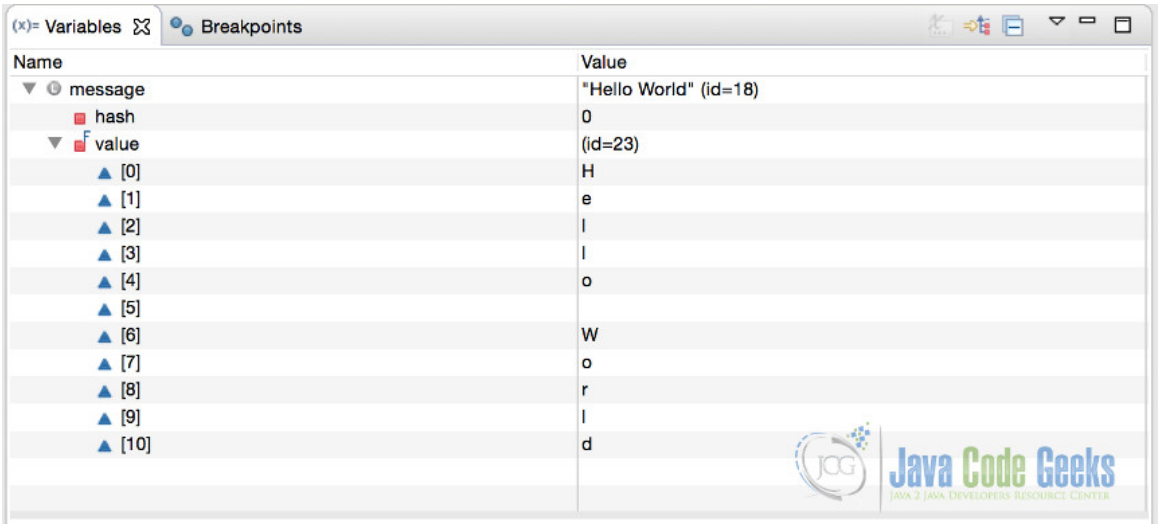

Figure 1.57: Select PrintText method name

Select “Open Call Hierarchy”, which opens the “Call Hierarchy” view in the bottom tabbed pane.

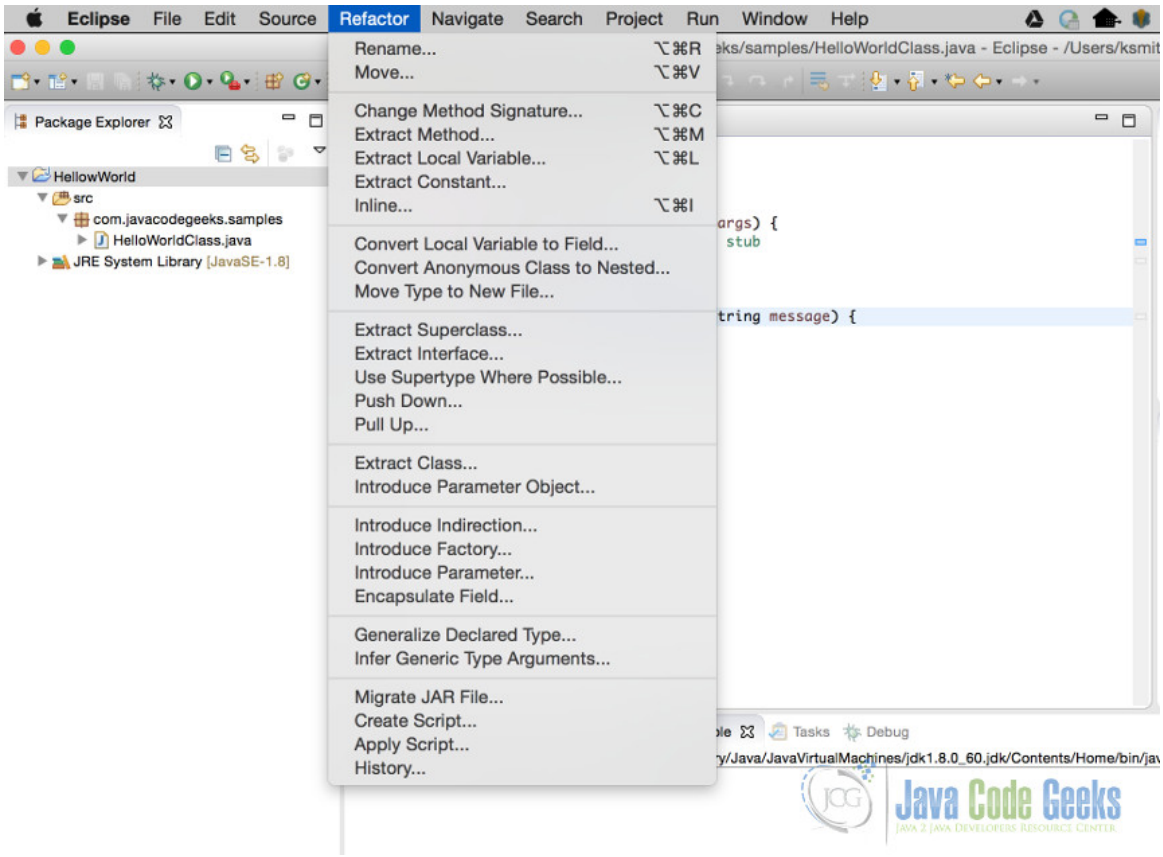

Figure 1.58: Open Call Hierarchy

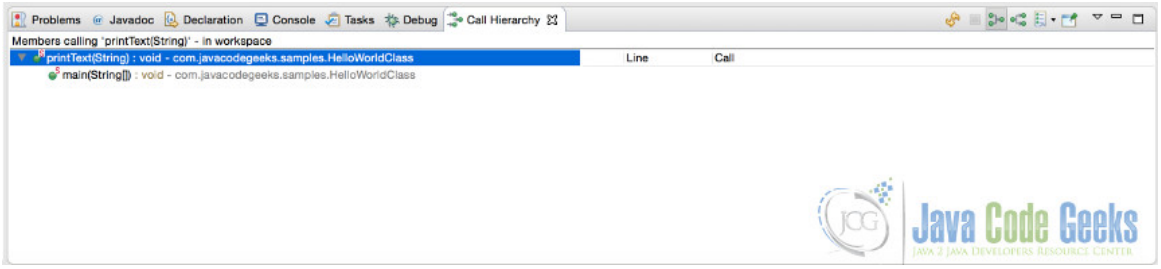

Figure 1.59: Call Hierarchy

1.8.4 Local History for Files

Eclipse IDE keeps a local history of files, which may prove useful when you need to review a previous version of a file that has not been committed to a version control system yet.

The local history for a file can be selected by right clicking on a file name in the Package Explorer and selecting “Compare With→Local History”.

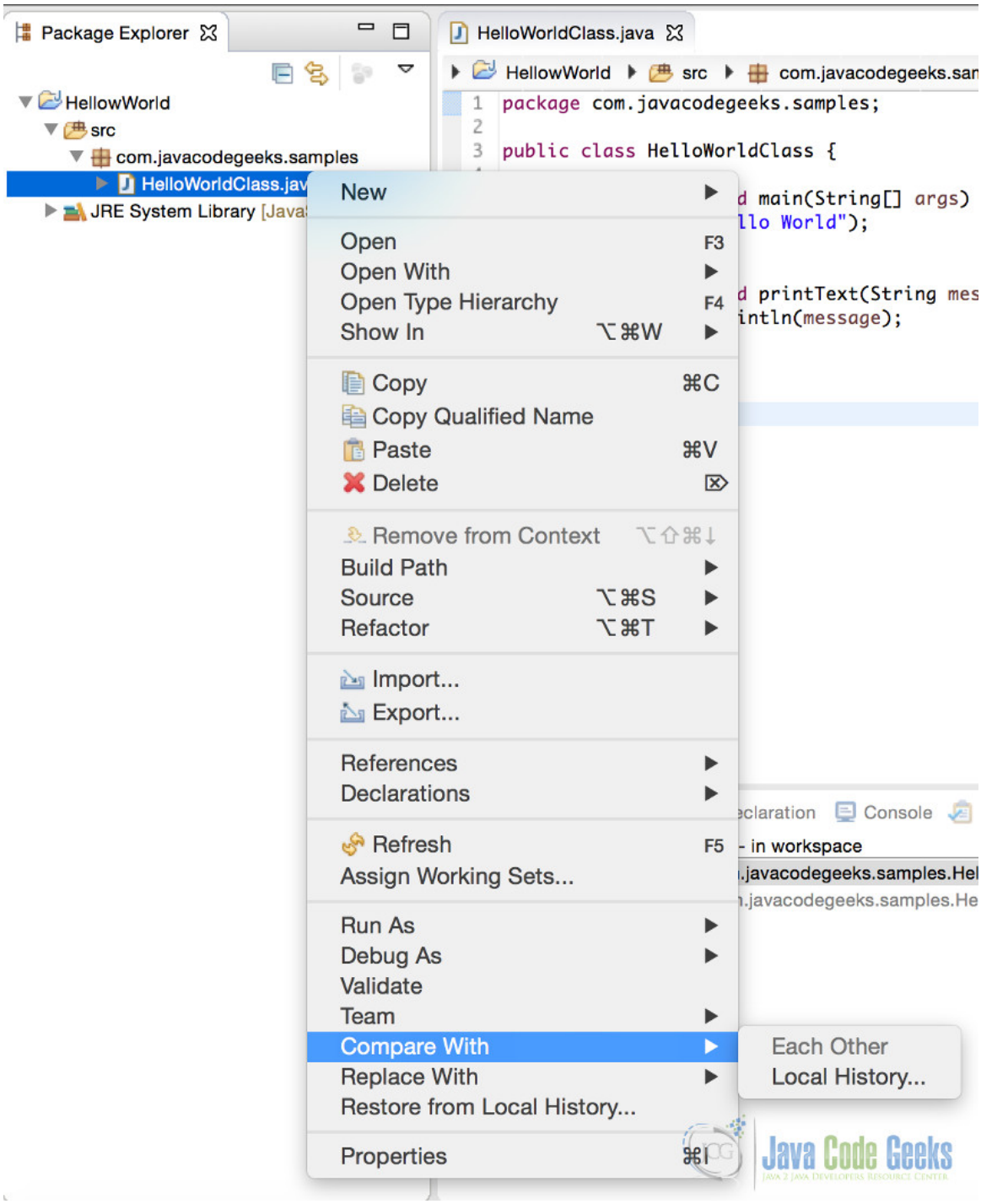

Figure 1.60: Compare With Local History

The “History” view displays in the bottom tabbed pane of Eclipse.

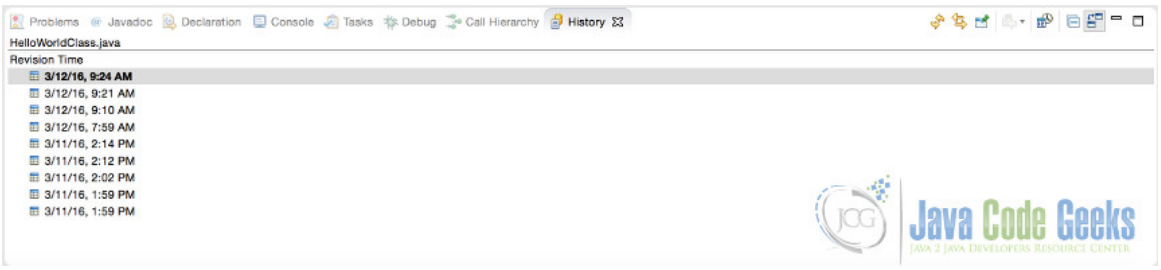

Figure 1.61: Local History

A previous version may be selected by double-clicking on it. In this example, I will double-click on “3/12/16, 7:59 AM”, which displays the following screen.

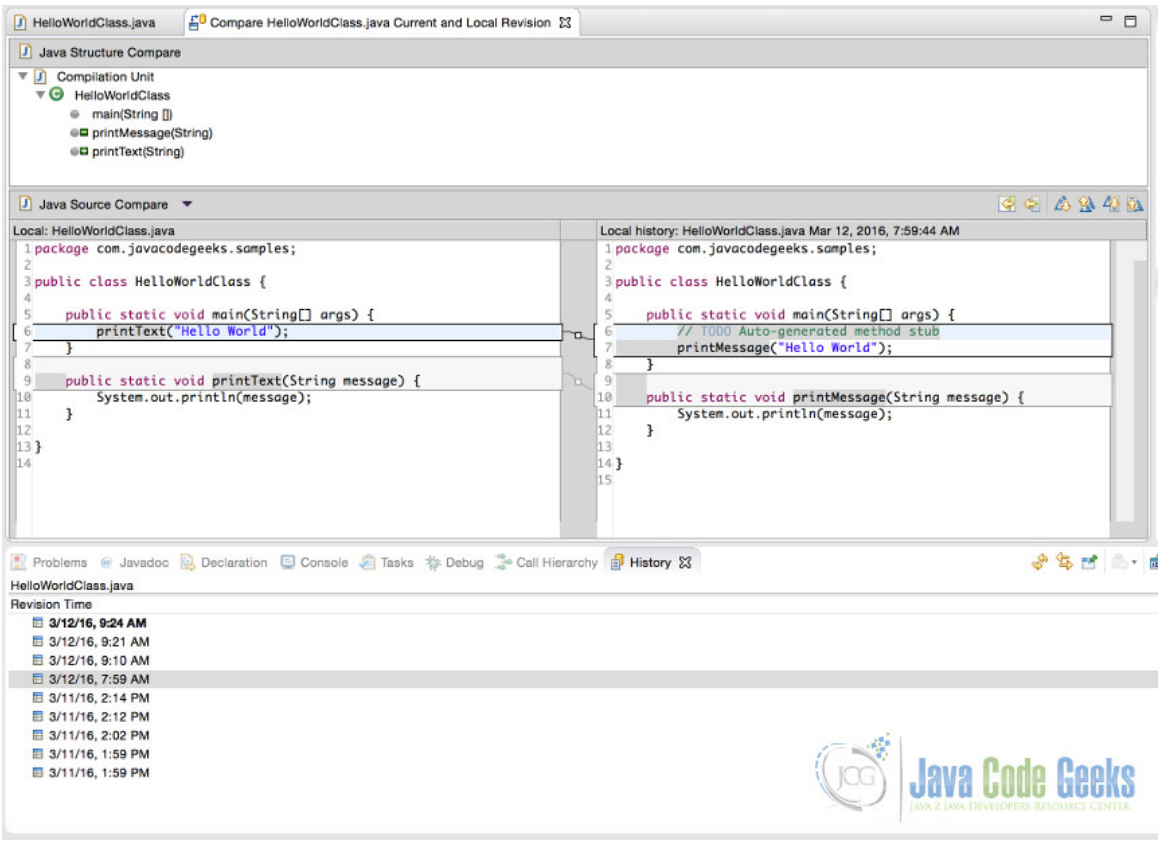

Figure 1.62: Previous Version Comparison

The current version is displayed on the left-hand side; the previous version is display on the right-hand side. Eclipse also provides the ability to replace the current version with a previous version by right clicking on a file name in the Package Explorer and selecting “Replace With→Local History” or “Restore From Local History”.

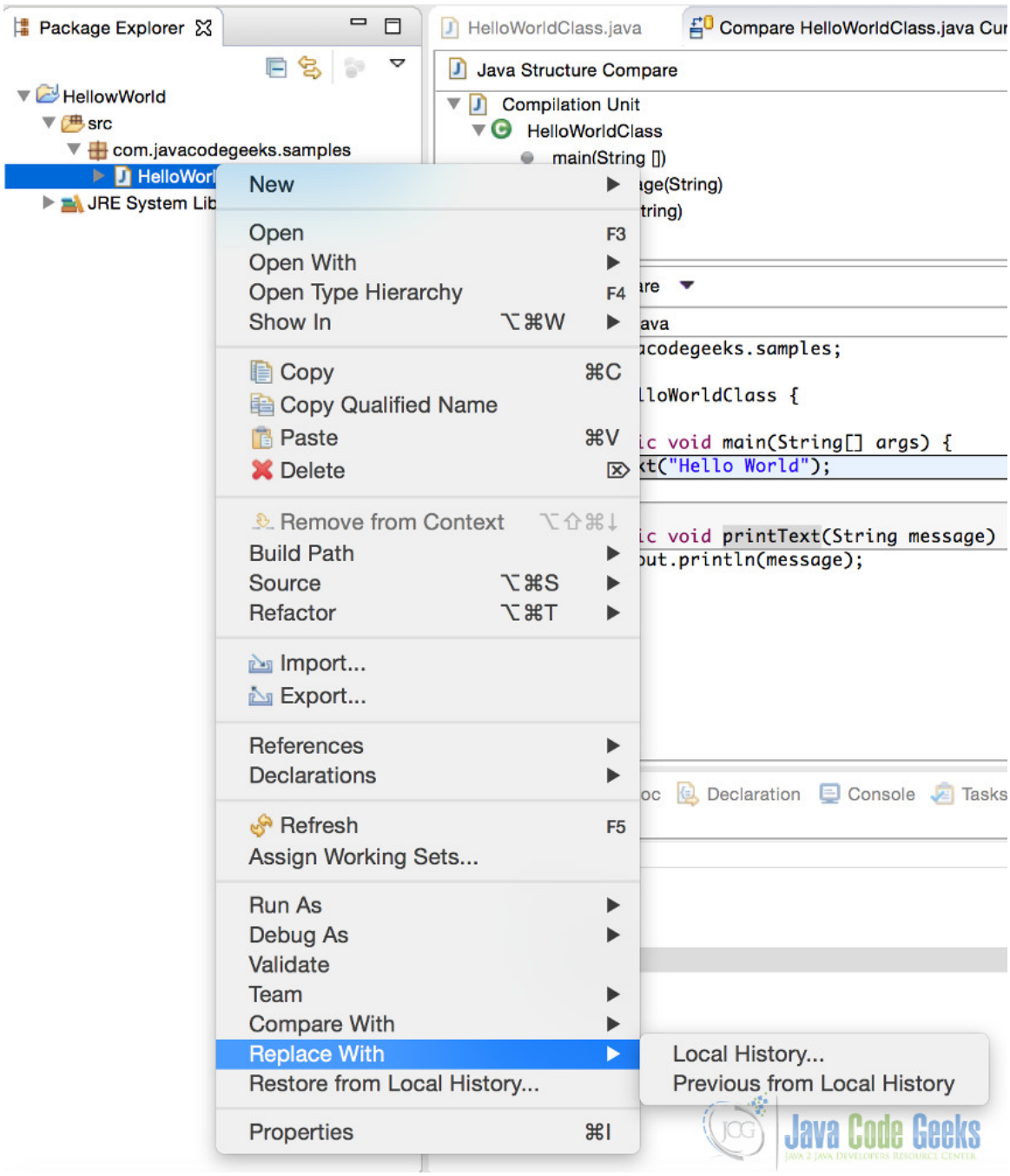

Figure 1.63: Restore From Local History

1.9 Download the Source Code

This was a tutorial on the Eclipse IDE for Java Developers.

Download You can download the full source code of this example here: **HelloWorld Project**

Chapter 2

Eclipse Environment Variable Setup Example

In this article we will see how to set the environment variables in Eclipse. For this example we will make use of Eclipse Luna 4.4.2.

2.1 Introduction

Eclipse is a Java-based open source platform that allows a software developer to create a customized development environment from plug-in components built by Eclipse members. Eclipse is managed and directed by the Eclipse.org Consortium. Eclipse is the most common Integrated Development Environment (IDE) used by Java developers. The best thing that eclipse is cabable of is that it uses plugins for adding more features. So if you don't need a feature you can simple ignore the required plugin.

Many operating systems use environment variables to pass configuration information to applications. Like properties in the Java platform, environment variables are key/value pairs, where both the key and the value are strings. The conventions for setting and using environment variables vary between operating systems, and also between command line interpreters

2.2 Create Simple Project

In this section we will see how to create a simple project. Then we will set the environment variable for this newly created project.

Click on File⇒New⇒Java Project.

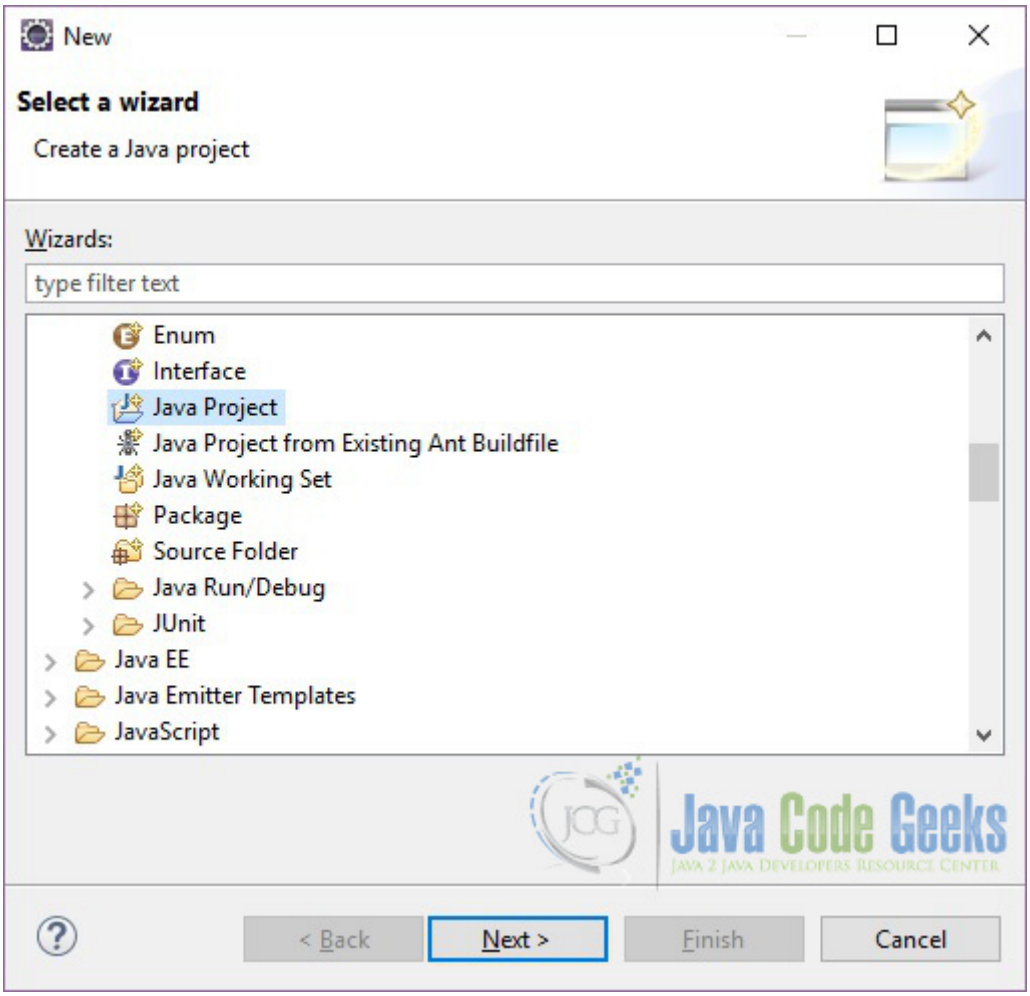

Figure 2.1: Java Project

Please note that you might need to click 'Other... ' link to go to the Java Project section. In the Project name text box give the name of the project. We will use EnvironmentVariableSetup. Leave the other dafault values as it is. Click the Finish button.

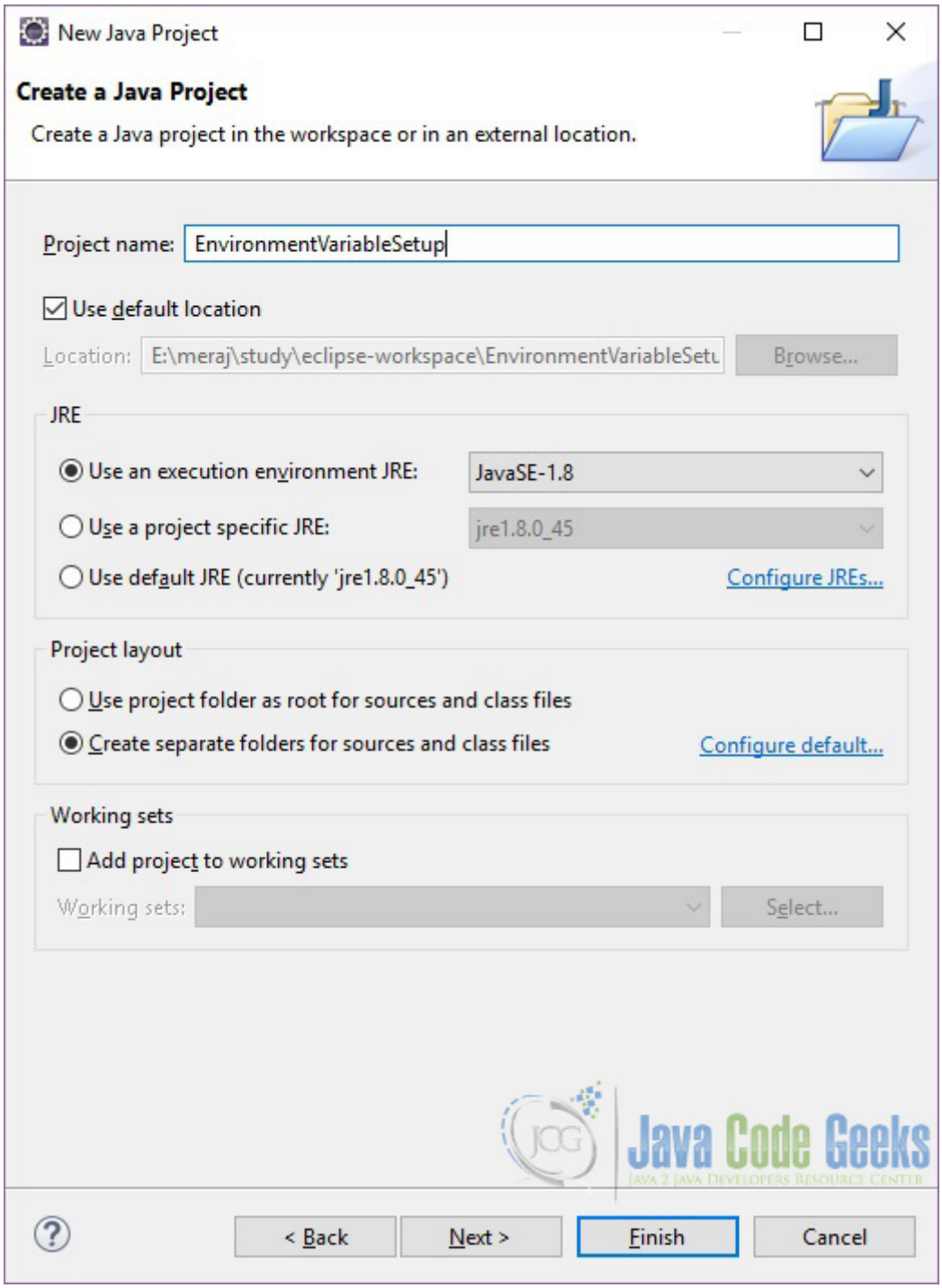

Figure 2.2: Project Details

Eclipse will create a default src folder and will include some jar files in the classpath (shown as below). Please note that the list of jar files included can be different depending on the java and Eclipse version.

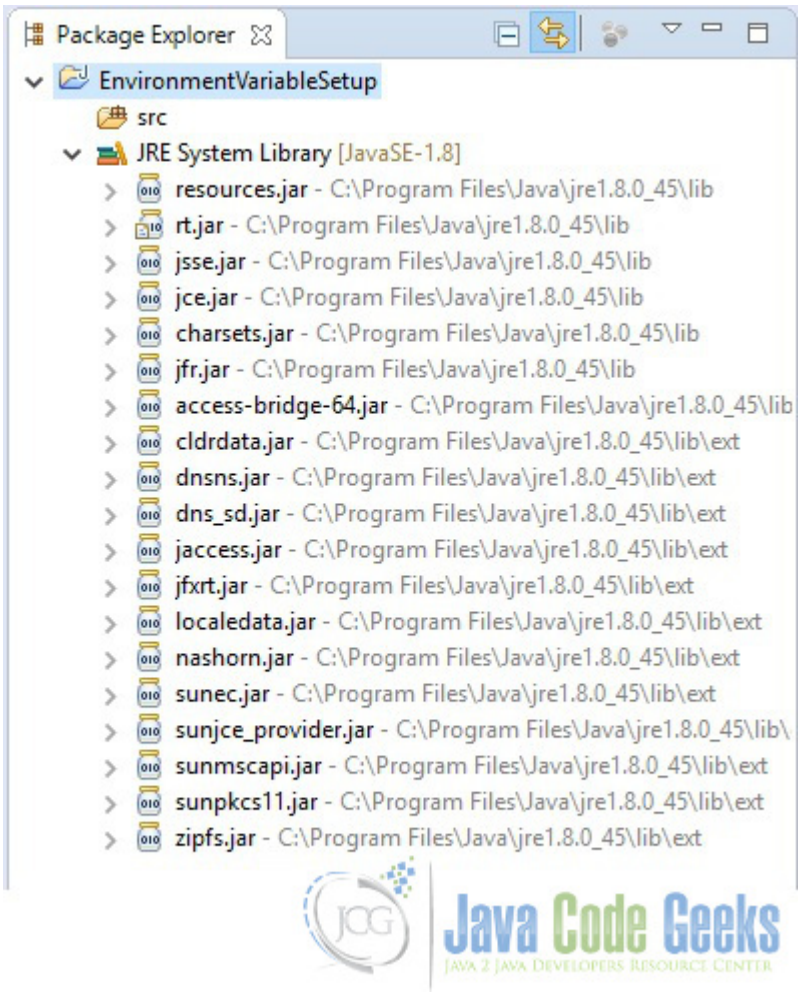

Figure 2.3: Jar files

2.2.1 Create New Package and Class

In this section we will see how to create a new java package in Eclipse.

Right click on the src folder and go to New⇒Package. In the Java Package pop-up give the package name. We will use com.javacodegeeks. Click Finish.

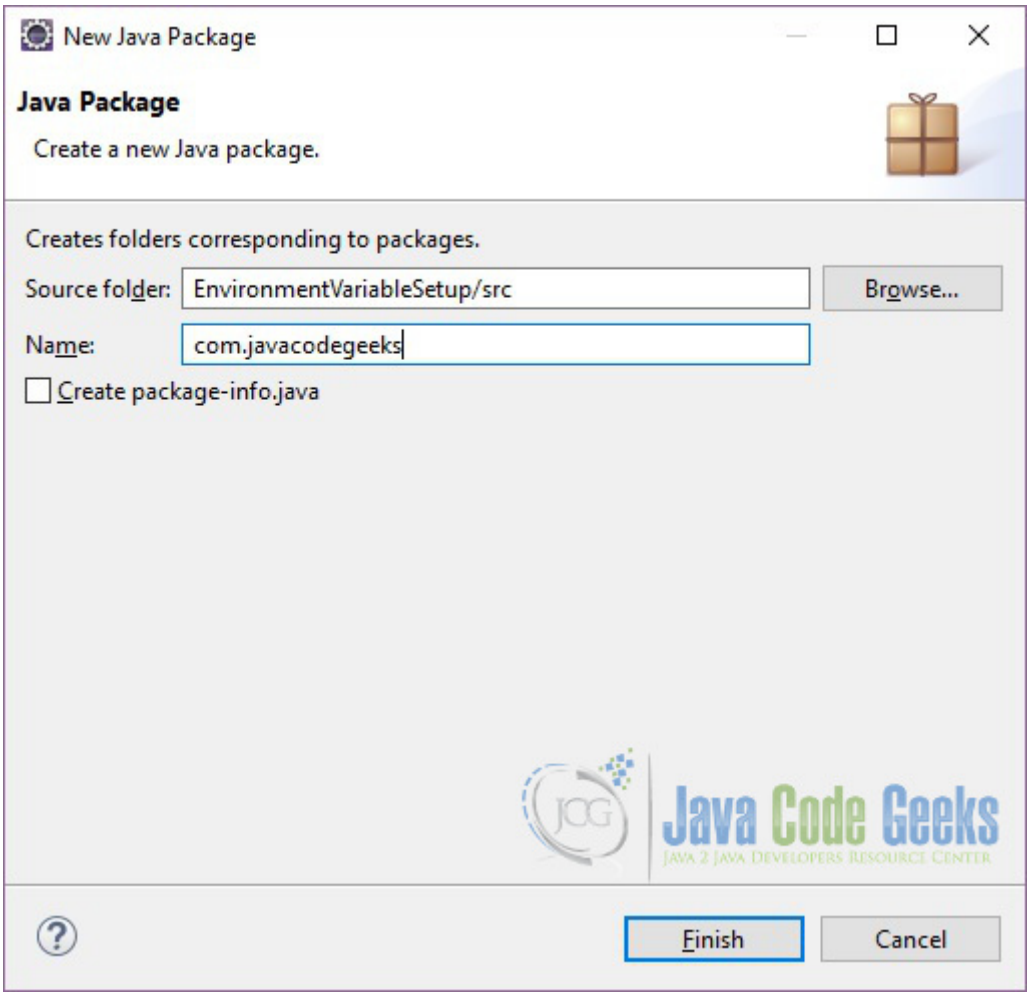

Figure 2.4: New Package

Now to create a new class in this package right click on the package and choose New⇒Class. Enter the Class name. You can choose some other options as well to configure the newly created class.

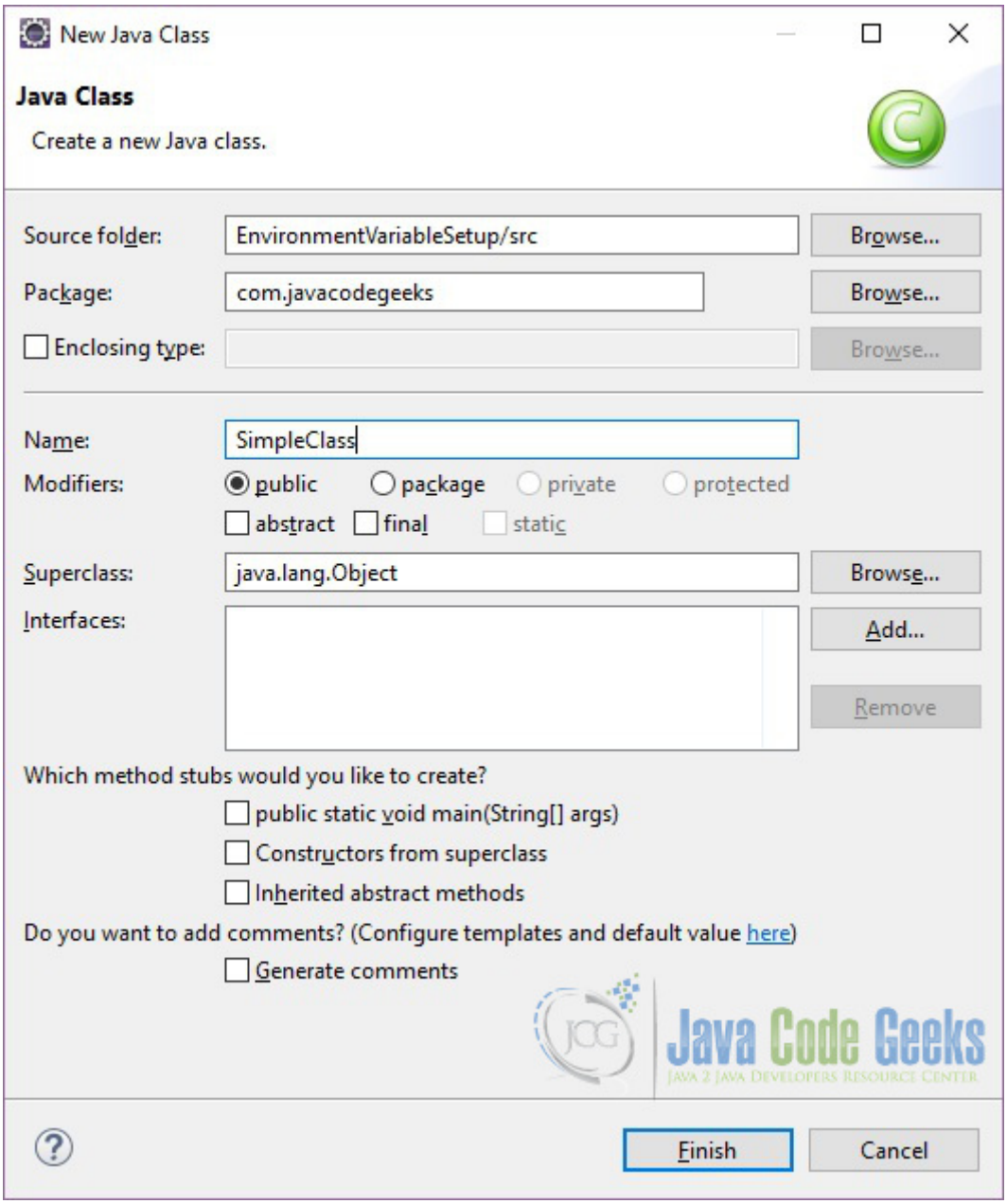

Figure 2.5: New Class

2.3 Set Environment variable

In this section we will see how to set an environment variable in eclipse. Right click on the class (SimpleClass), go to Run As⇒Run Configurations... Click on the Environment Tab.

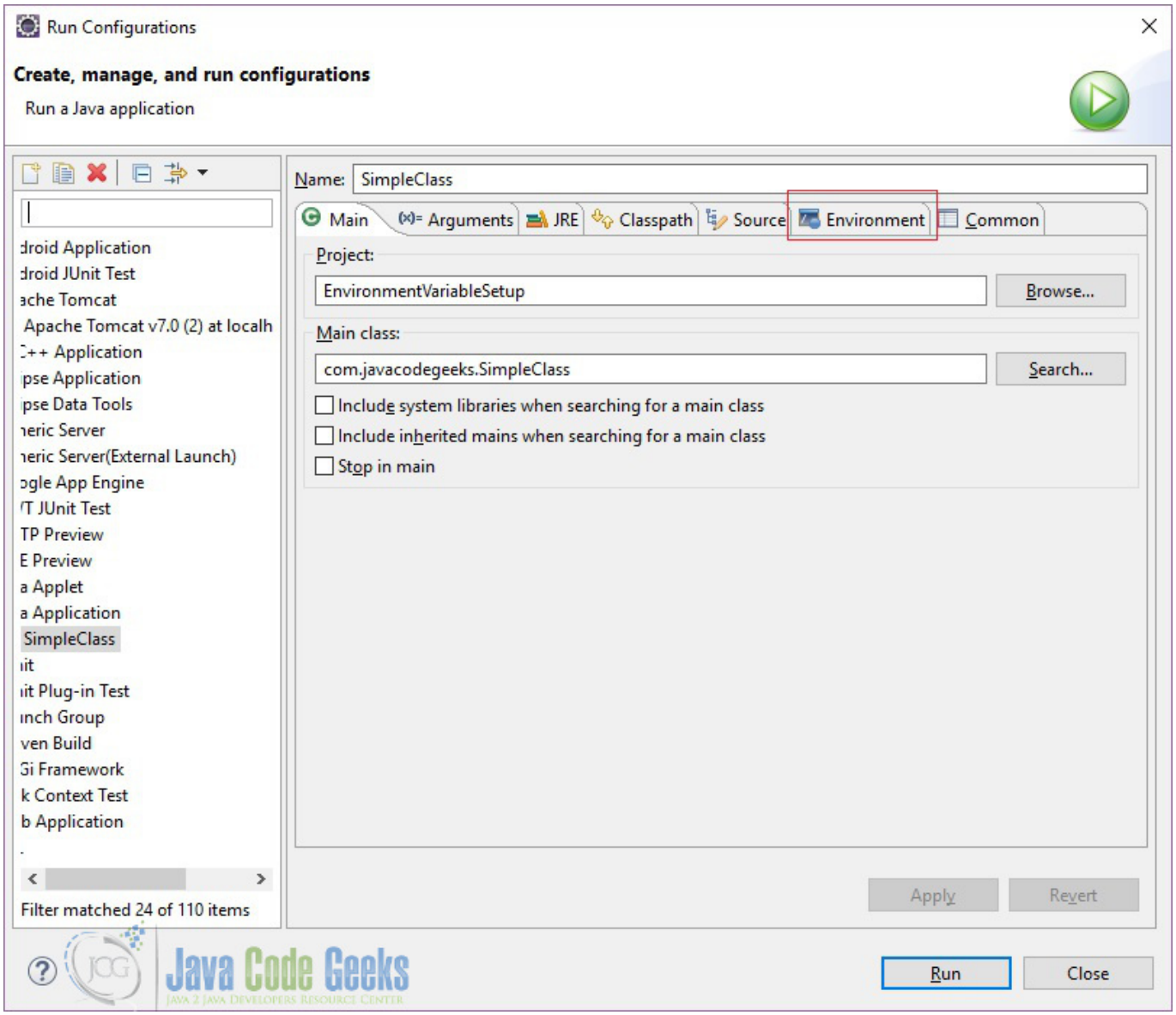

Figure 2.6: Run Configurations

On the Environment Tab click on the New… button. In the Name textbox enter the name of the environment variable - *TEST_ME*. In the Value textbox enter *Tested*. Click OK.

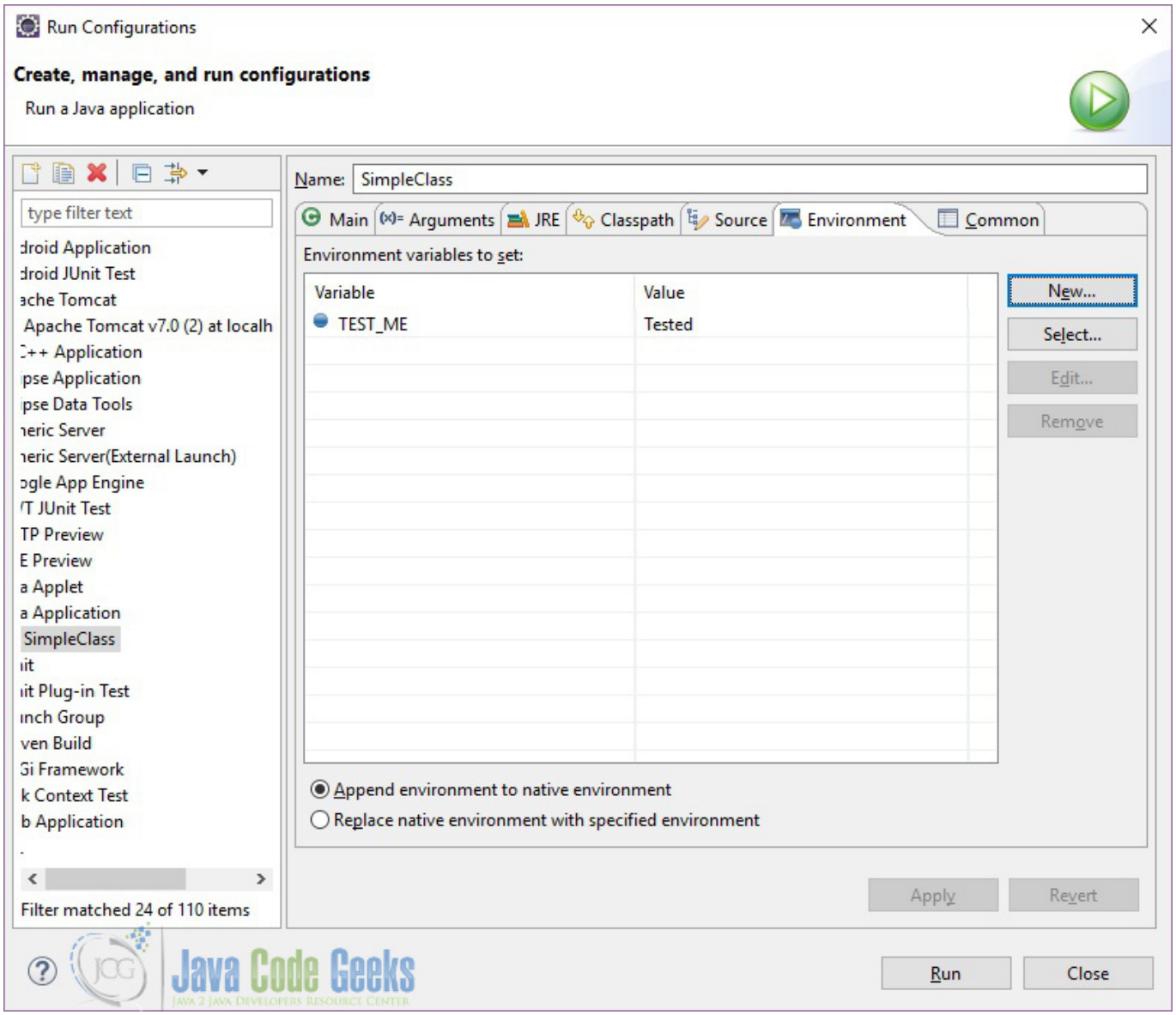

Figure 2.7: Environment Setup

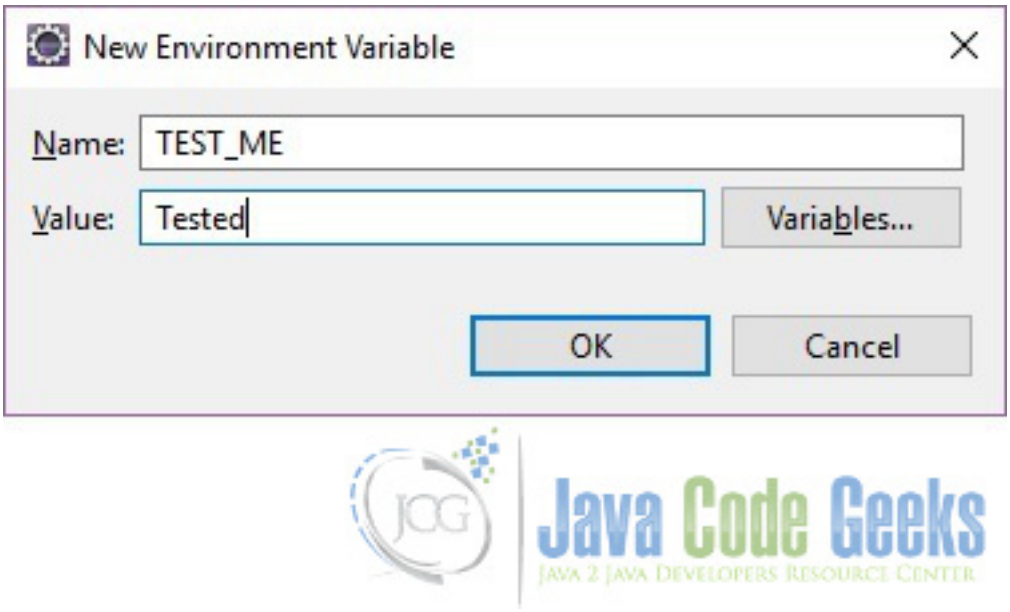

Figure 2.8: New Environment Variable

In the window above you can see all the environment variables that have been defined. You can also Edit or Remove these variable from this window.

2.4 Use environment Variable

In this section we will see how we can use the newly defined environment variable.

On the Java platform, an application uses `System.getenv` to retrieve environment variable values. Without an argument, `getenv` returns a read-only instance of `java.util.Map`, where the map keys are the environment variable names, and the map values are the environment variable values. With a String argument, `getenv` returns the value of the specified variable. If the variable is not defined, `getenv` returns `null`

In the class previously defined create a main method. In this main method we will use the `java.lang.System` class to get the value of environment variable.

```
String envValue = System.getenv("TEST_ME");
```

Now we will print the value returned. Below is the full code for this class

SimpleClass.java

```
package com.javacodegeeks;

public class SimpleClass {

  public static void main(String... args) {
    String envValue = System.getenv("TEST_ME");
    System.out.print(envValue);
  }
}
```

In the console you will see that the value of the variable will get printed (Tested).

2.5 Conclusion

In this article we saw how to setup a simple java project and how to create a new package and class. We then saw how we can setup environment variable using the Run Configuration setting of Eclipse. Wehave also examined how to use this newly created environment variable.

Chapter 3

Eclipse Web Development Tutorial

The Web development environment provides the tools we need to develop Web applications as defined in the Sun Microsystems Java Servlet 2.3 Specification and the Sun Microsystems JSP 1.2 Specification. Web applications can be simple (consisting of only static Web pages) or they can be more advanced and include JavaServer Pages (JSP) files and Java servlets.

These resources, along with an XML deployment descriptor (and other Web resources, are contained within a Web project during development. We deploy the Web project to the server in the form of a Web archive (WAR) file once it's ready. The end user can then view the Web application as a Web site from a URL.

The integrated Web development environment makes it easy to cooperatively create, assemble, publish, deploy and maintain dynamic, interactive Web applications.

In this tutorial we will see how we can develop a web application using eclipse.

3.1 Web resources

In most cases, all of the resources that we need to create for our Web application are developed during Web site or Web page design; However, there are additional resources that we may need to include in our Web project if we are using more advanced Web technologies in your application. These Web resources are not typical Web page files, and are often not the resources that we consider part of the final Web site. For example, tag libraries and Java resources, such as JAR files, are resources we might need to include in our Web project.

In fact, even the WAR file itself could be considered a Web resource, if we consider importing or exporting the resource.

3.2 Web page design

Web pages are an integral part of every Web application. Each Web page should serve to help achieve the overall goal of the entire Web site. There are many types of Web pages, ranging from simple HTML pages that contain no dynamic elements, to advanced Java-based pages that make use of servlets, scripts, forms, or data access components. A few of the many items you should consider when designing your pages are markup language, links, images, and style sheets.

3.3 Web projects

Web projects hold all of the Web resources that are created and used when developing your Web application. The first step to creating or importing a Web application is to create either a static or a dynamic Web project. **Static Web projects** are meant to contain only simple Web site resources, such as HTML files. **Dynamic Web projects** are used to structure Web applications that will use more complicated, dynamic Web technologies, such as JavaServer Pages files, and possibly data access resources.

Though the Web project is structured on your file system in compliance with the Java EE Web application standard for deployment purposes, the Project Explorer view is designed to show the most convenient display of project resources for use, while you are actually developing the Web application. When you are finished developing your Web application, you use the Web project to deploy the correct resources to the server. These resources will be packaged in a file called a Web archive, or WAR file.

3.4 Web archive (WAR)

A Web application is a group of HTML pages, JSP pages, servlets, resources and source file, which can be managed as a single unit. A Web archive (WAR) file is a packaged Web application. WAR files can be used to import a Web application into a Web server. In addition to project resources, the WAR file includes a Web deployment descriptor file. The Web deployment descriptor is an XML file that contains deployment information, MIME types, session configuration details, and other settings for a Web application. The Web deployment descriptor file (web.xml) provides information about the WAR file and is shared with the developers, assemblers, and deployers in a Java EE environment.

3.5 Creating a dynamic Web project

You create and maintain the resources for your Web applications in Web projects. Unlike with static Web projects, dynamic Web projects enable you to create resources such as JavaServer Pages and servlets. To create a new dynamic Web project, complete the following steps:

1. Open the Java EE perspective
2. Go to File ⇒ New ⇒ Dynamic Web Project

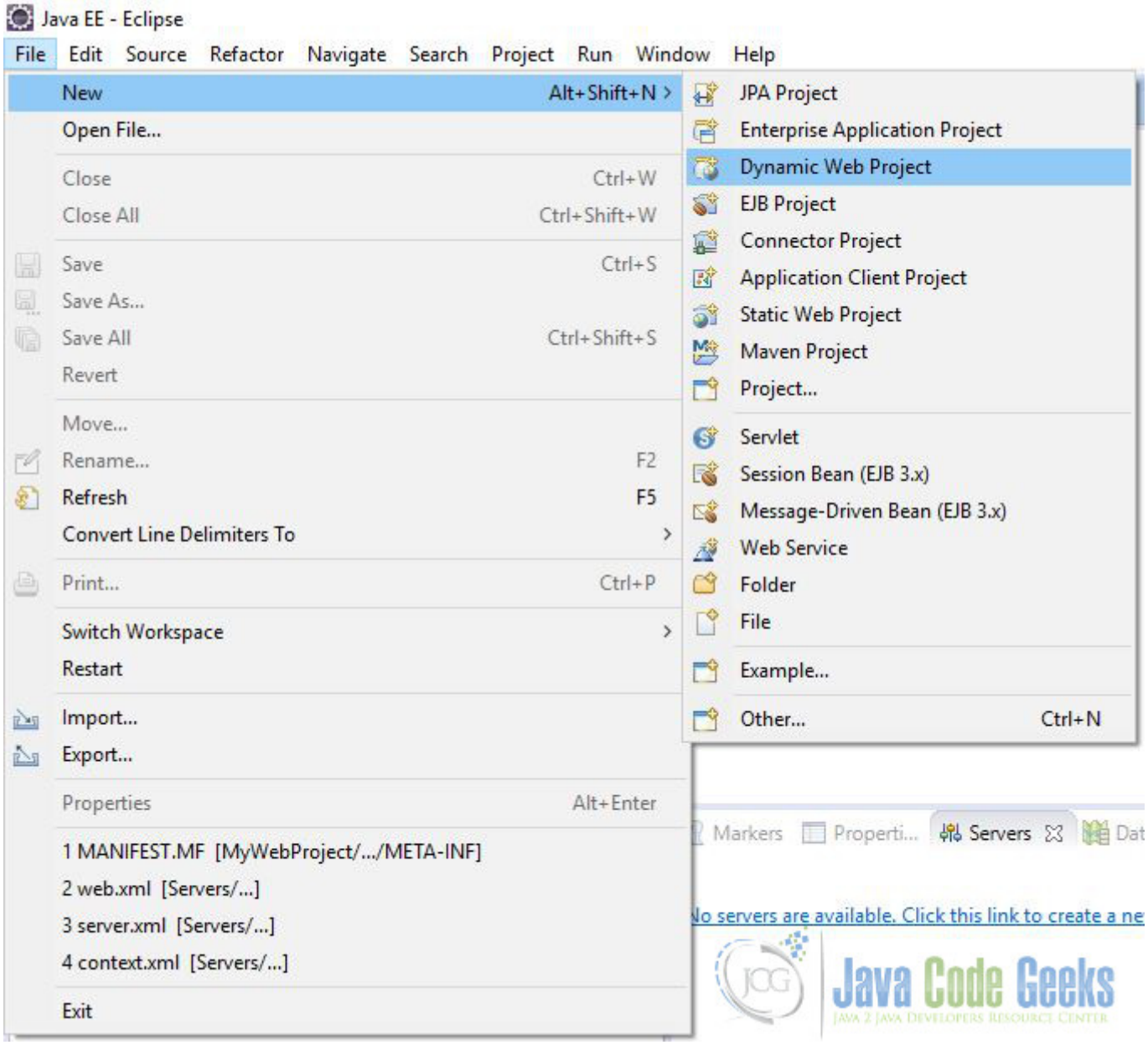

Figure 3.1: Dynamic Web Project

If you don't see the *Dynamic Web Project* option choose Other and in the Wizards text box start writing *Dynamic Web*.

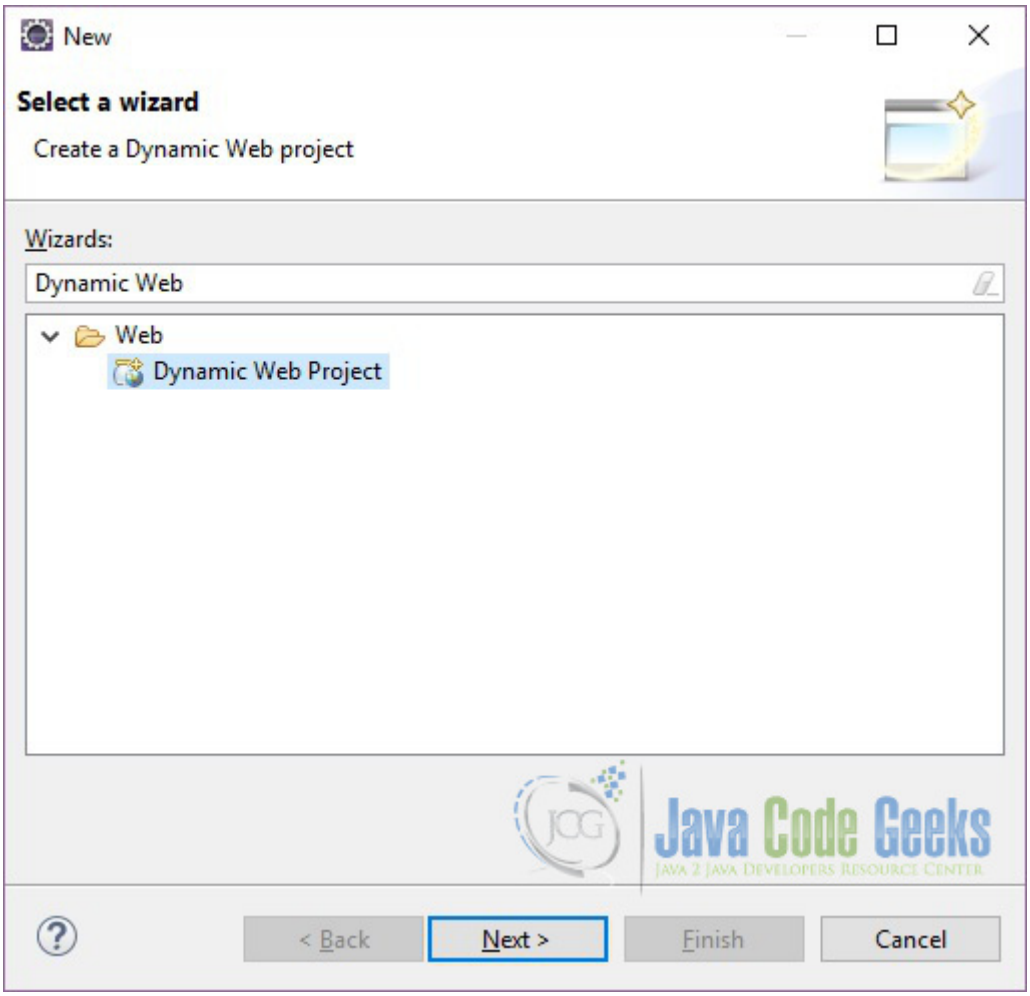

Figure 3.2: Other Dynamic Web Project

1. In the pop-up enter the Project Name. For our example we will choose MyFirstDynamicProject. Choose the project location and the Target runtime. Use this field to define a new installed runtime environment. Runtimes are used at build time to compile projects. For our example we will use Tomcat 7.0. If you haven't downloaded the Apache Tomcat you can do this from Tomcat 7.0. For the Dynamic web module version we will use 2.5. Leave the other fields as it is and click *Finish*

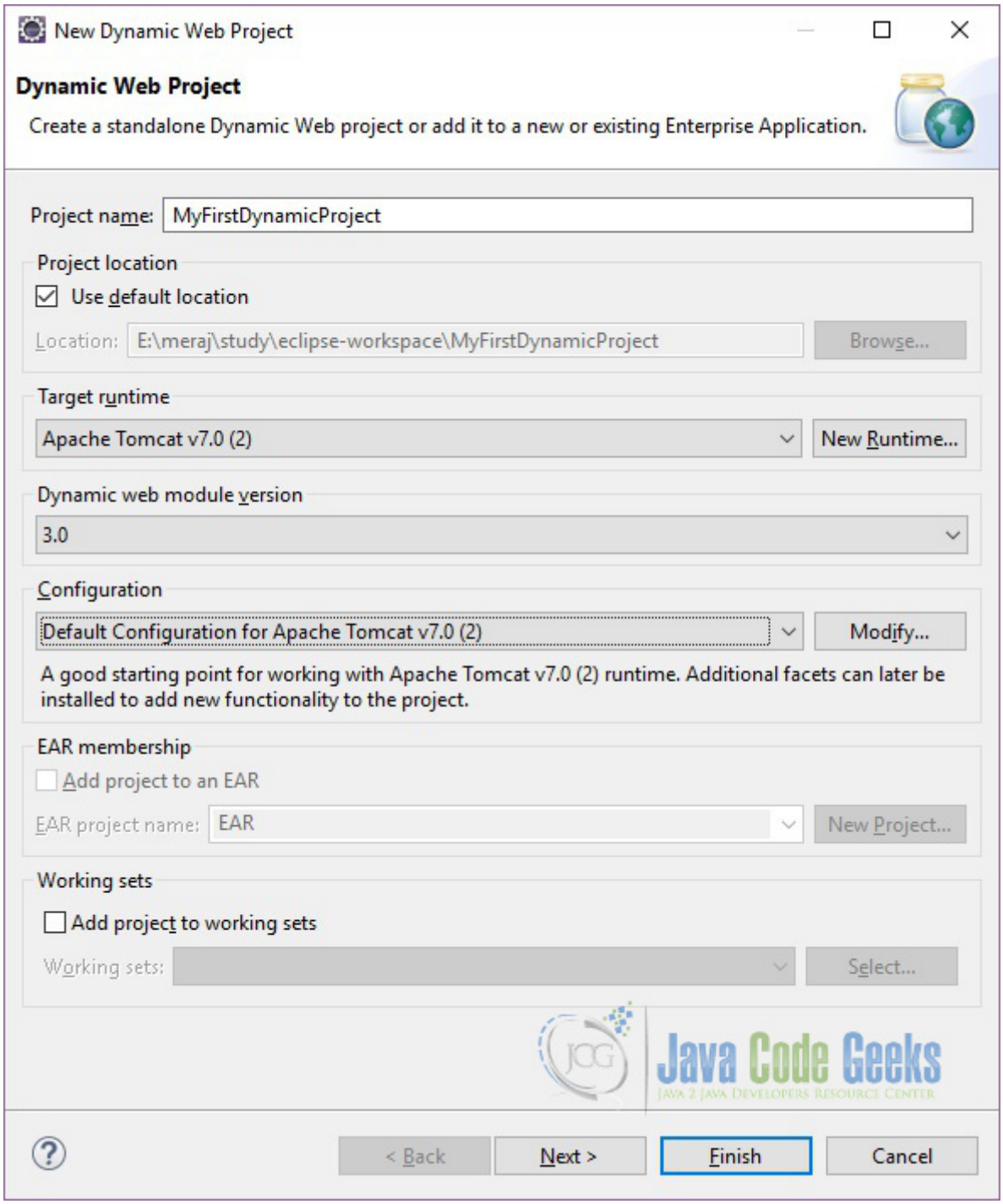

Figure 3.3: New Dynamic Web Project

On the next pop-up click *Next*. In the next pop-up (Web Module) you can define the Context root and content directory. For our example we will leave the default values as it is.

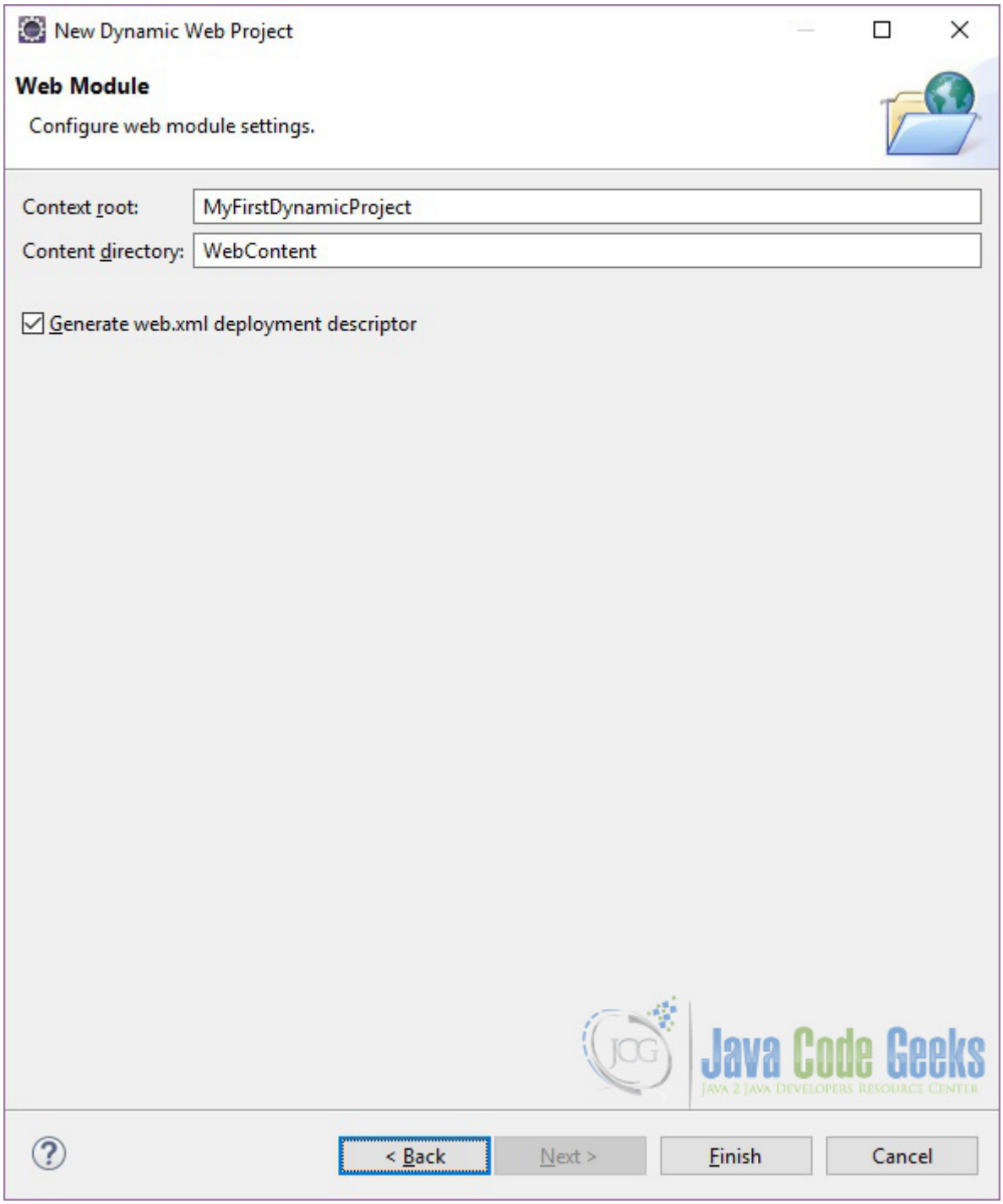

Figure 3.4: Web Module

Click *Finish*. Eclipse will generate some files.

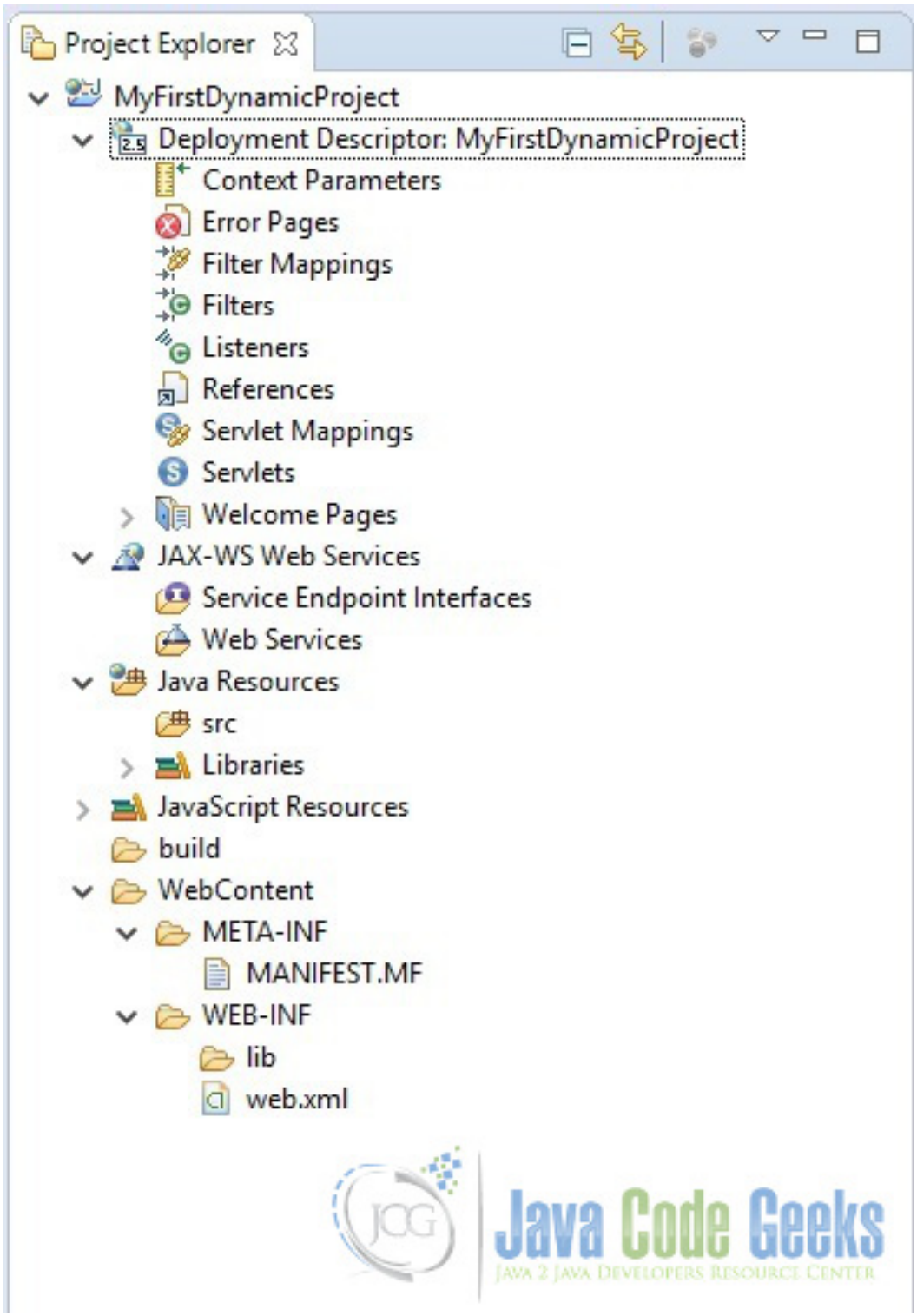

Figure 3.5: Generated Files

Now lets create a very simple servlet for our example. Right click on the *src* folder and choose New⇒Package. Give the package name (com.javacodegeeks). Click Finish. Now right click on the package and choose New⇒Servlet. Give the servlet name (MyServlet) and click *Finish*.

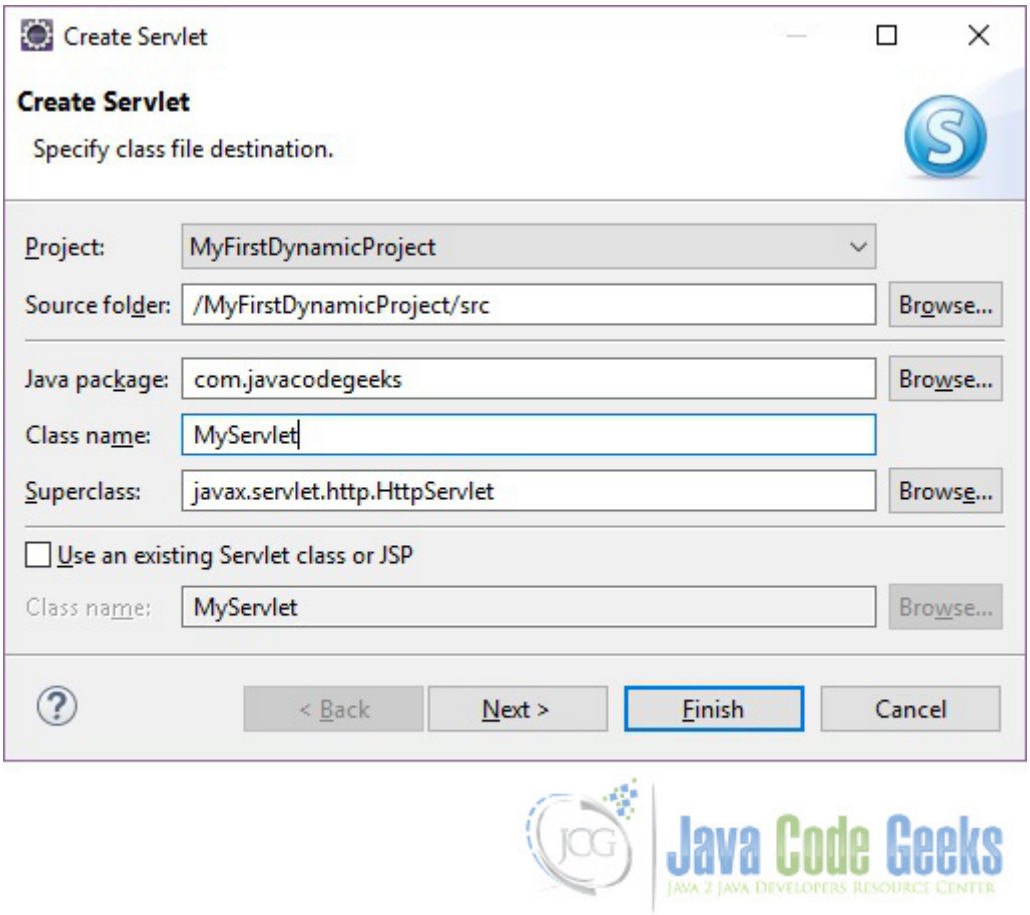

Figure 3.6: Create Servlet

Eclipse will create a sample MyServlet class with two methods - *doGet()* and *doPost()*. These are the two most important methods of any servlet. You can read more about these methods here: https://docs.oracle.com/javaee/5/api/javax/servlet/http/-HttpServlet.html#doGet(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)[doGet()], https://docs.oracle.co javaee/5/api/javax/servlet/http/HttpServlet.html#doPost(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)[Now lets update the doGet() method. First we will get the reference to the `PrintWriter` by calling the `response.getWr iter()` method. This returns a `PrintWriter` object that can send character text to the client. The `PrintWriter` uses the character encoding returned by `getCharacterEncoding()`. If the response's character encoding has not been specified as described in `getCharacterEncoding` (i.e., the method just returns the default value ISO-8859-1), `getWriter` updates it to ISO-8859-1. Calling `flush()` on the `PrintWriter` commits the response. Either this method or `getOutputStream()` may be called to write the body, not both.

The `doGet()` method will look like below:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws  ↩
    ServletException, IOException {
  PrintWriter pw = response.getWriter();
  pw.write("My Fist Dynamic Web Project");
}
```

Now lets test the application. Fist we need to start the server. To start the Server right click on the server and select Start.

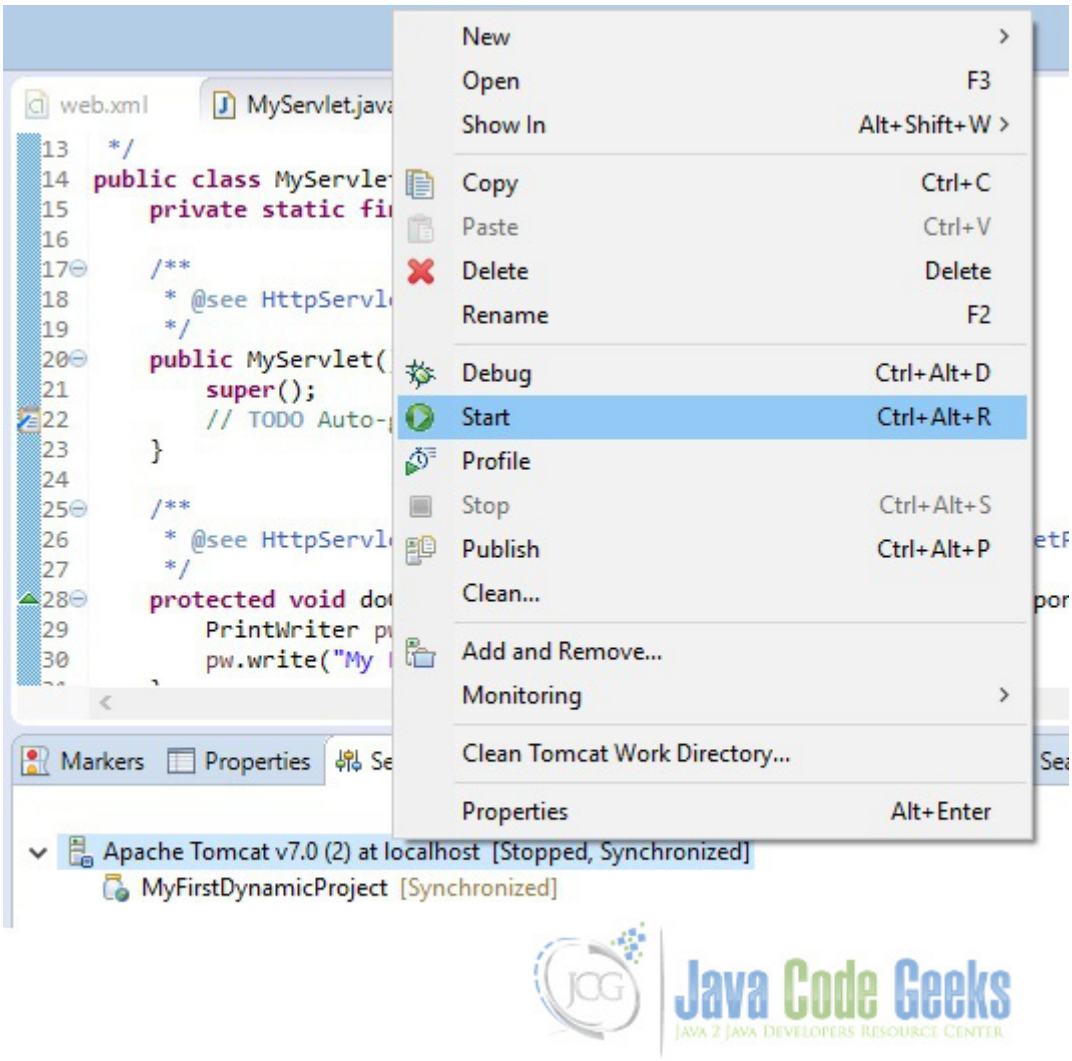

Figure 3.7: Start Server

Once the server is started successfully, go to your preferred browser and type this URL: https://localhost:8080/MyFirstDynamicProject/-MyServlet and press enter. You will see the text you set in the print writer is displayed. Now lets understand what the composition of URL. URL stands for Uniform Resource Locator. It is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it.

The first part of the URL is the **scheme**. In out case it's *http*. *http* is referred to as Hyper Text Transfer Protocol. *localhost* refer to the machine where our application is deployed. In our case it's the localhost. After the host, we provide the port where the application is listening to. After the post, we provide the context root. For our application it's the same as the project name. You must be wondering what the last part (MyServlet) is ? When we created the default Servlet using Eclipse, Eclipse update the web.xml with the below entries:

```
<servlet>
  <description></description>
  <display-name>MyServlet</display-name>
  <servlet-name>MyServlet</servlet-name>
  <servlet-class>com.javacodegeeks.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>MyServlet</servlet-name>
  <url-pattern>/MyServlet</url-pattern>
</servlet-mapping>
```

3.5.1 Project Facets

A facet represents a unit of functionality in a Web project. For example, the Dynamic Web Module facet enables the project to be deployed as a dynamic Web module. A brief description of a project facet appears in the wizard when you select it. Note that in many instances, you can view the constraints for a project facet by right click on the facet and select project constraints from the pop up menu.

3.5.2 Context Root

The context root is the Web application root, which is the top-level directory of your application when it is deployed to the Web server. You can change the context root after you create a project using the project Properties dialog, which you access from the project's pop-up menu. The context root can also be used by the links builder to ensure that your links remain ready to publish as you move and rename files inside your project.

3.6 Dynamic Web applications

There are two types of Web projects: dynamic and static. Dynamic web projects can contain dynamic Java EE resources such as servlets, JSP files, filters, and associated metadata, in addition to static resources such as images and HTML files. Static web projects only contain static resources. When you create Web projects, you can include cascading style sheets and JSP tag libraries (for dynamic Web projects), so that you can begin development with a richer set of project resources.

Dynamic Web projects are always embedded in Enterprise Application projects. The wizard that you use to create a dynamic Web project will also create an Enterprise Application (EAR) project if it does not already exist. The wizard will also update the application.xml deployment descriptor of the specified Enterprise Application project to define the Web project as a module element. If you are importing a WAR file rather than creating a dynamic Web project new, the WAR Import wizard requires that you specify a Web project, which already requires an EAR project.

3.6.1 WebContent folder

The mandatory location of all Web resources, includes HTML, JSP, graphic files, and so on. If the files are not placed in this directory (or in a subdirectory structure under this directory), the files will not be available when the application is executed on a server. The Web content folder represents the contents of the WAR file that will be deployed to the server. Any files not under the Web content folder are considered development-time resources (for example, .java files, .sql files, and .mif files), and are not deployed when the project is unit tested or published.

Though the default name given to the folder is WebContent, you can change the name in the Project Explorer by right-clicking the folder and selecting RefactorRename or from the Web page of the project's Properties dialog. In a dynamic Web project, changing the folder name will update the Java build output directory.

3.6.1.1 6.1.1. META-INF

This directory contains the MANIFEST.MF file, which is used to map class paths for dependent JAR files that exist in other projects in the same Enterprise Application project. An entry in this file will update the run-time project class path and Java build settings to include the referenced JAR files.

3.6.1.2 6.1.2. WEB-INF

Based on the Sun Microsystems Java Servlet 2.3 Specification, this directory contains the supporting Web resources for a Web application, including the web.xml file and the classes and lib directories.

3.6.1.3 6.1.3. Classes

This directory is for servlets, utility classes, and the Java compiler output directory. The classes in this directory are used by the application class loader to load the classes. Folders in this directory will map package and class names, as in: /WEB-INF/classes/com/mycorp/servlets/MyServlet.class. Do not place any .class files directly into this directory. The .class files are placed in this directory automatically when the Java compiler compiles Java source files that are in the Java Resources directory. Any files placed directly in this directory will be deleted by the Java compiler when it runs.

3.6.1.4 6.1.4. Lib

The supporting JAR files that your Web application references. Any classes in .jar files placed in this directory will be available for your Web application.

3.7 Testing and publishing on your server

The testing and publishing tools provides runtime environments where you can test JSP files, servlets, HTML files, Java classes and many more artifacts. You can use the workbench to test and publish resources from many types of projects. Here are some examples:

- Dynamic Web projects, which typically contain JSP files, HTML files, servlets, and JavaBeans
- Static Web projects, which typically contain HTML files and graphic files
- Enterprise Applications projects, which may contain Java Archive (JAR) files or Web Archive (WAR) files or both, and pointers to other Web or EJB projects
- EJB projects, which contain enterprise beans
- Application client projects

After testing your application, you can use the tools to publish the application.

3.7.1 Server definitions

The workbench defines servers to test and publish your projects. Servers are definitions that identify where you want to test your projects. You can either have the development environment create the servers automatically for you, or you can create them using the New Server wizard. To open the Server view go to Window⇒Show View⇒Servers. If there is no Server defined you will see a link saying *No servers are available. Click this link to create a new server...*. Click on this link

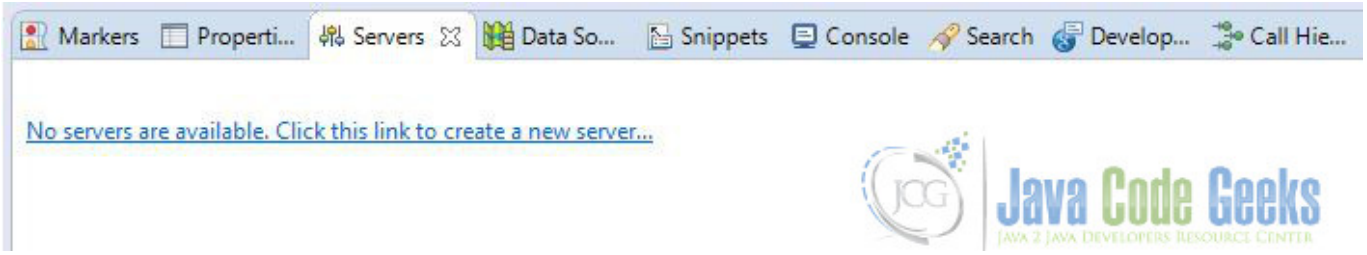

Figure 3.8: Servers View

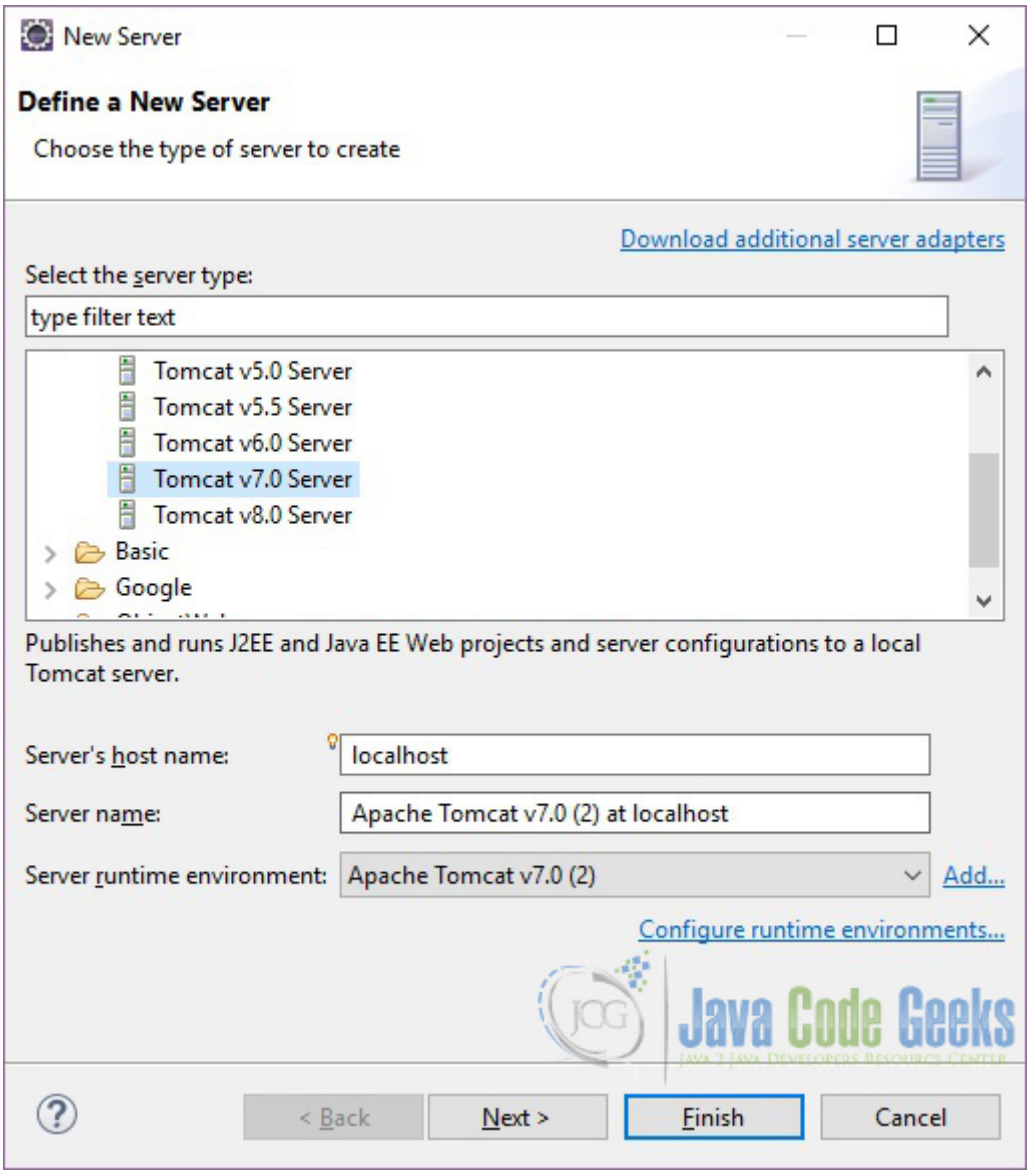

Figure 3.9: New Server

For our example we will choose *Tomcat v7.0 Server*. Leave the rest of the field values as default. Click *Next*. On the next screen select the project and click *Add* then click *Finish*

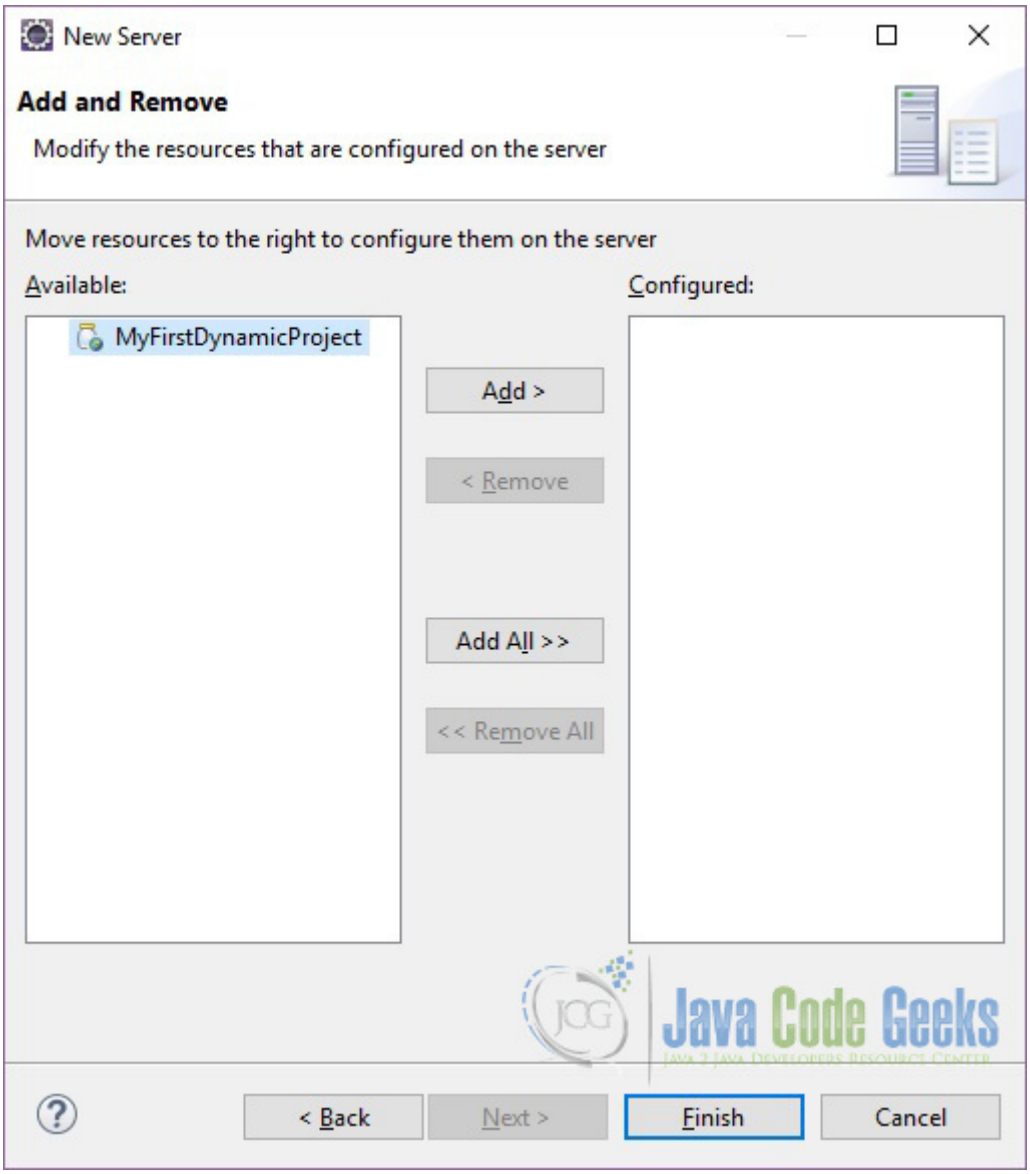

Figure 3.10: Add and Remove

You will see the server in the Servers tab and also in the Project Explorer tab. The Servers view (similar to the one shown below) allows you to manage the servers. This view displays a list of all your servers and projects that are associated with that server. A project displays under a server when a project from the workbench is added to the server. You can use this view to start, start in debug mode, restart, or stop the servers. In addition, you can use the Servers view to determine the current status and state of the server; and the projects added to the server from the workbench.

3.8 Conclusion

In this tutorial we saw how we can make use of the in-build features of Eclipse to create a web application. This is a simple example of the features which Eclipse provides. There are a lot other features which can be used for building much more complex applications.

Chapter 4

How to update Eclipse

In this article we will see how we can update Eclipse. Eclipse is the most popular Integrated Development Environment (IDE) used by Java developers. The Eclipse platform itself is structured as subsystems which are implemented in one or more plug-ins. The subsystems are built on top of a small runtime engine.

The term Workbench refers to the desktop development environment. The Workbench aims to achieve seamless tool integration and controlled openness by providing a common paradigm for the creation, management, and navigation of workspace resources. Each Workbench window contains one or more perspectives. Perspectives contain views and editors and control what appears in certain menus and tool bars. More than one Workbench window can exist on the desktop at any given time.

4.1 Introduction

If you are upgrading to a newer release of Eclipse from an older release, there are simple steps to follow to migrate your workspace to the new release. Your workspace is the directory on disk that contains all of your project files, as well as meta-data such as preferences you may have customized. The steps to follow for upgrading depend on whether or not you used the "-data" command line argument when starting Eclipse. The "-data" argument is recommended because it clearly specifies the location of your workspace.

The workspace chooser dialog allows you to choose the location of your workspace. This dialog appears on first startup in the absence of a -data argument. The default location provided by this dialog will be a "workspace" child of your home directory

Unless you have an existing workspace from a previous Eclipse version, you can keep this default or choose some other location. You should not store your workspace inside the Eclipse install directory, because that will make it more difficult to upgrade to a newer version of Eclipse. You should not copy or move the workspace directory, because it may contain metadata with absolute file system paths, which will be invalid if the workspace is copied elsewhere.

4.2 Add New Repository

If upgrading the platform itself to the next full release follow the steps below:

- Go to Window ⇒ Preferences ⇒ Install/Update ⇒ Available Software Sites

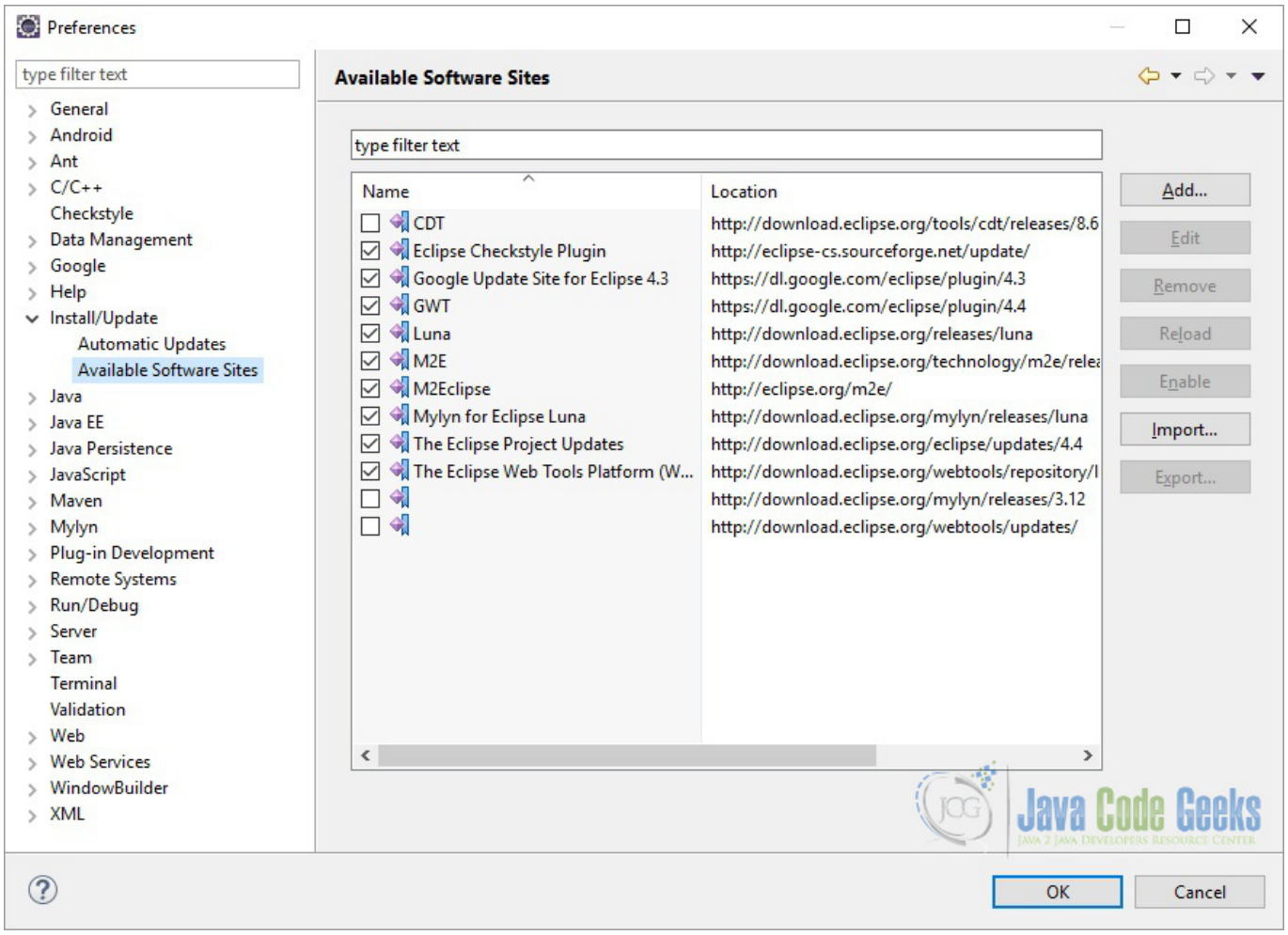

Figure 4.1: Available Software Sites

- Click *Add*. Enter the URL of the new repository (for example, https://download.eclipse.org/releases/mars/ for Mars (4.5)). Click OK

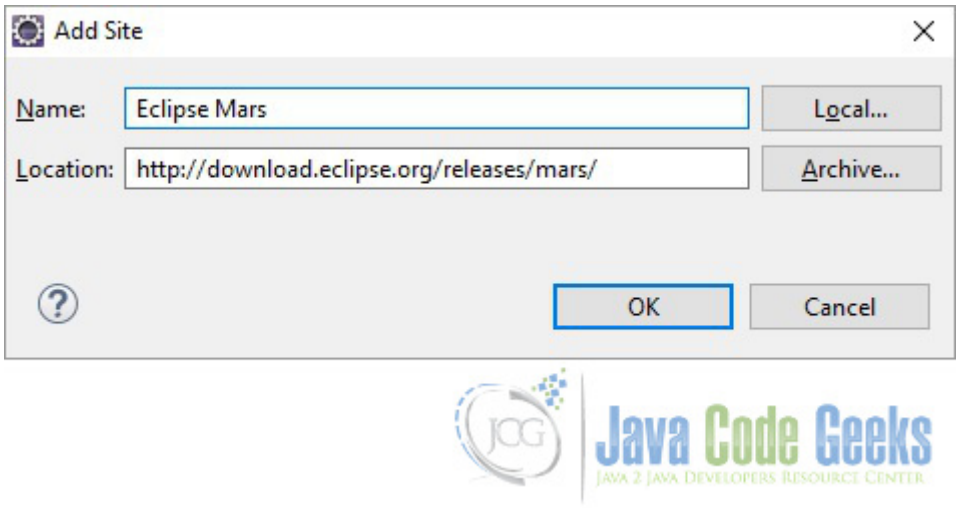

Figure 4.2: Add Site

4.3 Check for Updates

Eclipse provides the facility to check for any updates for the existing features and install those updates. To check for updates go to Help ⇒ Check For Updates.

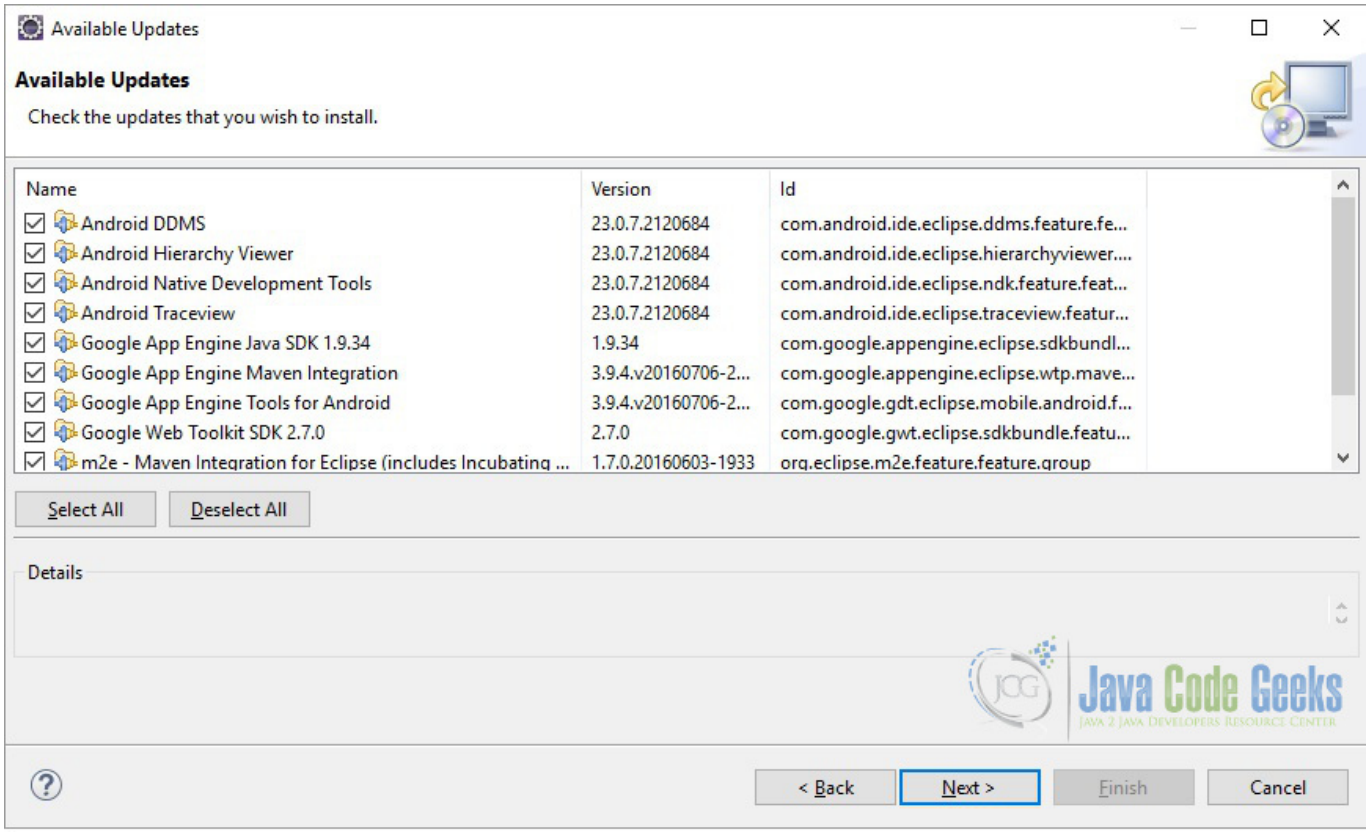

Figure 4.3: Available Updates

Here you can choose the items which you want to update. If upgrading the platform itself, when you are prompted to restart it is strongly recommended to do so. A restart may not be required when updating other features, but always select to restart if you are unsure. Check your Available Software Sites for release-specific update sites that may need updating as well, e.g. The Eclipse Project Updates URL changes with each release (typically release-specific sites hold the SDK/Source features and off-cycle hot fixes and are not required)

Occasionally you won't be able to upgrade Eclipse or certain features due to incompatible changes in the update technology. In these cases you will need to download a fresh install. Download a new build from the Eclipse download Web site Eclipse Download and unzip it in a new directory. Complete upgrade instructions are always included in the Eclipse readme_eclipse.html file included with every build in the readme directory.

Upgrades may require administrator privileges to succeed and may fail with error messages claiming "Only one of the following can be installed:" otherwise. Start Eclipse with "Run as administrator...".

4.4 Update Manager

The Update Manager allows you to find new plug-ins on your machine, your network, or the Internet, compare new plug-ins to your configuration, and install only those that are compatible with your current configuration. The Update Manager thinks in terms of features, a logical group of related plug-ins, and also provides support for managing configurations to undo a given installation or to automatically update all the features currently installed in your Eclipse configuration. Before plug-ins can be

installed by the Update Manager, they need to be collected into a feature. The feature itself has to be published using an update site.

The Update Manager is invoked by Help ⇒ Software Updates. Remember to have an *open* connection to the internet when updating. Proxy settings can prevent the update mechanism from reaching the locations you want/need. The proxy-settings can be changed using Windows ⇒ Preferences ⇒ General ⇒ Network Connections.

4.5 Conclusion

In this article we saw how we can update Eclipse. It is very important feature of any software as it allows new features to be integrated with existing ones without reinstalling the software again. We also saw how we can add new repositories for downloading new plugins.

Chapter 5

How to install plugin in Eclipse

Eclipse is a platform that has been designed from the ground up for building integrated web and application development tooling.

5.1 Introduction

By design, the platform does not provide a great deal of end user functionality by itself. The value of the platform is what it encourages: rapid development of integrated features based on a **plug-in** model. Eclipse provides a common user interface (UI) model for working with tools. It is designed to run on multiple operating systems while providing robust integration with each underlying OS.

Plug-ins can program to the Eclipse portable APIs and run unchanged on any of the supported operating systems. At the core of Eclipse is an architecture for dynamic discovery, loading, and running of plug-ins. The platform handles the logistics of finding and running the right code. The platform UI provides a standard user navigation model. Each plug-in can then focus on doing a small number of tasks well. What kinds of tasks? Defining, testing, animating, publishing, compiling, debugging, diagramming. . . the only limit is your imagination. There are various ways to install a plugin in Eclipse.

5.2 Plug it in

The Eclipse platform is structured as a core runtime engine and a set of additional features that are installed as platform **plug-ins**. Plug-ins contribute functionality to the platform by contributing to pre-defined **extension points**. The workbench UI is contributed by one such plug-in. When you start up the workbench, you are not starting up a single Java program. You are activating a platform runtime which can dynamically discover registered plug-ins and start them as needed.

When you want to provide code that extends the platform, you do this by defining system extensions in your plug-in. The platform has a well-defined set of extension points - places where you can hook into the platform and contribute system behavior. From the platform's perspective, your plug-in is no different than basic plug-ins like the resource management system or the workbench itself.

5.3 Eclipse Marketplace

In this section we will see how to use Eclipse Marketplace to install a new plugin.

Go to Help⇒Eclipse Marketplace. . . A pop-up will appear as shown below:

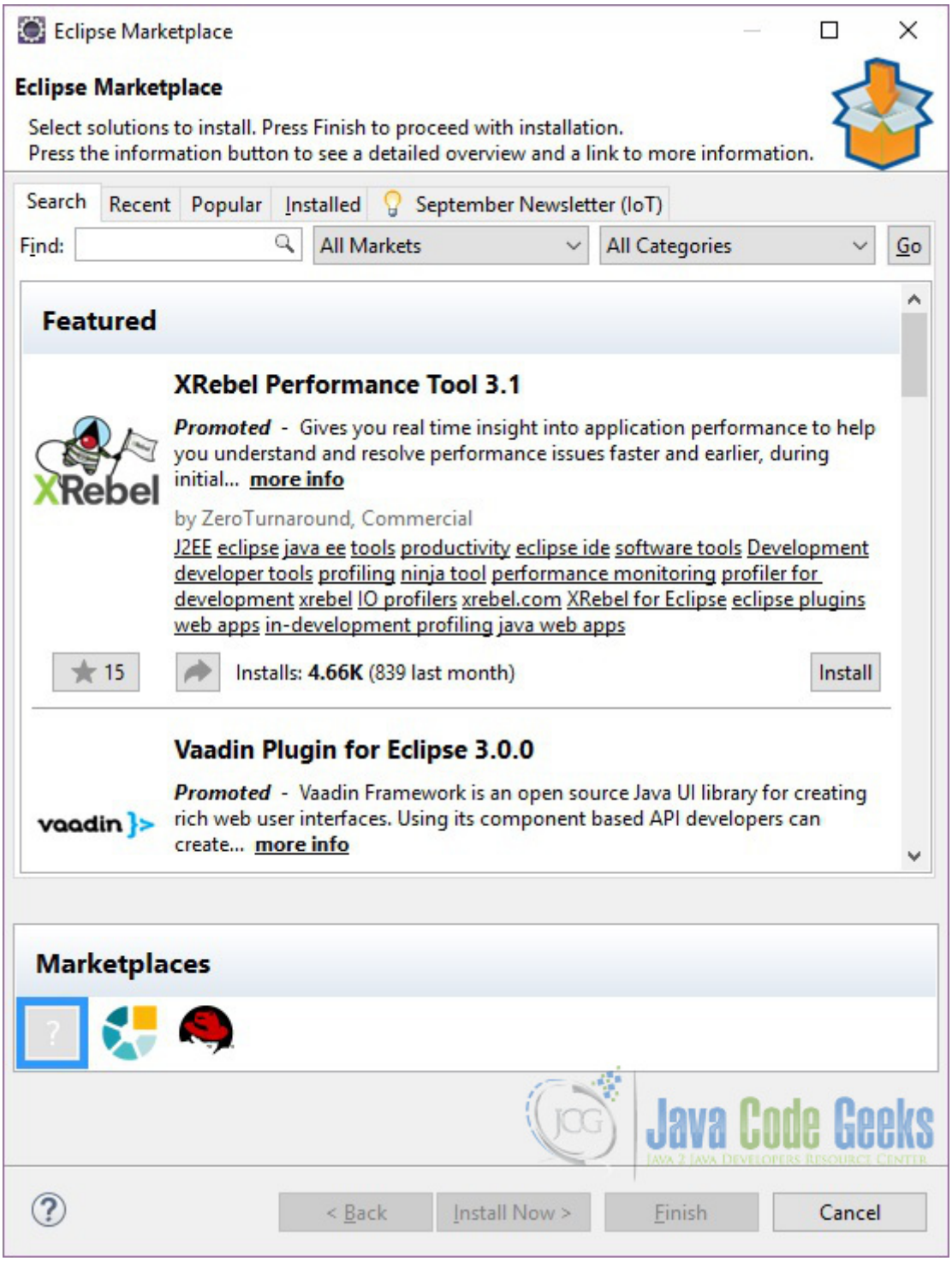

Figure 5.1: Eclipse Marketplace

In the Find text box write the name of the plugin you want to install and click the search icon. Let say we want to install the Subclipse plugin. We will type *Subclipse* in the search box and will click the search icon. Eclipse will display the results matching the criteria:

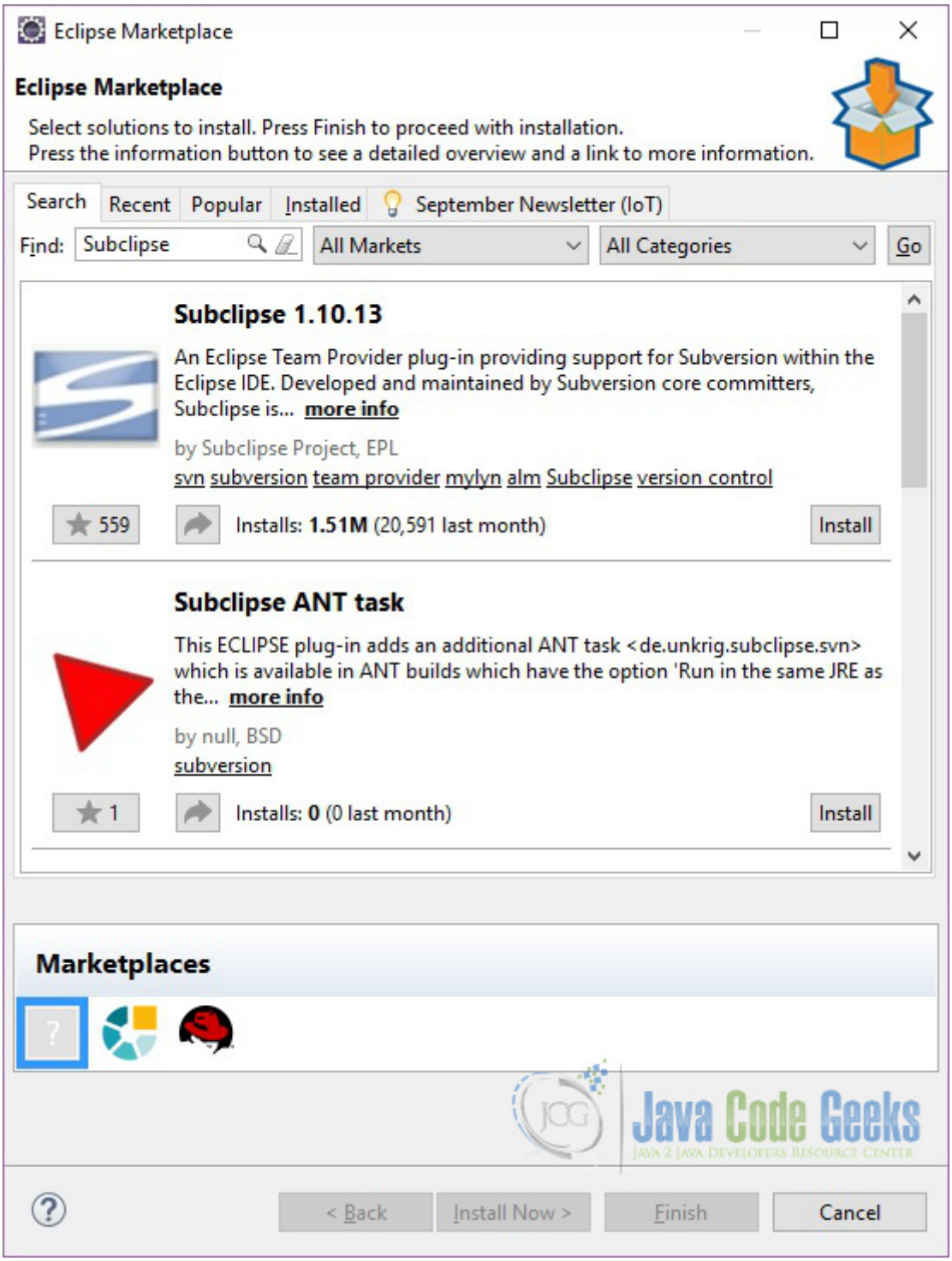

Figure 5.2: Search Results

Click on the Install button for the plugin you want to install. In the next window you will be required to Confirm the selected feature. Here you can unselect the Option features as well. Eclipse helps you to decide by putting the (Optional) text in front of the features which are not Required.

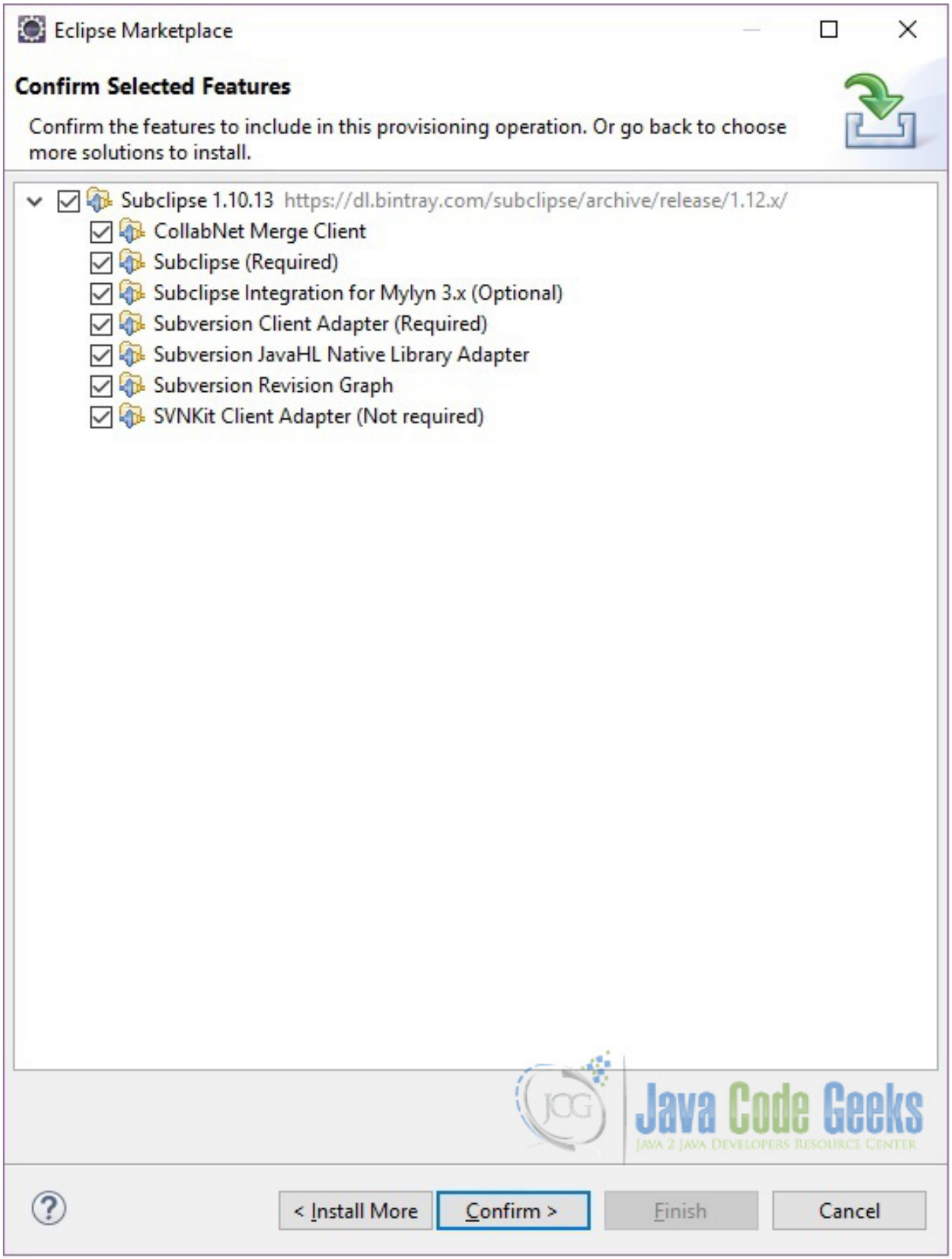

Figure 5.3: Confirm Selected Features

On the next window we need to Accept the licenses. Click on the *I accept the terms of the license agreements* radio button and then click Finish.

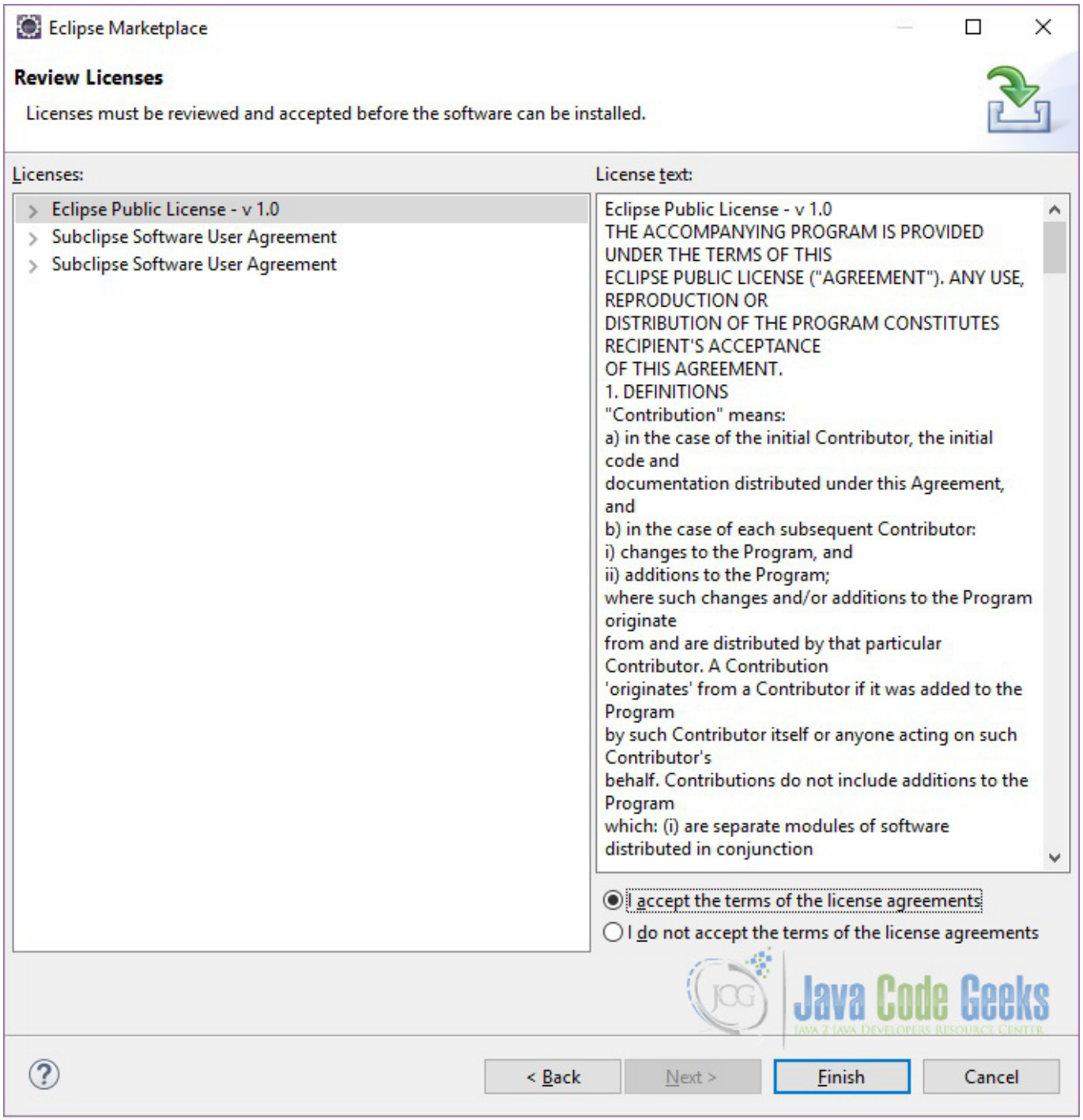

Figure 5.4: Review Licenses

Eclipse will start installing the plugin. Eclipse will display a Warning pop-up, click OK. Once the features are installed Eclipse will ask you to restart the Eclipse for the changes to take effect. Click Yes. Once the Eclipse has restarted you can verify whether the plugin was properly installed or not. Go to Help⇒Eclipse Marketplace... then click on the *Installed* tab. Here you can see the list of plugins that are been installed.

After the Find text box there are two other drop-downs which you can use to filter the search. The first one lets you choose where

you want to search the plugin and the second one lets you choose the category for the plugin.

5.4 Install New Software

The other way to install a plugin is you use the Available Software window. Go to Help⇒Install New Software... Eclipse will displace a pop-up window like below:

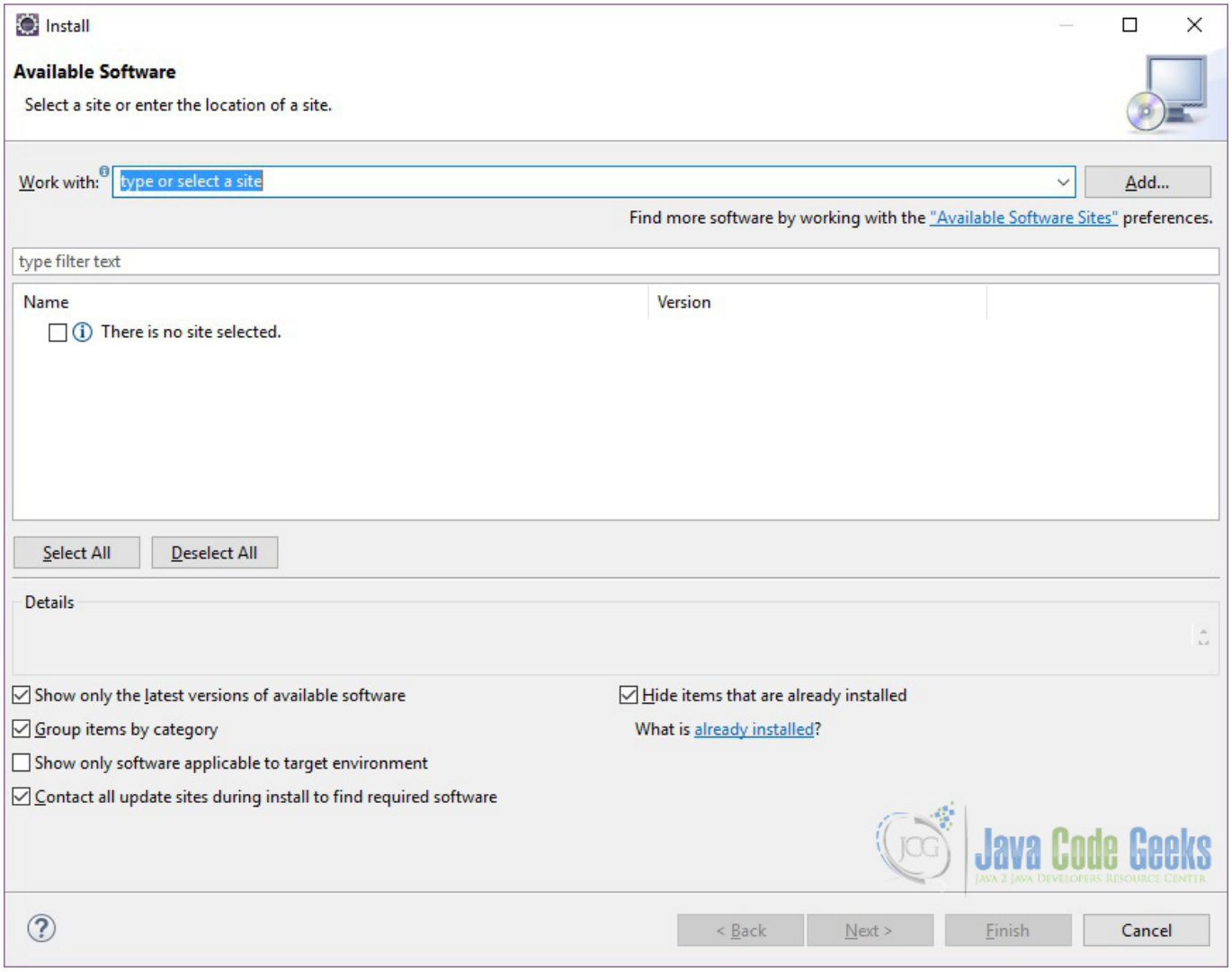

Figure 5.5: Available Software

In the Work with search text box type the site name. You can also select from the existing sites which can be found by clicking the drop-down button. Let's say we want to install the Subclipse, enter the URL: https://dl.bintray.com/subclipse/releases/subclipse/-4.2.x/ in the search box and click Add. Eclipse will display a pop-up window as below:

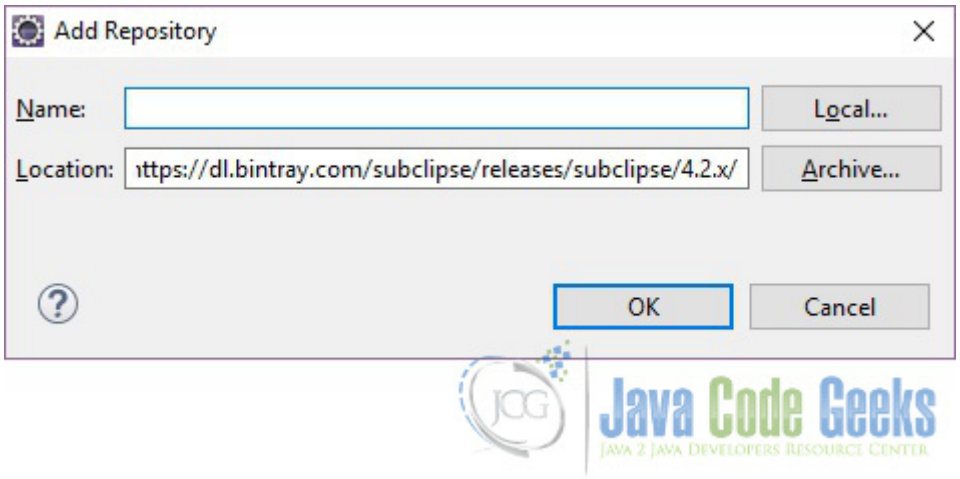

Figure 5.6: Add repository

Give the name of the repository. We will use Subclipse. Click OK. Eclipse will display the search result. Choose the features you want to install and click Next. IN the next window Eclipse will ask you you review the items to be installed. Click Next. In the next window accept the license and click Finish. Eclipse will install the selected plugin.

5.5 Installed Plugins

In this section we will see how to see the list of installed plugin. Go to Help⇒Installation Details. Here you will see the list of installed plugins:

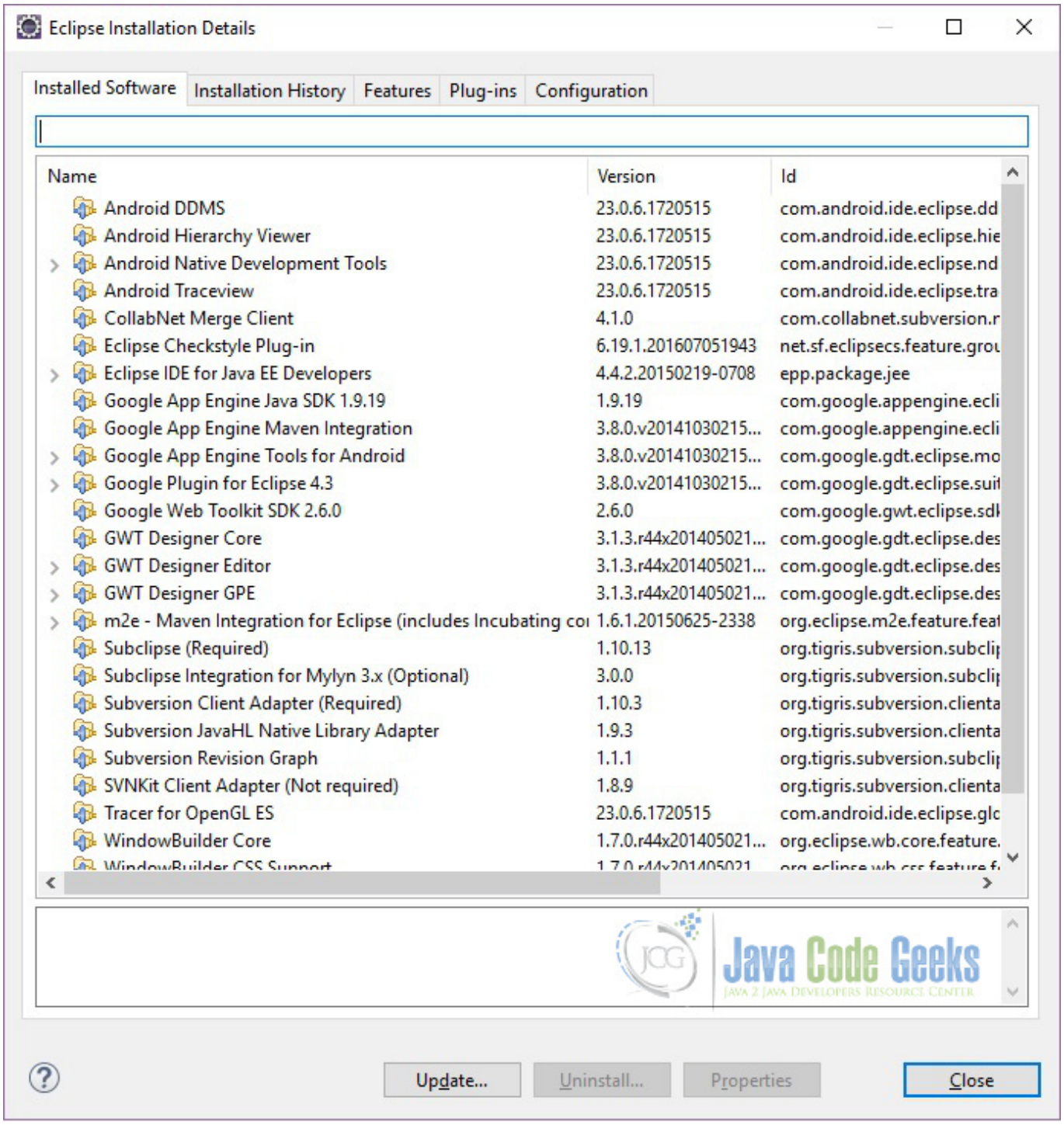

Figure 5.7: Installation Details

In this window you can see the Name, Version, Id and the Provider of the Plugin. You can search for a particular plugin by typing the name in the search text box on the top of the window. To see the History of the installation click on the *Installation History* tab:

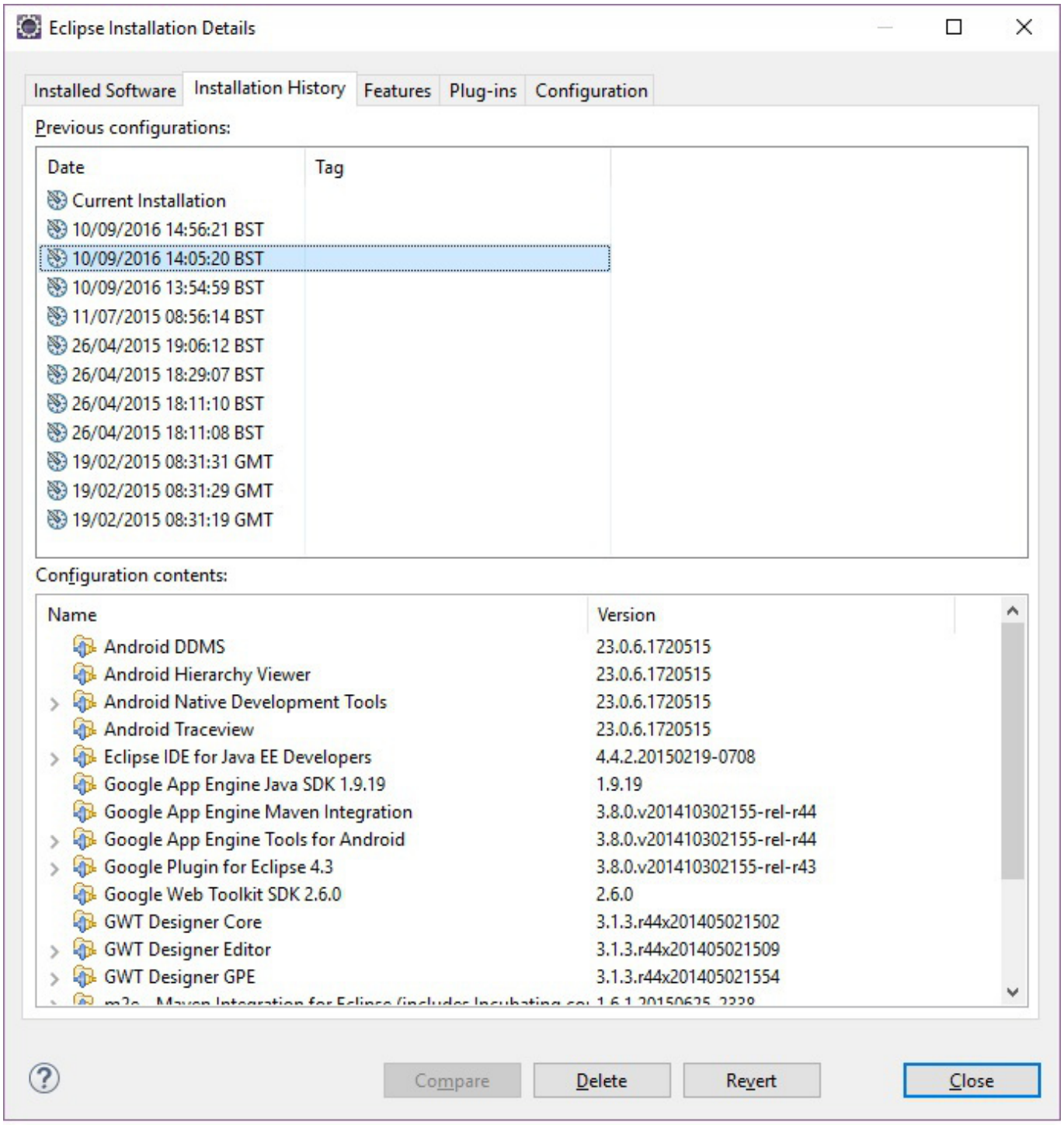

Figure 5.8: Installation History

In this window you can see the installation history by dates.

5.6 Update Plugin

To update a plugin, go to Help⇒Installation Details. Click on the plugin you want to update and then click on the *Update* button at the bottom of the window. Eclipse will update the plugin accordingly. If there is no updates Eclipse will display a pop-up saying *No updates were found*.

Figure 5.9: No Updates Found

You can also update the plugin using Eclipse Marketplace. Go to Eclipse Marketplace and in the search text box insert the plugin name which you want to update. Eclipse will display the *Update* button which you can use to update the plugin:

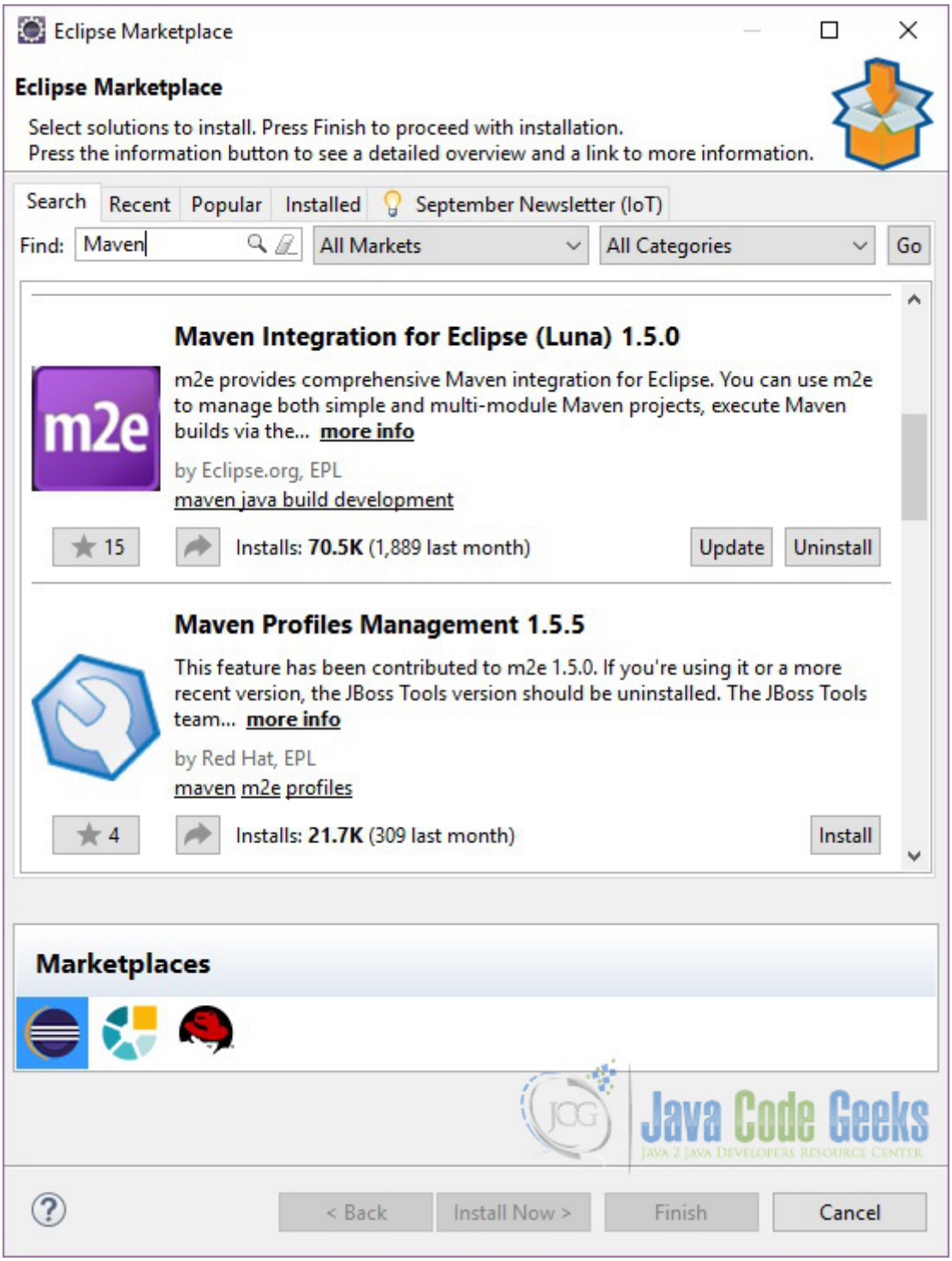

Figure 5.10: Update Plugin

5.7 Remove Plugin

In this section we will see how we can uninstall a plugin. To Uninstall a plugin go to Help⇒Installation Details. Now select the plugin which you want to uninstall and click the *Uninstall...* button. Eclipse will ask you to review and confirm that you want to uninstall the particular plugin. Click Finish. The plugin will be removed from the Eclipse. You will be required to restart the

Eclipse for the changes to take effect.

You can also uninstall a plugin using Eclipse Marketplace. Go to Eclipse Marketplace and in the search text box insert the plugin name which you want to uninstall. Eclipse will display the *Uninstall* button which you can use to uninstall the plugin

5.8 Conclusion

In this article we saw what is an Eclipse plugin and how useful it can be. We also saw how we can install/update/uninstall a plugin. We also discussed the various ways of using a plugin and how Eclipse Marketplace makes it easy for us to manage the plugins.

Chapter 6

Eclipse Window Builder Tutorial for GUI Creation

6.1 Introduction

In this example, we will show you how to develop Java GUI Application using Eclipse WindowBuilder plug-in.

Eclipse WindowBuilder is a powerful and easy to use bi-directional Java GUI designer that makes it very easy to create Java GUI applications without spending a lot of time writing code to display simple forms.

The bi-directional Java GUI designer means the developer can seamlessly move between a Drag n' Drop designer and the generated code.

Using Eclipse WindowBuilder, the developer will enjoy creating Java GUI based applications. One can create complicated windows in minutes using WindowBuilder.

WYSIWYG (What You See Is What You Get) layout tools in WindowBuilder are used to generate back-end java code by drag-and-drop of components to the container.

6.2 Simple Java Window Application

Now, we will see how fast a simple Java GUI application can be created uisng Eclipse WindowsBuilder.

6.2.1 2.1 System requirements

Tools required to run this example are:

6.2.1.1 2.1.1 Eclipse

WindowBuilder is built as a plug-in to Eclipse. *Eclipse for RCP and RAP Developers* is the default IDE bundled with *Windows Builder* plug-in. This IDE has a complete set of tools for developers who want to create Eclipse plug-ins, Rich Client Applications (RCA).

Download *Eclipse for RCP and RAP Developers* from here. Please refer the picture given below to identify the correct IDE.

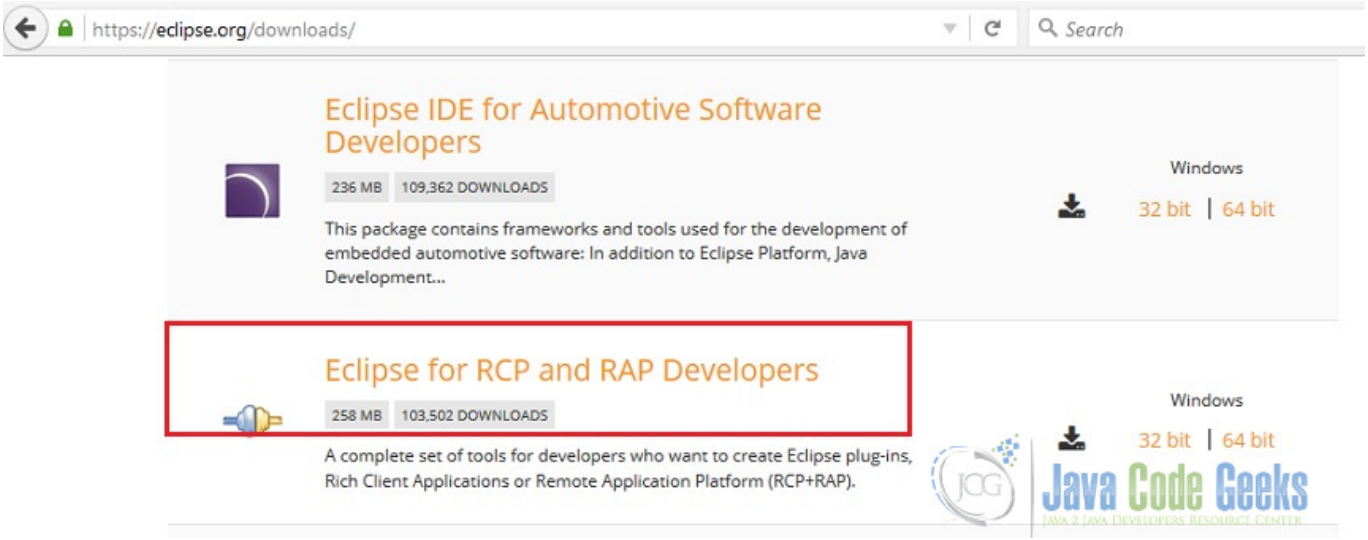

Figure 6.1: Eclipse IDE for RCP and RAD

6.2.1.2 2.1.2 Java

- Download Java SE 7 or above from here

6.3 Open New Project

Let us create a new 'SWT/JFace Java Project' to see the usage of WindowBuilder for building GUI components. Open 'File - New - Other' and then click *SWT/JFace Project* as depicted below

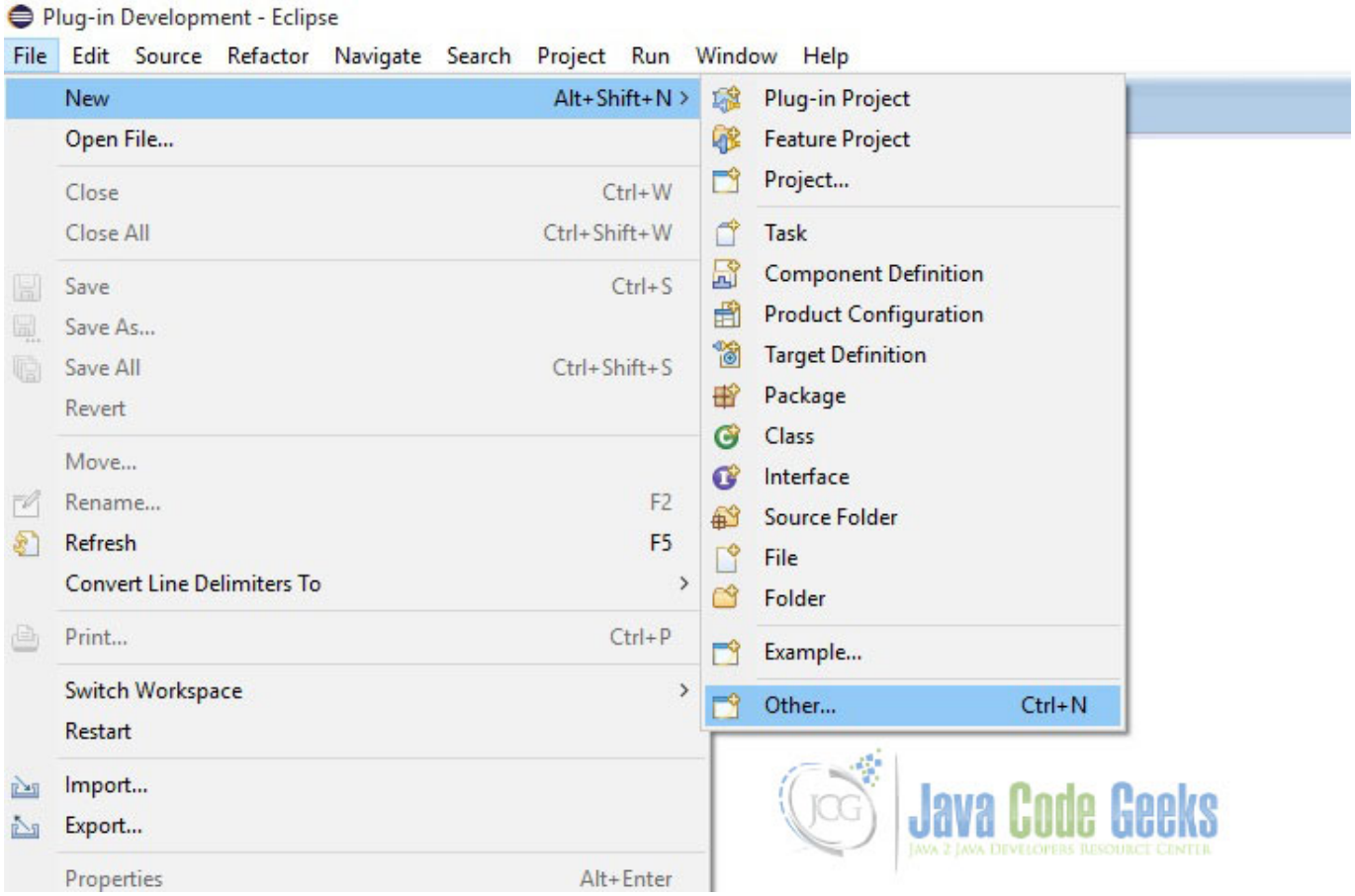

Figure 6.2: Open Project

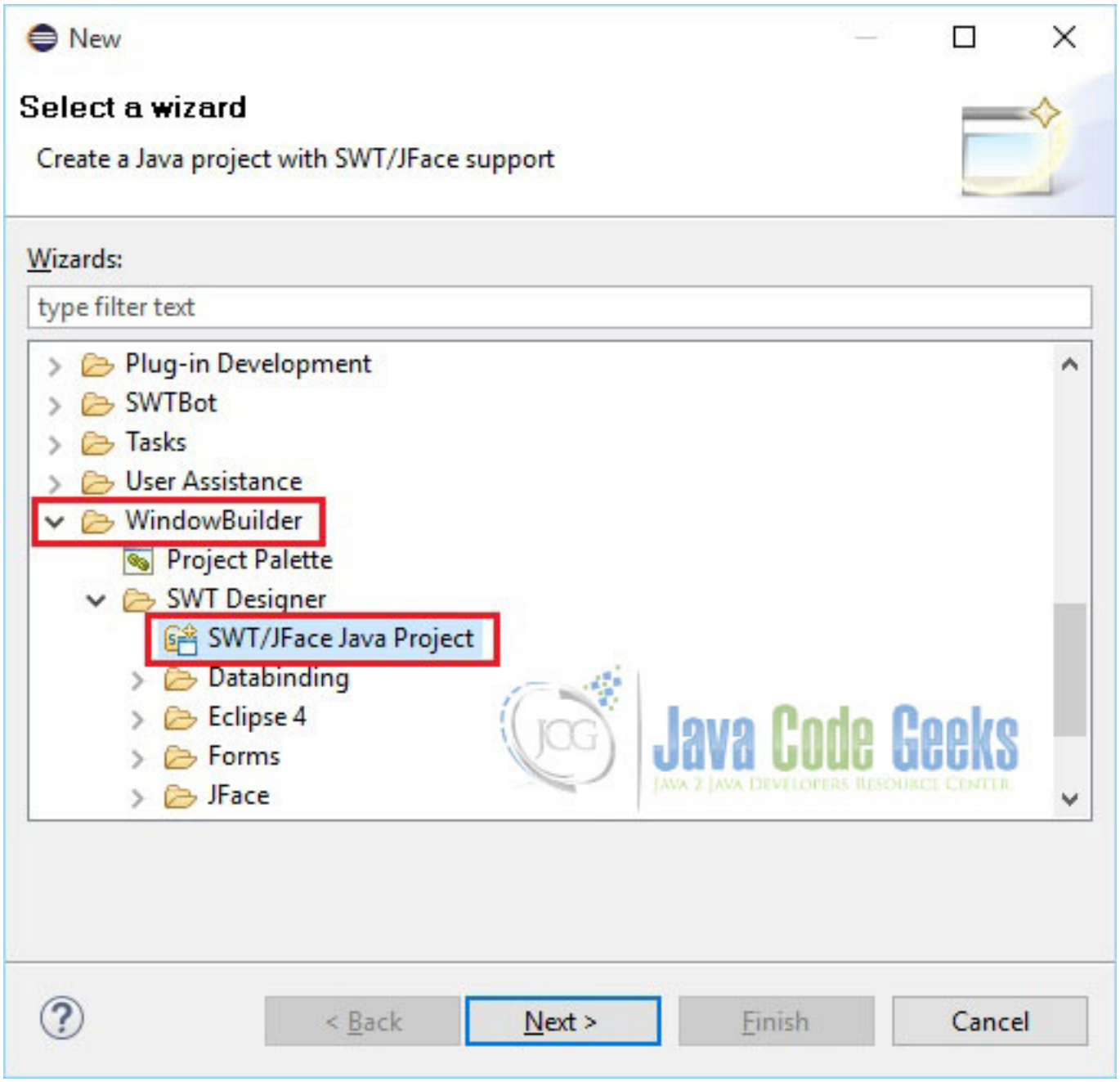

Figure 6.3: SWT JFace Java Project

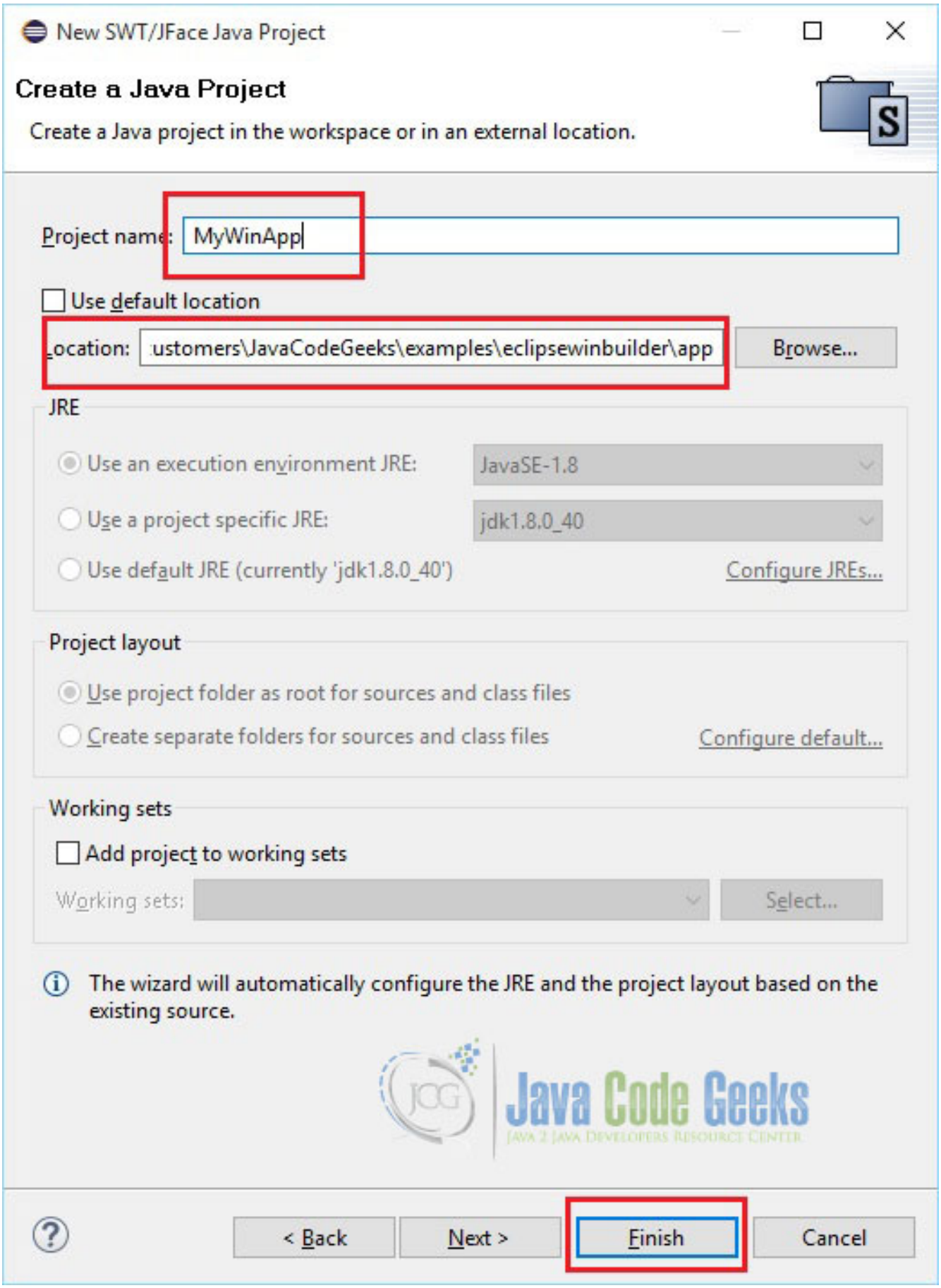

Figure 6.4: Project Name

The reason for creating new project as *SWT/JFace Java Project* is to have all the necessary JARs and native libraries included by the IDE itself. Otherwise, you have to add all these dependent JARs and native libraries on your own.

The Standard Widget Toolkit (SWT) is a graphical widget toolkit to be used with the Java platform. It provides a portable graphics API independent of the OS but that relies on the native widgets.

JFace is a UI toolkit with classes for handling many common UI programming tasks. JFace is window-system-independent in both its API and implementation, and is designed to work with SWT without hiding it.

JFace is a higher-level user interface toolkit that uses the raw SWT widgets to provide model-driven widgets, and to some extent some functionality that isn't available in the Swing libraries, such as advanced editors, dialog boxes, and wizards.

6.4 New SWT Application

Let us add widget to the project. As a main window, create Application Window as shown below. Right click on the project and select *New - Other - Window Builder - SWT Designer - SWT - Application Window*. And then click *Next*

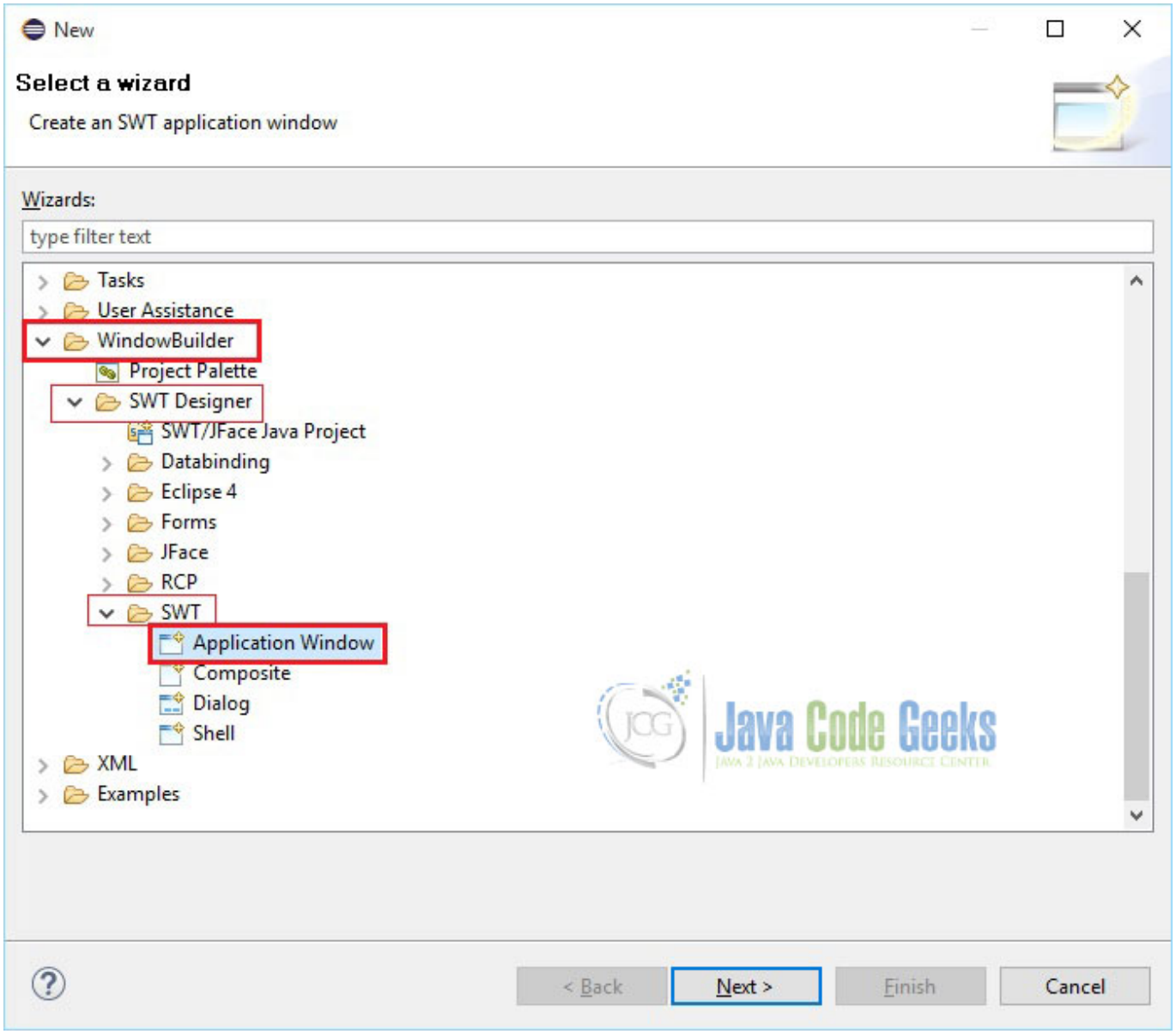

Figure 6.5: Application Window

Enter Class Name and click *Finish*

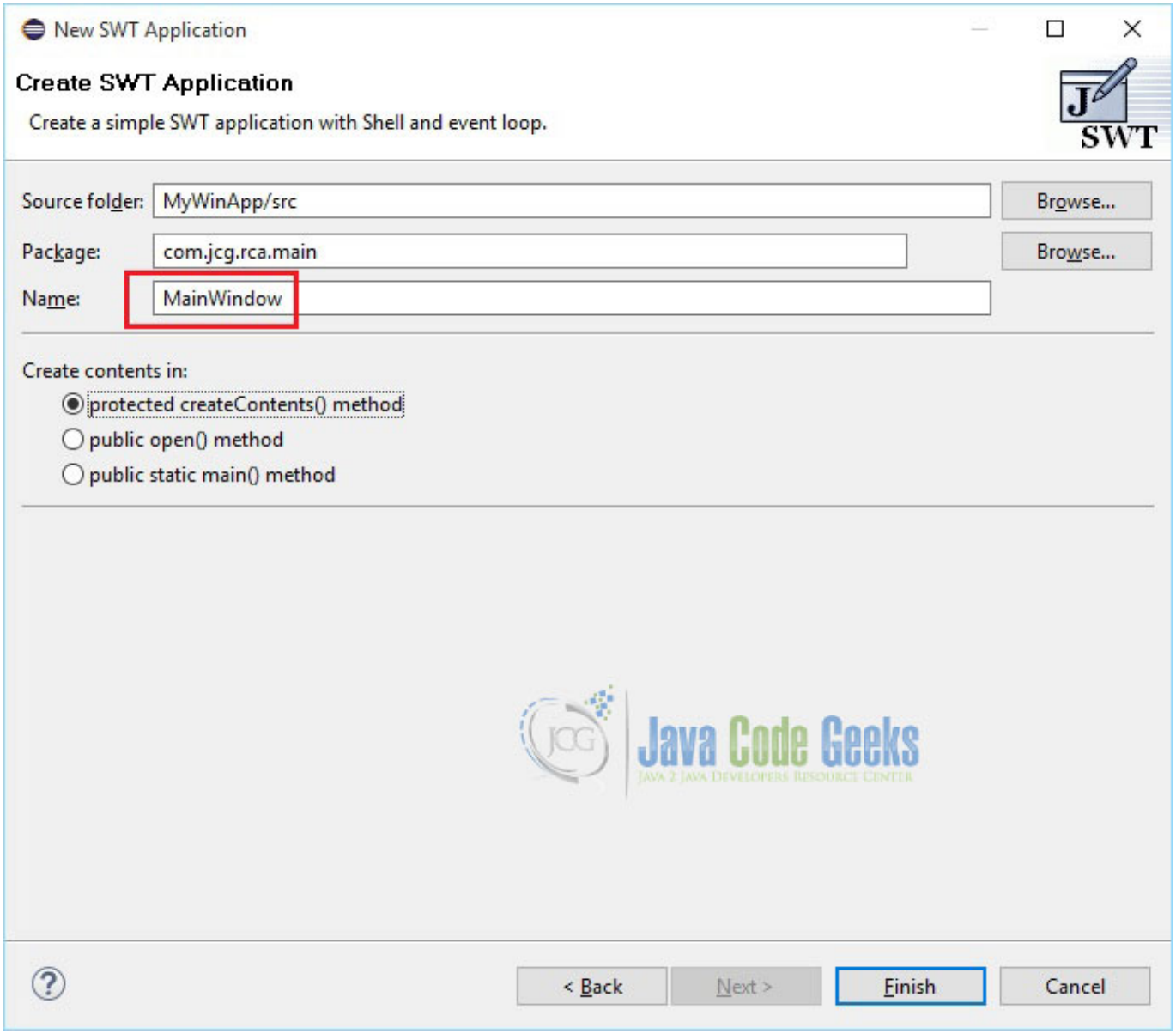

Figure 6.6: Class Name

A basic window application has been created. Window Builder can be used to get your UI up and running quickly. Click *Design* tab as shown below.

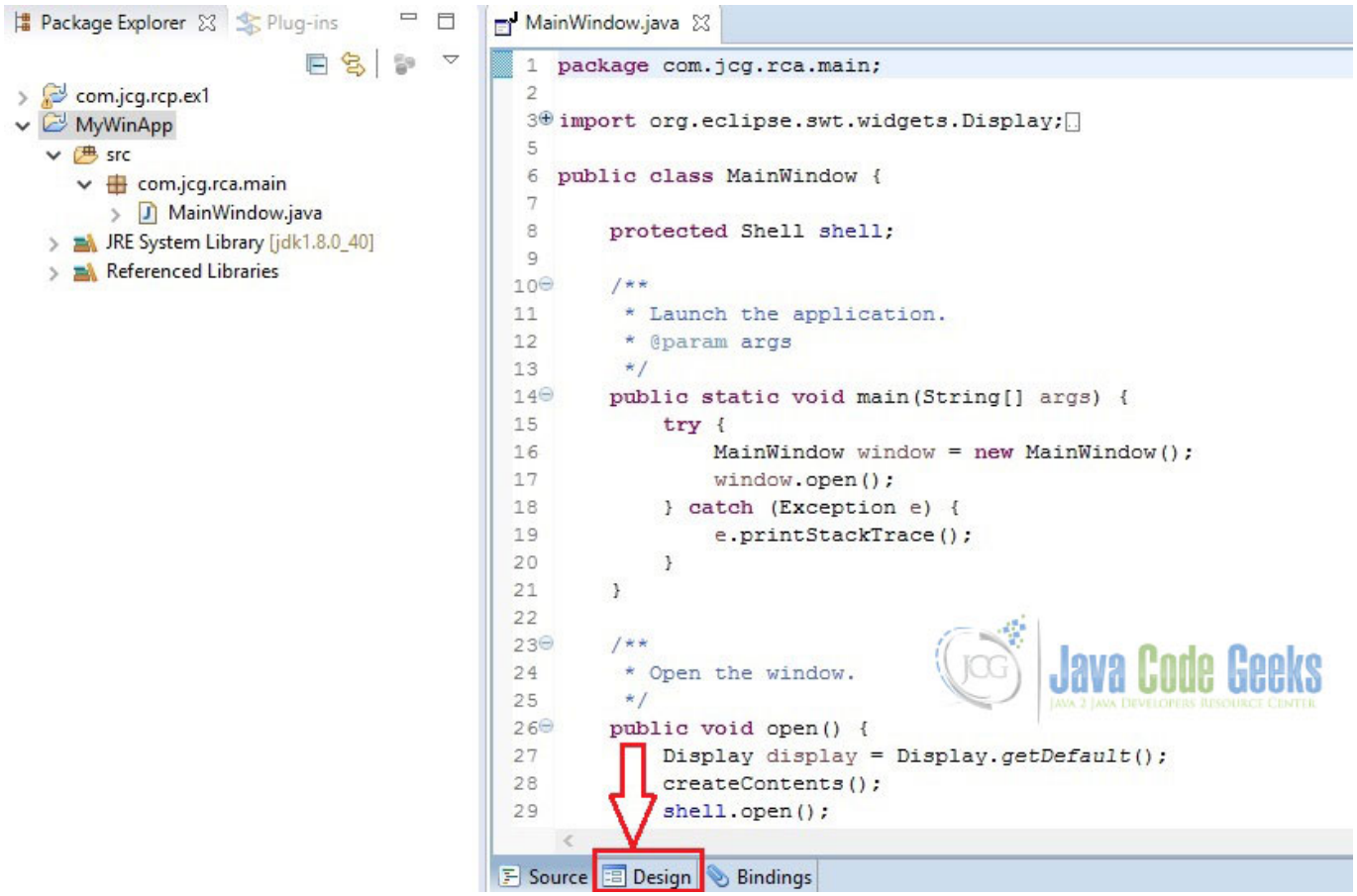

Figure 6.7: Basic Window Application

Now, you will see the graphical representation (Design View) of your code.

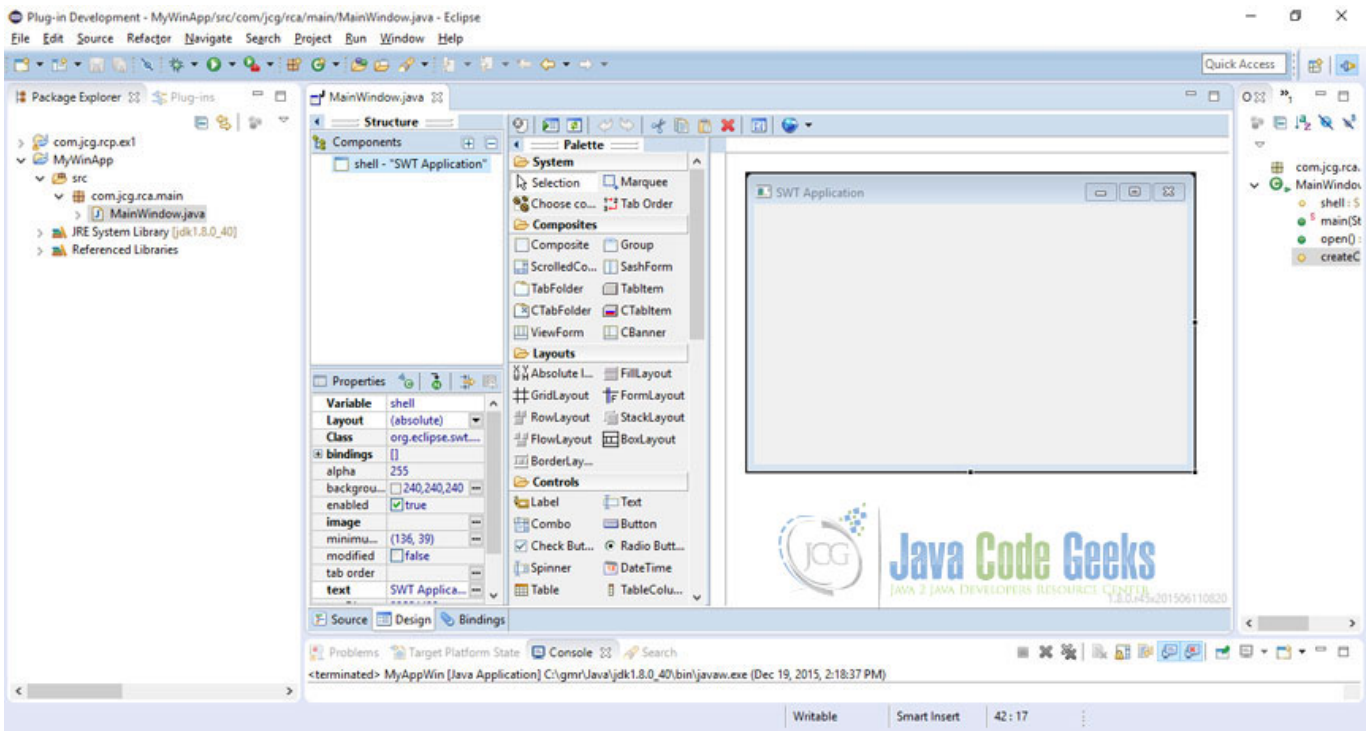

Figure 6.8: Design View

This application can be simply executed like any other java program with main method. Right click on the class name and *Run As - Java Application*

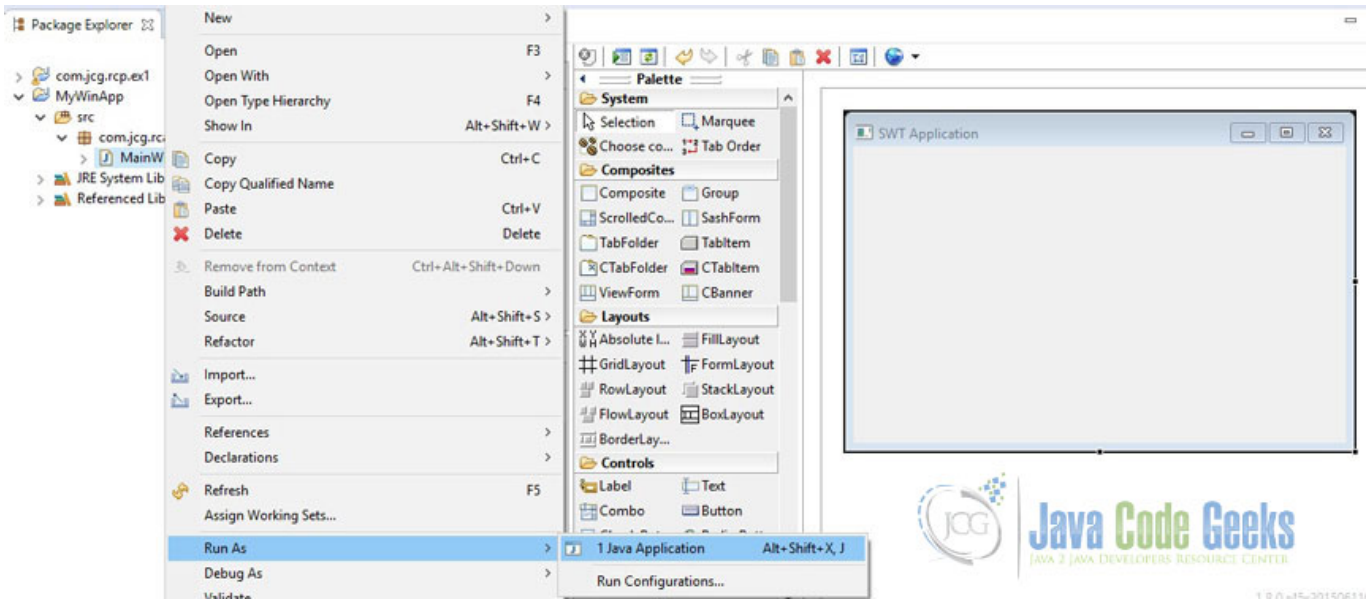

Figure 6.9: Run Application

As we have not yet added any other elements, you will see a simple window popping-up as a result of the execution.

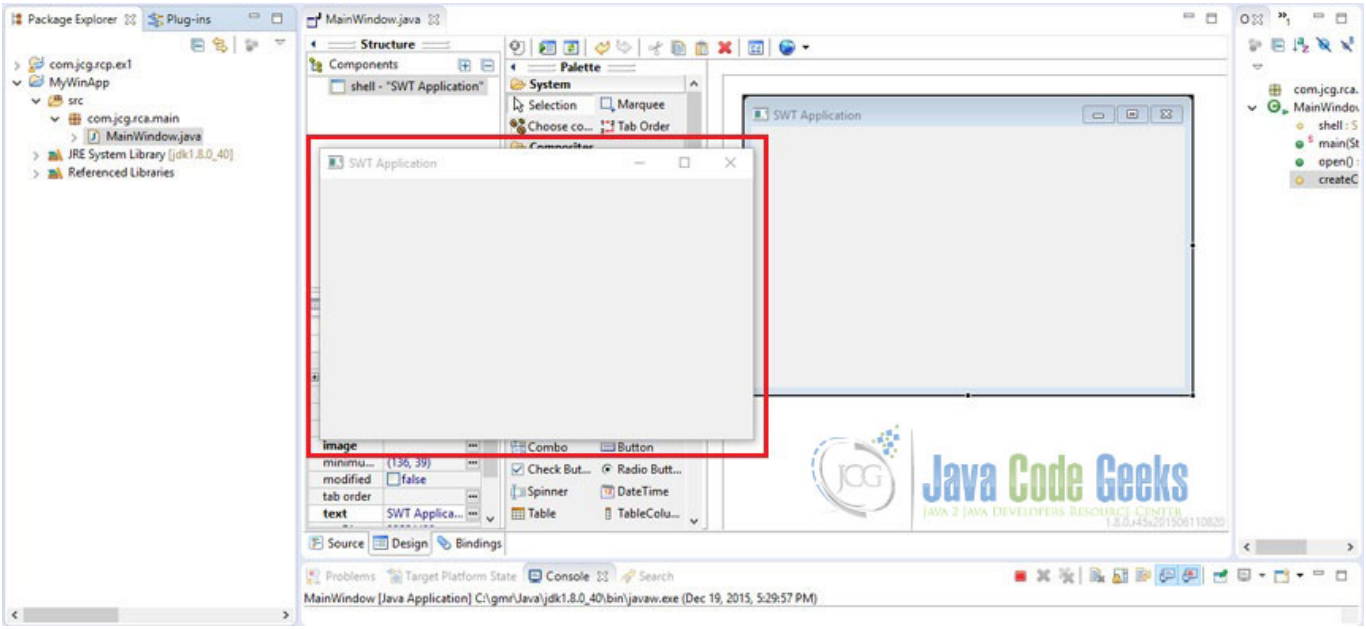

Figure 6.10: Executed Application

6.5 Components in the editor

As shown above, the editor is composed of the following major components:

- **Design View** - the main visual layout area.
- **Source View** - write code and review the generated code
- **Structure View** - composed of the **Component Tree** and the **Property Pane.**
  - **Component Tree** - shows the hierarchical relationship between all of the components.
  - **Property Pane** - displays properties and events of the selected components.
- **Palette** - provides quick access to toolkit-specific components.
- **Toolbar** - provides access to commonly used commands.
- **Context Menu** - provides access to commonly used commands.

6.6 Editor Features

The editor supports the following major features;

- **Bi-directional Code Generation** - read and write almost any format and reverse-engineer most hand-written code
- **Internationalization (i18n) / Localization** - externalize component strings, create and manage resource bundles.
- **Custom Composites & Panels** - create custom, reusable components.
- **Factories** - create custom factory classes and methods.
- **Visual Inheritance** - create visual component hierarchies.

- **Event Handling** - add event handlers to your components.
- **Menu Editing** - visually create and edit menubars, menu items and popup menus.
- **Morphing** - convert one component type into another.

6.7 Layouts in SWT

Layouts are non-visible widgets used to give GUI windows a specific look and it helps to control the position and size of children in a *Composite*.

To make sure the GUI application developed in one environment works perfect in another platform, Java provides a system of portable layout managers. We use these layout managers to specify rules and constraints for the layout of the UI in a way that will be portable.

Layout managers gives you the advantages as given below,

- Correctly positioned components that are independent of fonts, screen resolutions, and platform differences.
- Intelligent component placement for containers that are dynamically resized at runtime.
- Ease of translation. If a string increases in length after translation, the associated components stay properly aligned.

SWT Designer supports the following layout managers.

AbsoluteLayout	AbsoluteLayout or Null Layout helps to specify the exact position, the width and the height of components. In a generic environment where the size of the screens may vary, this layout manager should be avoided.
FillLayout	FillLayout is the simplest layout class. It lays out controls in a single row or column, forcing them to be the same size.
RowLayout	Puts the widgets in rows or columns and allows you to control the layout with options, e.g., wrap, spacing, fill and so on.
GridLayout	Arranges widgets in a grid.
FormLayout	Arranges the widgets with the help of the associated attachments.
StackLayout	A StackLayout object is a layout manager for a container. It treats each component in the container as a card. Only one card is visible at a time, and the container acts as a stack of cards.
BorderLayout	BorderLayout lays out a container, arranging and resizing its components to fit in five regions: north, south, east, west, and center. Each region may contain no more than one component, and is identified by a corresponding constant: NORTH, SOUTH, EAST, WEST, and CENTER.
BoxLayout	BoxLayout allows multiple components to be laid out either vertically or horizontally. The components will not wrap so, for example, a vertical arrangement of components will stay vertically arranged when the frame is resized. Nesting multiple panels with different combinations of horizontal and vertical gives an effect similar to GridBagLayout, without the complexity.
FlowLayout	A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel. It will arrange buttons left to right until no more buttons fit on the same line.

6.8 New UI Page

We will now design a new Login UI page using Window Builder. For this normal size screen, we will continue with the default (absolute) layout. We are going to have an image, two labels, one text field, one password field and a button on the screen.

To display image use CLabel widget. CLabel supports aligned text and/or an image and different border styles.

As shown below, click *CLabel* once and keep your cursor on the screen and click. Now, the *CLabel* is placed on the screen.

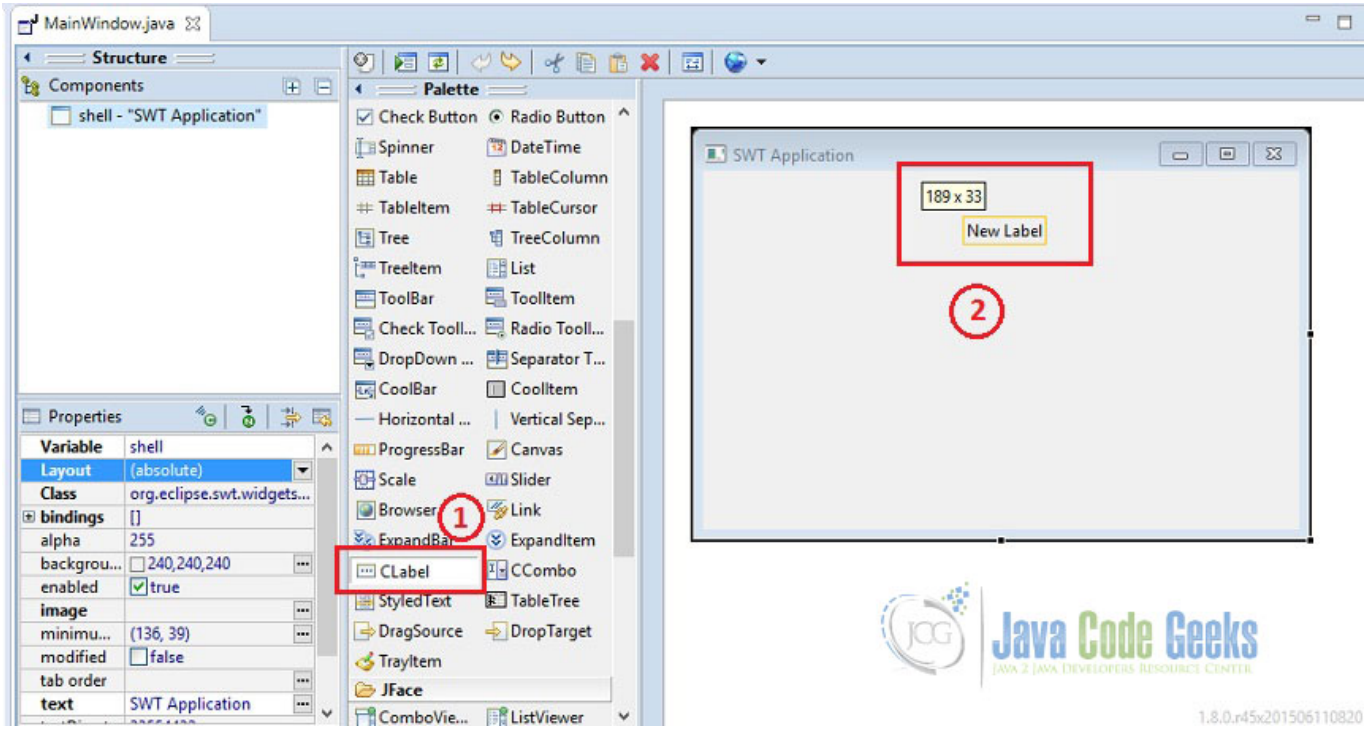

Figure 6.11: New Login UI

Let us attach an image with *CLabel*. For this, you need to have an image in the folder where your *MainWindow* source file is placed. For this example, I have used eclipse logo.

Click on the *CLabel* and then, in the *Properties* window select *image*.

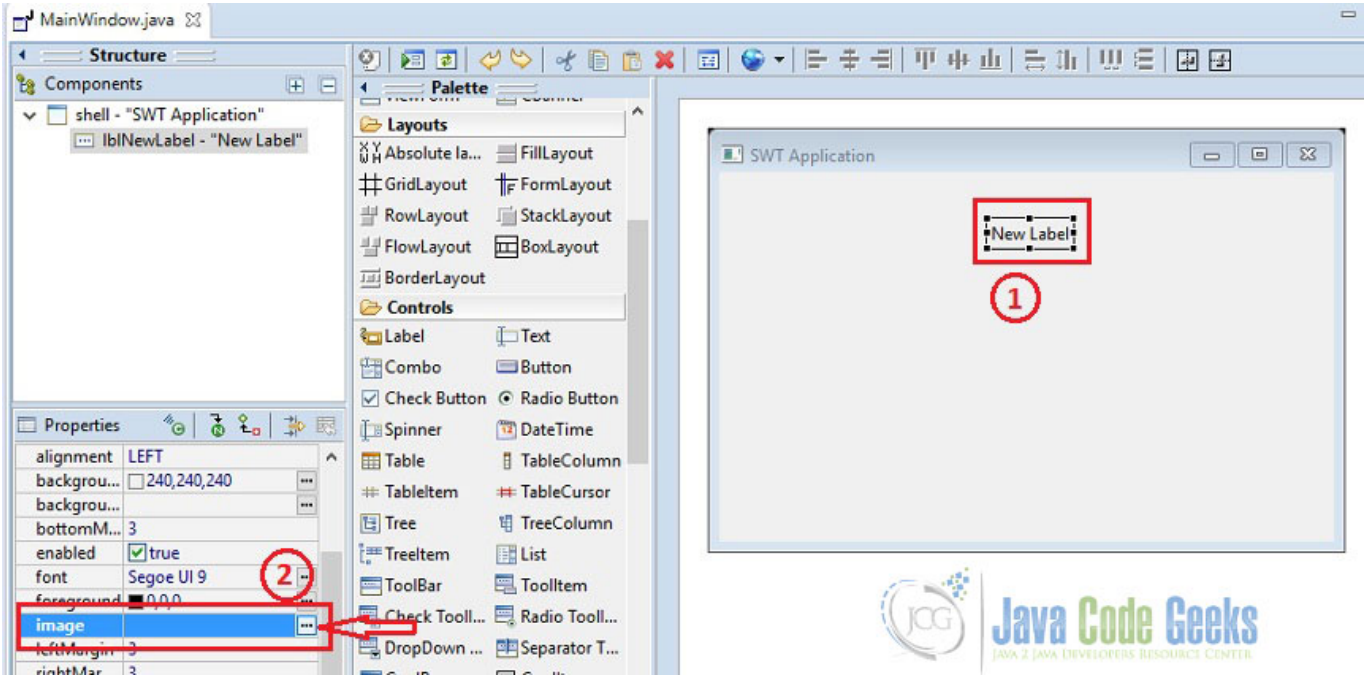

Figure 6.12: CLabel Image

You will now see the Image chooser window pops up. Select *Classpath resource* option and navigate to the image file, select it and then click *OK*.

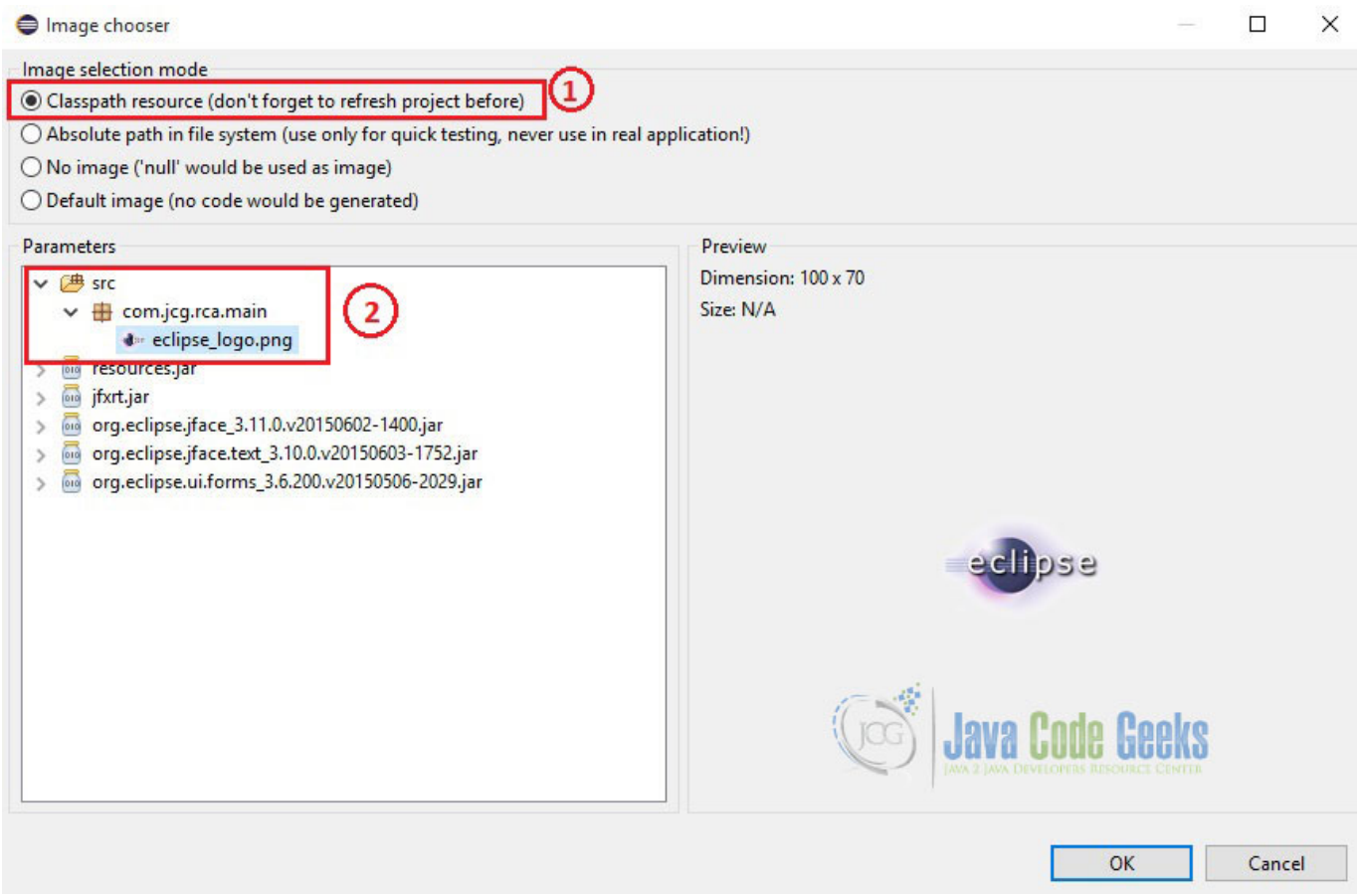

Figure 6.13: Select Image

Adjust the field bounds according to the size of the logo so that the image is visible on the screen.

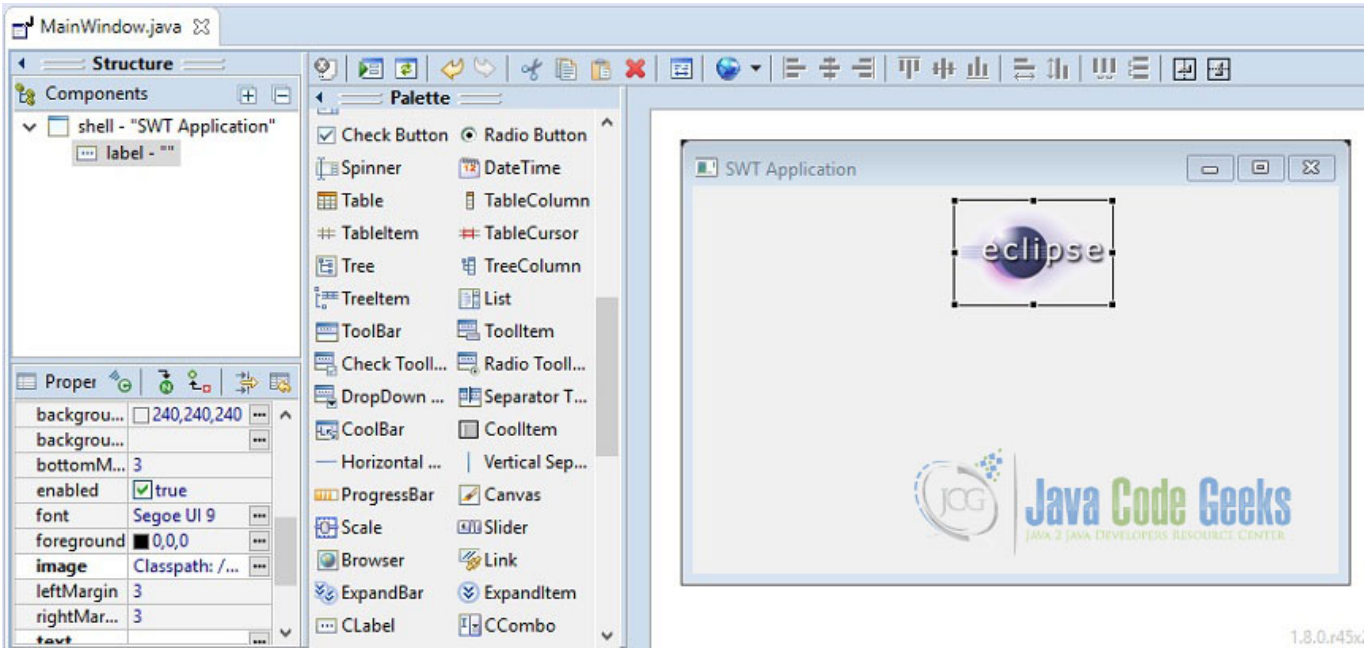

Figure 6.14: Image Attached

Similarly, add Labels, Text Fields and a Button. Finally the screen will be looking like the one shown below.

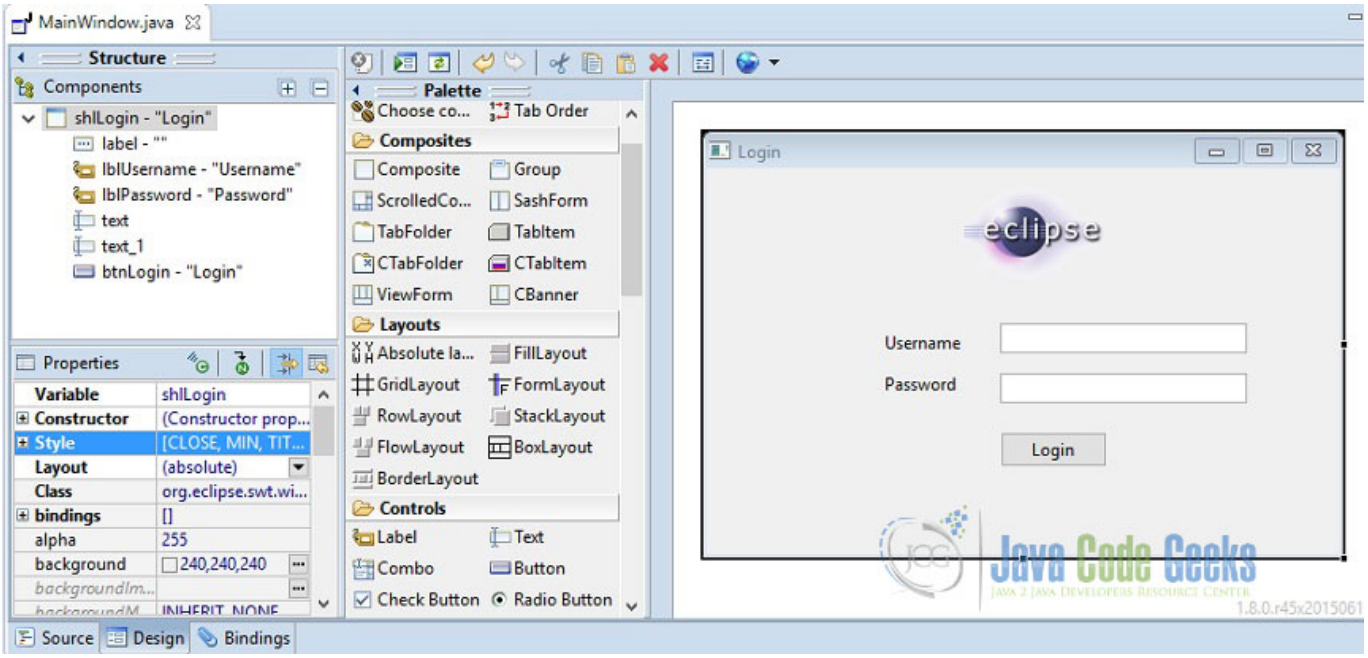

Figure 6.15: Login UI Page

To test this design, right click on the window and select *Test/Preview* from the popup menu.

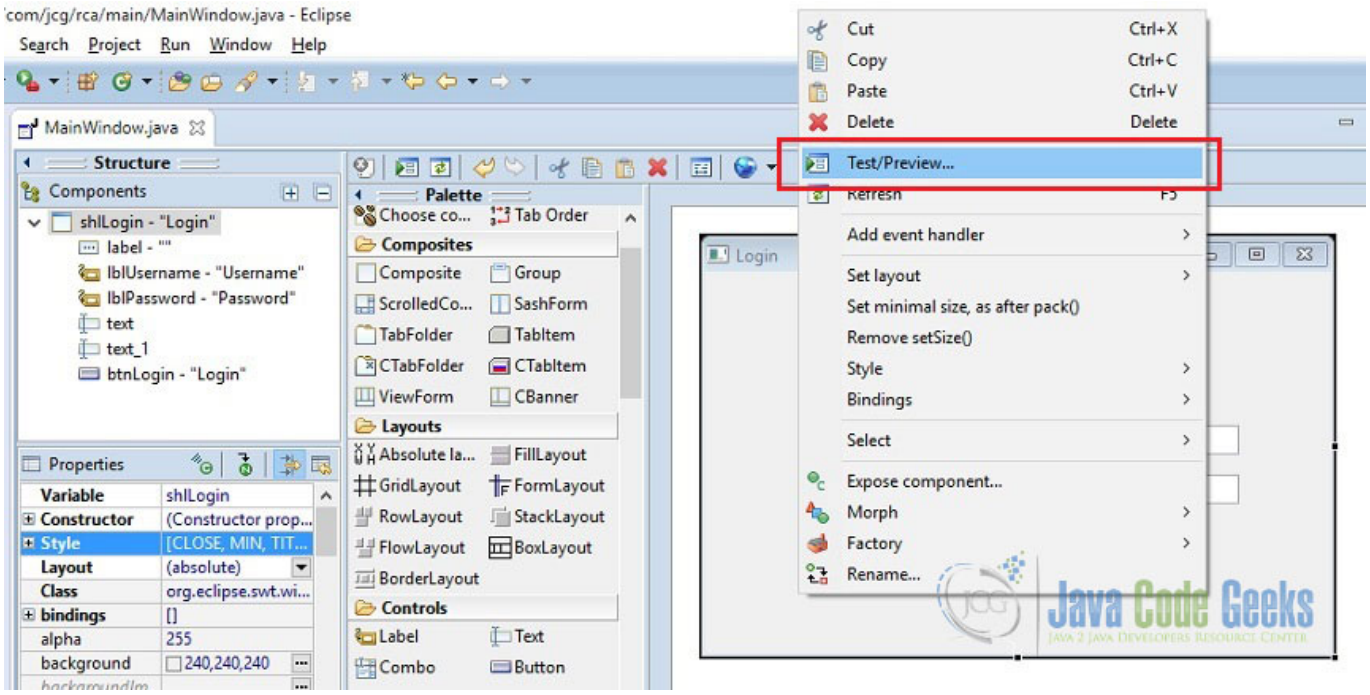

Figure 6.16: Test GUI

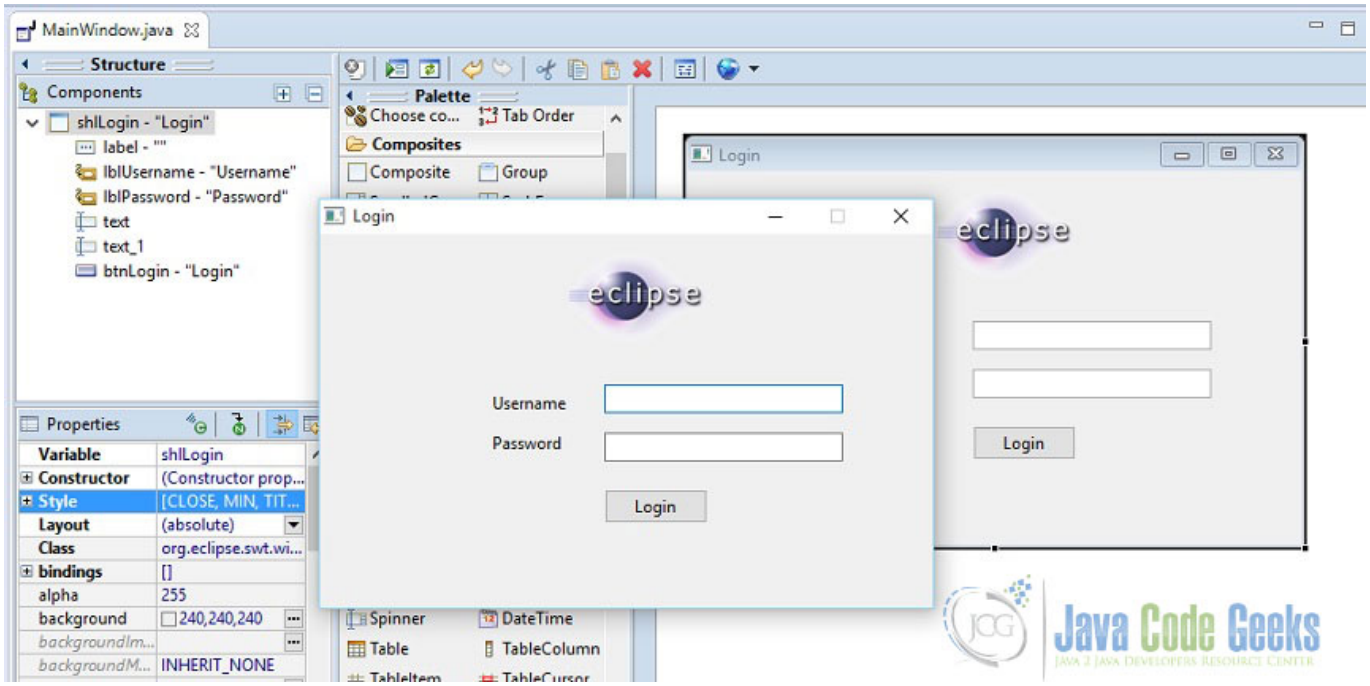

Figure 6.17: Test Preview

6.9 Source View

Click *Source* tab to see the code generated by the IDE. Single line of code in this was not written manually.

```
protected void createContents() {
    shlLogin = new Shell(SWT.CLOSE | SWT.TITLE | SWT.MIN);
    shlLogin.setSize(450, 300);
    shlLogin.setText("Login");

    CLabel label = new CLabel(shlLogin, SWT.NONE);
    label.setImage(SWTResourceManager.getImage(MainWindow.class, "/com/jcg/rca/main/eclipse_logo.png"));
    label.setBounds(176, 10, 106, 70);
    label.setText("");

    Label lblUsername = new Label(shlLogin, SWT.NONE);
    lblUsername.setBounds(125, 115, 55, 15);
    lblUsername.setText("Username");

    Label lblPassword = new Label(shlLogin, SWT.NONE);
    lblPassword.setBounds(125, 144, 55, 15);
    lblPassword.setText("Password");

    text = new Text(shlLogin, SWT.BORDER);
    text.setBounds(206, 109, 173, 21);

    text_1 = new Text(shlLogin, SWT.BORDER);
    text_1.setBounds(206, 144, 173, 21);

    Button btnLogin = new Button(shlLogin, SWT.NONE);
    btnLogin.setBounds(206, 185, 75, 25);
    btnLogin.setText("Login");
```

Figure 6.18: Source View

6.10 Button Listener

Attach listener with the button to validate field entries. Refer the source code of the main file given below.

MainWindow.java

```
package com.jcg.rca.main;

import org.eclipse.swt.SWT;
import org.eclipse.swt.custom.CLabel;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.Event;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Listener;
import org.eclipse.swt.widgets.MessageBox;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;
import org.eclipse.wb.swt.SWTResourceManager;

public class MainWindow {

        protected Shell shlLogin;
        private Text userNameTxt;
        private Text passwordTxt;

        private String userName = null;
        private String password = null;

        /**
         * Launch the application.
```

```
	 *
	 * @param args
	 */
	public static void main(String[] args) {
		try {
			MainWindow window = new MainWindow();
			window.open();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	/**
	 * Open the window.
	 */
	public void open() {
		Display display = Display.getDefault();
		createContents();
		shlLogin.open();
		shlLogin.layout();
		while (!shlLogin.isDisposed()) {
			if (!display.readAndDispatch()) {
				display.sleep();
			}
		}
	}

	/**
	 * Create contents of the window.
	 */
	protected void createContents() {
		shlLogin = new Shell(SWT.CLOSE | SWT.TITLE | SWT.MIN);
		shlLogin.setSize(450, 300);
		shlLogin.setText("Login");

		CLabel label = new CLabel(shlLogin, SWT.NONE);
		label.setImage(SWTResourceManager.getImage(MainWindow.class, "/com/jcg/rca/ ↩
		    main/eclipse_logo.png"));
		label.setBounds(176, 10, 106, 70);
		label.setText("");

		Label lblUsername = new Label(shlLogin, SWT.NONE);
		lblUsername.setBounds(125, 115, 55, 15);
		lblUsername.setText("Username");

		Label lblPassword = new Label(shlLogin, SWT.NONE);
		lblPassword.setBounds(125, 144, 55, 15);
		lblPassword.setText("Password");

		userNameTxt = new Text(shlLogin, SWT.BORDER);
		userNameTxt.setBounds(206, 109, 173, 21);

		passwordTxt = new Text(shlLogin, SWT.BORDER | SWT.PASSWORD);
		passwordTxt.setBounds(206, 144, 173, 21);

		Button btnLogin = new Button(shlLogin, SWT.NONE);
		btnLogin.setBounds(206, 185, 75, 25);
		btnLogin.setText("Login");

		btnLogin.addListener(SWT.Selection, new Listener() {
			public void handleEvent(Event event) {
```

```
                                userName = userNameTxt.getText();
                                password = passwordTxt.getText();

                                if (userName == null || userName.isEmpty() || password ==  ↩
                                    null || password.isEmpty()) {
                                        String errorMsg = null;
                                        MessageBox messageBox = new MessageBox(shlLogin,  ↩
                                            SWT.OK | SWT.ICON_ERROR);

                                        messageBox.setText("Alert");
                                        if (userName == null || userName.isEmpty()) {
                                                errorMsg = "Please enter username";
                                        } else if (password == null || password.isEmpty())  ↩
                                            {
                                                errorMsg = "Please enter password";
                                        }
                                        if (errorMsg != null) {
                                                messageBox.setMessage(errorMsg);
                                                messageBox.open();
                                        }
                                } else {
                                        MessageBox messageBox = new MessageBox(shlLogin,  ↩
                                            SWT.OK | SWT.ICON_WORKING);
                                        messageBox.setText("Info");
                                        messageBox.setMessage("Valid");
                                        messageBox.open();
                                }
                        }
                });

        }

}
```

6.11 Conclusion

From this example, we learned how quickly a Java GUI application can be developed using Eclipse Window Builder. WindowBuilder Engine provides a rich API for creating UI designers. It supports Java-based UI frameworks such as Swing, SWT/RCP, eRCP, GWT etc. It also supports XML-based UI frameworks like XWT, GWT UiBinder, Android etc.

6.12 Download the Code Project

This was a Tutorial about Eclipse Window Builder for GUI Creation.

Download You can download the full source code of this example here: **WindowBuilderExample**

Chapter 7

Eclipse Rich Client Platform (RCP) Tutorial

7.1 Introduction

In this example, we will learn how to use Eclipse Rich Client Platform (RCP) to develop stand alone applications written in Java and built on top of Eclipse platform technologies. Using RCP, programmers can develop customized window applications, menus, tool bars, palettes, wizards and other specialized features.

7.1.1 What is Rich Client Platform?

The minimal set of plug-ins needed to build a rich client application is collectively known as the Rich Client Platform.This is a platform for building client applications with rich functionality.

7.1.2 Why Eclipse RCP?

RCP is a collection of lower-level frameworks. It is a well-suited platform for Java based desktop applications. The basic advantage of Eclipse RCP is module re-usability. Not just class re-usability but full component reuse. Eclipse architecture make this very much possible than ever before.

Applications written with RCP are completely portable and will run equally well on Windows, Mac or Linux. RCP development has been made simple with Eclipse 4.x API compared to Eclipse 3.

This example is tested with *Eclipse (Mars) for RCP and RAP Developers* IDE. Before we start, please make sure you have the tools mentioned below are installed in your system. This example assumes basic programming knowledge in Java programming language using Eclipse IDE is a plus.

System requirements

Tools required to run this example are:

Eclipse

Download *Eclipse for RCP and RAP Developers* from here. Please refer the picture given below to identify the correct IDE.

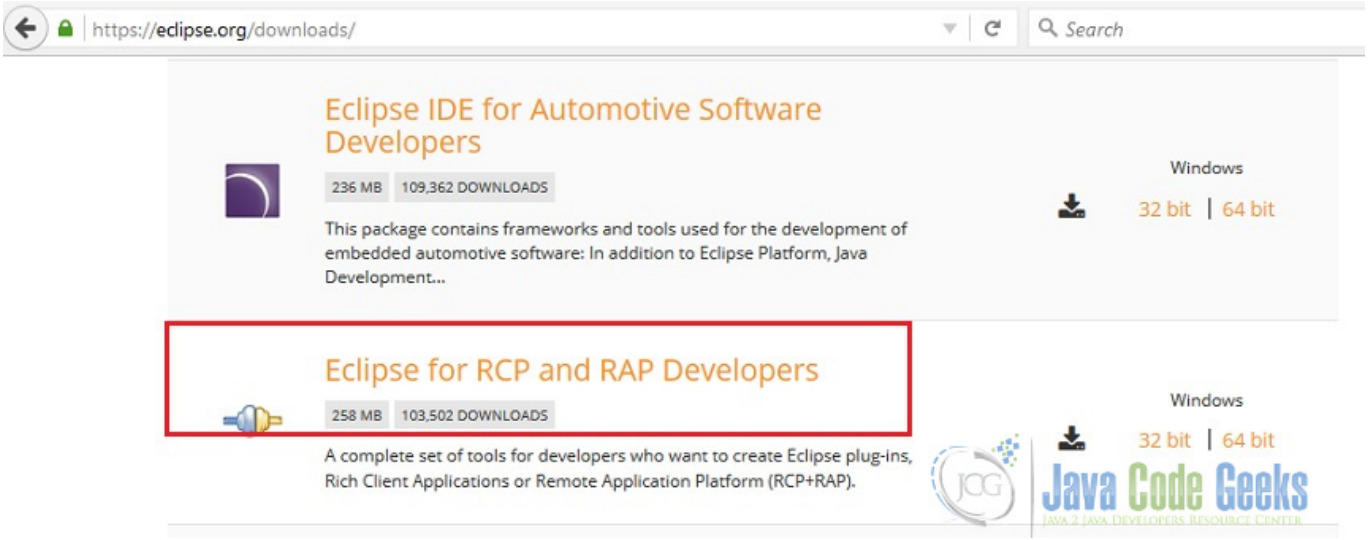

Figure 7.1: Eclipse IDE for RCP and RAD

Java

Download Java SE 7 or above from here

Let's start:

7.2 Open New Project

This needs to be created as new *Eclipse 4 Application Project*. For that open *File - New - Other*

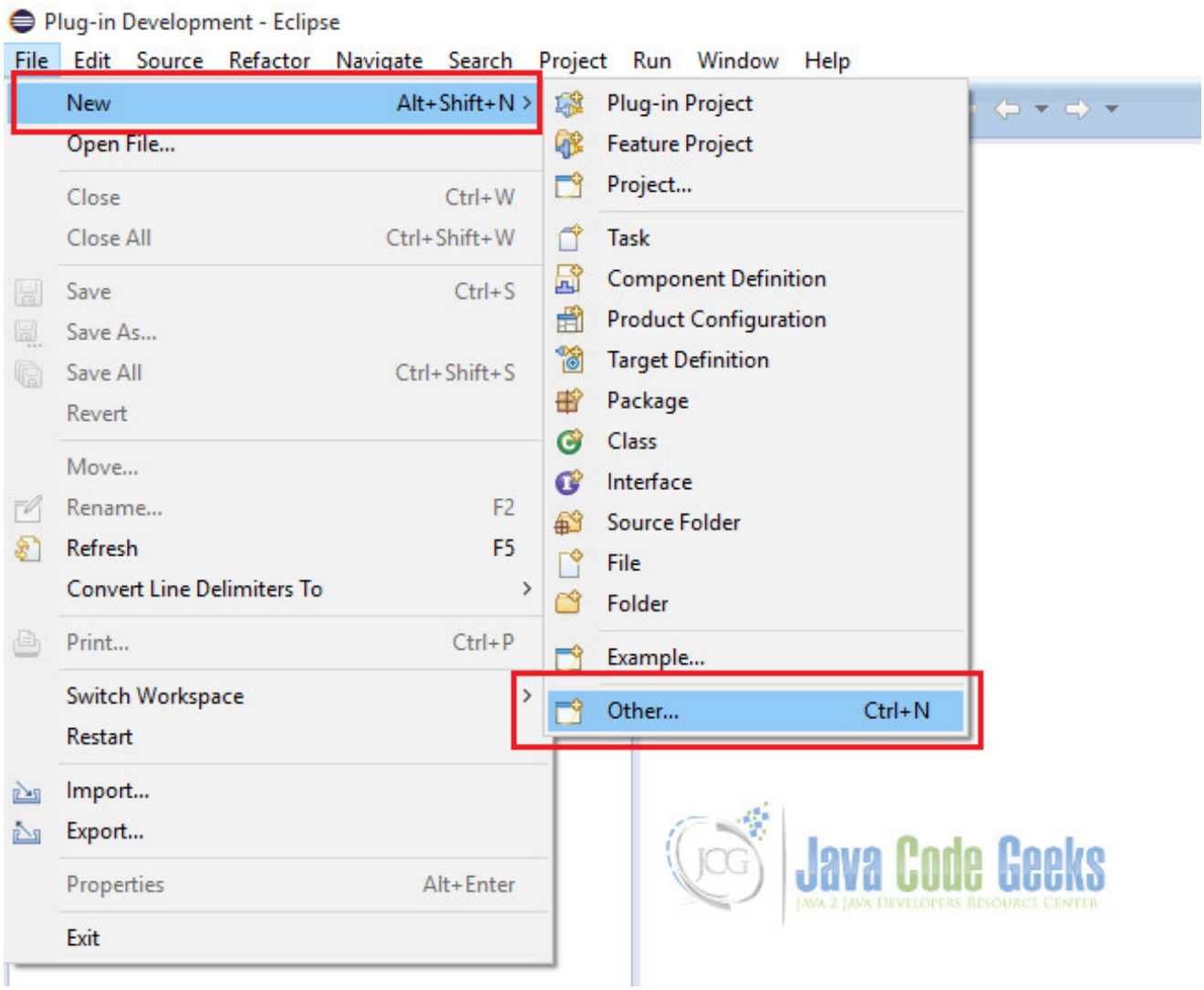

Figure 7.2: Open IDE

7.3 Eclipse 4 Application Project

Select *Eclipse 4 Application Project* and click Next

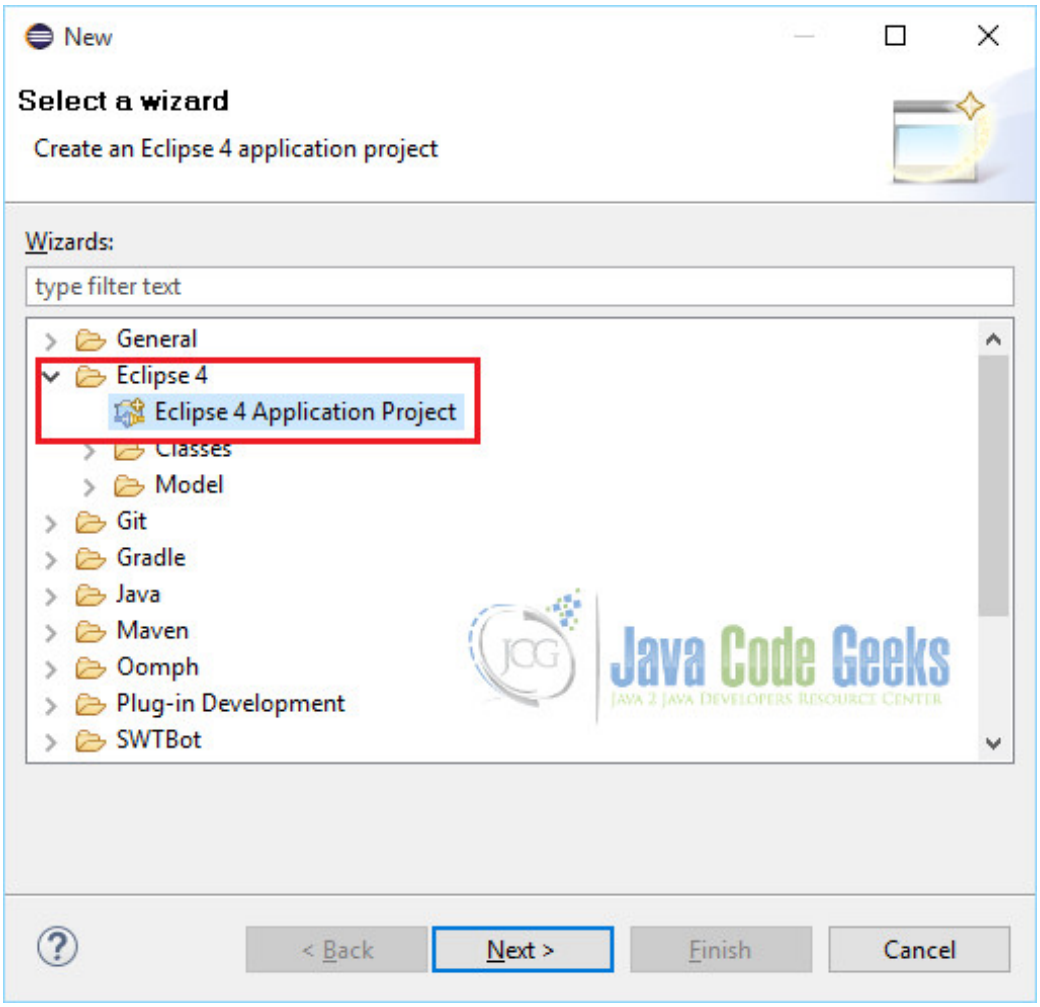

Figure 7.3: Eclipse 4 Application Project

7.3.1 Enter Project Name

Leave other default values and click *Next*

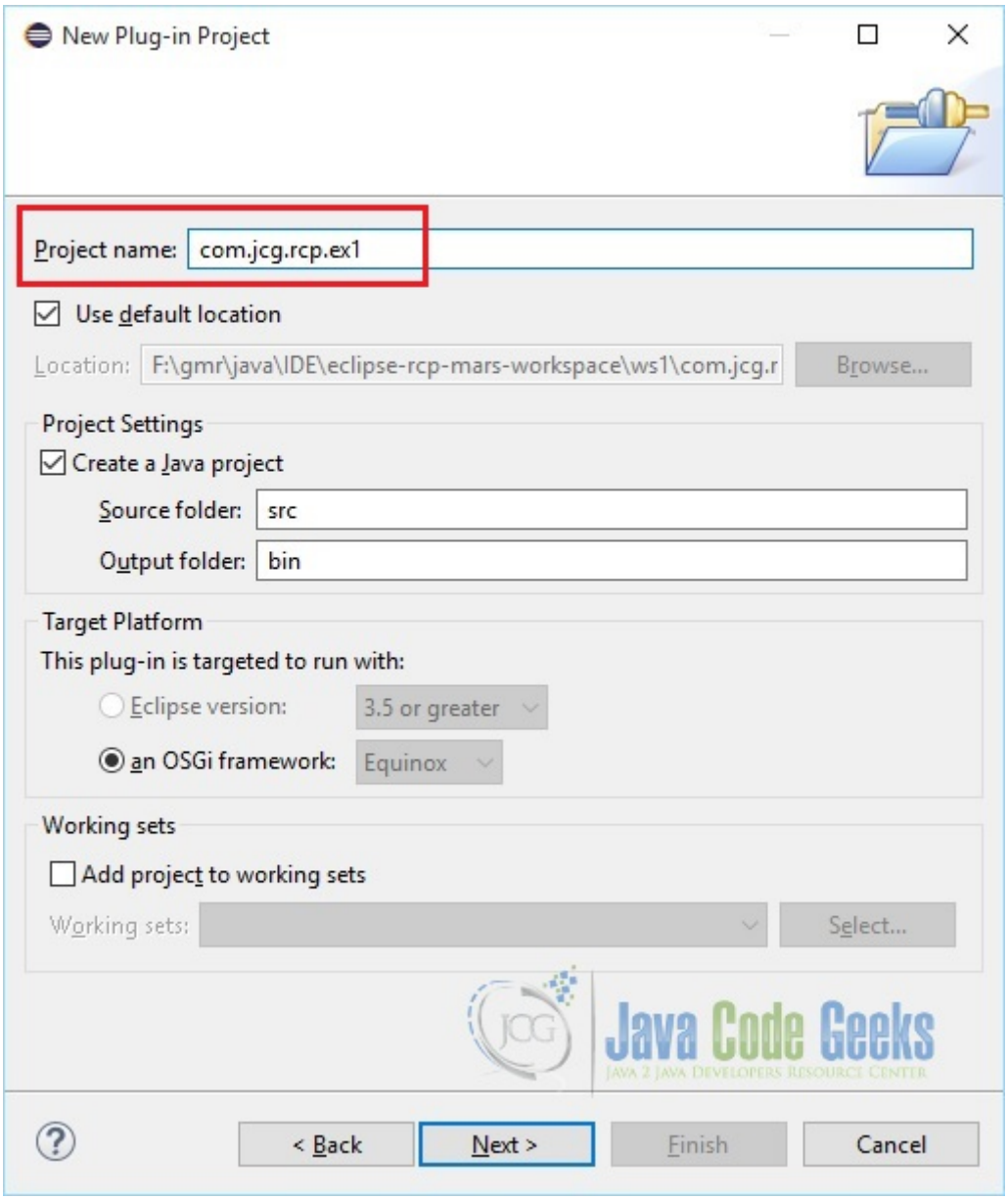

Figure 7.4: Project Name

7.4 Project Properties

Leave other default values and click *Next*

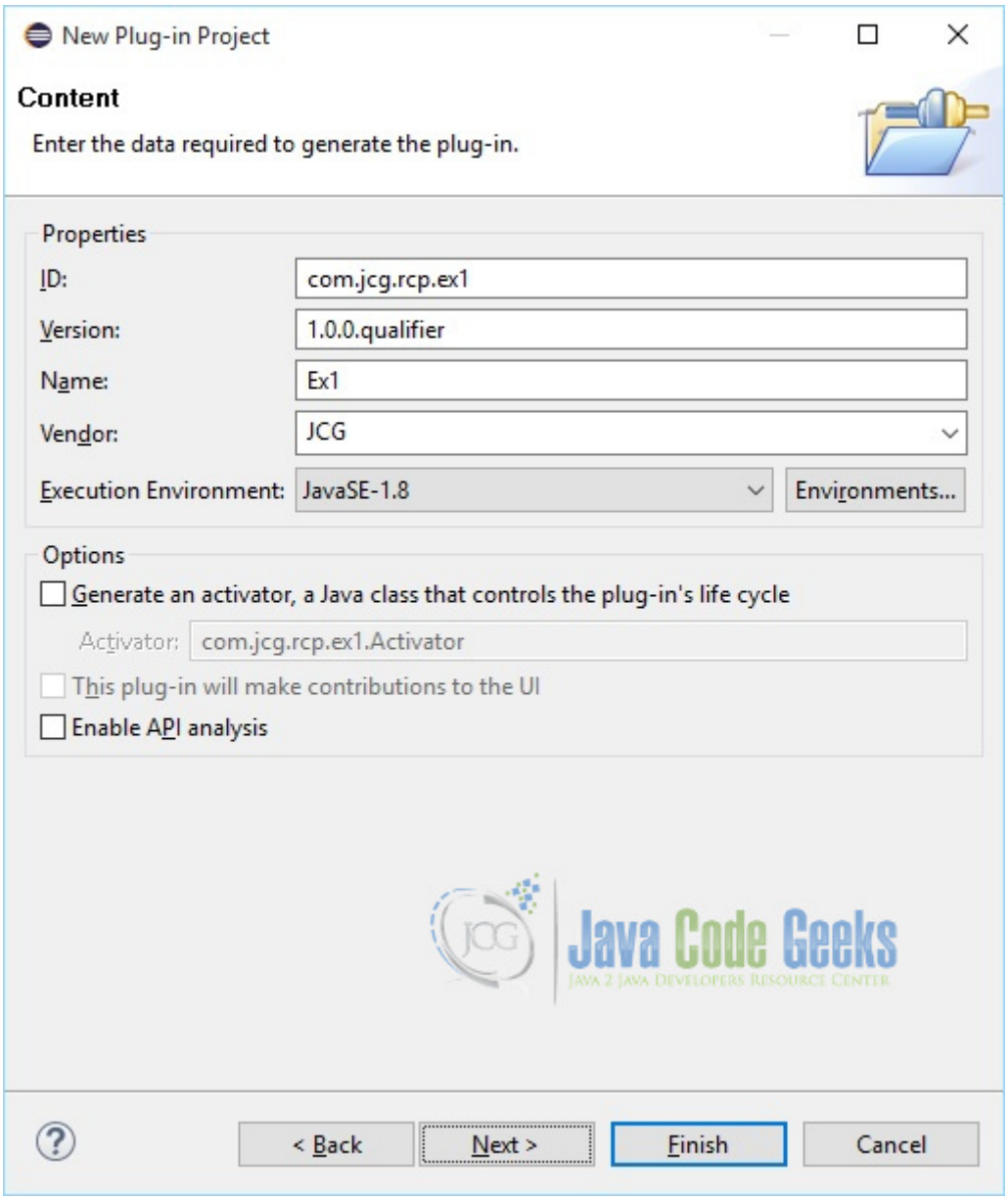

Figure 7.5: Project Properties

7.5 Project Configuration

Make sure *Create sample content(parts, menu etc.)* check box is checked and click *Finish*

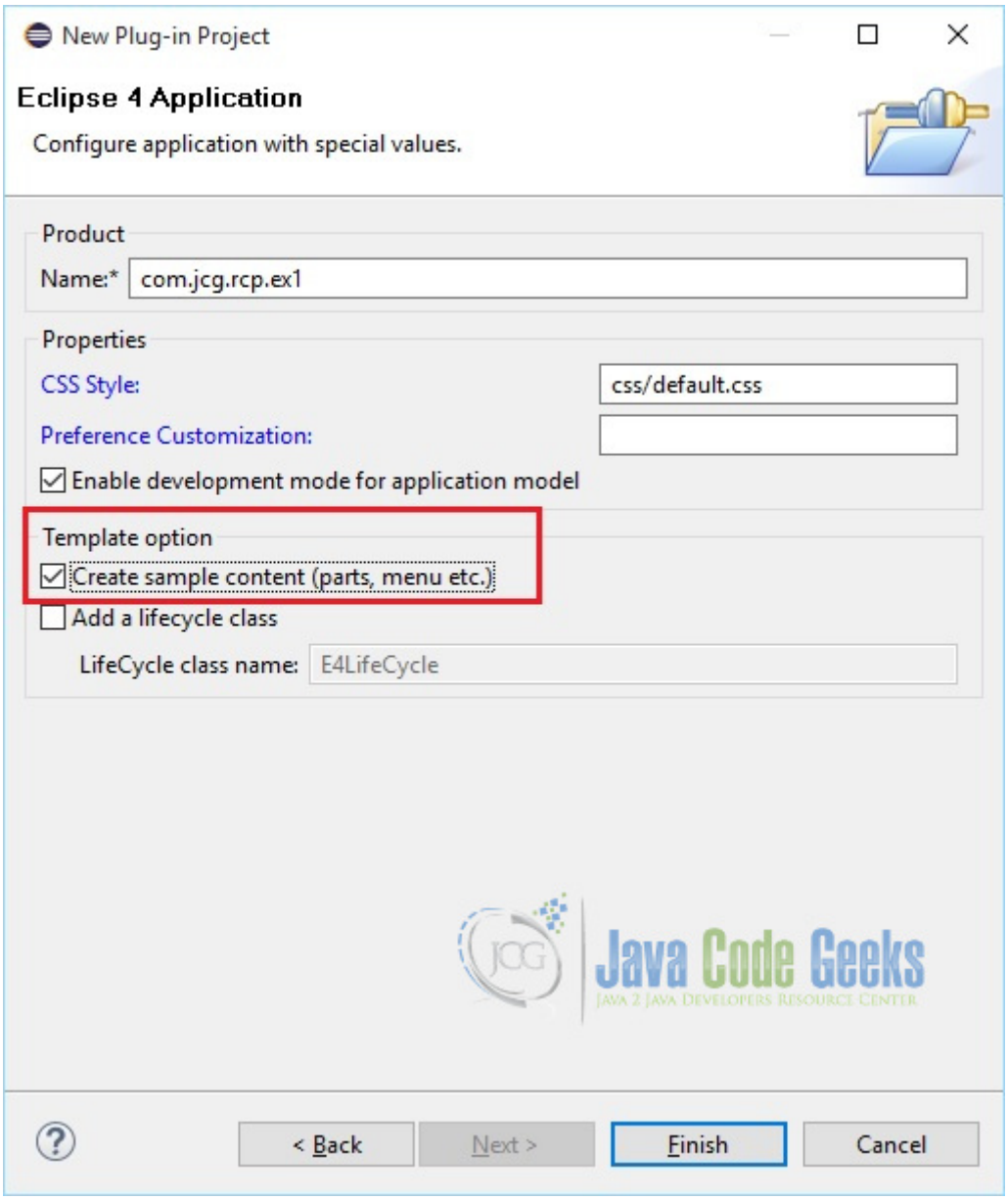

Figure 7.6: Application Configuration

7.6 RCP Application

Yes, a basic RCP application using built-in template has been created.

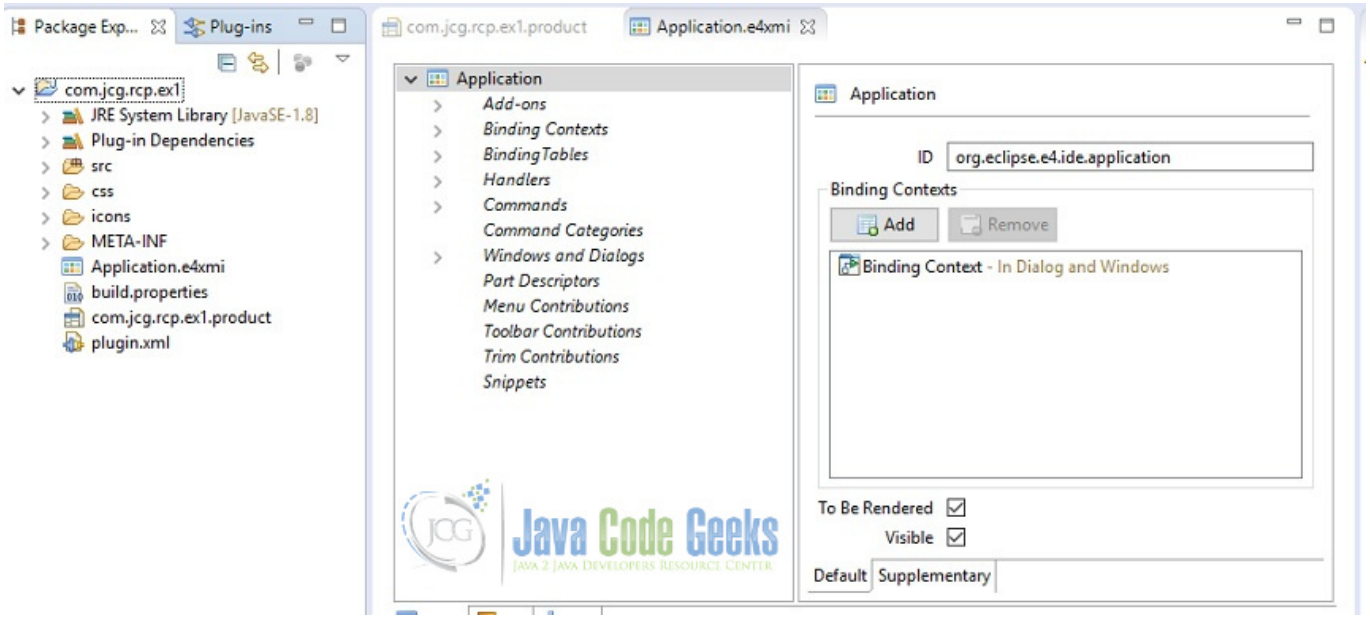

Figure 7.7: Basic Application

7.7 Structure of Eclipse 4 RCP application

Eclipse Version 4 has introduced many new concepts and APIs, such as the application model, dependency injection (DI), and CSS styling. The structure of the application is described via the application model in the Application.e4xmi file. Views, menus, and toolbars of your application can be defined in this file.

Open Application.e4xmi file and navigate to *Part - Sample Part* as shown in the picture below. This is the default class gets executed and shown on the window as view part.

As per Eclipse 4 application model, Parts are the UI components which can be used to navigate and modify data. All Parts can be stacked next to each other or it can be positioned.

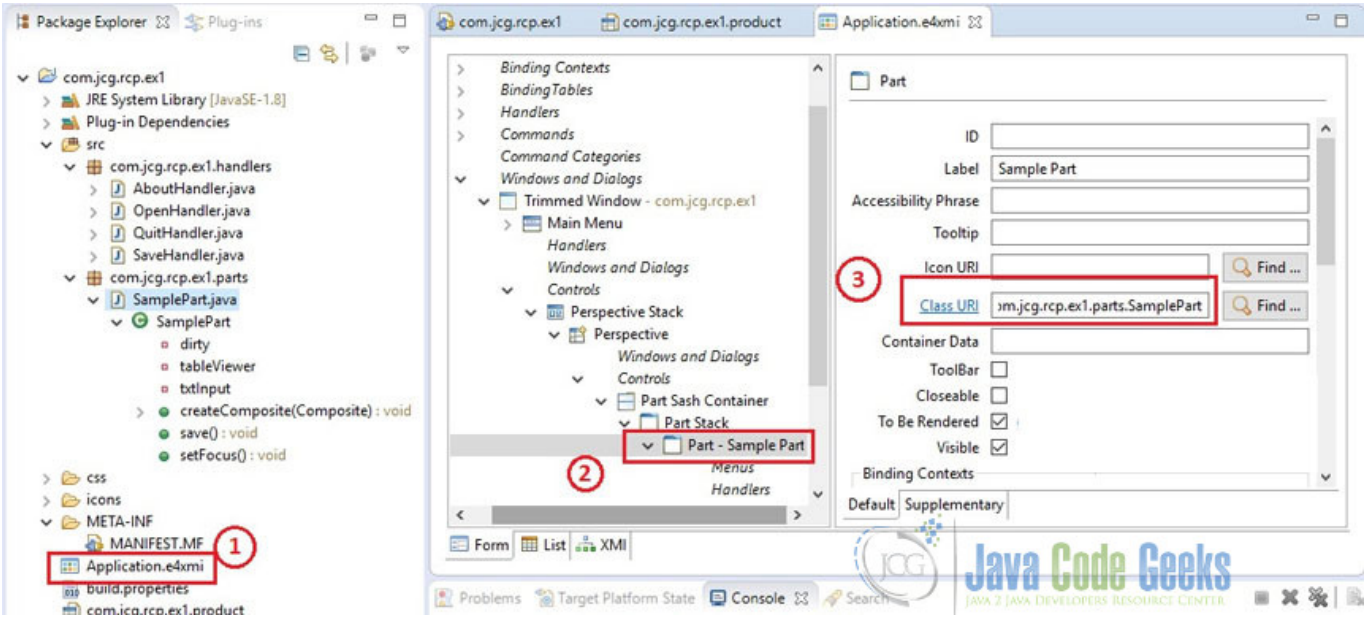

Figure 7.8: Structure of the Application

7.8 Run RCP Application

We will see how to run this application before we add our own components in this application. To run, double click on the product file as shown in the picture below (1). Launch your Eclipse application by pressing on *Launch an Eclipse application* hyperlink (2) from the *Overview* tab.

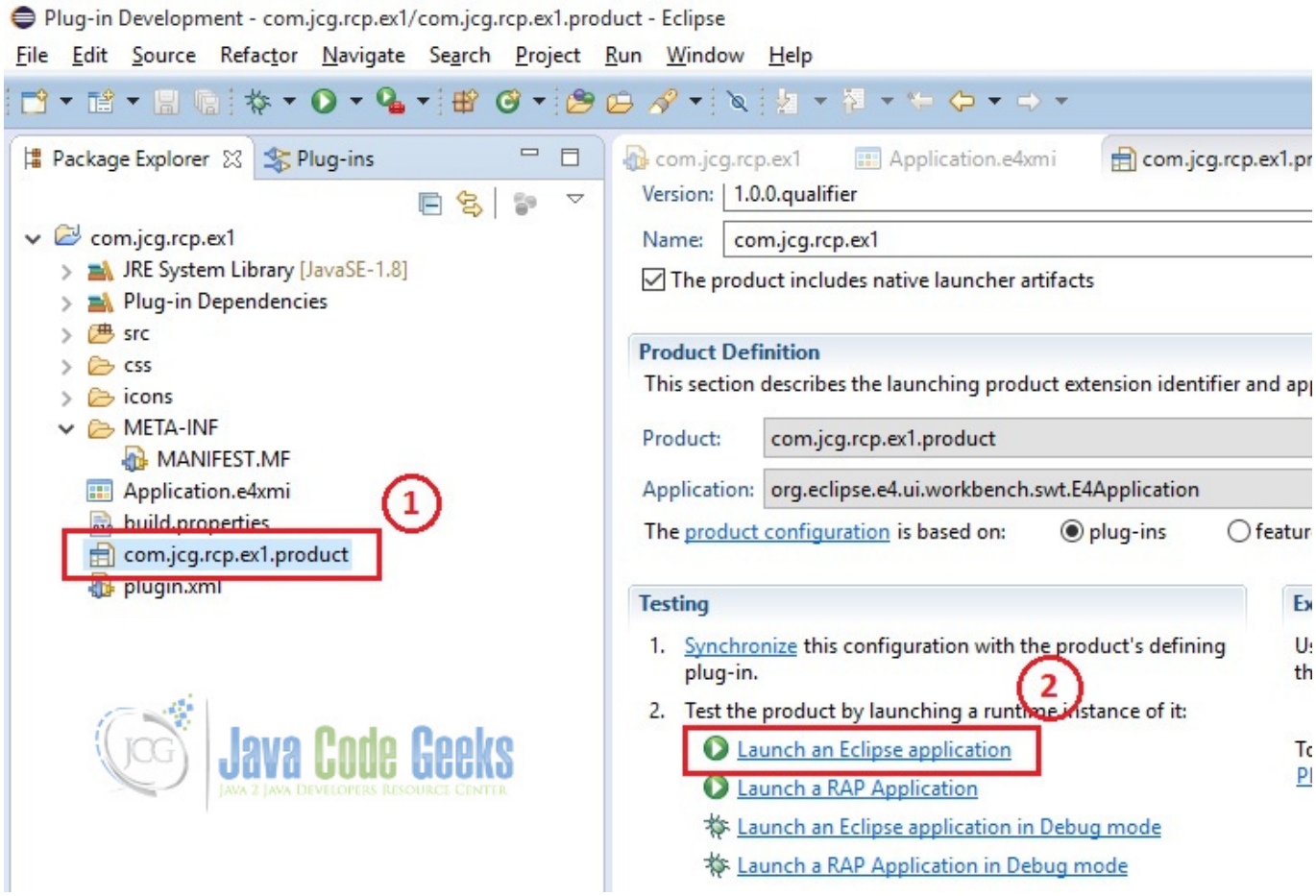

Figure 7.9: Run Application

It can also be executed by selecting *Run As* --→ *Eclipse Application* as depicted below.

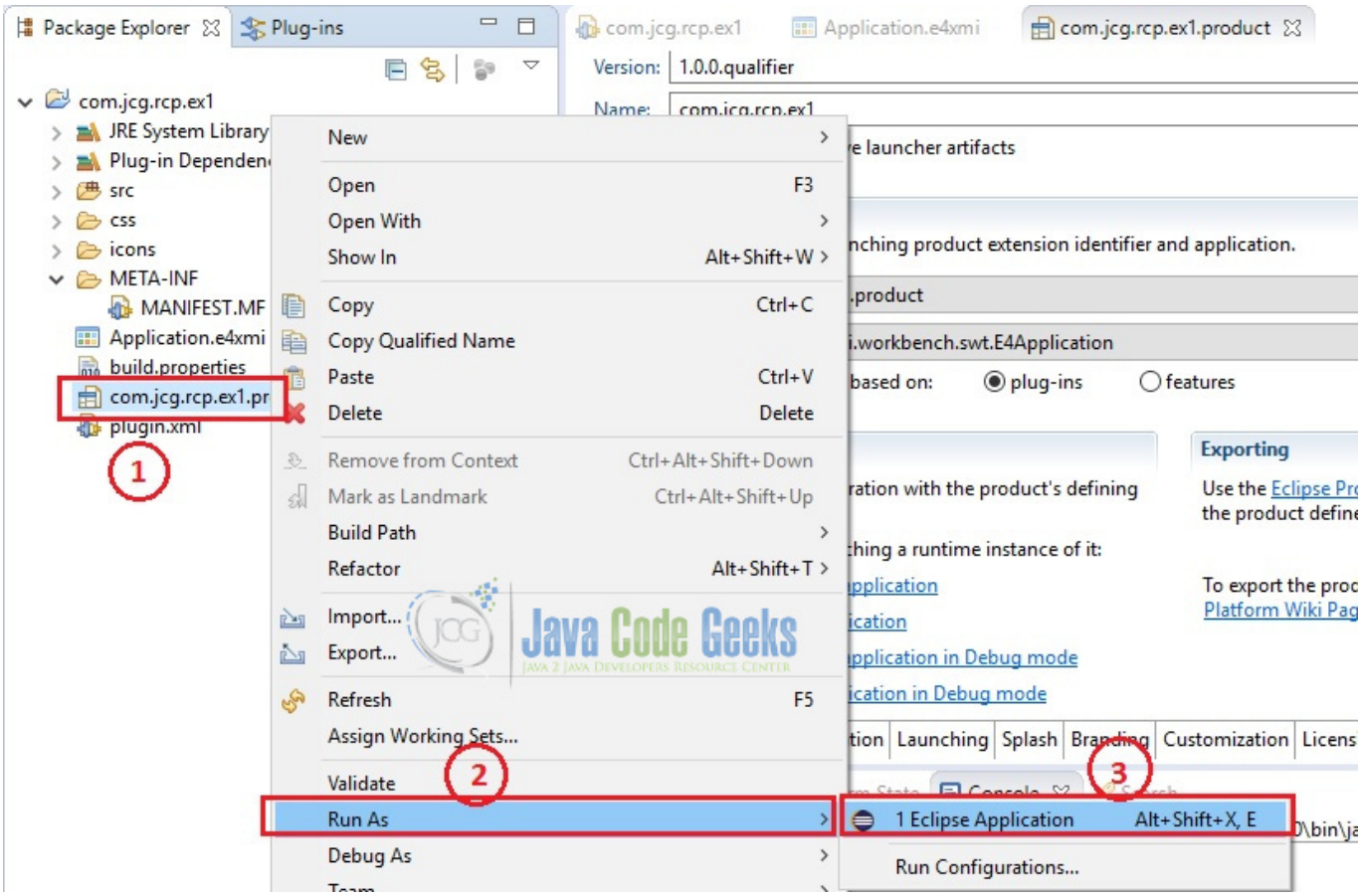

Figure 7.10: Run As Eclipse Application

7.9 Stand-alone application

Separate desktop application is up and running.

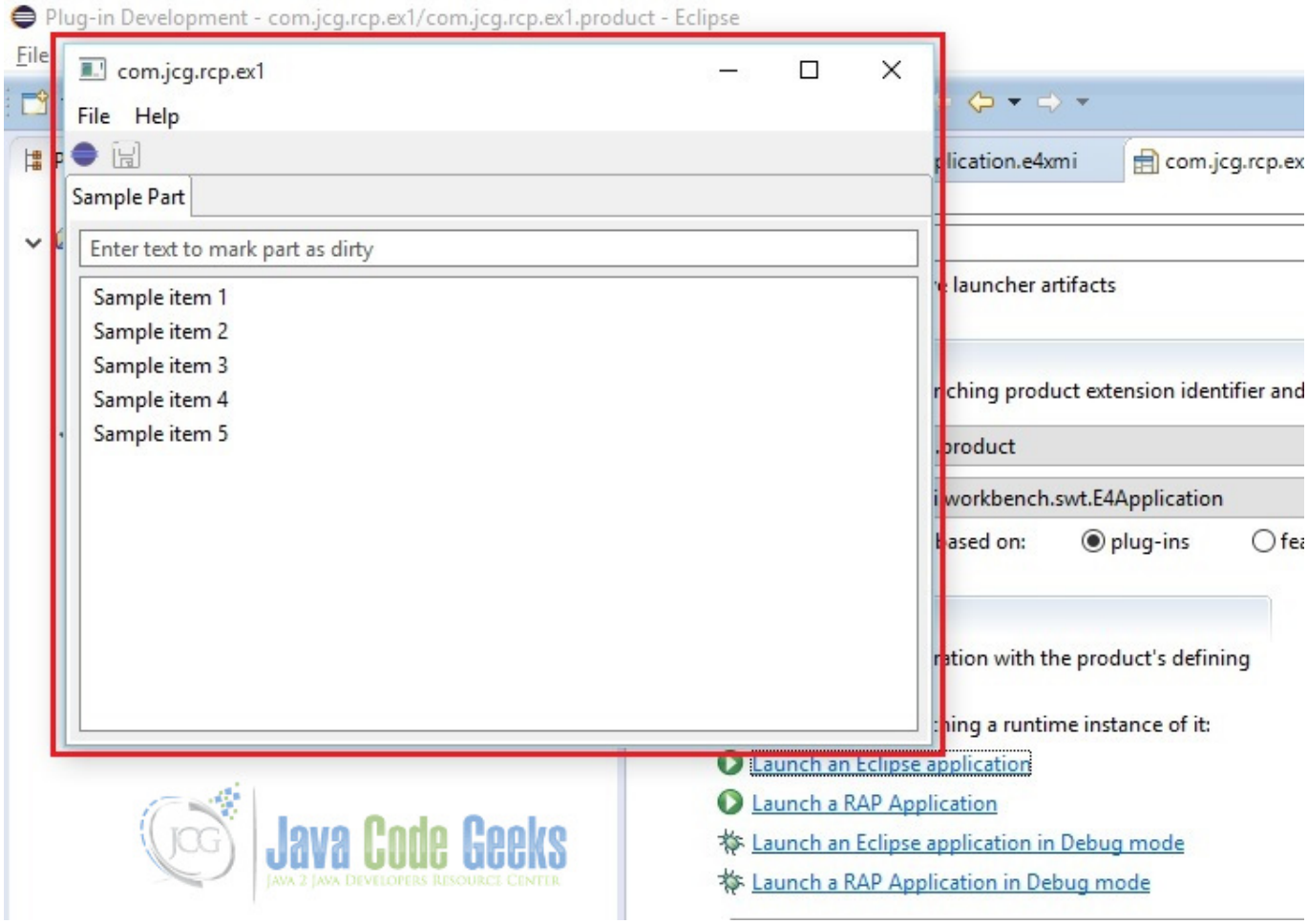

Figure 7.11: Running Application

7.10 Create New Part

Now, we will see how to add our own Part in the application. As shown in the picture, click on the product name - New - Other:

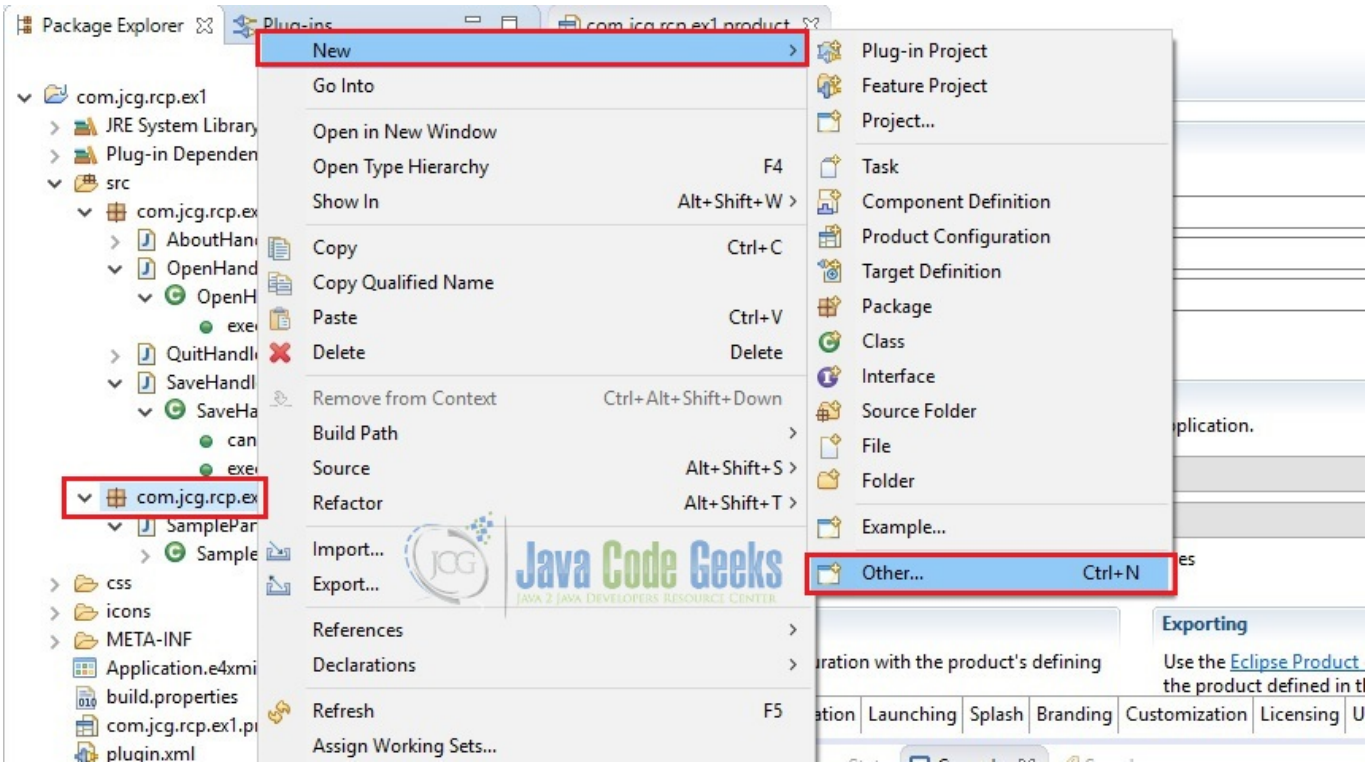

Figure 7.12: Add New Part

Select *New Part Class* and click Next

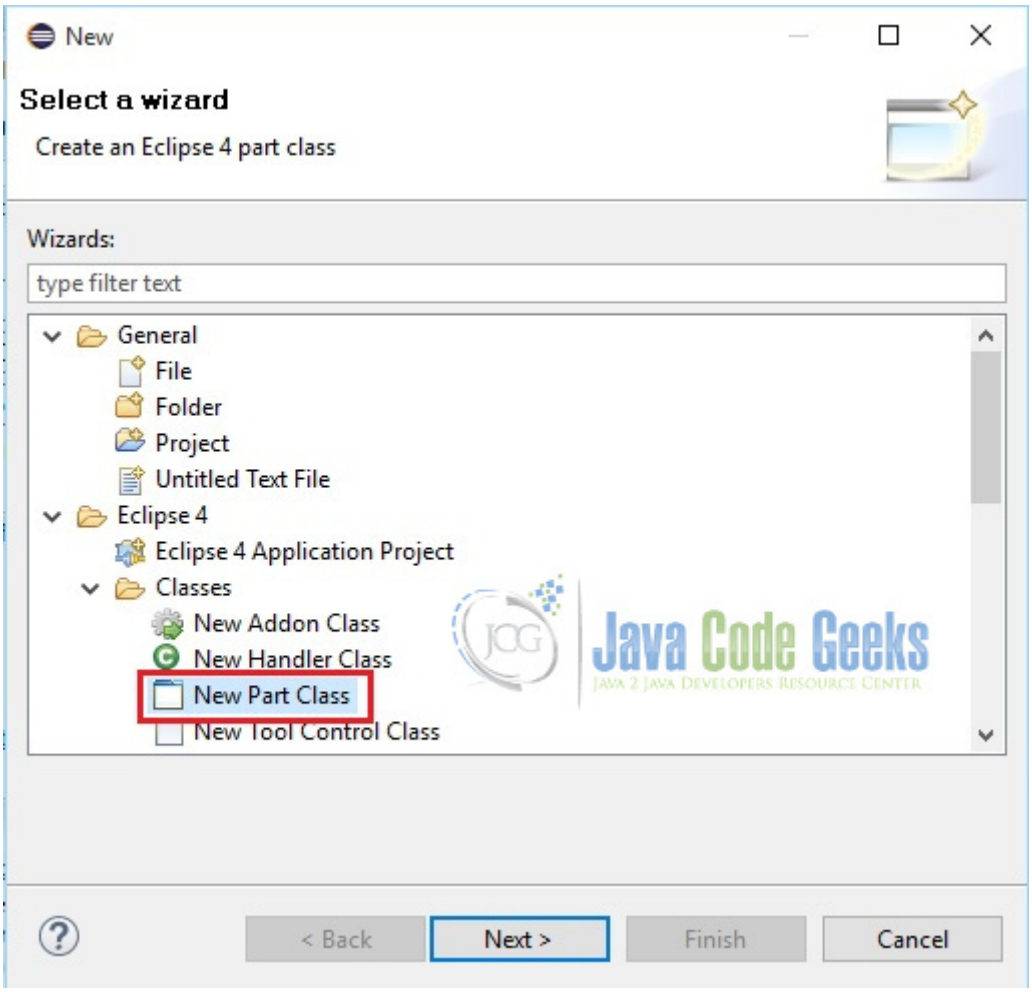

Figure 7.13: New Part Class

Enter class name and click Finish

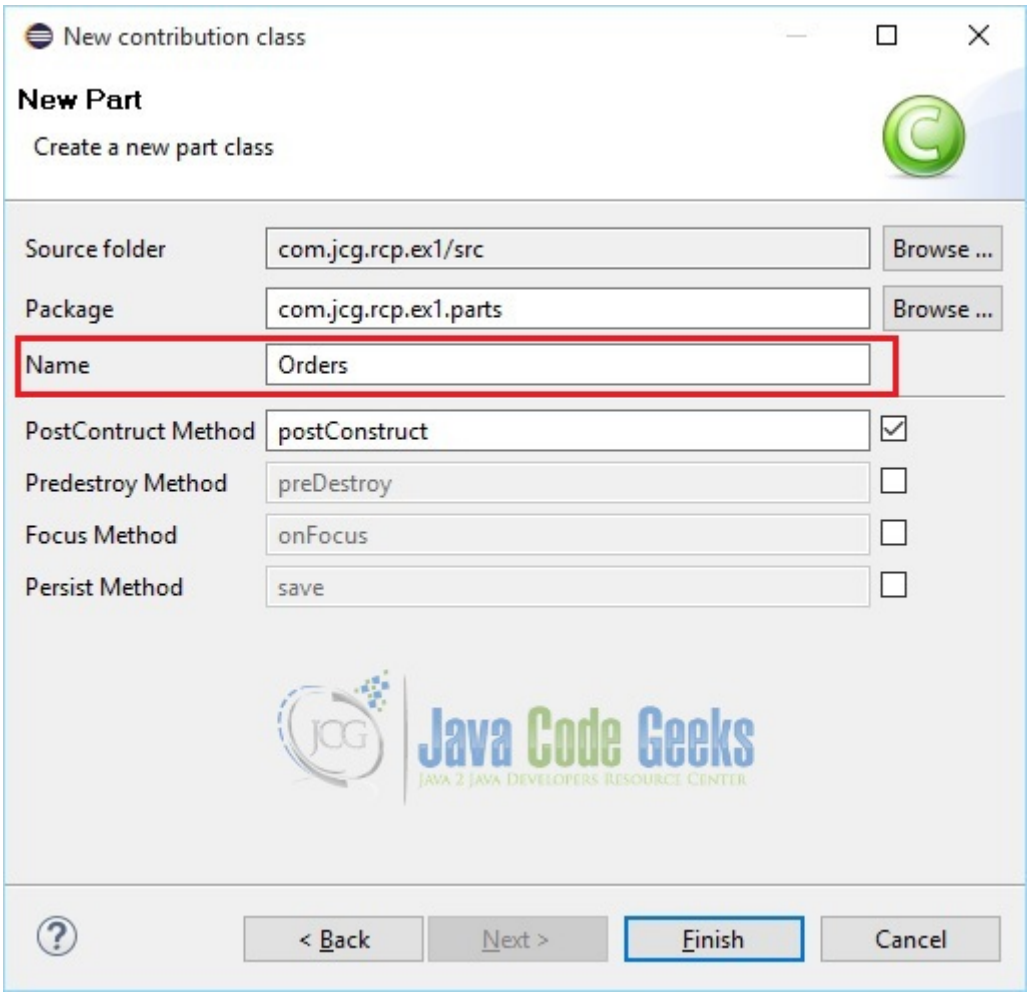

Figure 7.14: Part Class Name

New Part is created with two annotations `@Inject` and `@PostConstruct`. `@Inject` marks a constructor, method, or field as being available for injection and methods annotated with `@PostConstruct` are called after an object has been fully injected.

Figure 7.15: New Part Class

7.11 Add New Part

New Part created above should be attached to the stack to view. Open Application.e4xmi and navigate to *Part Stack*. Click *Add* button.

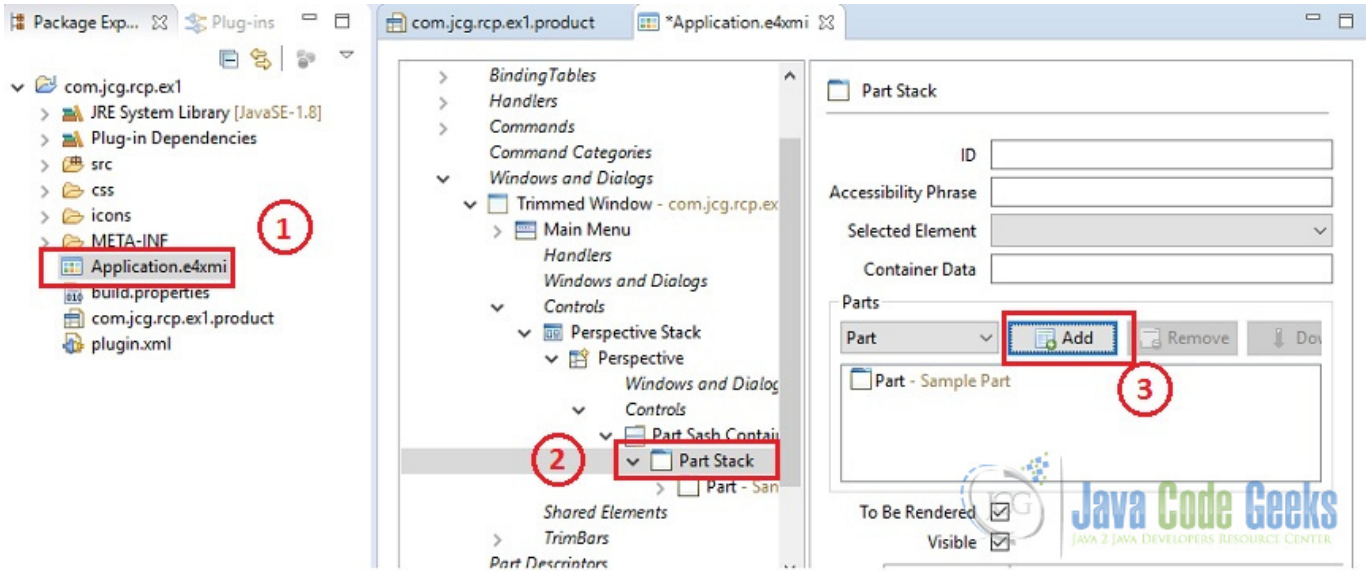

Figure 7.16: Attach Part

New Part form is opened. Enter Part name and click *Find* button of *Class URI*. Select the new part *Orders* just created and click

OK.

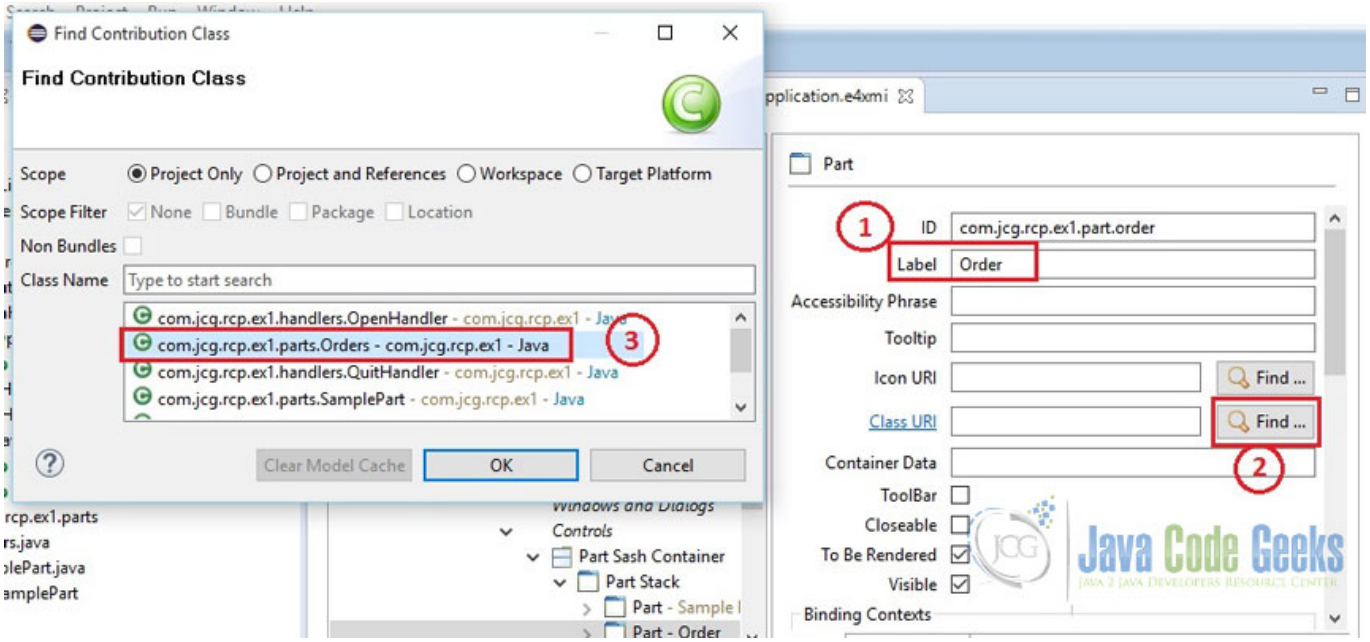

Figure 7.17: New Part Class

Now, newly created Part has been attached with the application. Click *Ctrl+S* to save and run the application as explained in step 8. You can see the new Part has been attached next to *Sample Part*

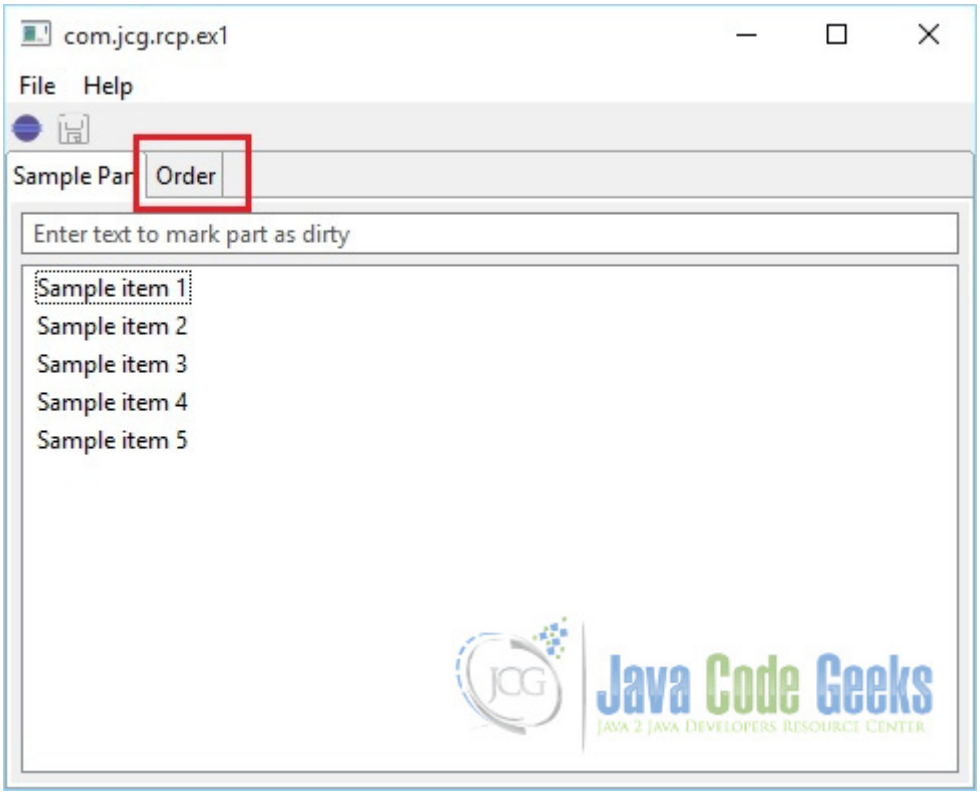

Figure 7.18: New Part Attached

7.12 Add Controls on Part

We will use WindowBuilder Editor to add controls on Part. To open WindowBuilder Editor, right click on the newly created class name *Orders* and open with WindowBuilder Editor.

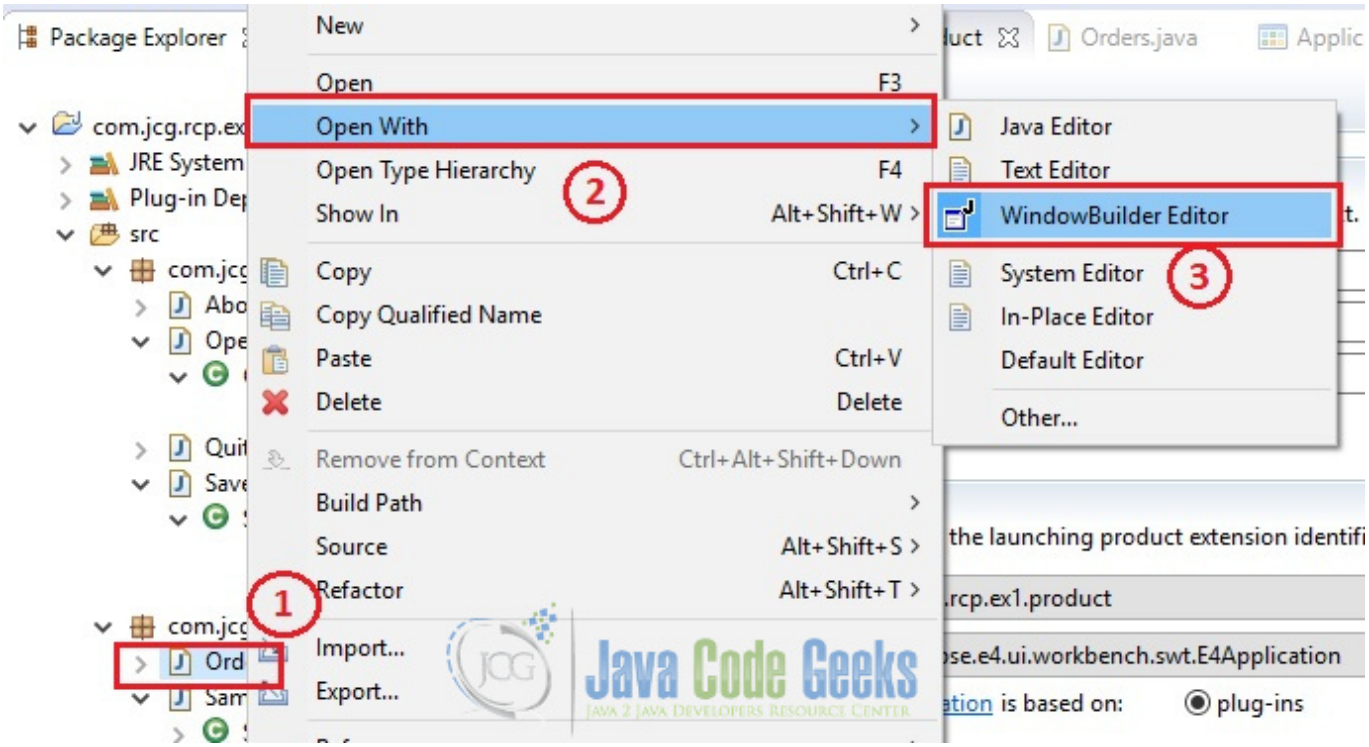

Figure 7.19: WindowBuilder Editor

Click on *Design* tab

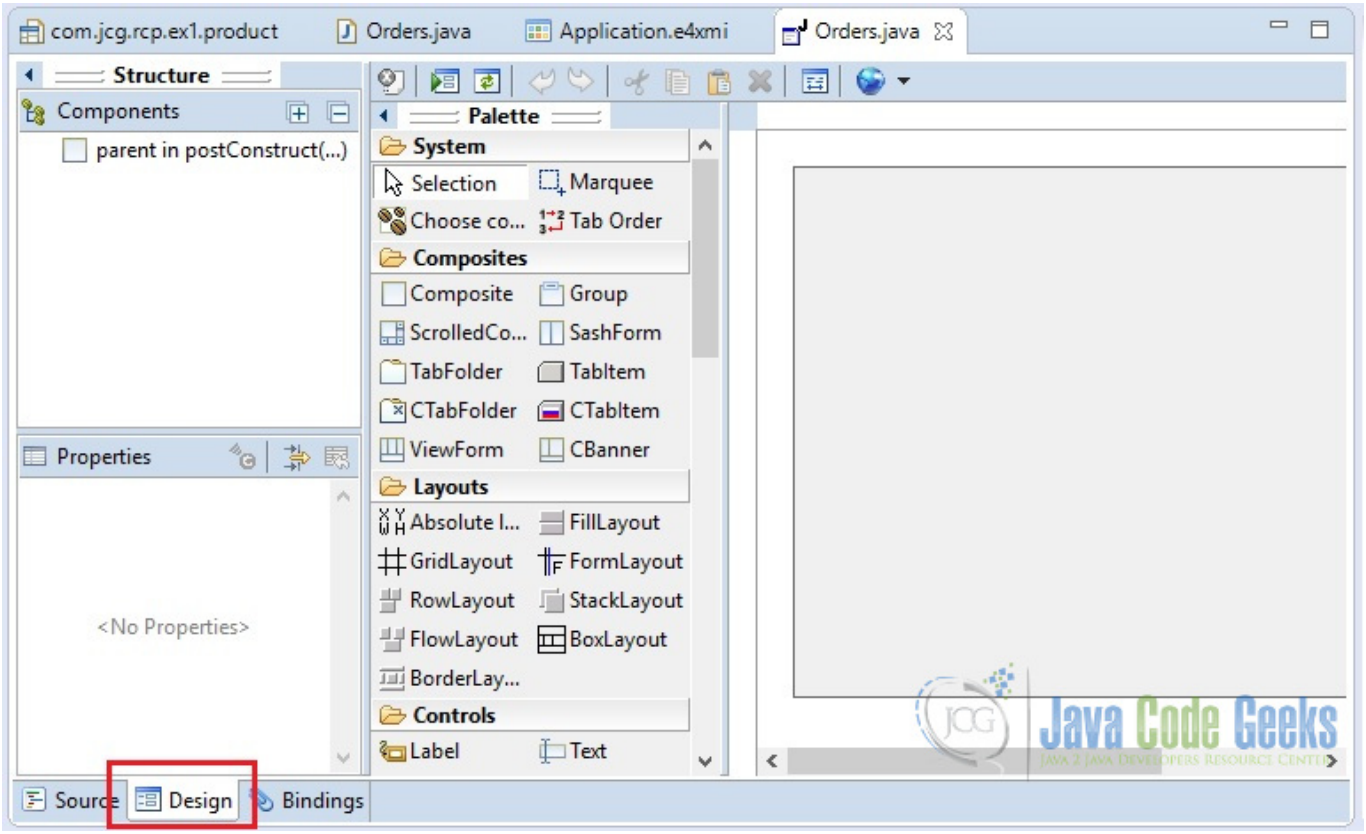

Figure 7.20: Design Palette

Now, your Part can be decorated with required controls and design elements.

7.13 Export application

Yes, finally we want our application to be executed as a separate application away from Eclipse platform. RCP application can be exported as a separate product and executed out of eclipse platform. To export click on the *Eclipse Product export wizard* hyperlink from the same *Overview* tab.

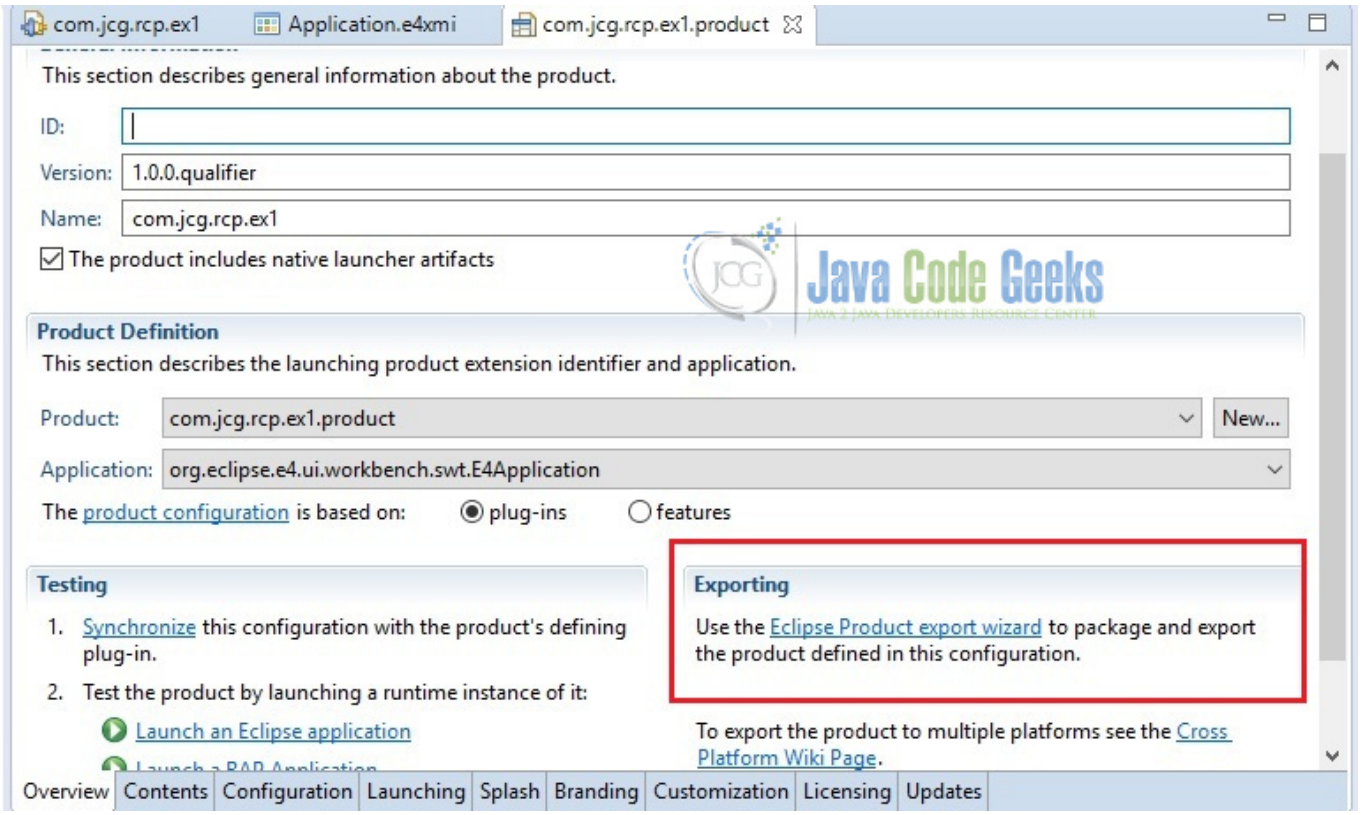

Figure 7.21: Export Application

Export pop-up window appears. Enter destination directory path where this stand-alone application needs to be exported. Click *Finish*

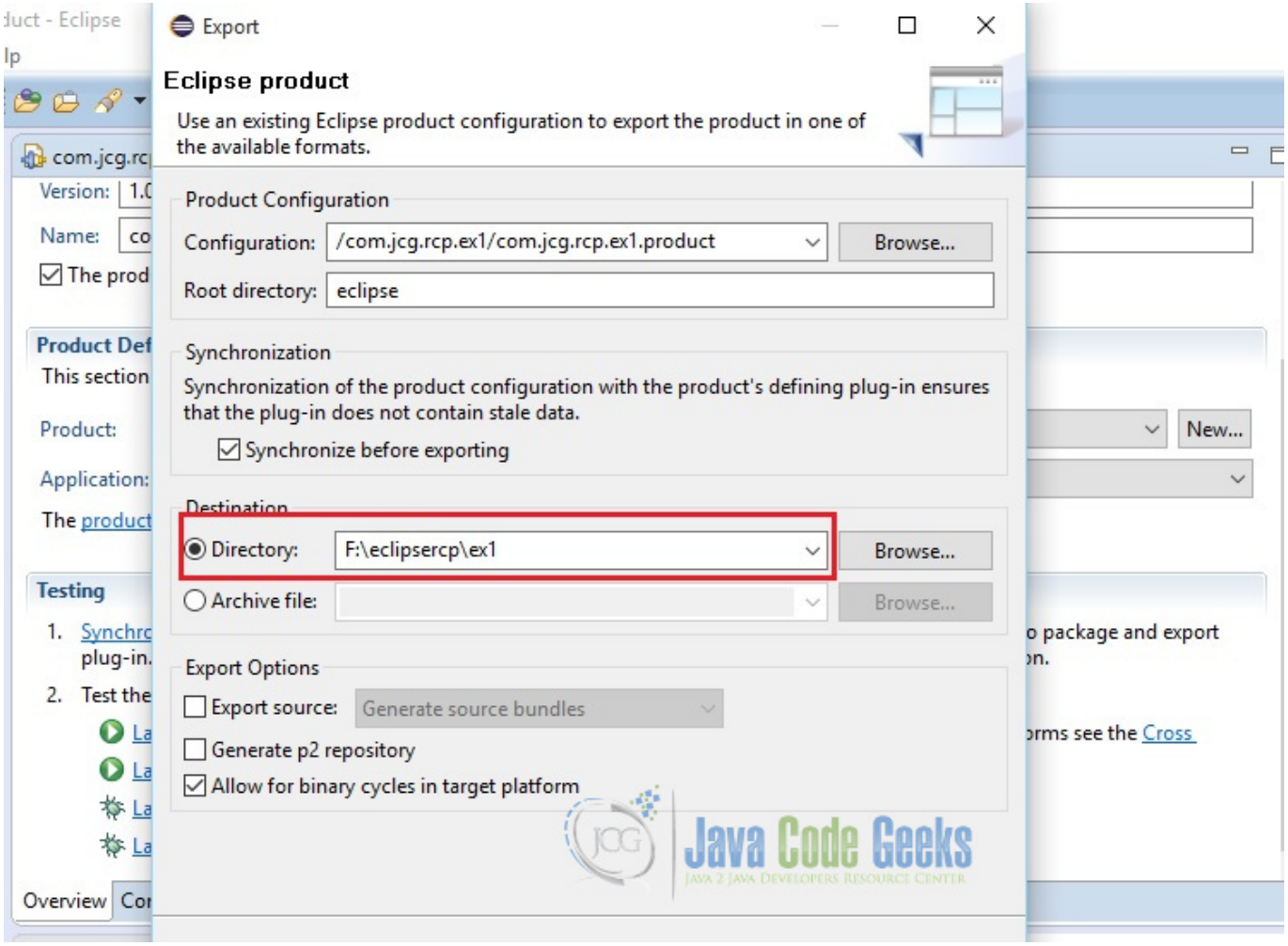

Figure 7.22: Product Configuration

7.14 Completed Application

The application is built and by default the windows version of this application is copied in the destination directory. Click on the *eclipse* icon to run the application away from eclipse platform.

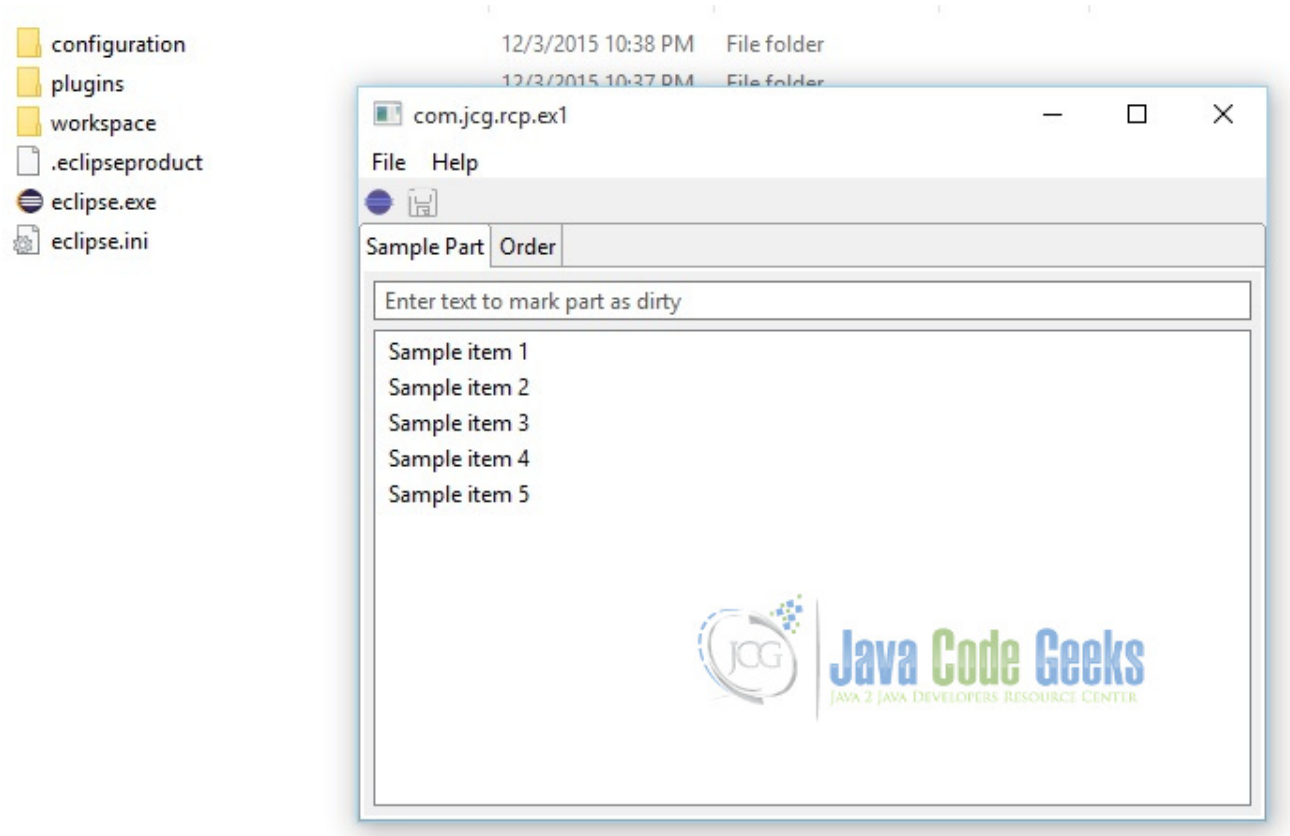

Figure 7.23: Run Separate Application

7.15 Conclusion

We have shown you a way to define the general design of an application in a consistent way using Eclipse 4 RCP API. The Eclipse 4 Application Platform provides you the foundation to build whatever you want as a plug-in or as a stand alone application. In this example we have seen how to create our own view Part and attach with the application.

Chapter 8

How to Install and Use the Eclipse Marketplace Plugin

In this example, we will see how to install and use Eclipse Marketplace Client Plugin from within Eclipse IDE.

8.1 Introduction

The Eclipse community has many third-party plugins and these plugins can be added to the individual Eclipse installation. But, in earlier version of Eclipse, this was not an easy way to discover and install these solutions from within Eclipse.

The Eclipse Foundation operates a website, called Eclipse Marketplace, the *App Store* for Eclipse apps, that provides a listing of Eclipse-based solutions. The listings allow each solution provider to specify a P2 repository for their solution. Eclipse users now have a central catalog to find Eclipse solutions but the install process is still not tightly integrated with the Eclipse workspace.

To make this process simple, Eclipse MarketPlace Client (MPC) provides the tight install integration between the Eclipse workspace and Eclipse Marketplace, plus other third party solution listings. It is a new feature that allows Eclipse users to discover and install Eclipse solutions directly into their Eclipse installation.

8.2 Install Eclipse MarketPlace Client

From Eclipse Juno, the MPC is already included. But, it would be a great idea if we can have MPC on older versions also.

Please follow the steps below to configure market place in the older versions of Eclipse IDE.

Open *Help* → *Install New Software*

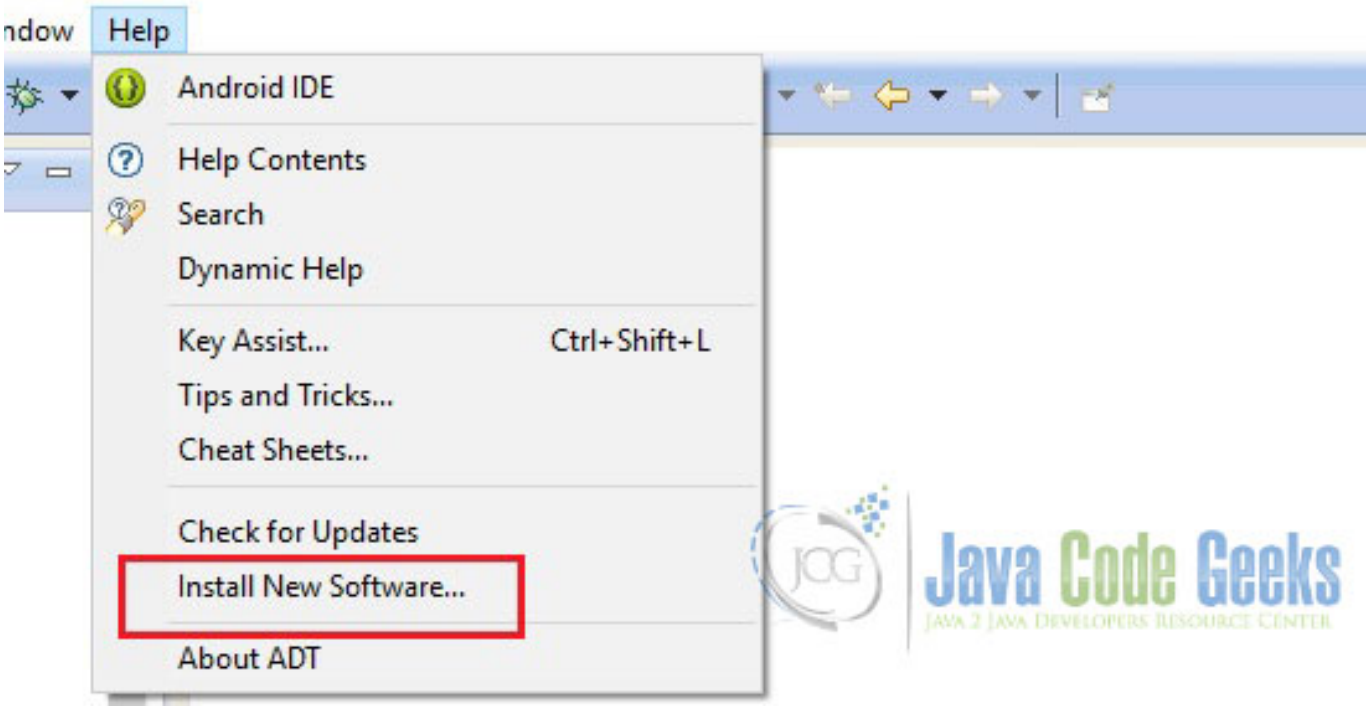

Figure 8.1: Install New Software

And select *Kepler* - *https://download.eclipse.org/releases/kepler*, or *Helios* - *https://download.eclipse.org/releases/helios* from the *Work with* field according to the version of your IDE.

Then select *General Purpose Tools* - *Marketplace Client* as shown below.

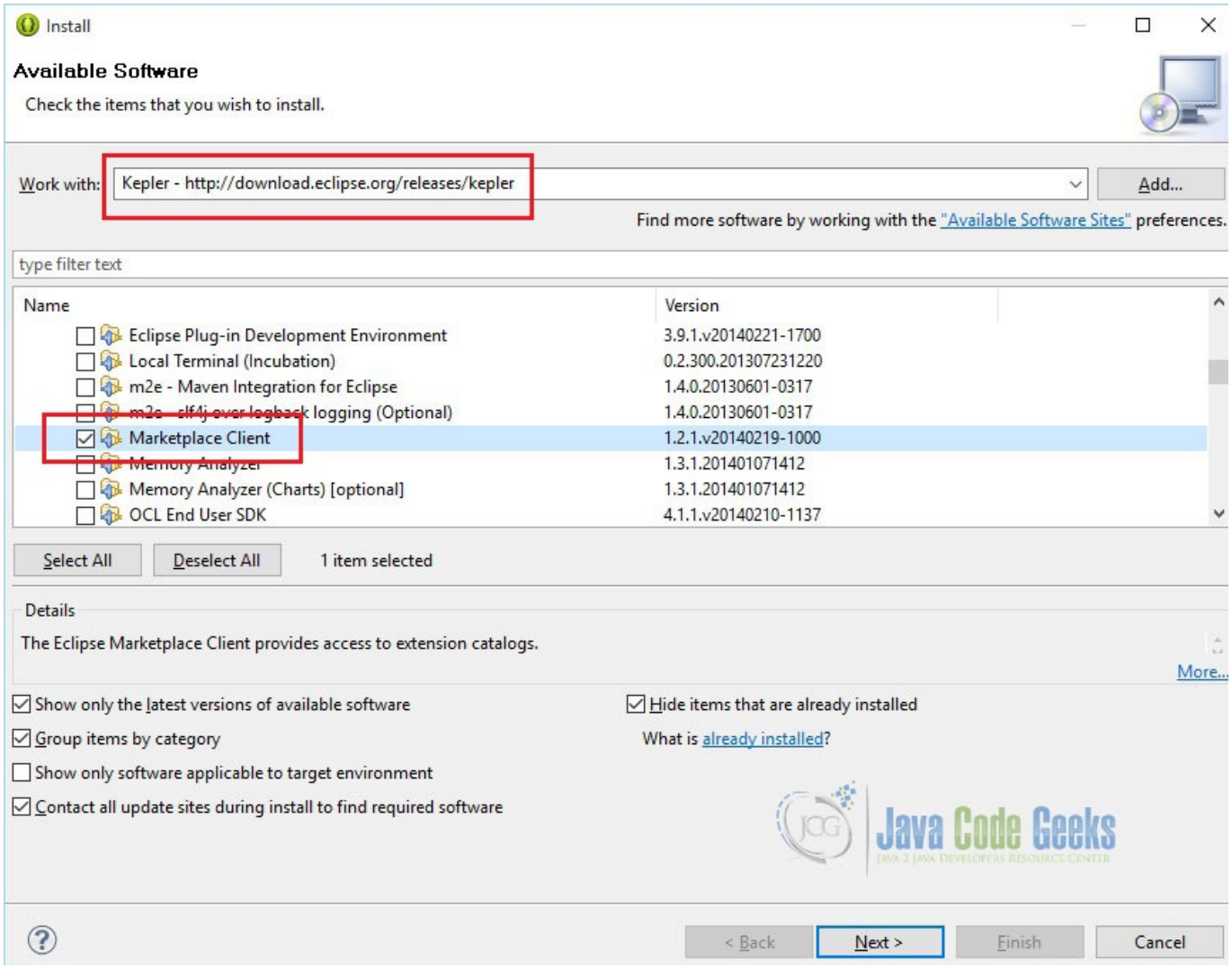

Figure 8.2: Select MPC

Accept terms and install the plug-in. You will be asked to restart the IDE. Please do so.

Now, you will see a new menu *Eclipse Marketplace...* under help menu as shown below. Click on it. .Marketplace Client Menu Marketplace Client Menu

8.3 Eclipse MarketPlace Wizard

Yes, you have Eclipse Marketplace within your IDE now. You can install plugins as per your requirement from the Marketplace shown in the wizard.

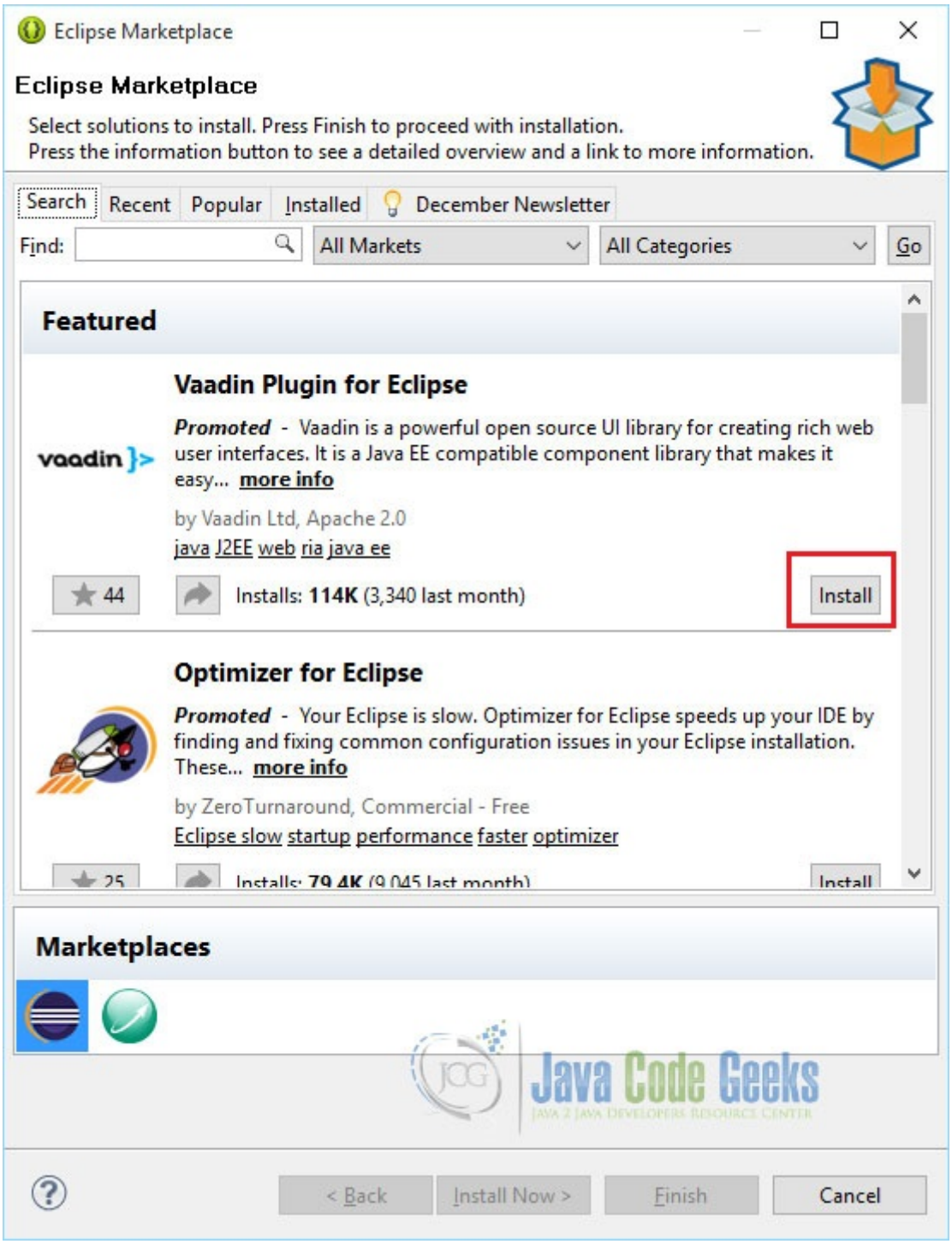

Figure 8.3: Eclipse Marketplace Client

8.4 Search Eclipse Solutions

There is a very effective feature of searching the solutions in the MarketPlace.

In the existing IDE, there is no support for Java 8. Now, from this MarketPlace, we will install Java 8 support plugin.

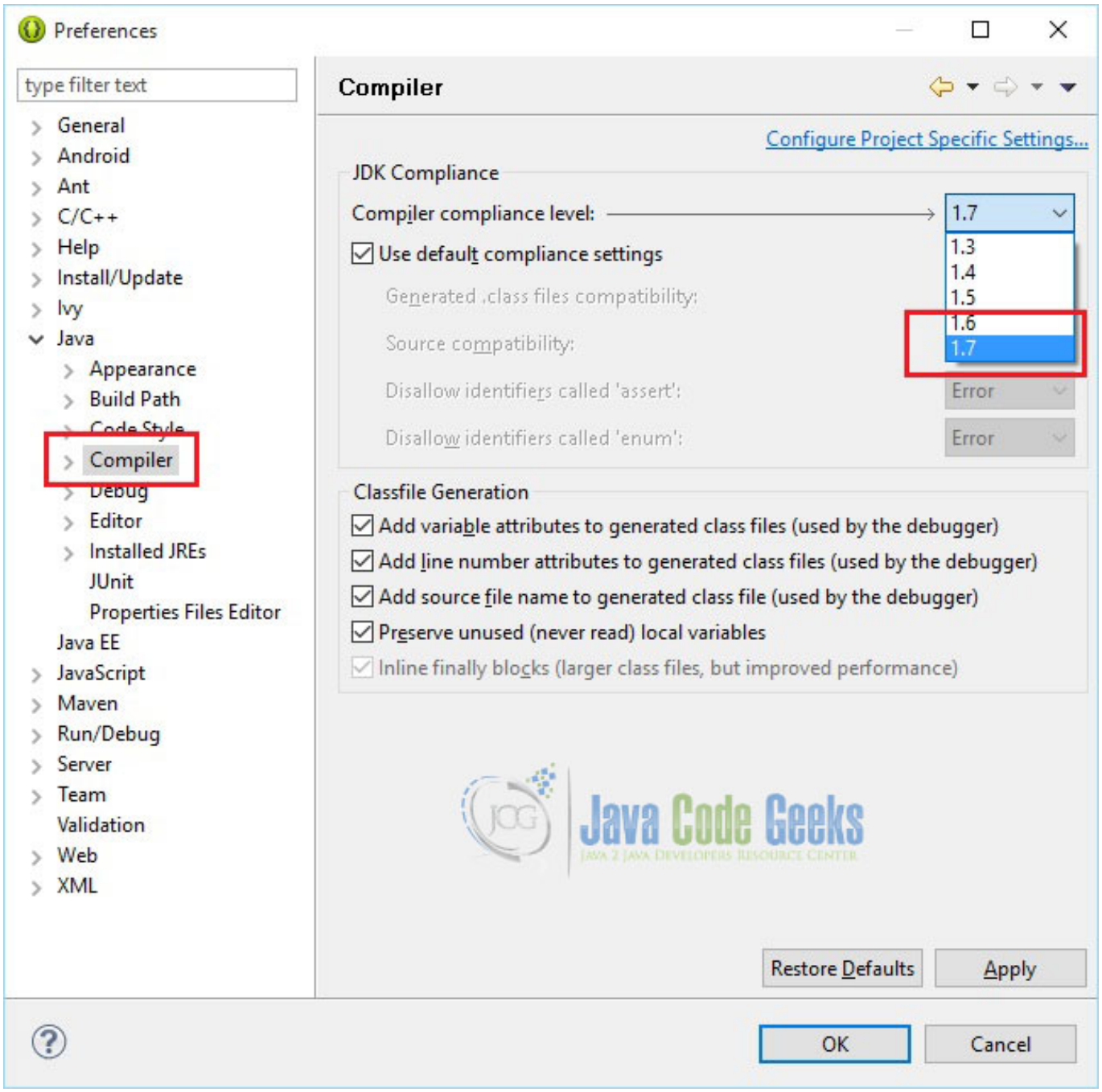

Figure 8.4: Java 7 Latest

Please look at the picture given below. To enable Java 8 support in the existing IDE which was not there by default, enter Java 8 and click search icon.

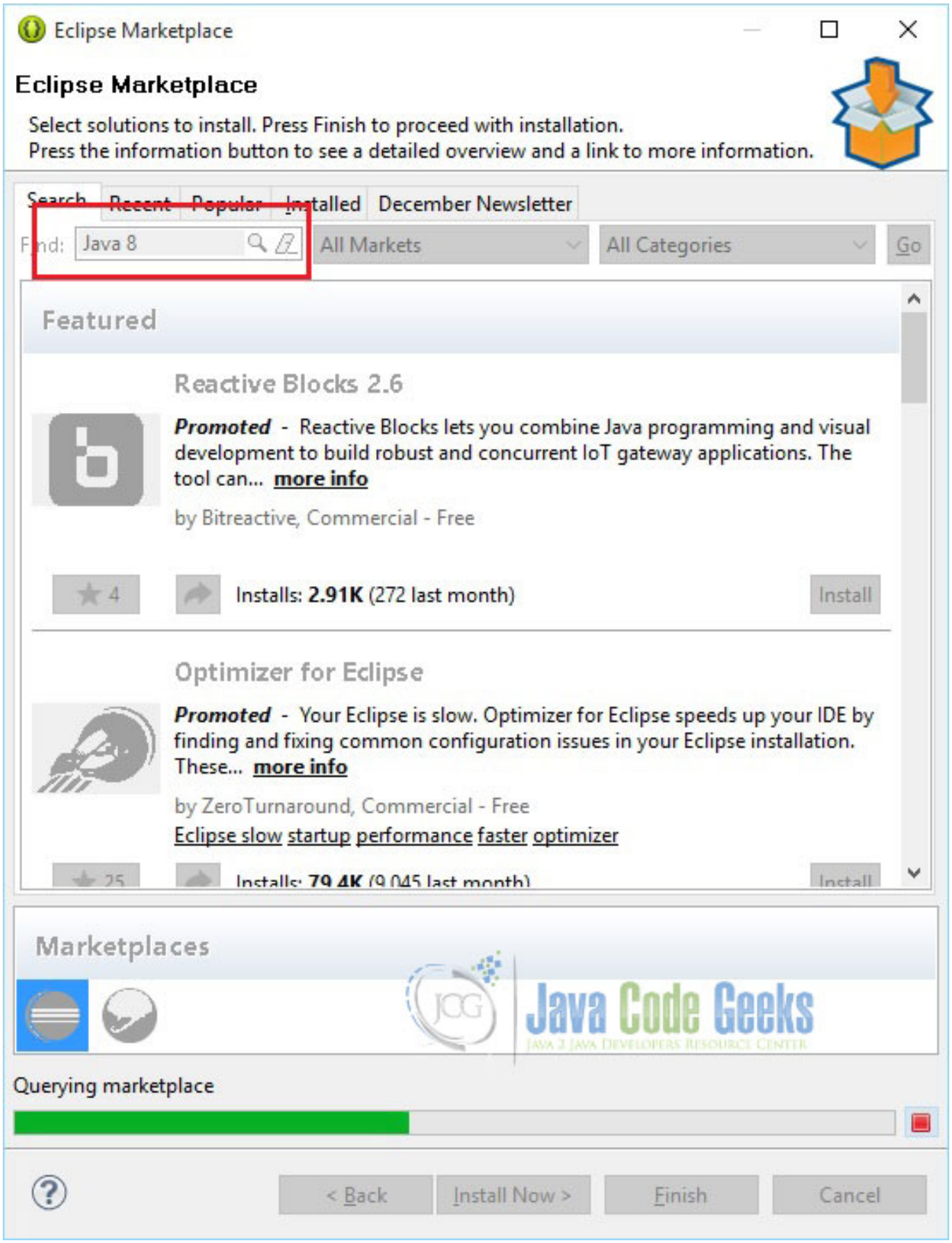

Figure 8.5: Search Solution

In the wizard, you will see available plugins to enable Java 8 support. Click *Install* button of *Java 8 support for Eclipse Kepler SR2* plugin.

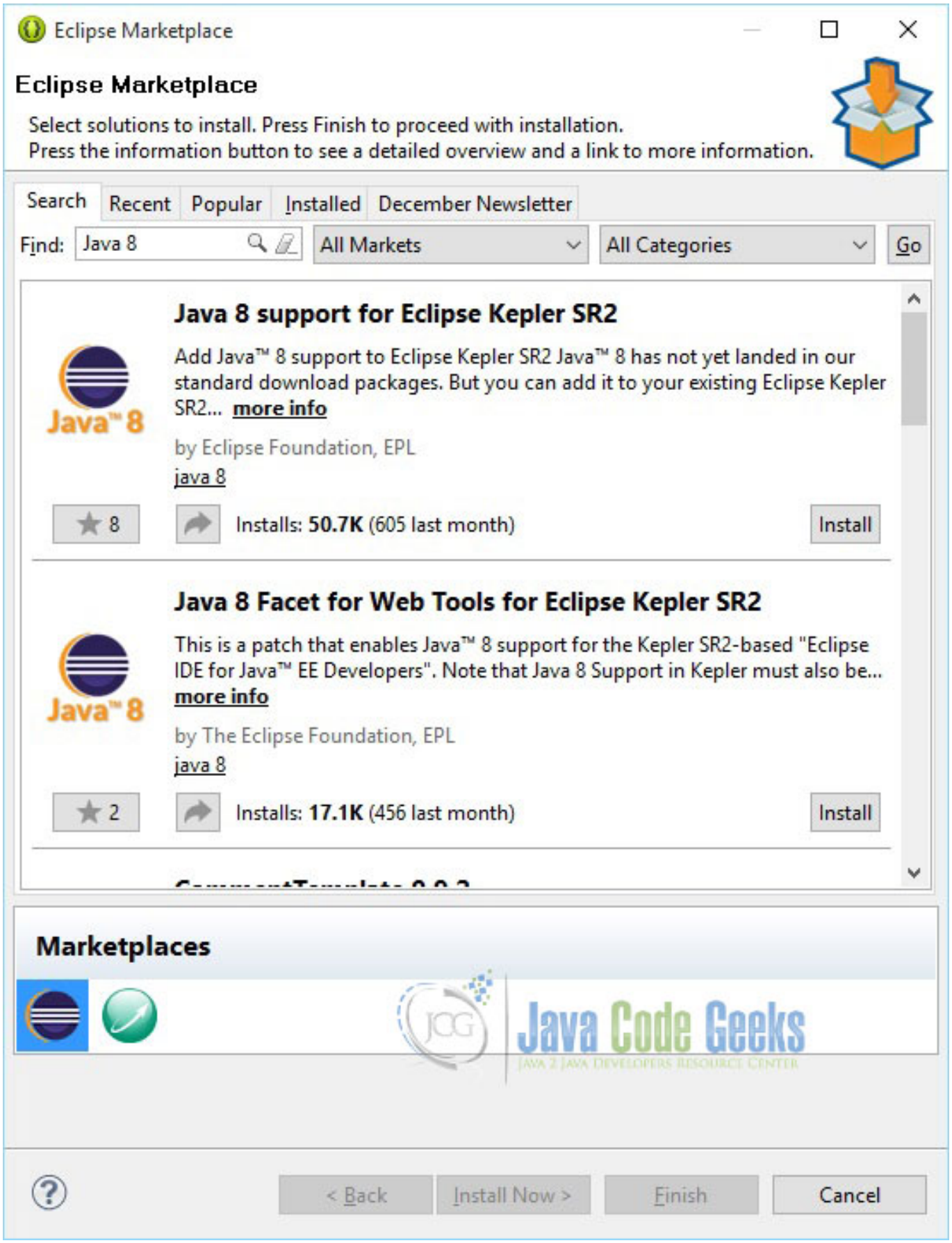

Figure 8.6: Java 8 Support

Select required features as shown below and click *Confirm* button.

Also, it is good to notice that you can install multiple plugins on the same flow by pressing *Install More* button.

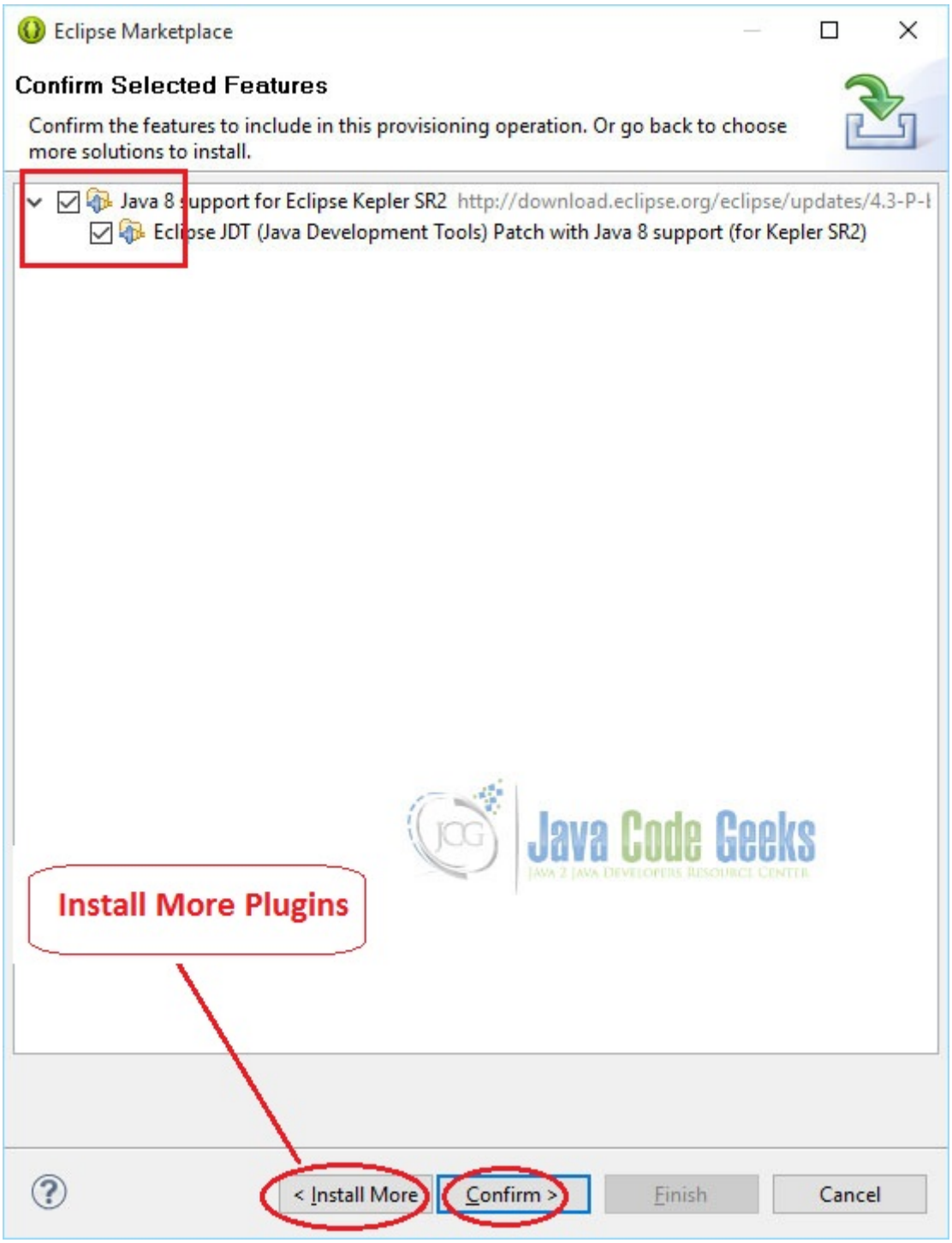

Figure 8.7: Select Plugins

Accept terms and click *Finish*

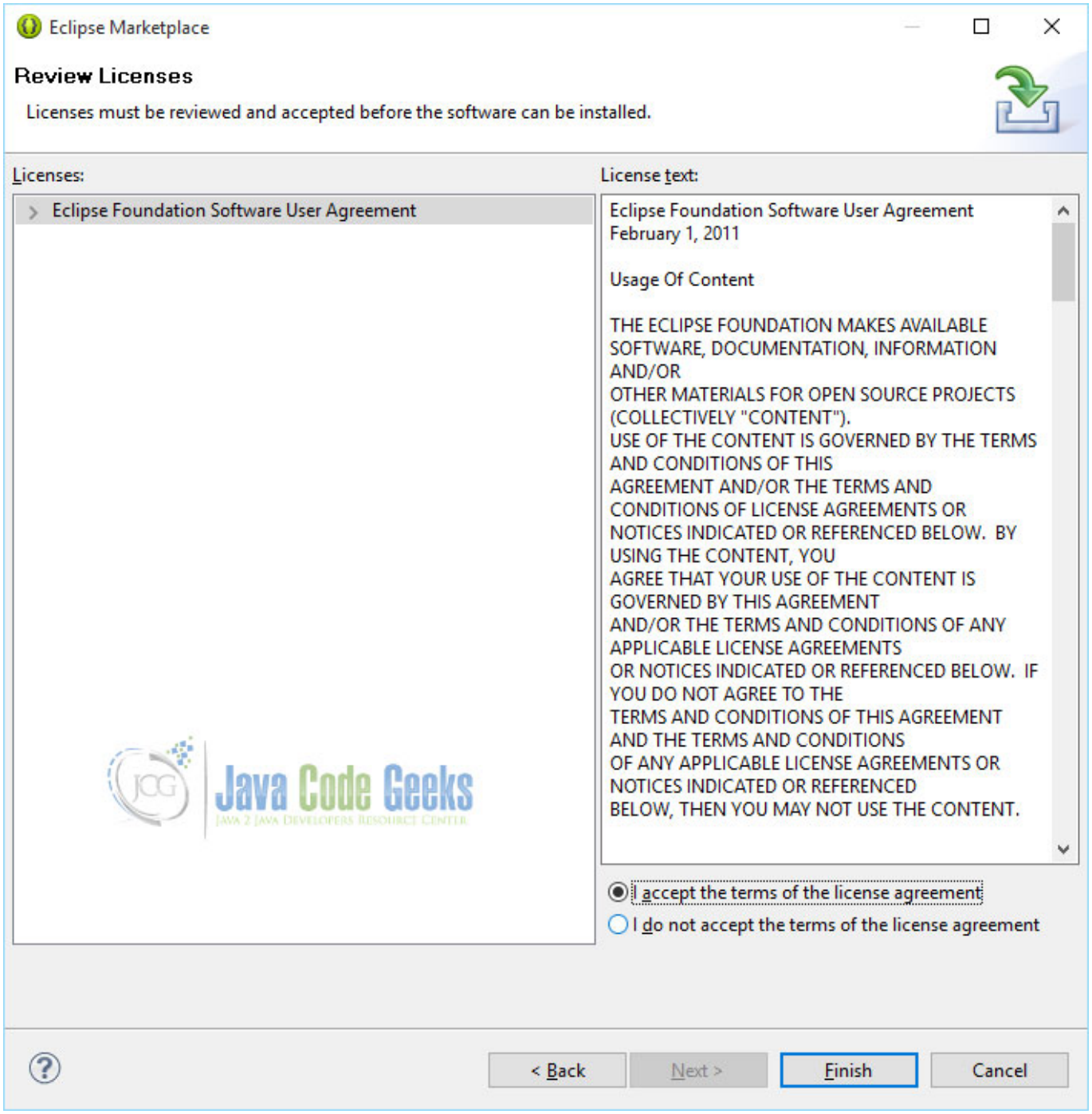

Figure 8.8: Accept Terms

It may take some time to update your IDE with Java 8 support plug-in. You will be asked to restart the IDE. Now, after restarting the IDE if you look at the compiler version, you will be able to see Java 8 compliance level is enabled as depicted below.

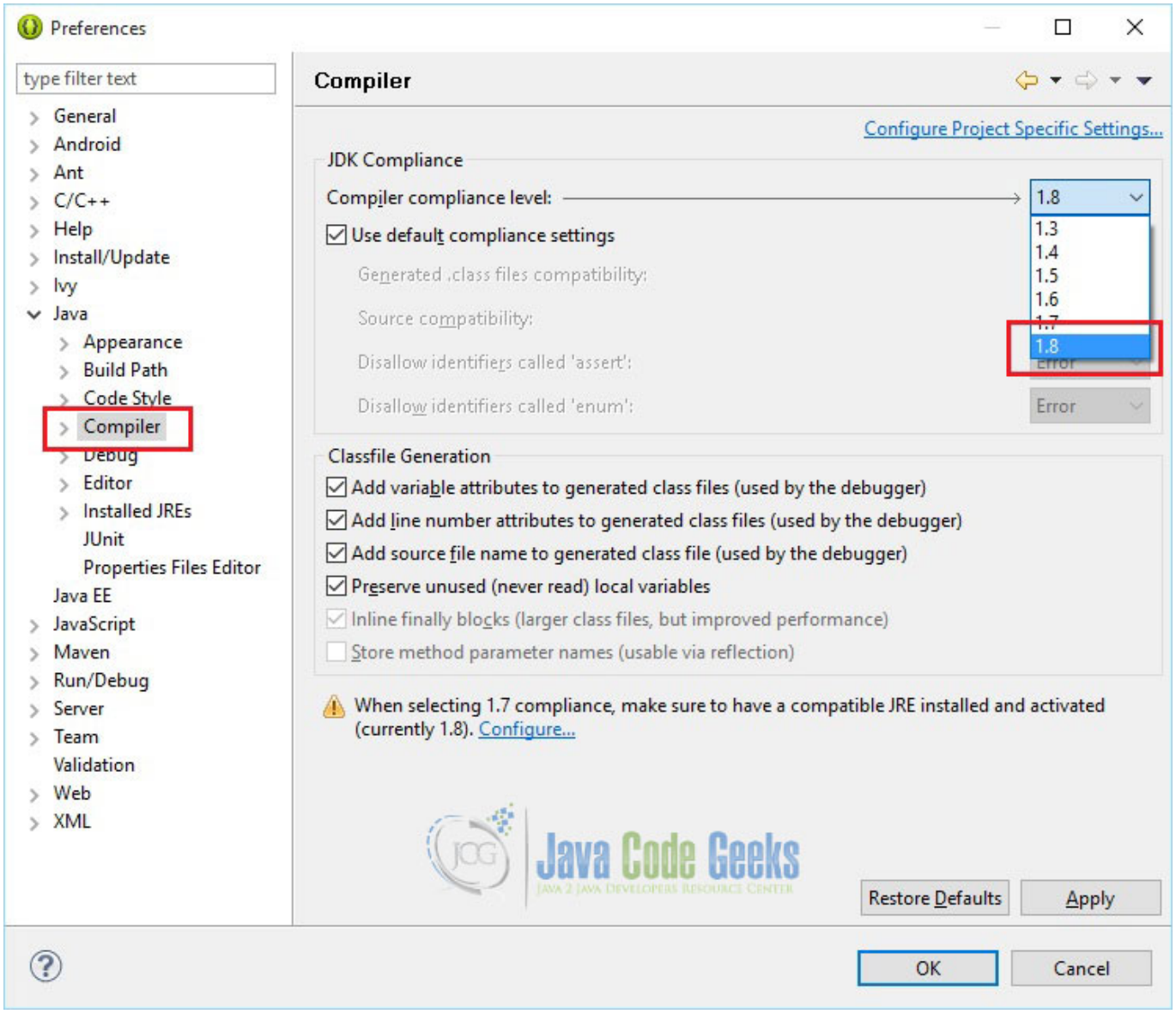

Figure 8.9: Java 8 Support

8.5 Search by Market

The search conditions can be still reduced by specifying Market or Category.

Figure 8.10: Search By Market

8.6 Conclusion

In this example, we have seen how to install Eclipse Marketplace Client plug-in used for browsing and installing the Eclipse based solutions listed on the Eclipse Marketplace portal from within the Eclipse IDE.