



**FREE eBook**

# LEARNING eclipse

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

**#eclipse**

# Table of Contents

|  |           |
|--|-----------|
| About.....   | 1         |
| <b>Chapter 1: Getting started with eclipse.....</b>              | <b>2</b>  |
| Remarks.....   | 2         |
| Versions.....  | 2         |
| Examples.....  | 2         |
| Installation and Setup.....                                      | 3         |
| Install Marketplace in Eclipse.....                              | 4         |
| Useful Keyboard Shortcuts.....                                   | 5         |
| <b>Manage Files and Projects.....</b>                            | <b>5</b>  |
| <b>Editor Window.....</b>  | <b>5</b>  |
| <b>Navigate in Editor.....</b>                                   | <b>5</b>  |
| <b>Edit Text.....</b>  | <b>5</b>  |
| <b>Search and Replace.....</b>                                   | <b>6</b>  |
| <b>Move a block of code.....</b>                                 | <b>6</b>  |
| Creating and Running a Java HelloWorld Program.....              | 6         |
| Create a new Java project.....                                   | 6         |
| Create a new Java class.....                                     | 8         |
| Run your Java class.....   | 10        |
| Importing Existing Projects.....                                 | 10        |
| <b>Chapter 2: Configuring Eclipse.....</b>                       | <b>13</b> |
| Examples.....  | 13        |
| Increasing maximum heap memory for Eclipse.....                  | 13        |
| Specifying the JVM.....  | 13        |
| How to configure the font size of views in Eclipse on Linux..... | 14        |
| <b>Chapter 3: Create a new workspace in Eclipse.....</b>         | <b>17</b> |
| Examples.....  | 17        |
| How to create a workspace.....                                   | 17        |
| <b>Chapter 4: Debugging Java programs in Eclipse.....</b>        | <b>18</b> |
| Examples.....  | 18        |

|   |           |
|---|-----------|
| Evaluating expressions within a debugging session.....                      | 18        |
| Remote debugging of a Java application.....                                 | 20        |
| <b>Chapter 5: Eclipse Shortcuts.....</b>                                    | <b>22</b> |
| Introduction.....   | 22        |
| Examples.....   | 22        |
| Comment/Uncomment code.....   | 22        |
| Open Resouce Dialog.....  | 22        |
| To get a println.....   | 22        |
| Generate Getters and Setters.....   | 22        |
| Refactor Highlighted Text.....  | 22        |
| Format xml.....   | 22        |
| <b>Chapter 6: How Eclipse Remote Debugging works behind the scenes.....</b> | <b>23</b> |
| Examples.....   | 23        |
| How does Eclipse Remote Debugging work behind the scences.....              | 23        |
| <b>Chapter 7: Remote Debugging in Eclipse.....</b>                          | <b>24</b> |
| Examples.....   | 24        |
| Configure Eclipse Remote Debugging for an application.....                  | 24        |
| <b>Chapter 8: Setting up Eclipse for C++.....</b>                           | <b>26</b> |
| Examples.....   | 26        |
| Linux + CMake ("Unix Makefiles" generator) + Qt (optional).....             | 26        |
| Qt (optional).....  | 26        |
| Workspace.....  | 26        |
| Attaching Sources to the Project.....                                       | 26        |
| CMake generator.....  | 26        |
| Build.....  | 27        |
| Re-running CMake (to re-generate the makefiles).....                        | 27        |
| <b>Chapter 9: Tomcat deployment procedure.....</b>                          | <b>28</b> |
| Examples.....   | 28        |
| Procedure when nothing else helps.....                                      | 28        |
| <b>Credits.....</b>   | <b>29</b> |

---

# About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [eclipse](#)

It is an unofficial and free eclipse ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official eclipse.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to [info@zzzprojects.com](mailto:info@zzzprojects.com)

---

# Chapter 1: Getting started with eclipse

## Remarks

This section provides an overview of what eclipse is, and why a developer might want to use it.

It should also mention any large subjects within eclipse, and link out to the related topics. Since the Documentation for eclipse is new, you may need to create initial versions of those related topics.

## Versions

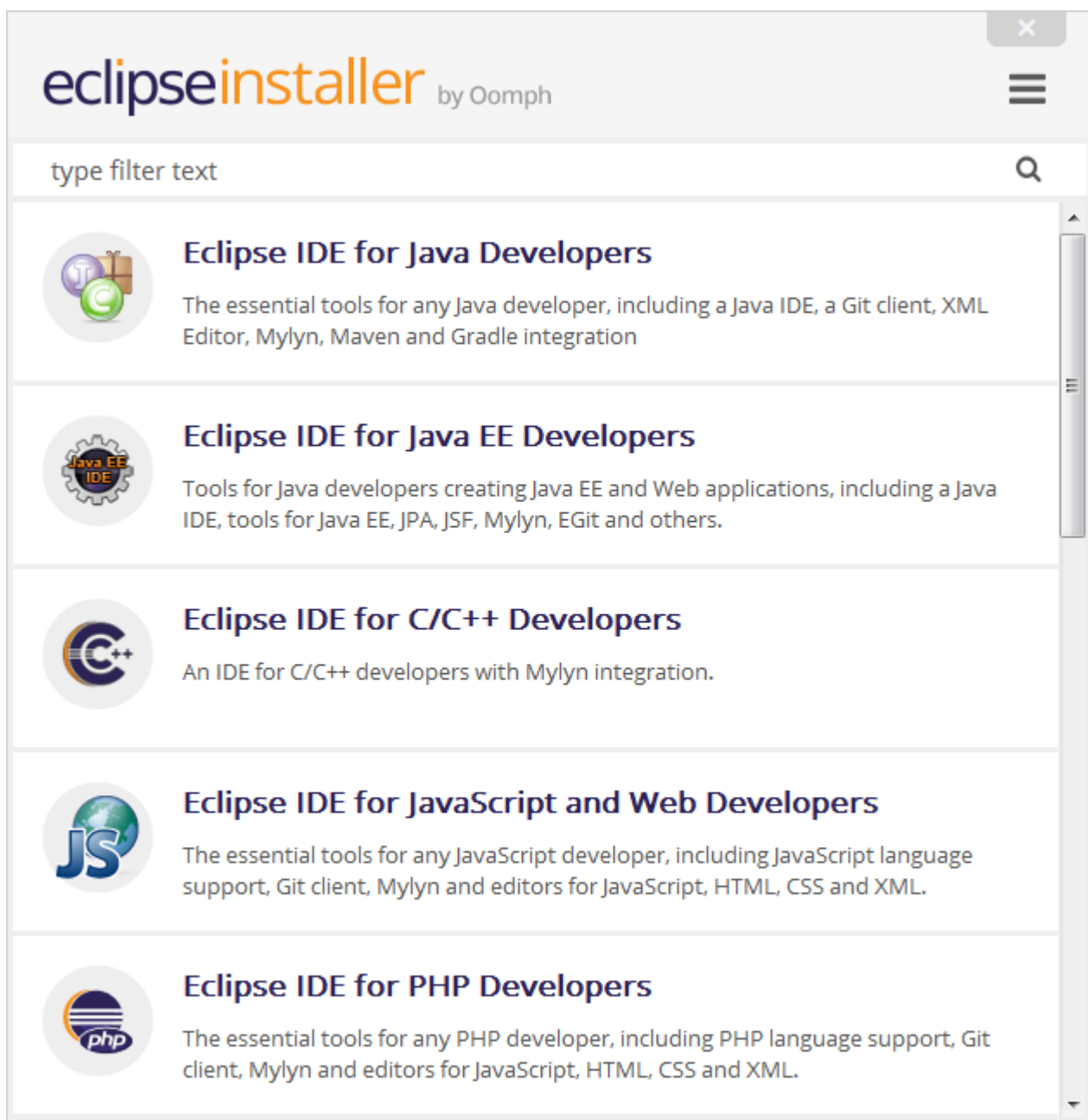
| Version     | Name             | Release Date |
|-------------|------------------|--------------|
| 3.0         |                  | 2004-06-21   |
| 3.1         |                  | 2005-06-28   |
| 3.2         | Callisto         | 2006-06-30   |
| 3.3         | Europa           | 2007-06-29   |
| 3.4         | Ganymede         | 2008-06-25   |
| 3.5         | Galileo          | 2009-06-24   |
| 3.6         | Helios           | 2010-06-23   |
| 3.7         | Indigo           | 2011-06-22   |
| 3.8 and 4.2 | Juno             | 2012-06-27   |
| 4.3         | Kepler           | 2013-06-26   |
| 4.4         | Luna             | 2014-06-25   |
| 4.5         | Mars             | 2015-06-24   |
| 4.6         | Neon             | 2016-06-22   |
| 4.7         | Oxygen           | 2017-06-28   |
| 4.8         | Photon (Planned) | 2018-06-01   |

## Examples

## Installation and Setup

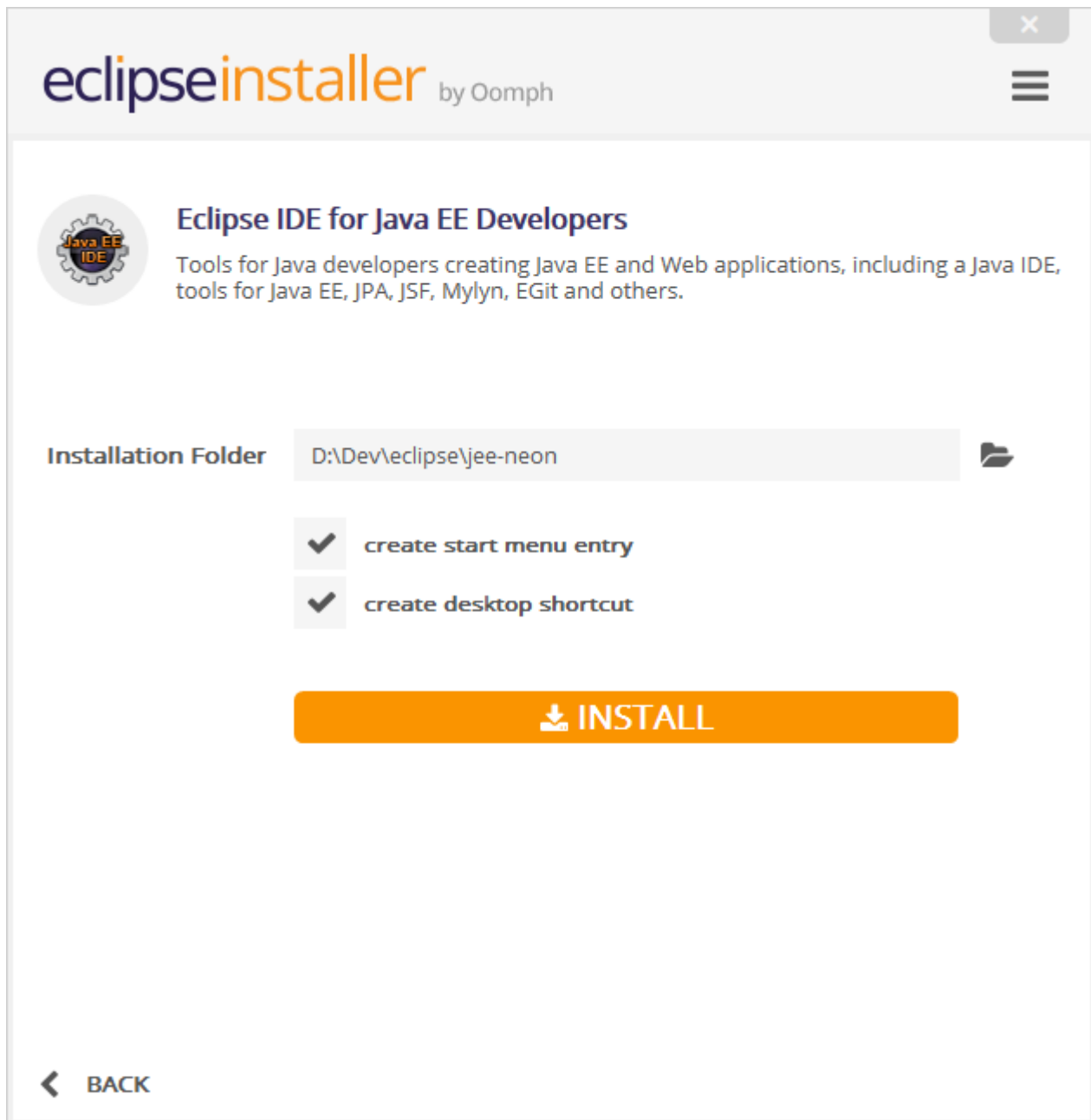
To install Eclipse, go to the [Eclipse Downloads](#) Web page where there is usually a direct link to download the latest version of Eclipse. Starting Eclipse Mars (version 4.5), an installer can be downloaded which guides you through the installation procedure, as opposed to downloading the whole installation as a compressed file (this option is still available, however). There are also links to download old Eclipse packages.

Eclipse comes in several different packages that target different users as shown in the below screenshot from the installer. For instance, the **Eclipse IDE for Java Developers** contains basic tools that support developing, debugging and building Java applications, as well as basic support for version control such as a plugin that allows versioning projects using Git, while the **Eclipse for Android Developers** provides an environment for creating Android applications.



Once a package is selected, the next page allows the user to select the installation directory, along with other options. The following screenshot illustrates the procedure on a Windows

machine.



Clicking the Install button will start the installation of the Eclipse package into that directory.

If the machine already has Java properly installed, Eclipse should launch fine and no configuration or setup is usually required. However, it is a good practice to change some configuration options for Eclipse, for example to specify in which JVM Eclipse should run, and to configure minimum and maximum memory for that JVM. To do so, a file called `eclipse.ini` exists in the installation directory, where this startup configuration is located. [This page](#) contains details about how to configure Eclipse using that file.

## Install Marketplace in Eclipse

Few of the eclipse classic versions don't come pre-installed with marketplace, this maybe installed using the following steps:

1. Goto Help → Install new Software
2. Add new Repository(site specified below)
3. General Purpose Tools → Marketplace Client
4. Click Finish and you are done.

Marketplace update sites:

```
Oxygen - http://download.eclipse.org/releases/oxygen/  
Neon - http://download.eclipse.org/releases/neon/  
Mars - http://download.eclipse.org/releases/mars/  
Luna - http://download.eclipse.org/mpc/luna  
Helios - http://download.eclipse.org/releases/helios  
Juno - http://download.eclipse.org/releases/juno/
```

## Useful Keyboard Shortcuts

---

# Manage Files and Projects

- **Ctrl+Shift+R** : Open Resource (file, folder or project)
- **Ctrl+Shift+S** : Save all files
- **Ctrl+W** : Close current file
- **Ctrl+Shift+W** : Close all files

---

# Editor Window

- **F12** : Jump to Editor Window
- **Ctrl+E** : Show list of open Editors. Use arrow keys and enter to switch
- **Ctrl+Page Down/Up** : Switch to next editor / switch to previous editor
- **Ctrl+M** : Maximize or minimize current Editor Window

---

# Navigate in Editor

- **Ctrl+L** : Go to line
- **Ctrl+Q** : Jump to last location edited
- **Ctrl+Shift+P** : With a bracket selected: jump to the matching closing or opening bracket
- **Ctrl+Arrow Down/Up** : Scroll Editor without changing cursor position

---

# Edit Text

- **Ctrl+D** : Delete Line
- **Alt+Shift+Y** : Wrap selected text (fit text width to screen)
- **Alt+Shift+S** : Open Source menu options
- **Alt+Shift+R** : Refactor highlighted word across all files



- **Ctrl+Alt+Up/Ctrl+Alt+Down** : Copy the selected lines to top/down.

---

## Search and Replace

- **Ctrl+J** : Type a term to search then use Ctrl+J / Ctrl+shift+J to go up/back
- **Ctrl+K/Ctrl+Shift+K** : Ctrl+H then close find window. Then Find previous / next occurrence of search term.

---

## Move a block of code

- **Shift+tab** : Move to the left
- **Alt+Up/Alt+Down** : Move to top/down.

### Creating and Running a Java HelloWorld Program

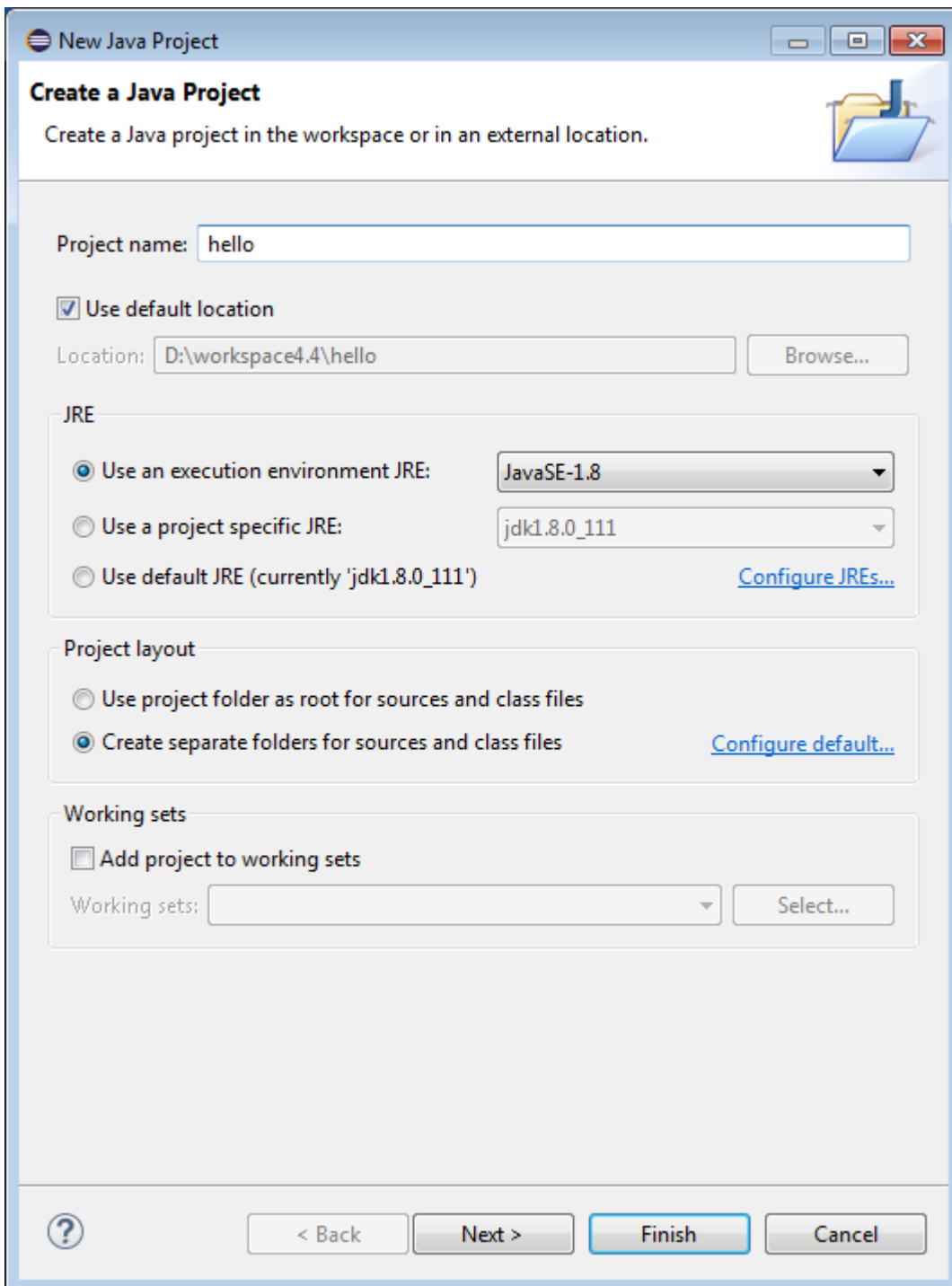
From the tool-bar open the Java Perspective.



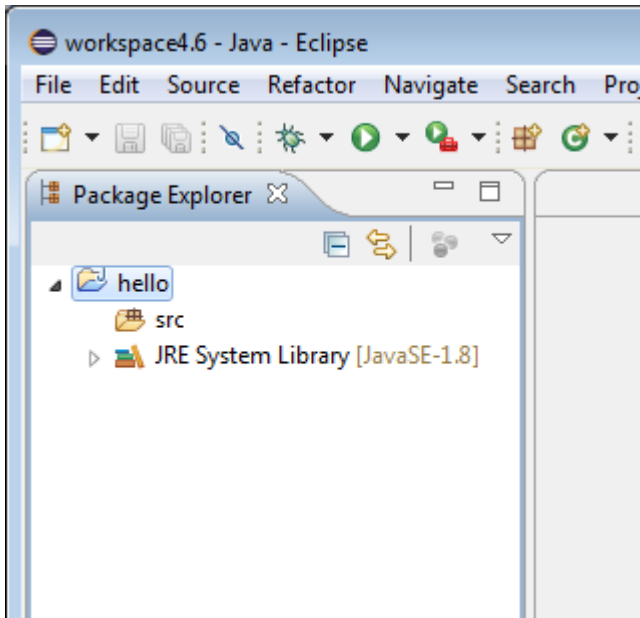
### Create a new Java project

Right-click into the Package Explorer, and from the menu select `New -> Java Project`

In the upcoming dialog enter a project name, then click `Finish`.



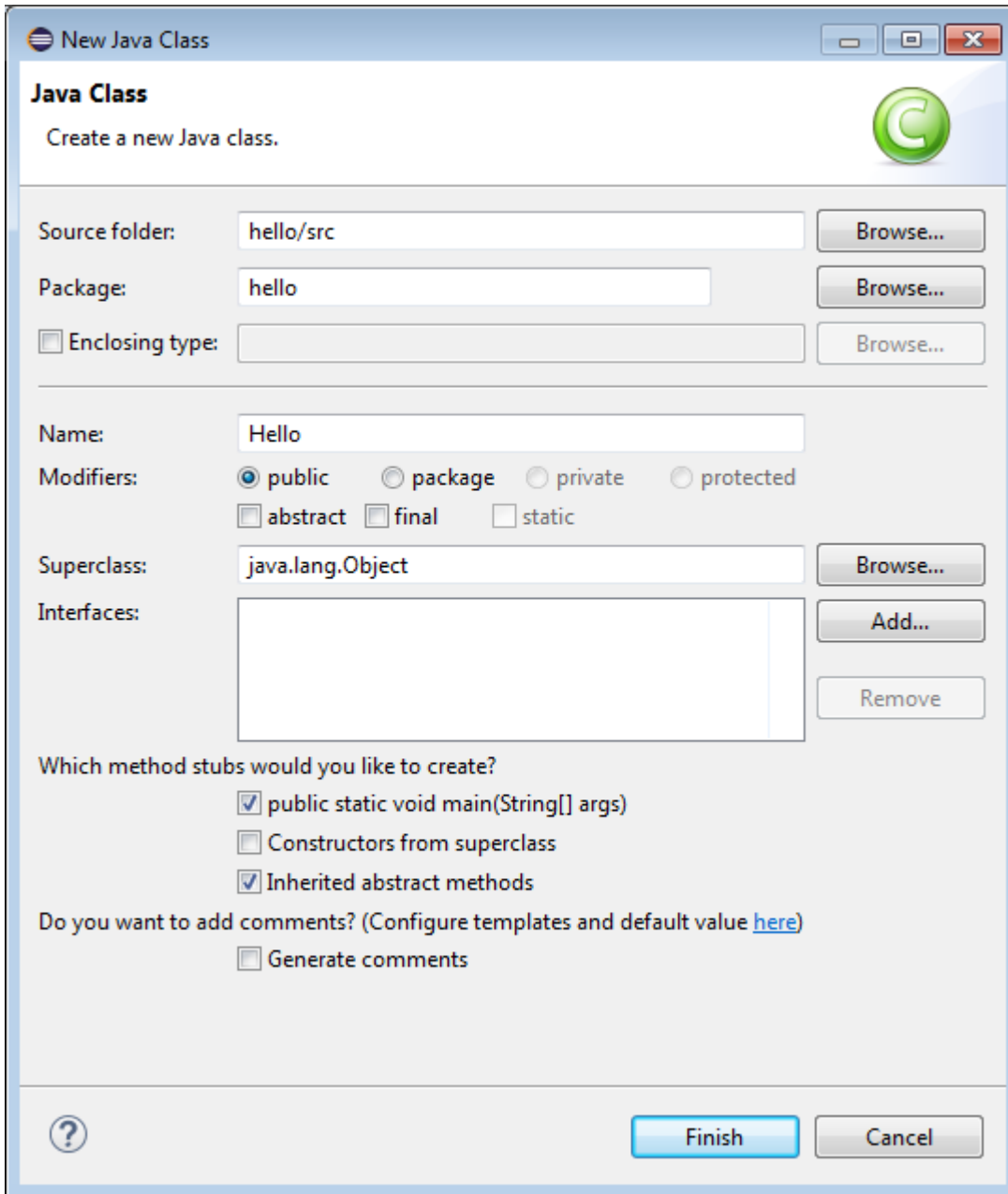
Now you have the new project in your workspace.



## Create a new Java class

Right-click on your project, and from the menu select `New -> Class`.

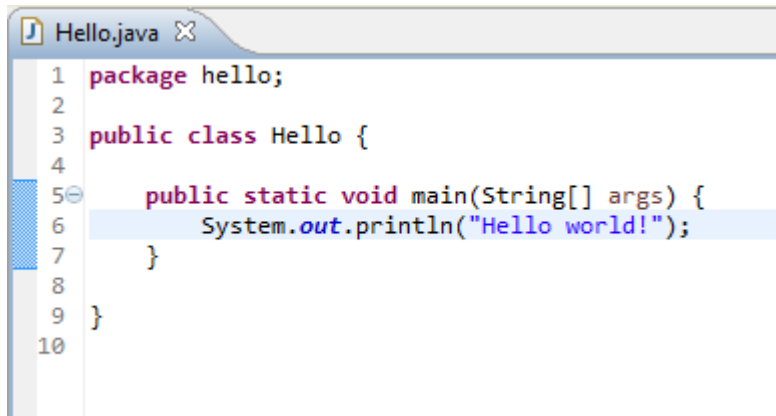
In the dialog type in the class' name (it should begin with a capital letter), also select the checkbox `public static void main(String[] args)`, then click `Finish`.



Now you have the first Java file in your project. The editor will automatically open this new file.

```
Hello.java
1 package hello;
2
3 public class Hello {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8     }
9
10 }
11
```

Within the `main` method type in some code to print `Hello world!`.

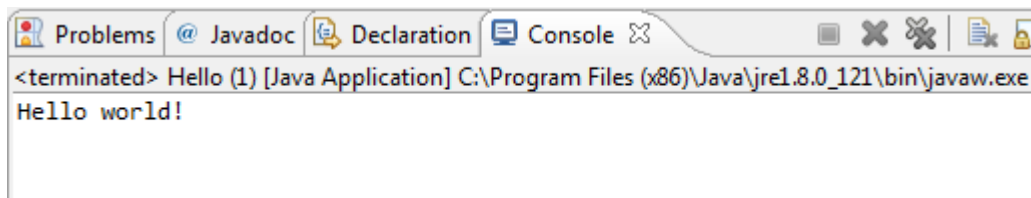


```
1 package hello;
2
3 public class Hello {
4
5     public static void main(String[] args) {
6         System.out.println("Hello world!");
7     }
8
9 }
10
```

## Run your Java class

Right-click on your Java class, and from the menu select `Run as -> Java application`.

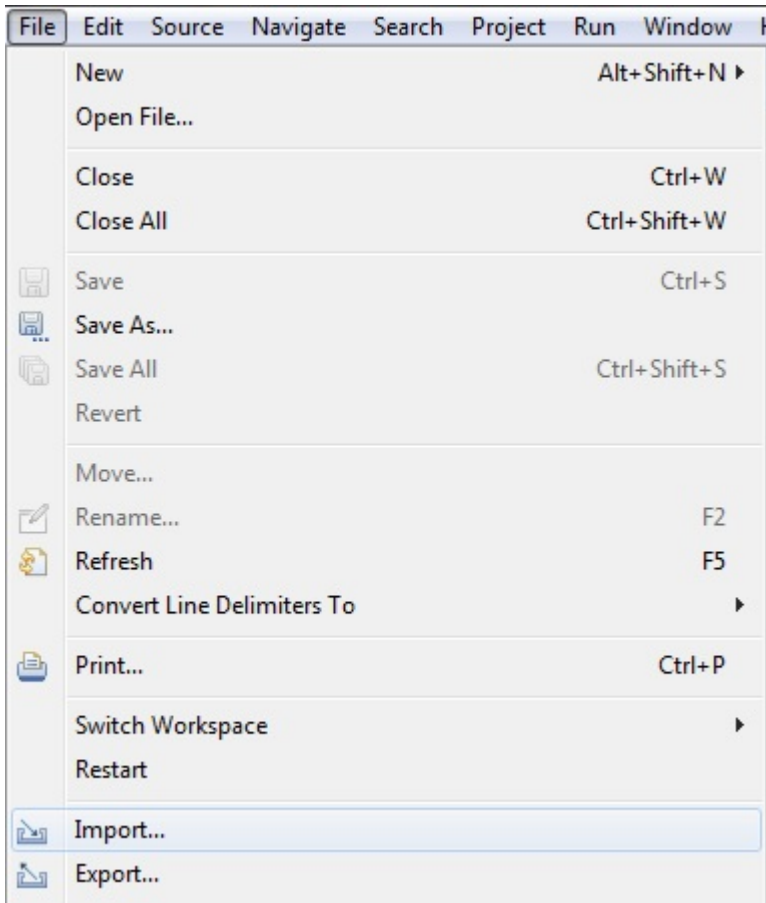
Voila, you see the output of your Java program in the Console.



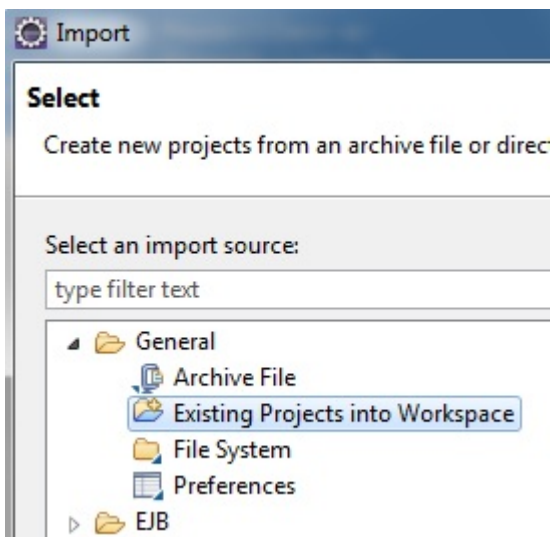
```
<terminated> Hello (1) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_121\bin\javaw.exe
Hello world!
```

## Importing Existing Projects

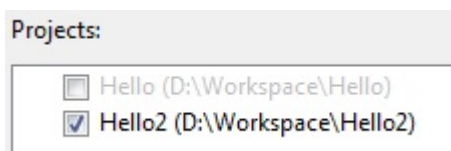
In the File menu, choose the 'Import...' option.



This opens up the Import dialog box, which asks for the type of project/file you want to import. For a basic Java project, choose 'Existing Projects into Workspace' from the 'General' folder.

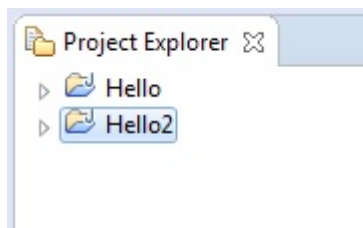


Next, select the directory where the project(s) is located using the 'Browse' button. All projects that can be imported into Eclipse will show up in the 'Projects:' section. If the project has already been imported, it will still be displayed but the checkbox will be disabled.



You can also import projects directly from a compressed file by choosing 'Select archive file' and then clicking the 'Browse' button.

Once you click 'Finish' the project is now visible in your Project Explorer and ready to use.



Read [Getting started with eclipse online](https://riptutorial.com/eclipse/topic/1143/getting-started-with-eclipse): <https://riptutorial.com/eclipse/topic/1143/getting-started-with-eclipse>

---

# Chapter 2: Configuring Eclipse

## Examples

### Increasing maximum heap memory for Eclipse

To increase the maximum amount of heap memory used Eclipse, edit the `eclipse.ini` file located in the Eclipse installation directory.

This file specifies options for the startup of Eclipse, such as which JVM to use, and the options for the JVM. Specifically, you need to edit the value of the `-Xmx` JVM option (or create one if it does not exist).

Below is an example configuration that sets a maximum heap memory of 1 GB (1024m). The relevant line is `-Xmx1024m` - this would replace the existing `-Xmx*` line in your configuration:

```
-startup
plugins/org.eclipse.equinox.launcher_1.3.200.v20160318-1642.jar
--launcher.library
C:/Users/user1/.p2/pool/plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.400.v20160518-1444
-product
org.eclipse.epp.package.java.product
--launcher.defaultAction
openFile
-showsplash
org.eclipse.platform
--launcher.appendVmargs
-vmargs
-Xms256m
-Xmx1024m
```

### Specifying the JVM

A common issue that users of Eclipse encounter is related to the default system JVM.

A typical situation is a 64 bit Windows which has both 32 and 64 bit versions of Java installed, and a 32 bit Eclipse. If the 64 bit version of Java is the system default, when Eclipse is launched an error dialog is shown.

Specifying the JVM explicitly in `eclipse.ini` will resolve this. The `-vm` entry should be added directly above the `-vmargs` section.

The example below shows how to use a 32 bit JVM on a 64 bit Windows:

```
-startup
plugins/org.eclipse.equinox.launcher_1.3.200.v20160318-1642.jar
...
-vm
C:/Program Files (x86)/Java/jdk1.7.0_71/bin/javaw.exe
```



```
-vmargs  
-Xms256m  
-Xmx1024m
```

## How to configure the font size of views in Eclipse on Linux

Eclipse does not give you the possibility to change the font size of the views like 'Project Explorer' or 'Servers', which looks ugly on Linux since Eclipse uses the default (desktop) font size. But you can edit specific configuration files to get the proper font sizes.

To fix this annoying font size, go to

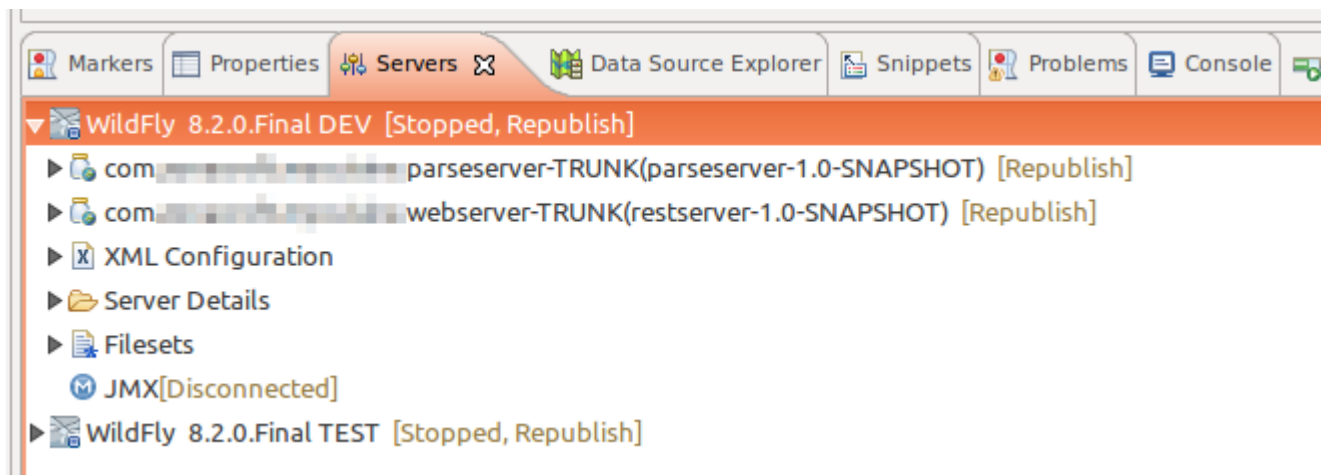
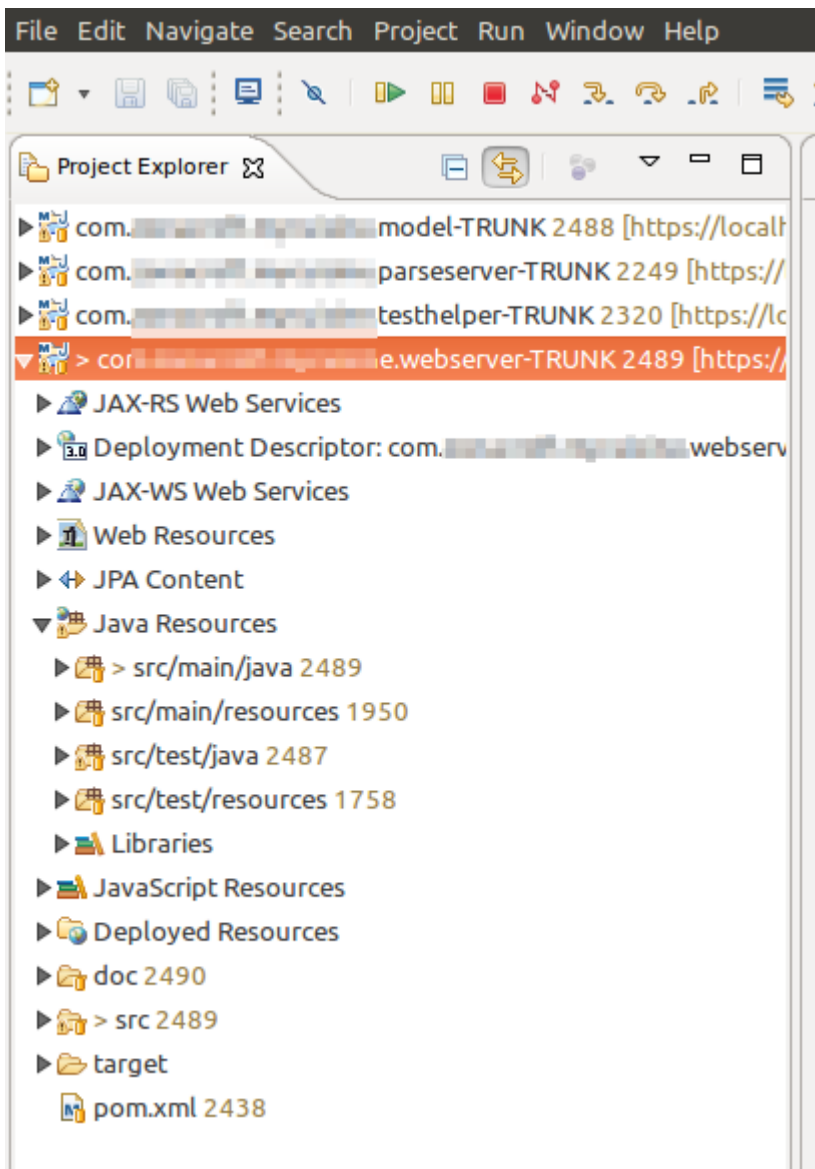
`/[YOUR_INST_DIR]/eclipse/plugins/org.eclipse.ui.themes_[LATEST_INSTALLATION]/css` and add this content...

```
.MPart Tree{  
    font-family: Sans;  
    font-size: 8px;  
}
```

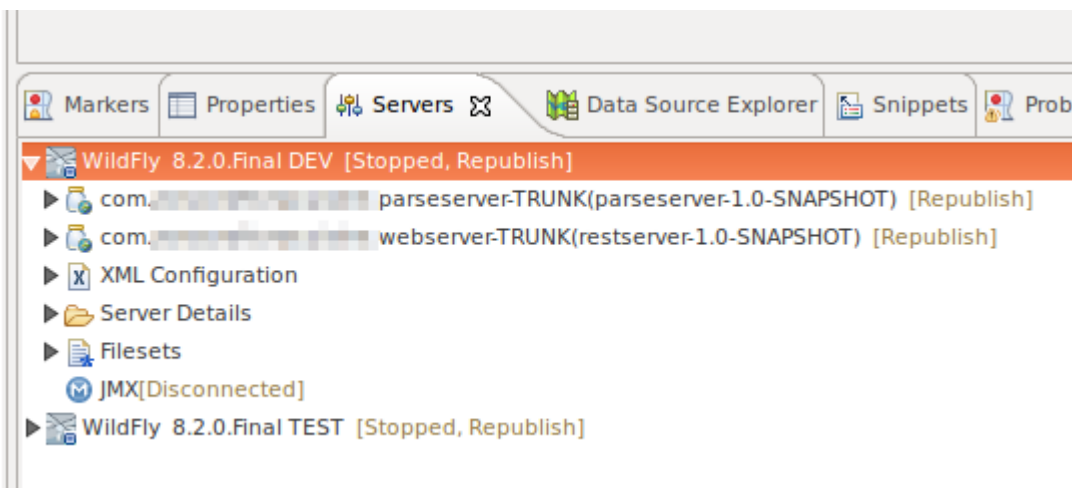
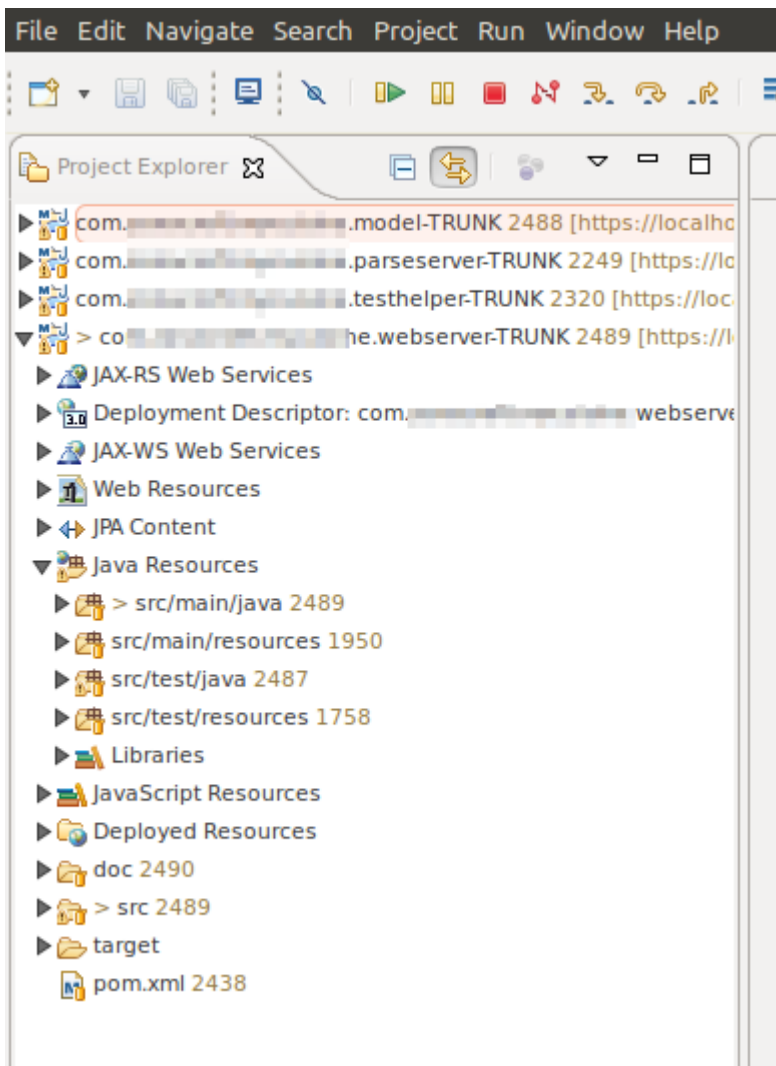
to the bottom of the following files:

```
e4_classic_winxp.css  
e4_classic_win7.css
```

### **BEFORE CHANGE**



## AFTER CHANGE



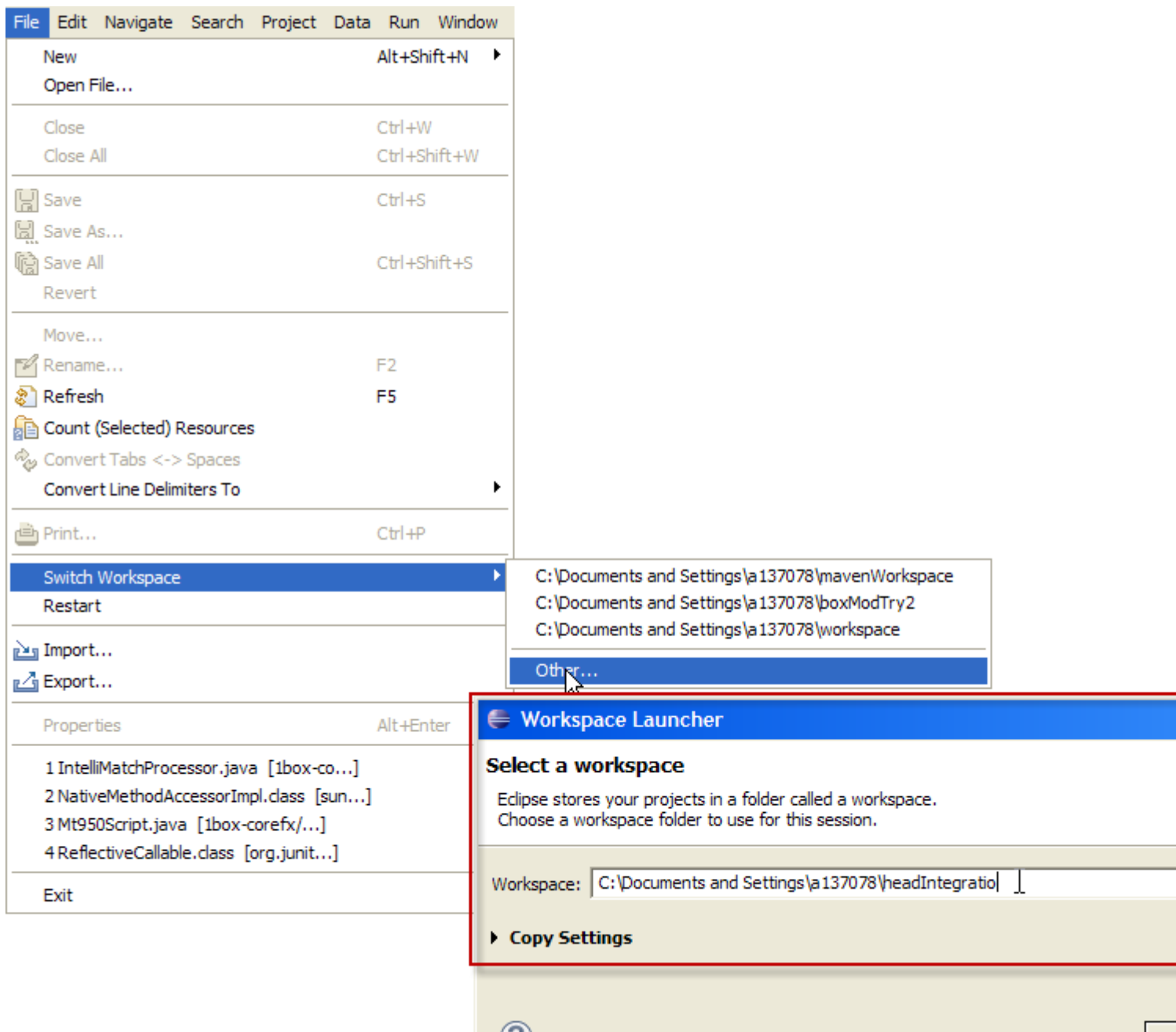
Read Configuring Eclipse online: <https://riptutorial.com/eclipse/topic/2112/configuring-eclipse>

# Chapter 3: Create a new workspace in Eclipse

## Examples

### How to create a workspace

Go to File -> Switch Workspace -> Other... and type in your new workspace name.



Read [Create a new workspace in Eclipse](https://riptutorial.com/eclipse/topic/6345/create-a-new-workspace-in-eclipse) online: <https://riptutorial.com/eclipse/topic/6345/create-a-new-workspace-in-eclipse>

# Chapter 4: Debugging Java programs in Eclipse

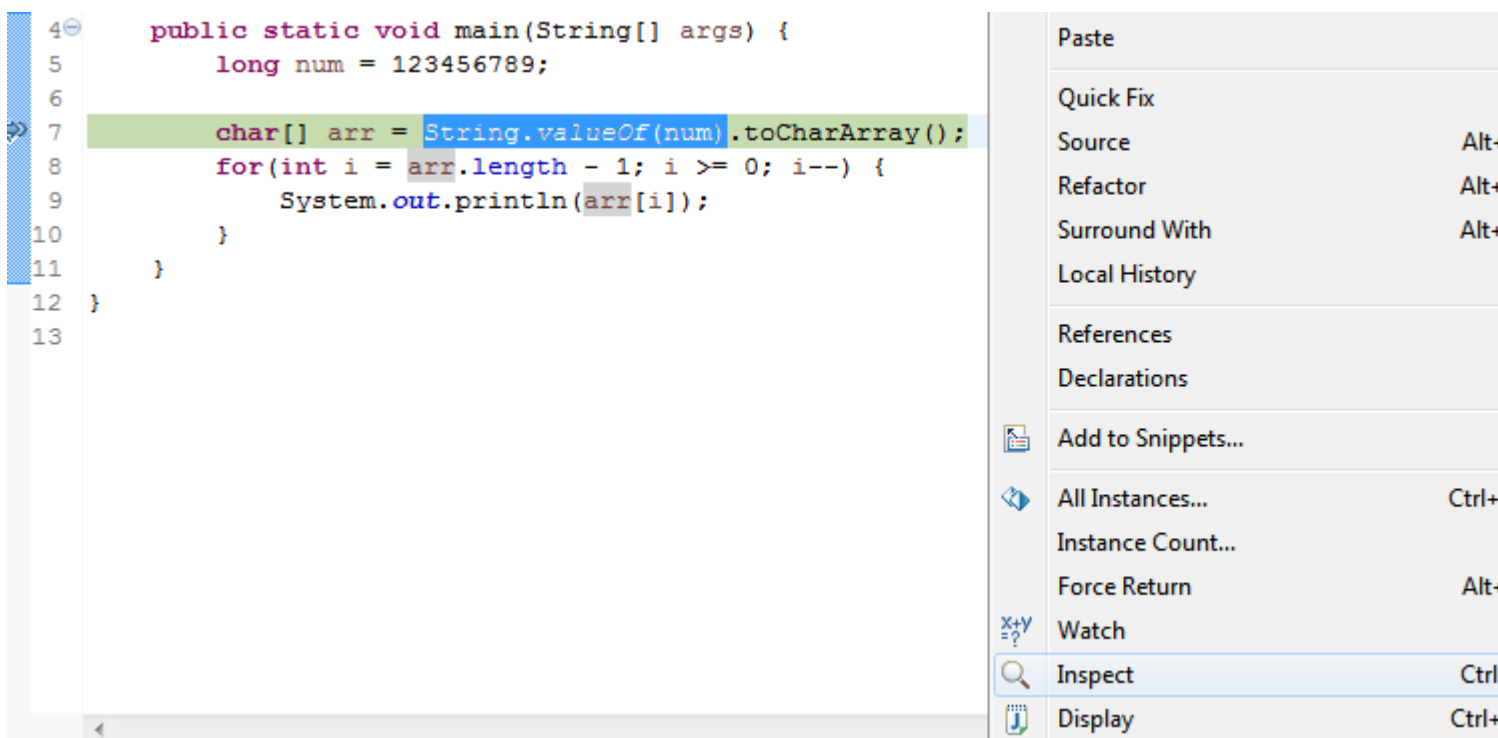
## Examples

### Evaluating expressions within a debugging session

There are several to evaluate a certain expression when debugging a Java application.

#### 1. Manually inspecting an expression

When the program execution is suspended at a certain line (either due to a breakpoint or manually stepping through the debugger), you can manually evaluate an expression by selecting the expression in the code, then right-clicking and selecting **Inspect** as shown in the below screenshot. Alternatively, do `Ctrl+Shift+I` after selecting the expression.



#### 2. Watching an expression in the Expressions view

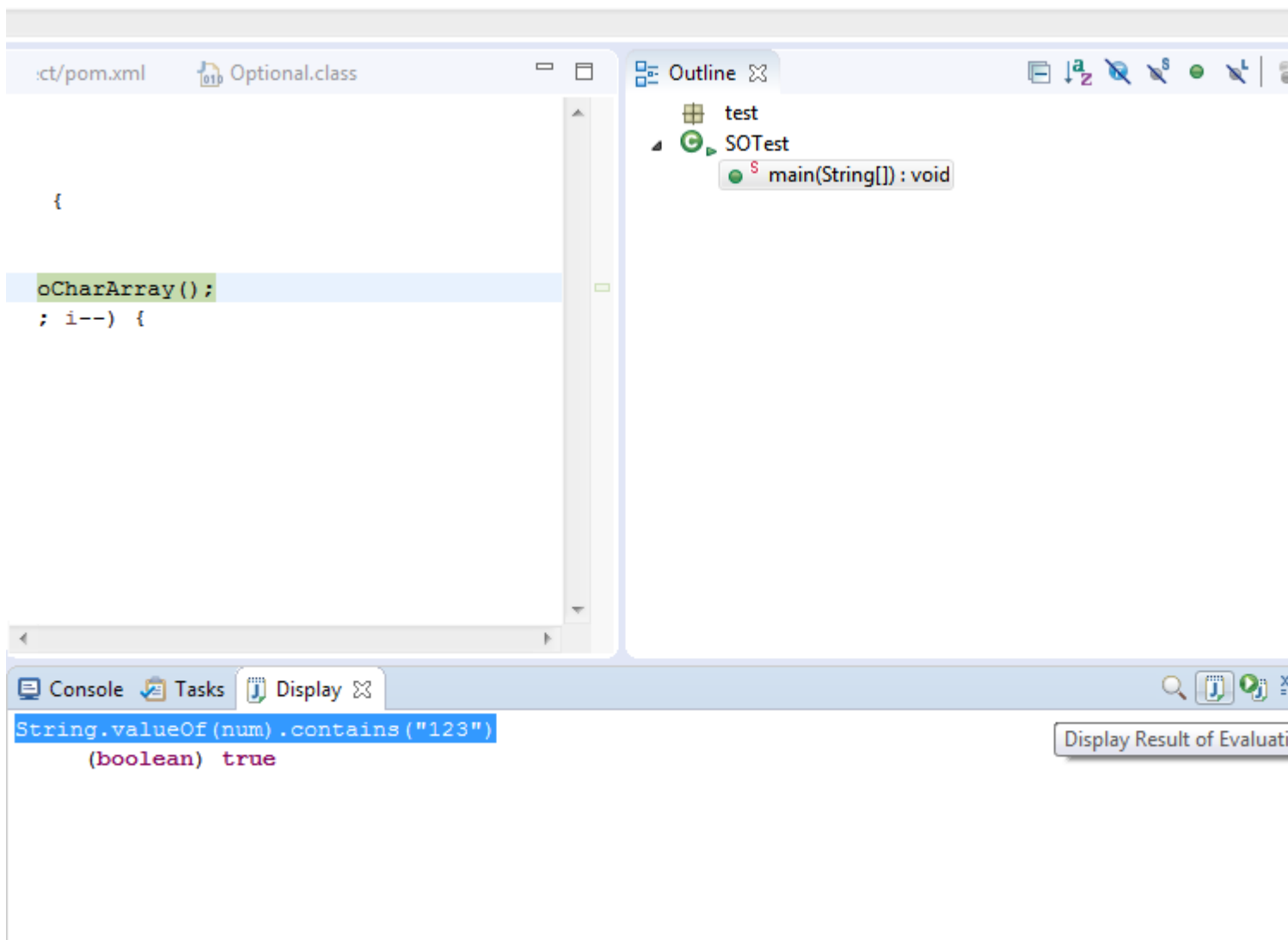
If you want to continuously evaluate an expression, say because it is within in a loop, you can watch it in the Expressions view. This way its value will be displayed whenever the program is suspended at each iteration of the loop. To do this, select the desired expression, then right-click and select **Watch**. This will open the **Expressions** view and show the value of the expression (see below image). You can also manually write the expression in the view.

| Name                             | Value     |
|----------------------------------|-----------|
| <code>String.valueOf(num)</code> | 123456789 |
| + Add new expression             |           |
|                                  |           |
|                                  |           |
|                                  |           |
|                                  |           |
|                                  |           |
|                                  |           |
|                                  |           |

### 3. Using the Display view to evaluate and execute statements

The **Display** view allows you to write your own expressions, statements or any code in general that would be evaluated or executed in context with the suspended program code. This can be useful if you want to evaluate complex expressions without changing your original and restart the debugging.

To open the Display view, select **Window > Show View > Display**. Then write your expression or statements in the view, and select one of the options in the toolbar of the view, for example to execute the written statements, or display the result of evaluating them in the view as shown in the below image. The code written in the Display view can also be inspected or executed by selecting it, then right-clicking and selecting the desired action.



## Remote debugging of a Java application

In order to debug a remote Java application, it should be launched with some extra arguments to instruct the JVM to execute it in debug mode. This is done as follows:

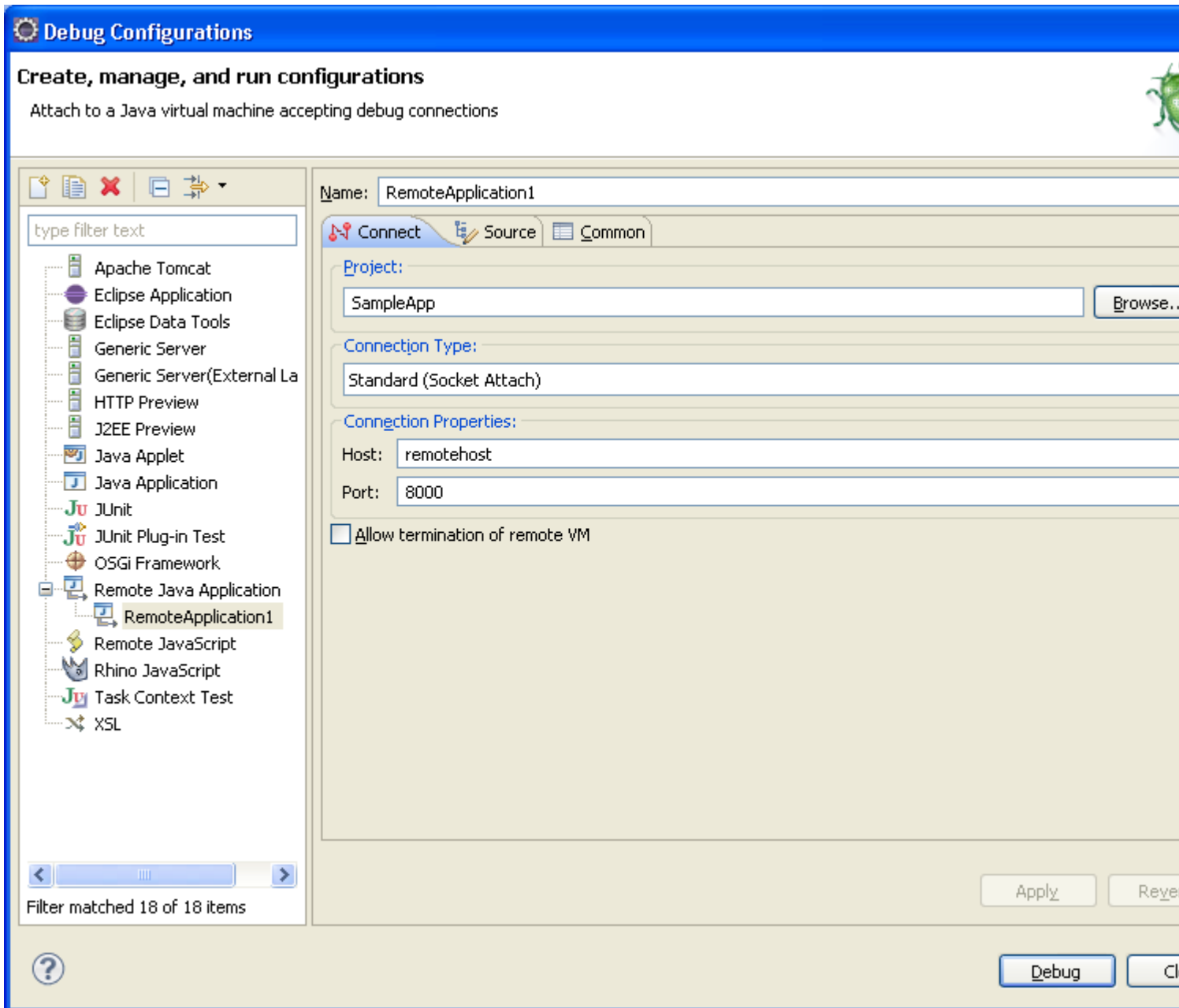
```
java -Xdebug -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=8000 -jar sampleApp.jar
```

The above command tells the JVM to start the application `sampleApp.jar` while having a server socket listening at port `8000` for a debugger to attach to it. The `suspend` argument tells whether you want the JVM to wait until a debugger attaches to the port number before the application effectively runs.

After launching the remote application with the above parameters, the next step is attach your Eclipse debugger to the remote application. This is done as follows:

1. Go to **Run > Debug Configurations...**
2. In the **Debug Configurations** dialog (see figure below), select the **Remote Java Application** section and click **New launch configuration** action.

3. Give your debug configuration a name, then select the project that contains the code of your remote application. Set the hostname or IP address of the remote machine and the port to which the debugger should attach.
4. Click **Debug**. The Eclipse debugger will now attempt to attach to the remote application and suspend at breakpoints set in the source code.



Read Debugging Java programs in Eclipse online:

<https://riptutorial.com/eclipse/topic/4548/debugging-java-programs-in-eclipse>



---

# Chapter 5: Eclipse Shortcuts

## Introduction

Eclipse has many shortcuts to make your life easier.

## Examples

### Comment/Uncomment code

To comment or uncomment code select the lines and use **Ctrl + Shift + C** or **Ctrl + Shift + /**

### Open Resource Dialog

To access the Open Resource dialog use **Ctrl + Shift + R**. From here you can start typing a resource name and it will find all matches in the workspace, this makes it easier to find a file when you don't know exactly where it is.

### To get a println

If you want `System.out.println()`; but don't want to type the whole thing out you can just type **syso** and hit **Ctrl + Spacebar**. It will type the rest and set the cursor between the parenthesis.

### Generate Getters and Setters

Eclipse can generate basic getters and setters for you. Right click in your class file and go to **Source - Generate Getters and Setters** (ALT+SHIFT+S). This will open a dialog where you can choose which fields you would like to have getters and setters generated for.

### Refactor Highlighted Text

Renaming a variable or class is usually a tedious task, by searching for all the locations where it is used. This can be significantly speeded up by highlighting the word, pressing **Alt+Shift+R** and then typing the new word. Eclipse will automatically rename the word in every file where it is called.

### Format xml

When you add entries to an xml or copy from other sources, there often tends to be uneven tabs and spaces around the entries.

When you press **Ctrl + Shift + F**, you easily align the entire document and remove the extra tabs as well. Thus the text gets formatted and eventually becomes readable.

Read Eclipse Shortcuts online: <https://riptutorial.com/eclipse/topic/9387/eclipse-shortcuts>

---

# Chapter 6: How Eclipse Remote Debugging works behind the scenes

## Examples

### How does Eclipse Remote Debugging work behind the scenes

Eclipse debugging starts with what is referred to as Agents.

The **JVM**, which runs the compiled `.class` sources has a feature that allows externally libraries (written in either Java or C++) to be injected into the JVM, just about runtime. These external libraries are referred to as Agents and they have the ability to modify the content of the `.class` files been run. These Agents have access to functionality of the JVM that is not accessible from within a regular Java code running inside the JVM and they can be used to do interesting stuff like injecting and modify the running *source code*, *profiling* etc. Tools like **JRebel** makes use of this piece of functionality to achieve their magic.

And to pass an Agent Lib to a JVM, you do so via start up arguments, using the

```
agentlib:libname[=options] format.
```

We were actually passing an Agent Lib named `jdwp` to the JVM running Tomcat. The `jdwp` is a JVM specific, optional implementation of the *JDWP (Java Debug Wire Protocol)* that is used for defining communication between a debugger and a running JVM. It's implementation, if present is supplied as a native library of the JVM as either `jdwp.so` or `jdwp.dll`

### So what does it do?

In simple terms, the `jdwp` agent we pass is basically serving the function of being a link between the JVM instance running an application and a Debugger (which can be located either remote or local). Since it is an Agent Library, It does have the ability to intercept the running code, create a bridge between the JVM and a debugger, and have the functionality of a debugger applied on the JVM. Since in the JVM architecture, the debugging functionality is not found within the JVM itself but is abstracted away into external tools (that are aptly referred to as debuggers), these tools can either reside on the local machine running the JVM being debugged or be run from an external machine. It is this de-coupled, modular architecture that allows us to have a JVM running on a remote machine and using the JDWP, have a remote debugger be able to communicate with it. In short, this is how Eclipse debugger works.

Read How Eclipse Remote Debugging works behind the scenes online:

<https://riptutorial.com/eclipse/topic/6247/how-eclipse-remote-debugging-works-behind-the-scenes>

---

# Chapter 7: Remote Debugging in Eclipse

## Examples

### Configure Eclipse Remote Debugging for an application

The following are the steps to start an Eclipse remote debugger. This is useful when the application is not started from a server instance within Eclipse. This feature is really powerful and can also help debugging code which resides in the test or production environment. Let's have a look at the settings:

#### Eclipse Settings:

1. Click the Run Button
2. Select the Debug Configurations
3. Select the "Remote Java Application"
4. New Configuration
  - a) Name : GatewayPortalProject
  - b) Project : GatewayPortal-portlet
  - c) Connection Type: Socket Attach
  - d) Connection Properties:
    - i) localhost ii) 8787

#### For JBoss:

1. Change the `/path/toJBoss/jboss-eap-6.1/bin/standalone.conf` in your vm as follows: Uncomment the following line by removing the #:

```
JAVA_OPTS="$JAVA_OPTS -agentlib:jdwp=transport=dt_socket,address=8787,server=y,suspend=n"
```

#### For Tomcat :

In `catalina.bat` file :

Step 1:

```
CATALINA_OPTS="-Xdebug -Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n"
```

Step 2:

```
JPDA_OPTS="-agentlib:jdwp=transport=dt_socket,address=8000,server=y,suspend=n"
```

Step 3: Run Tomcat from command prompt like below:

```
catalina.sh jpda start
```

Then you need to set breakpoints in the Java classes you desire to debug.

Read Remote Debugging in Eclipse online: <https://riptutorial.com/eclipse/topic/3502/remote-debugging-in-eclipse>

---

# Chapter 8: Setting up Eclipse for C++

## Examples

### Linux + CMake ("Unix Makefiles" generator) + Qt (optional)

You should have a plain CMake project **myproject**, and we are going to make an Eclipse workspace outside of it:

```
myproject/  
  .git/  
  CMakeLists.txt  
  src/  
    main.cpp  
workspace/  
  myproject/  
    Release/  
    Debug/
```

## Qt (optional)

- Get latest Eclipse CDT and then install the Qt package in it through "Help -> Install New Software".

## Workspace

- Create an empty "workspace" directory alongside your CMake project source directory.
- Launch Eclipse and switch to that "workspace" directory.
- Create a C++ project (for Qt with Eclipse older than Neon: create "Qt Makefile Project" and then delete \*.pro file, makefile and main.cpp from it)

## Attaching Sources to the Project

- Go to Project Properties -> Paths and Symbols -> Source Location -> Link Folder.
- Check "Advanced" and link the source folder of CMake project like that: `../../myproject/src/`. It works because the workspace is just outside the CMake project directory.

## CMake generator

- Create `Release` folder in the project.
- Go to "Make Target" view (`Ctrl+3` and then type "Make Target" if it's hard to find). "Make Target" view looks similarly to project view.
- Right click on the "Release" folder and then click "New...".
  - Uncheck "Same as target name".
  - Uncheck "Use builder settings".

- Type in "Release" into "Target name" field.
  - Leave "Make target" empty.
  - Set "Build command" to something like `cmake ../../../../myproject/`.
  - Click ok.
- Double click on this "Release" make target that was just created in the Release folder. That will run cmake generation.

## Build

- Go to Project Properties and create a "Release" configuration.
- Make "Release" configuration active.
- For "Release" configuration uncheck "Generate Makefiles automatically".
- Set Build directory to "Release".
- Enable parallel build.

Now, you can build the project from Eclipse with a usual `Ctrl+b` "Build".

## Re-running CMake (to re-generate the makefiles)

- Remove everything from the "Release" directory.
- Go to "Make Target" view.
- Double-click on the "Release" target.

Read [Setting up Eclipse for C++](https://riptutorial.com/eclipse/topic/7028/setting-up-eclipse-for-cplusplus) online: <https://riptutorial.com/eclipse/topic/7028/setting-up-eclipse-for-cplusplus>

---

# Chapter 9: Tomcat deployment procedure

## Examples

### Procedure when nothing else helps

Once a while concecut deploys to internal tomcat start giving constant error, without any clear cause (Listener start or ClassNotFoundException). When nothing seems to cure it, this procedure saves the world:

- 1 delete Servers folder
- 2 restart Eclipse
- 3 create new server, add project and start

Works like charm and is not so lengthy.

If from some reason this fails, my original lengthy procedure (where the other is a short cut that should do the same) is here:

- 1 stop server
- 2 project -> clean
- 3 project build (I had automatic build disabled)
- 4 delete server
- 5 delete Servers folder
- 6 restart Eclipse
- 7 create new server, add project and start.

With this seven step thing problems with deploy never come out of your code and control.

#### **Note:**

*You dont't need else than page refresh if all goes smoothly. This procedure is done once per error message to be sure you get rid of the unclear error, if your code looks ok and you kind of did nothing to receive the error. The error is either containing word ClassNotFoundException or ListenerStart, depending on environment in use. Note also that this does not cure ClassNotFoundExceptions caused by missing libraries in a project.*

Read Tomcat deployment procedure online: <https://riptutorial.com/eclipse/topic/6092/tomcat-deployment-procedure>

# Credits

| S. No | Chapters   | Contributors  |
|-------|--|---|
| 1     | Getting started with eclipse                         | <a href="#">Aaron Vigal</a> , <a href="#">Ala Eddine JEBALI</a> , <a href="#">Aleksandr M</a> , <a href="#">Ani Menon</a> , <a href="#">Community</a> , <a href="#">Latsuj</a> , <a href="#">manouti</a> , <a href="#">Ray</a> , <a href="#">Thomas Fritsch</a> |
| 2     | Configuring Eclipse                                  | <a href="#">Bevor</a> , <a href="#">manouti</a> , <a href="#">mecsko</a> , <a href="#">romeara</a>  |
| 3     | Create a new workspace in Eclipse                    | <a href="#">rajah9</a>  |
| 4     | Debugging Java programs in Eclipse                   | <a href="#">manouti</a>   |
| 5     | Eclipse Shortcuts                                    | <a href="#">Latsuj</a> , <a href="#">Ray</a> , <a href="#">Srishti Sinha</a> , <a href="#">user7491506</a> , <a href="#">Yurii COjocari</a>   |
| 6     | How Eclipse Remote Debugging works behind the scenes | <a href="#">Pritam Banerjee</a>   |
| 7     | Remote Debugging in Eclipse                          | <a href="#">Pritam Banerjee</a>   |
| 8     | Setting up Eclipse for C++                           | <a href="#">Velkan</a>  |
| 9     | Tomcat deployment procedure                          | <a href="#">mico</a>  |