GRADLE BUILD AUTOMATION HANDBOOK

Hot Recipes for Gradle Automation

Gradle

ANDRES CESPEDES

Java Code Geeks
JAVA 2 JAVA DEVELOPERS RESOURCE CENTER

Gradle Build Automation Handbook

Contents

Preface

Gradle is an open source build automation system that builds upon the concepts of Apache Ant and Apache Maven and introduces a Groovy-based domain-specific language (DSL) instead of the XML form used by Apache Maven of declaring the project configuration. Gradle uses a directed acyclic graph ("DAG") to determine the order in which tasks can be run.

Gradle was designed for multi-project builds which can grow to be quite large, and supports incremental builds by intelligently determining which parts of the build tree are up-to-date, so that any task dependent upon those parts will not need to be re-executed. The initial plugins are primarily focused around Java, Groovy and Scala development and deployment, but more languages and project workflows are on the roadmap. (Source: https://en.wikipedia.org/wiki/Gradle)

In this ebook, we provide a compilation of Gradle examples that will help you kick-start your own projects. We cover a wide range of topics, from installation and configuration, to how to use various plugins and how to integrate Gradle with 3rd party tools. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

About the Author

Andres is a Java Software Craftsman from Medellin Colombia, who strongly develops on DevOps practices, RESTful Web Services, Continuous integration and delivery. Andres is working to improve software process and modernizing software culture on Colombia.

Chapter 1

Gradle “Hello World” Tutorial

In this post we’ll look at Gradle, its installation and configuration, and how to automate stages of development and release of software through its base concept, the Gradle tasks.

1.1 What is Gradle ?

Gradle is a build and automation tool, that can automate our building, testing, deploying tasks and many more. Gradle is the next generation build system for Java technologies that includes some advantages from older tools like ant or maven. Let’s have a look:

- Allows declarative and expressive domain-specific-language (DSL). This is a powerful concept because it allows us to write a custom language that is more friendly than Java.
- Is Groovy-based. This means that your configuration is made in Groovy statements instead of xml blocks, making it very easy to define the tasks to be performed.
- Supports legacy scripts in Ant or Maven, and has full support to Ivy repository infrastructure.
- It’s designed to take advantage of convention over configuration.
- Works on non-java projects too.
- Easily customizable and scalable.

1.2 Why Gradle ? I really need a build tool?

Today, we work on large projects that need automated release process to mitigate risks and failures; a building system as Gradle, permits you to structure a process from the compilation to the deployment in your application server. This approach has several advantages, like spending time on more important tasks for our project (like modeling and coding features) and delegating repetitive tasks to Gradle, or also the use of Groovy dynamic language scripts (DSL) instead of too long xml files.

If you want to do continuous delivery and make your release process automatic, Gradle is a nice tool to carry out these goals.

1.3 Downloading Gradle

- At time of this tutorial, Gradle is in your 2.3 version. You can download from here.
- We need a JDK 1.6 before installing Gradle, if you don’t have, you can download from here.

- So, unzip the file in any directory that you choose, in this example we set Gradle in C:/Desarrollo/Lib/gradle-2.3.

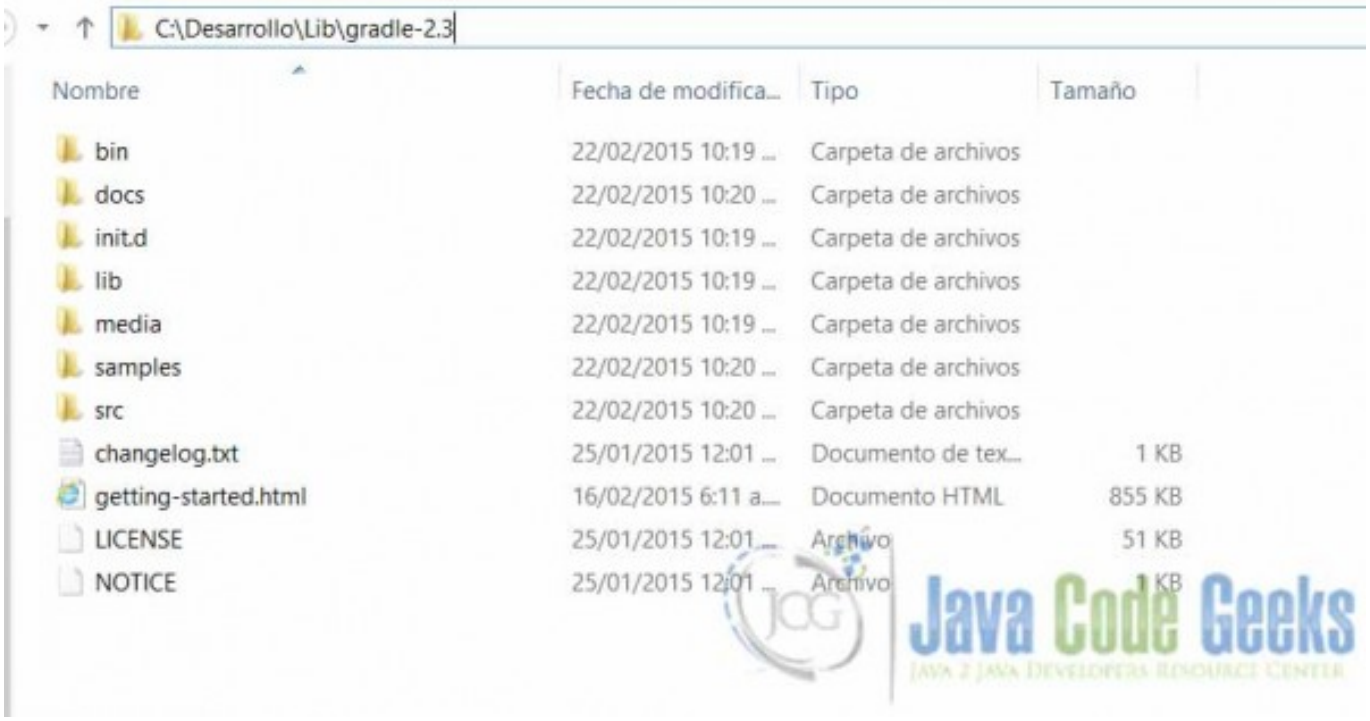

Figure 1.1: Gradle Directory Installation

1.4 Setting environment variables

Then, we have to set enviroment variables to get a full access to Gradle, so create the GRADLE_HOME variable that point to the earlier directory that you set.

Note: The screenshots shown here are taken from Windows 8. Your version of Windows may vary.

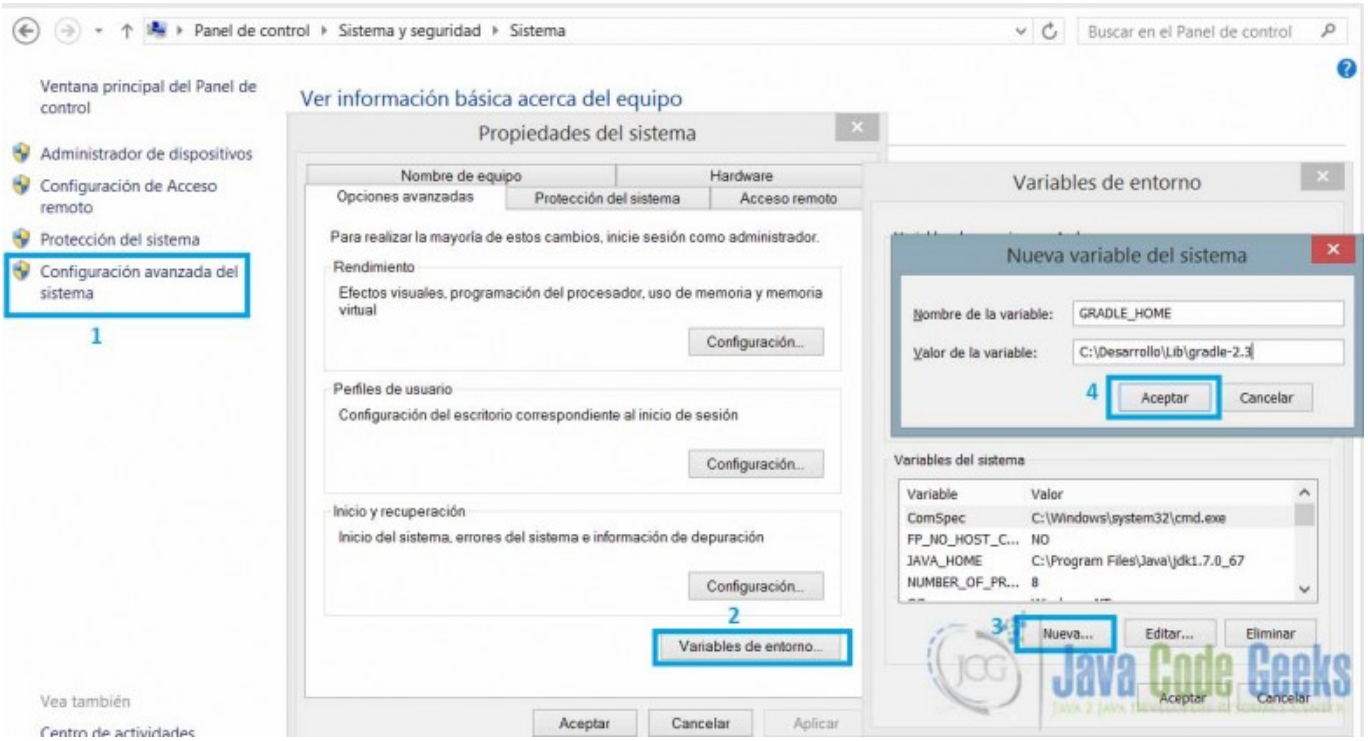

Figure 1.2: Environment Variable Configuration Step By Step

Next, in the PATH variable add the bin directory of the Gradle installation with `%GRADLE_HOME%bin`, with this we can run Gradle from any directory.

Figure 1.3: Path Variable Configuration

To verify that Gradle was successfully installed, go to the console (cmd shell) and run this command: `gradle -v`

```
C:\Users\Andres>gradle -v

------------------------------------
Gradle 2.3
------------------------------------

Build time:   2015-02-16 05:09:33 UTC
Build number: none
Revision:     586be72bf6e3df1ee7676d1f2a3afd9157341274

Groovy:       2.3.9
Ant:          Apache Ant(TM) version 1.9.3 compiled on December 23 2013
JVM:          1.7.0_67 (Oracle Corporation 24.65-b04)
OS:           Windows 8.1 6.3 amd64
```

1.5 Hello World! Gradle

The Gradle's starting point is the `build.gradle` file. Any task or project starts with this script. With the default naming convention, this file is called, but we can define any name to our build gradle scripts.

In the example, the first task of gradle is called helloWorld, with using Groovy language we call the Java's method `System.out.println` with the Groovy's shorter equivalent `println` to print a short message in console.

build.gradle

```
task helloWorld << {
   println 'Welcome to JCG Gradle Tutorial'
}
```

To execute the build script, go to the directory where you saved the file and execute the task, running this command:

`gradle helloWorld`, this will be the output.

```
E:\JavaCodeGeeks JCG\Gradle Tutorial>gradle helloWorld
:helloWorld
Welcome to JCG Gradle Tutorial
BUILD SUCCESSFUL
Total time: 1.838 secs
```

1.6 Gradle JVM Options

As any Java based tool, Gradle can set JVM Options to manage the memory space and another stuff. We can use the environment variables `GRADLE_OPTS` or `JAVA_OPTS`, this configuration prevents an `OutOfMemoryError` setting stable values. For example, we can set the maximum memory allocation pool adding the value `-Xmx512m` in the `GRADLE_OPTS` variable.

1.7 Gradle Basic Concepts

In Gradle we have 2 top concepts, that are Projects and Tasks. Any Gradle script is made up of one or more projects, and every project is made up of one or more tasks.

- A Gradle project is any goal what we want to do with Gradle, assemble a JAR, compile a Java project, run Unit tests, deploy an application.
- A Gradle task is the minimum unit of work, represents an atomic piece of work. The main goal is define some tasks to accomplish a Project.

In our first basic task `helloWorld` we can see what happens:

- When you execute the Gradle helloWorld, Gradle will lookup a task with that name in the default build script build.gradle. If Gradle finds the task, it executes the code with the Groovy engine .
- Every line of code (LOC) between the braces composes the task
- The double << , is the shorthand to define a Gradle task, the longhand is thus:

```
task helloWorld {
        doLast {
                println 'Welcome to JCG Gradle Tutorial'
        }
}
```

Yes, the double << is the short form to define the doLast task's block. We suggest you to use the shorthand way.

Gradle has basic commands that help us to write a more readable and clean code or scripts. Every command have to the shorthand and longhand way to run them.

- --help or -h : Prints out the helper messages.
- --info or -i : Set the logger level to INFO, this level prints a high level of information.

- --debug or -d : Set the Gradle logger level to DEBUG, this is useful for troubleshooting build problems.
- --quiet or -q : Only shows the error messages.
- tasks : Show all availables tasks in the current build script, also displayed the tasks defined by the plugin.
- --gui : Launches the Gradle GUI.

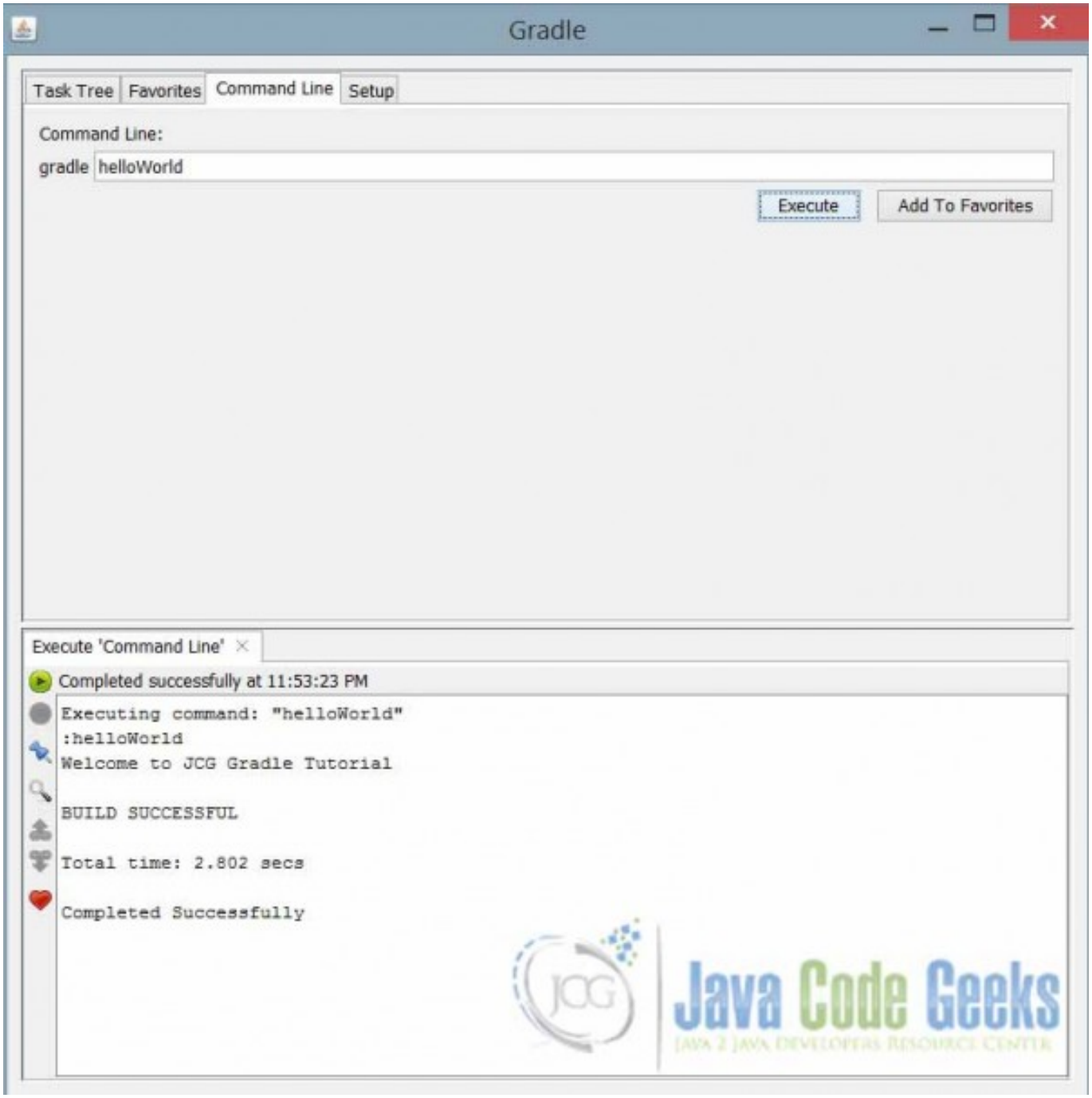

Figure 1.4: Gradle GUI

1.8 Working with Gradle Tasks

In this post we have only worked with the most basic concept tasks, but can we do more than just print some text on the screen?

If we execute the command Gradle tasks, we get the following output:

```
E:\JavaCodeGeeks JCG\Gradle Tutorial>gradle tasks -q

------------------------------------------
All tasks runnable from root project
------------------------------------------

Build Setup tasks
-----------------
init - Initializes a new Gradle build. [incubating]
wrapper - Generates Gradle wrapper files. [incubating]

Help tasks
----------
components - Displays the components produced by root project 'Gradle Tutorial'. [ ↩
    incubating]
dependencies - Displays all dependencies declared in root project 'Gradle Tutorial'.
dependencyInsight - Displays the insight into a specific dependency in root project 'Gradle ↩
     Tutorial'.
help - Displays a help message.
projects - Displays the sub-projects of root project 'Gradle Tutorial'.
properties - Displays the properties of root project 'Gradle Tutorial'.
tasks - Displays the tasks runnable from root project 'Gradle Tutorial'.

Other tasks
-----------
helloWorld

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task
```

Displays the list of tasks that we do, including default tasks and in other tasks that we define. So, Gradle is like an agent that perform tasks to carry out the projects that we model.

1.8.1 Default Tasks

Another important concept is the Default Tasks, which are the tasks that run if no task name indicated. Modify the build.gradle script like this:

```
defaultTasks 'beforHelloWorld'

task helloWorld << {
        println 'Welcome to JCG Gradle Tutorial'
}

task beforHelloWorld << {
        println 'Setting the previous configuration...'
}
```

If we run `gradle` in the console, it will execute the `beforHelloWorld` task.

```
E:\JavaCodeGeeks JCG\Gradle Tutorial>gradle
:beforHelloWorld
Setting the previous configuration...
```

```
BUILD SUCCESSFUL

Total time: 2.685 secs
```

1.8.2 Task Dependency

The biggest basic concept that we cover in this tutorial is task dependency. What does that mean? Gradle adds prevalence on execution of dependency-on task instead of the task that depends on it. Modify the build.gradle file as follows:

```
defaultTasks 'beforHelloWorld'

task helloWorld << {
        println 'Welcome to JCG Gradle Tutorial'
}

task beforHelloWorld (dependsOn:helloWorld) << {
        println 'Setting the previous configuration...'
}
```

If we run `gradle -q` The output will be:

```
E:\JavaCodeGeeks JCG\Gradle Tutorial>gradle -q
Welcome to JCG Gradle Tutorial
Setting the previous configuration...
```

1.8.3 Abbreviated Task Execution

The last basic and useful tip about Gradle tasks, is the abbreviated calling.

In Gradle, if our task's name is so long, we don't need to write the complete name to execute them, only specifying the initials in a camelCase format can execute the tasks.

If we run `gradle -q hW` the output will be:

```
E:\JavaCodeGeeks JCG\Gradle Tutorial>gradle -q hW
Welcome to JCG Gradle Tutorial
```

Gradle using regular expressions matches the name of task (helloWorld) and execute it.

1.9 Conclusions

- Gradle combining the capabilities of earlier build tools such as Ant and Maven allows you to create flexible and maintainable scripts that are interoperable with Java-based technologies.
- The most important concept are the tasks, they are the unit of work in Gradle.
- With predefined tasks we don't need to make everything by hand.
- Gradle is a user-friendly framework, which can be adopted very quickly compared to its predecessors.

1.10 Download the Gradle Scripts

Download

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

Chapter 2

Gradle Properties: Build and Configuration Example

In this detailed Gradle Properties tutorial, we shall see how to access several default properties in Gradle, and how to set our custom properties. Also, what methods are to set these properties and how Gradle processes them.

2.1 Why should we use Gradle Properties?

As developers, we provide high-quality code that is scalable and easy to read; Gradle's properties allow us to perform these tasks, in addition to some cases optimize the compilation and execution of them.

2.2 What do we need?

As described in the previous example (Gradle Hello World Tutorial), we need Gradle fully functional. Please review the part of the installation and then continue with this example.

2.3 Types of Gradle's Properties

In Gradle there are two types of properties, `System properties` and `Project properties`. The first of them, are properties used to configure the JVM that supports the execution of the builds, and the second type are for parameterizing variables in our projects, like names, paths, and other useful approaches.

2.3.1 System Properties

We can set these properties via environment variables, setting the GRADLE_OPTS or JAVA_OPTS values to manage, for example the Java home, Heap and PermGem Spaces, or Gradle daemon activation.

Then, go to the environment variables in windows users, and set the `GRADLE_OPTS` variable with the value `-Dorg.gradle.daemon=true`.

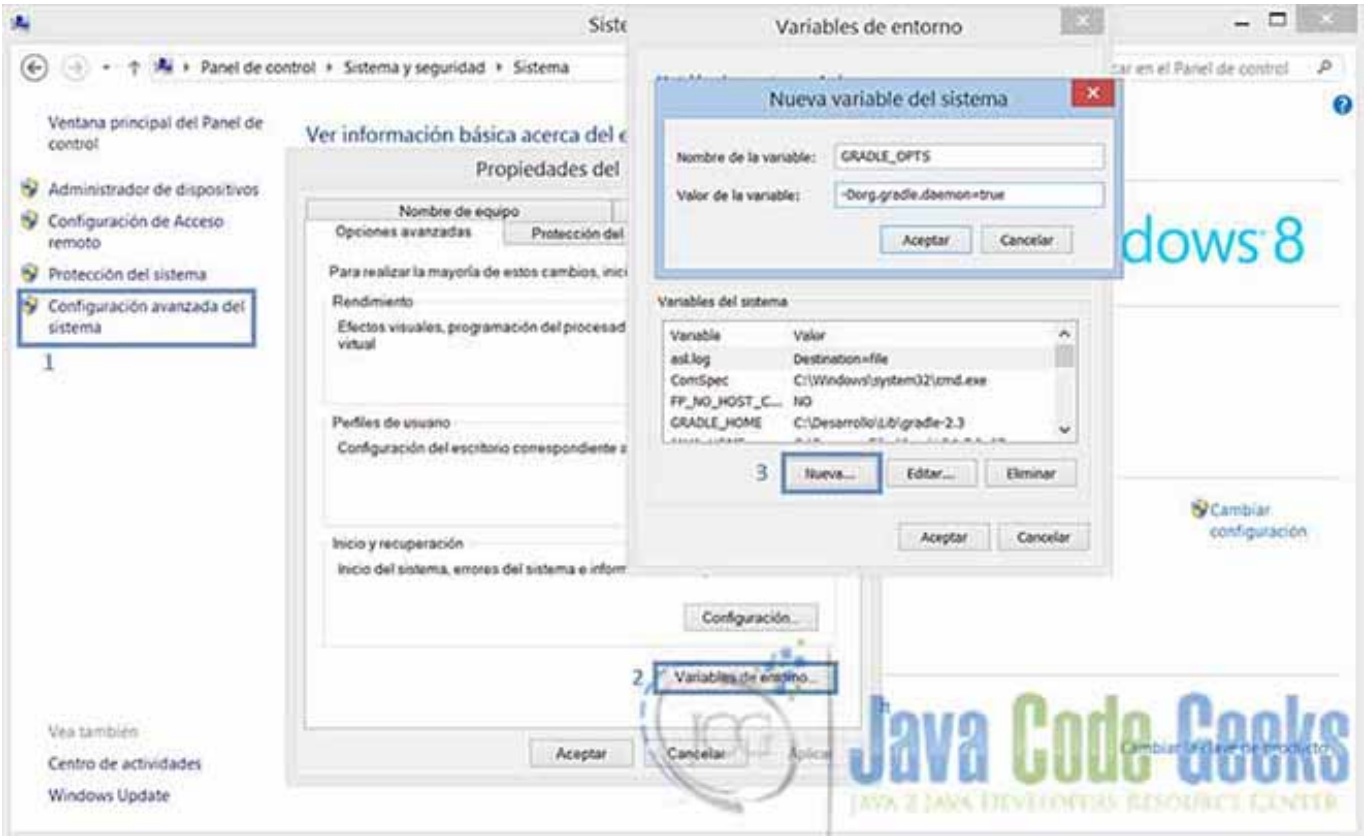

Figure 2.1: Gradle System Properties Configuration

With this property, we active a Gradle daemon that loads all the libraries and essential classes for the build, and when we run Gralde, only the build is executed, because the JVM and all other required libraries are already running, establishing a powerful improvement.

See how this affects the build's time.

Without daemon.

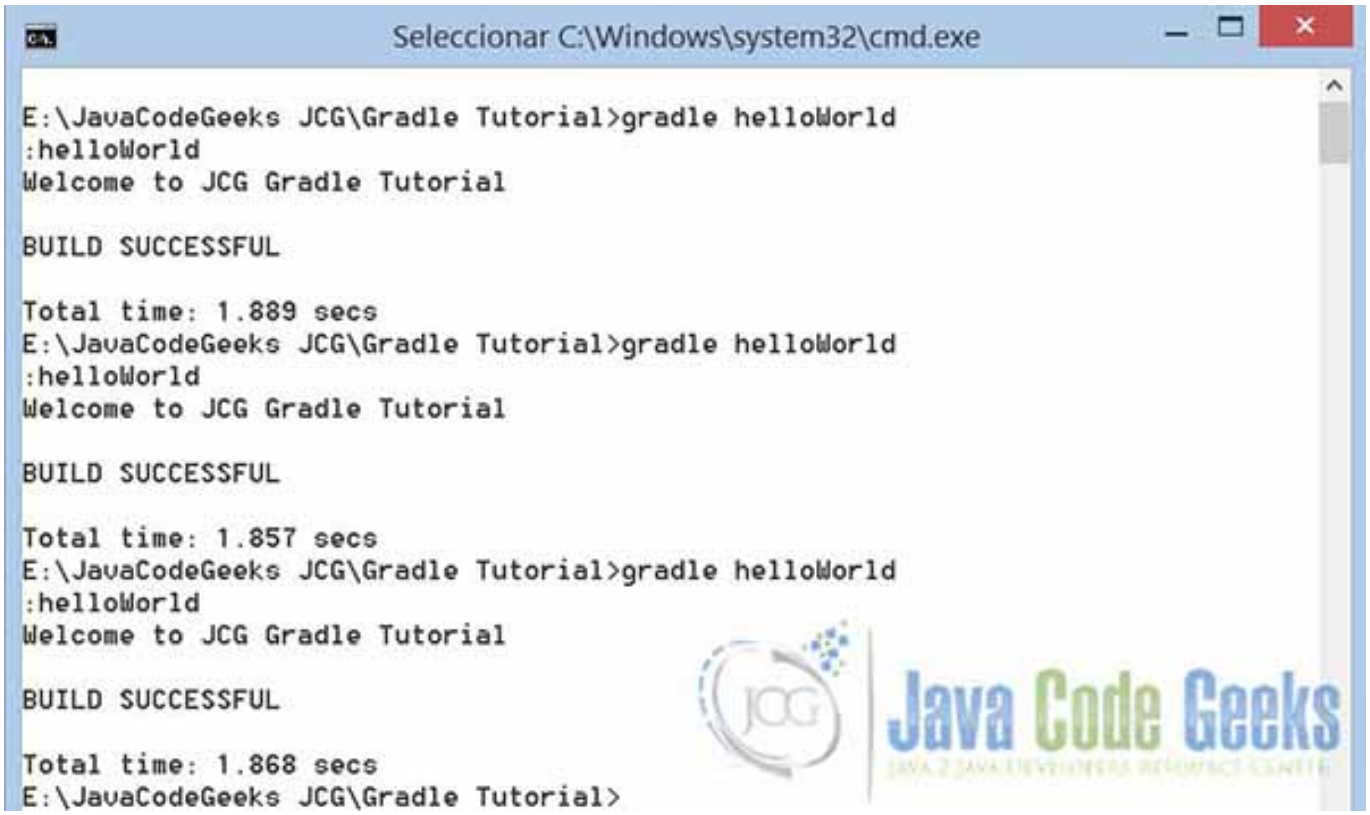

Figure 2.2: Gradle Build no Daemon

Daemon activated.

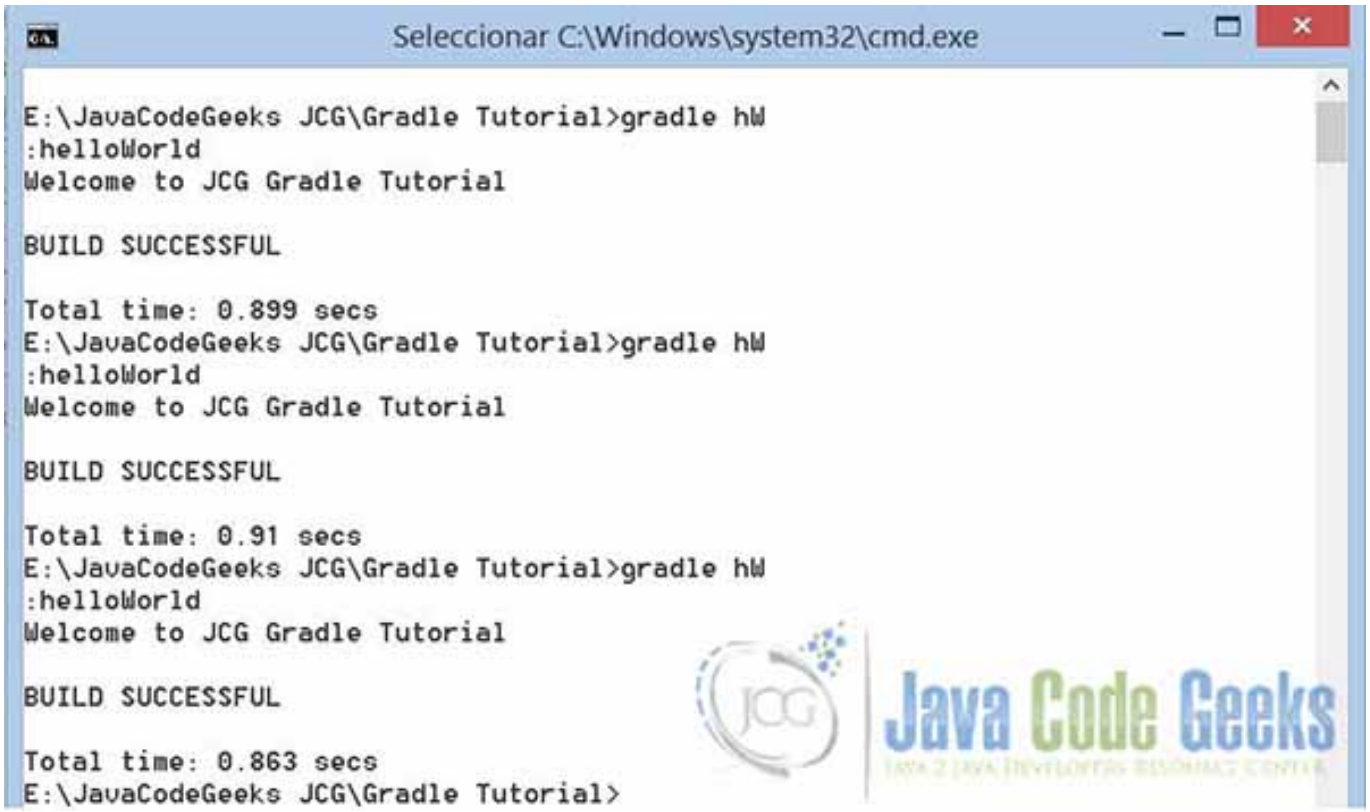

Figure 2.3: Gradle Build Daemon ON

Setting system properties, we can manage common properties to all projects, and how to improve the performance with JVM memory management and gradle's daemon.

In addition, we can set these properties in console with the -D Java command option. See how to do that.

```
E:\JavaCodeGeeks JCG\Gradle Properties Tutorial>gradle helloWorld -Dorg.gradle.daemon=false
:helloWorld
Welcome to JCG Gradle Tutorial

BUILD SUCCESSFUL

Total time: 1.873 secs
```

There are some other System Properties:

- **org.gradle.java.home**, Specifies the Java Home for the Gradle build process.
- **org.gradle.jvmargs**, Manage the memory settings for the JVM
- **org.gradle.configureondemand**, Enables a new incubating and building mode, that's an advanced feature when we work with multi-projects, which results in a faster builds.
- **org.gradle.parallel**, Enables the parallel mode for incubating mode.

2.4 Project Properties

2.4.1 Gradle's Properties, hello!

Gradle offers a wide list of default properties for the current build, we can print all running this simple command in the console: `gradle properties`. This is equivalent to the Properties in the Gradle "Hello World" Example.

```
E:\JavaCodeGeeks JCG\Gradle Properties Tutorial>gradle properties
:properties

------------------------------------------
Root project
------------------------------------------

allprojects: [root project 'Gradle Properties Tutorial']
ant: org.gradle.api.internal.project.DefaultAntBuilder@33666fbb
antBuilderFactory: org.gradle.api.internal.project.DefaultAntBuilderFactory@7e10
cc38
artifacts: org.gradle.api.internal.artifacts.dsl.DefaultArtifactHandler_Decorate
d@4065a018
asDynamicObject: org.gradle.api.internal.ExtensibleDynamicObject@7ff37476
baseClassLoaderScope: org.gradle.api.internal.initialization.DefaultClassLoaderS
cope@68a2866d
beforHelloWorld: task ':beforHelloWorld'
buildDir: E:\JavaCodeGeeks JCG\Gradle Properties Tutorial\build
buildFile: E:\JavaCodeGeeks JCG\Gradle Properties Tutorial\build.gradle
buildScriptSource: org.gradle.groovy.scripts.UriScriptSource@608a1b4e
buildscript: org.gradle.api.internal.initialization.DefaultScriptHandler@65037d6
3
childProjects: {}
class: class org.gradle.api.internal.project.DefaultProject_Decorated
classLoaderScope: org.gradle.api.internal.initialization.DefaultClassLoaderScope
@5ba0df26
components: []
configurationActions: org.gradle.configuration.project.DefaultProjectConfigurati
onActionContainer@57628c1c
configurations: []
convention: org.gradle.api.internal.plugins.DefaultConvention@18740ad1
defaultTasks: [beforHelloWorld]
dependencies: org.gradle.api.internal.artifacts.dsl.dependencies.DefaultDependen
cyHandler_Decorated@3f5aa04b
depth: 0
description: null
ext: org.gradle.api.internal.plugins.DefaultExtraPropertiesExtension@2b90cc9
extensions: org.gradle.api.internal.plugins.DefaultConvention@18740ad1
fileOperations: org.gradle.api.internal.file.DefaultFileOperations@5a6bc081
fileResolver: org.gradle.api.internal.file.BaseDirFileResolver@3e3155a8
gradle: build 'Gradle Properties Tutorial'
group:
helloWorld: task ':helloWorld'
inheritedScope: org.gradle.api.internal.ExtensibleDynamicObject$InheritedDynamic
Object@247eb222
logger: org.gradle.api.logging.Logging$LoggerImpl@767c9c02
logging: org.gradle.logging.internal.DefaultLoggingManager@6aed5294
modelRegistry: org.gradle.model.internal.registry.DefaultModelRegistry@49d7572c
module: org.gradle.api.internal.artifacts.ProjectBackedModule@556edb6b
name: Gradle Properties Tutorial
parent: null
parentIdentifier: null
path: :
pluginManager: org.gradle.api.internal.plugins.DefaultPluginManager_Decorated@26
```

```
1bf0a0
plugins: [org.gradle.api.plugins.HelpTasksPlugin@751f265a]
processOperations: org.gradle.api.internal.file.DefaultFileOperations@5a6bc081
project: root project 'Gradle Properties Tutorial'
projectDir: E:\JavaCodeGeeks JCG\Gradle Properties Tutorial
projectEvaluationBroadcaster: ProjectEvaluationListener broadcast
projectEvaluator: org.gradle.configuration.project.LifecycleProjectEvaluator@703
91e42
projectRegistry: org.gradle.api.internal.project.DefaultProjectRegistry@600e7a58

properties: {...}
repositories: []
resources: org.gradle.api.internal.resources.DefaultResourceHandler@3e4eede1
rootDir: E:\JavaCodeGeeks JCG\Gradle Properties Tutorial
rootProject: root project 'Gradle Properties Tutorial'
scriptHandlerFactory: org.gradle.api.internal.initialization.DefaultScriptHandle
rFactory@6867946a
scriptPluginFactory: org.gradle.configuration.DefaultScriptPluginFactory@579ab80
1
serviceRegistryFactory: org.gradle.internal.service.scopes.ProjectScopeServices$
5@40f76015
services: ProjectScopeServices
standardOutputCapture: org.gradle.logging.internal.DefaultLoggingManager@6aed529
4
state: project state 'EXECUTED'
status: release
subprojects: []
tasks: [task ':beforHelloWorld', task ':helloWorld', task ':properties']
version: unspecified

BUILD SUCCESSFUL

Total time: 0.958 secs
```

These are a lot of properties, we only want to know how to read the default project properties.

2.4.2 Custom Project Properties in Script

If we want to add our custom properties, we have to define them in a build script with the `ext{}` block.

```
ext{
        myFirstProperty = 'myCustomPropertyValue'
}

task showPropertiesTask << {
        println myFirstProperty
}
```

After running the build file, we get this output:

```
E:\JavaCodeGeeks JCG\Gradle Properties Tutorial>gradle sPT
:showPropertiesTask
myCustomPropertyValue

BUILD SUCCESSFUL

Total time: 1.344 secs
```

2.4.3 Setting Properties in Command Line

Instead of setting the properties in a build script (that's not a good practice, the best approach is a properties file), we can use the -P command-line option to add any custom property.

Then, the build script has a task that prints the value of two properties.

```
task showPropertiesTask << {
        println "Version: $version"
        println "Custom property: $customProperty"
}
```

Running the script with this command `gradle -Pversion=2.3.3 -PcustomProperty=myCustomValue showPropertiesTask`, we get the following output.

```
E:\JavaCodeGeeks JCG\Gradle Properties Tutorial>gradle -Pversion=2.3.3 -PcustomProperty= ↩
    myCustomValue showPropertiesTask
:showPropertiesTask
Version: 2.3.3
Custom property: myCustomValue

BUILD SUCCESSFUL

Total time: 2.67 secs
```

2.4.4 Setting Properties using an external file

So, we look at the optimal way to set our project properties, through an external file. The file needs to be named `gradle.properties` and it should be a plain text file with the name of the property and its value in the same line.

We now create a gradle.properties file in the same directory of our build.gradle file, with the following contents:

gradle.properties

```
version = 3.0
customProperty = This value is obtained from gradle.properties
```

We use our previous build script to show the properties, after running the gradle script we get the following output:

```
E:\JavaCodeGeeks JCG\Gradle Properties Tutorial>gradle showPropertiesTask
:showPropertiesTask
Version: 3.0
Custom property: This value is obtained from gradle.properties

BUILD SUCCESSFUL

Total time: 0.943 secs
```

A useful concept, is the main Gradle directory. If we put the gradle.properties there, these properties take precedence over those that are in the project. In windows this directory is `$USER_HOME/.gradle`.

Figure 2.4: Gradle User Home Properties

gradle.properties

```
version = 4.0
customProperty = This value is obtained from gradle.properties in user home
```

Then, we add another gradle.properties file in this directory and run the gradle build, we get the following output.

```
E:\JavaCodeGeeks JCG\Gradle Properties Tutorial>gradle showPropertiesTask
:showPropertiesTask
Version: 4.0
Custom property: This value is obtained from gradle.properties in user home

BUILD SUCCESSFUL

Total time: 1.45 secs
```

2.5 Key Points

So, here are some tips to take note of:

- We can set 2 types of properties, for tunning memory management and to customize our projects.
- We can pass both properties in command-line with the -D and -P options.
- A property defined in the properties file in the Gradle user home directory overrides the property values defined in a properties file in the project directory.
- We can set our gradle properties in the Gradle home user directory or in a project directory.
- The first directory where gradle lookup for properties is $USER_HOME/.gradle
- Setting the Gradle daemon, we get a better performance in the build process.

2.6 Download the Gradle Scripts

Download

You can download the full source code of this example here: **Gradle Configuration Properties Tutorial**

Chapter 3

Gradle GWT Integration Example

This example offers a complete tutorial on how to integrate Gradle & GWT. Then, we see how build a GWT project with Gradle and how to run them on a Jetty Server.

3.1 Introduction to Gradle GWT Integration

GWT is the Google's Toolkit to develop ajax applications using the Java language, these applications like Java EE's Applications need a server, but, in this case, is a lightweight web server like Jetty to make them run. So, and why I need Gradle here? Gradle facilitates and improves the build process with the automatization of the tasks and allowing the application's uncoupling of an IDE, making your release cycle easier!

Before we start with this example, we will cover some essentials topics about Gradle.

3.1.1 Gradle Plugins

Gradle as a build tool offers a set of basic tasks to perform simple compilations, such as wars generation, copying files and creating directories, but in more complex projects like GWT we need to add some plugins to execute advanced tasks. With adding this line at the top of your gradle build script, you are able to use another task.

```
apply plugin:'PLUGIN_NAME'
```

For this example, we need to apply various plugins like war, java and eclipse plugins; with them we can execute this tasks:

- `compileJava:` Compile Java Source files using javac.
- `classes:` Assembles the production classes directory.
- `jar:` Assembles the JAR file.
- `war:` Generates a war file with all the compiled classes.
- `eclipse:` Generates all Eclipse files.

3.1.2 Dependency Management

This is a Gradle's critical feature with a main purpose, to centralize and automate management of libraries dependency, people thinks that dependency management is only an automated fetching of dependencies from a remote source, but this feature wants more like to includes automatically fetching dependencies, detecting transitive dependency conflicts, and so on. For this example, we work declaring dependencies via DSL.

```
dependencies {
compile 'com.google.gwt:gwt-user:2.7.0'
}
```

3.1.3 Source Sets (Java Plugin)

This is a key point from the Java Plugin, a source set is a collection of source files which are compiled together. But for this example we need to know that for this Java Plugin, source sets has a default configuration to works:

- Java source code directory: `src/main/java`
- Production resources: `src/main/resources`
- Test sources: `src/test/java`
- Test resources: `src/test/resources`

3.2 What do we need?

- As IDE: Eclipse Luna 4.4
- Eclipse Gradle Plugin (Optional)
- GWT Eclipse Plugin
- JDK 1.7_75 or higher
- Gradle 2.3
- GWT 2.7.0
- GWT Steffen Schaefer plugin (Compile GWT Plugin)

We work in Eclipse as a prefer IDE, then we need to install Gradle and GWT Plugins for Eclipse. To make easier the integration we use GWT plugin, to do compilation from Java to Javascript files.

3.3 Environment Configuration

3.3.1 Gradle's Installation

Start, with downloading the Gradle 2.3 version and setting environment variables correctly thus:

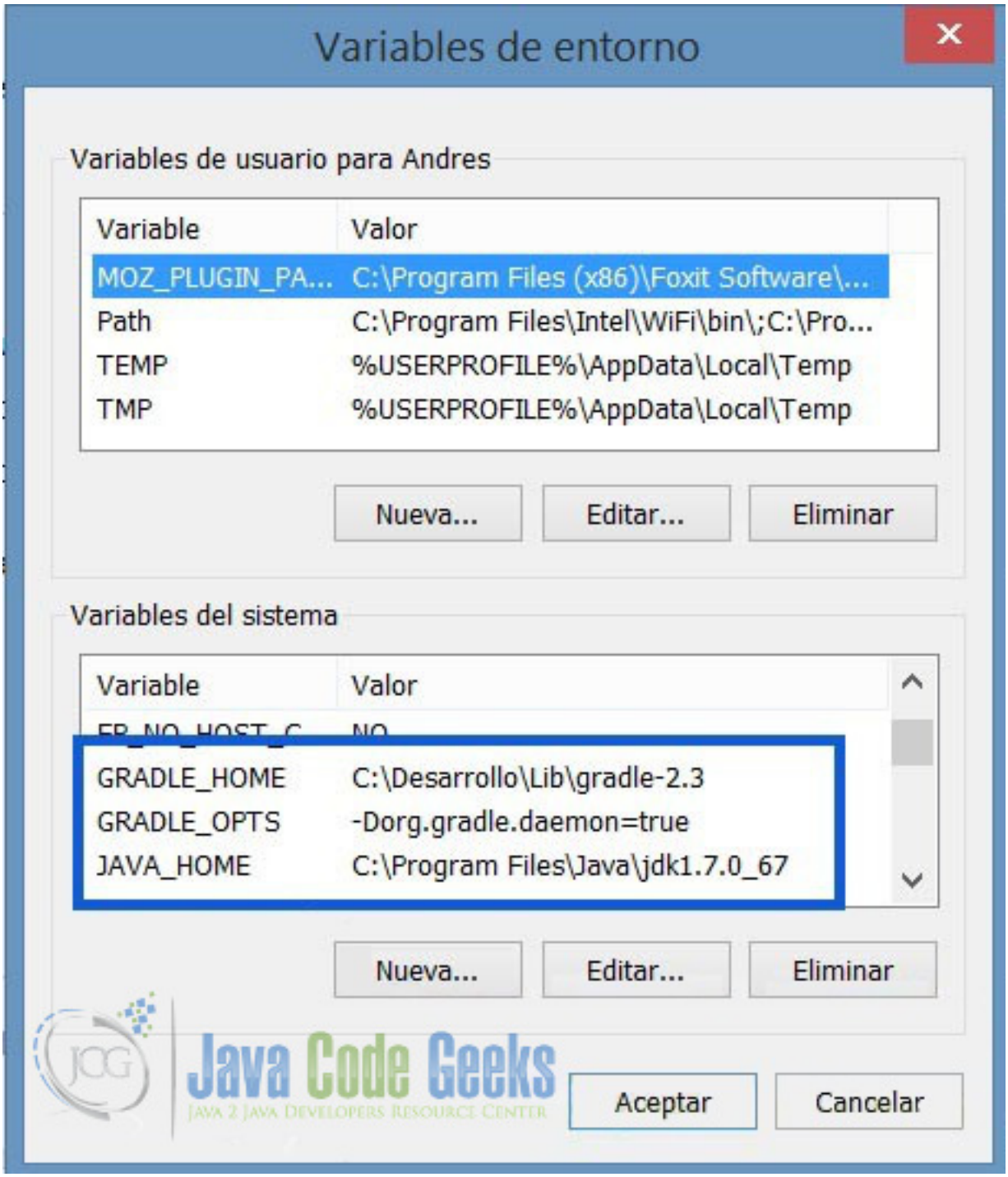

Figure 3.1: Gradle Environment Variable Setting

3.3.2 Eclipse Gradle & GWT Plugins

In the Eclipse Marketplace just type: *Eclipse Integration* and install the plugin that matches with your Eclipse version (for this example 4.4).

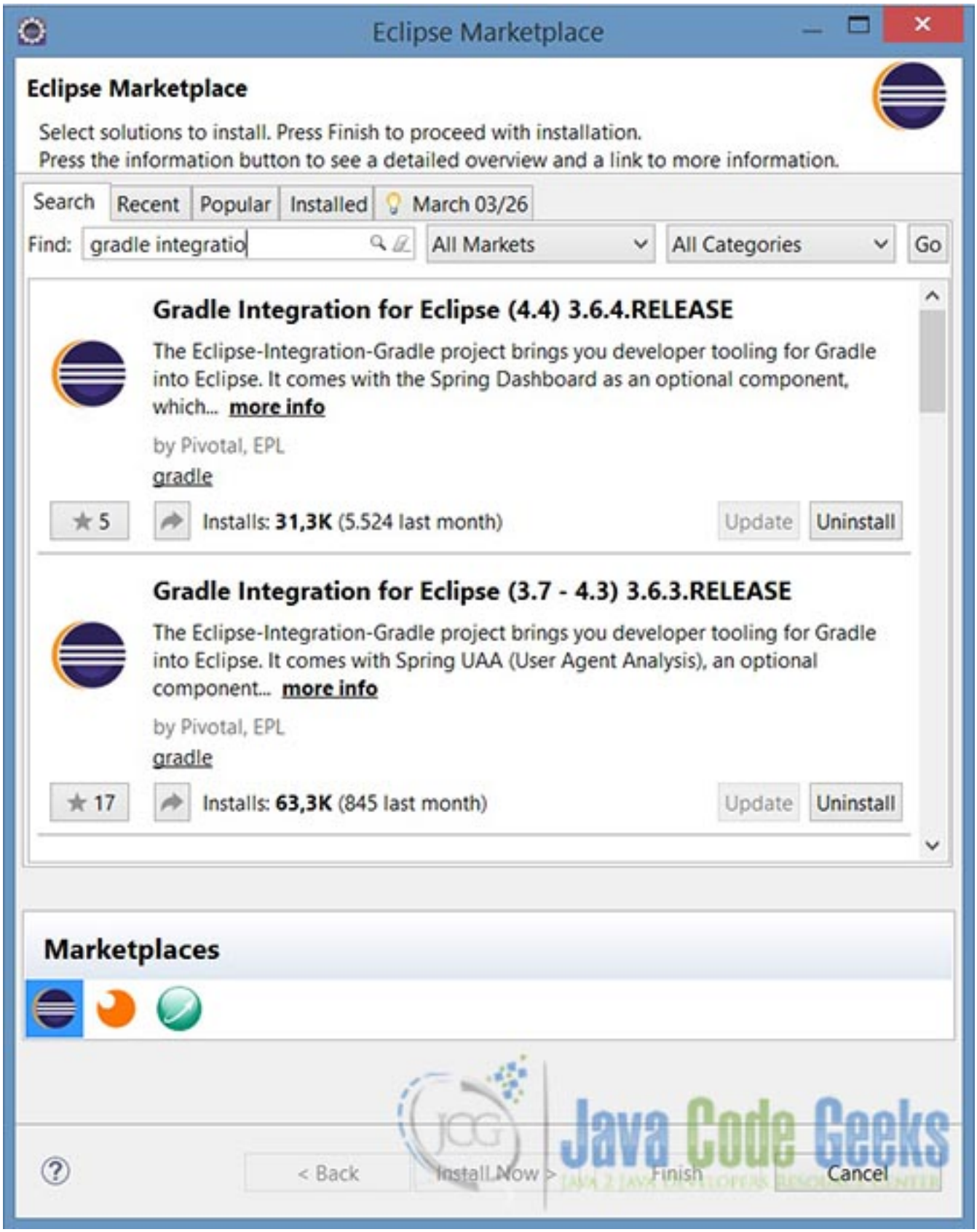

Figure 3.2: Gradle Eclipse Plugin

To install the GWT Plugin you have to go Help > Install New Software, then add this site `https://dl.google.com/eclipse/plugin/4.4`

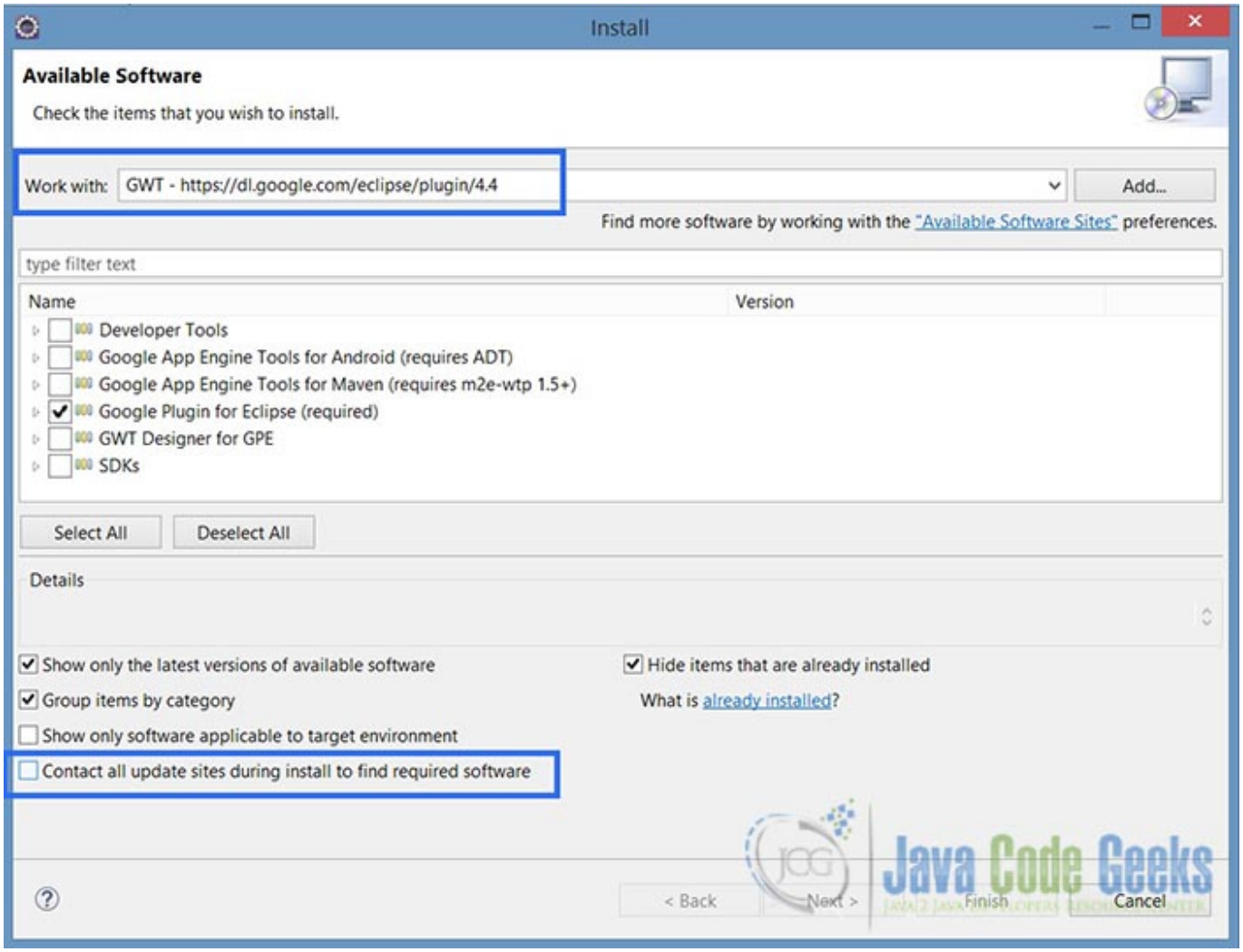

Figure 3.3: GWT Plugin Installation

Please unmark the selected checkbox to avoid problems.

3.3.3 Install and Configure GWT SDK

Finally, download GWT 2.7 distribution and unzip in some directory, then set as the default GWT SDK in Eclipse. Go to the Menu, Window > Preferences > Google > Web Toolkit, so add your GWT specifying the unzip location

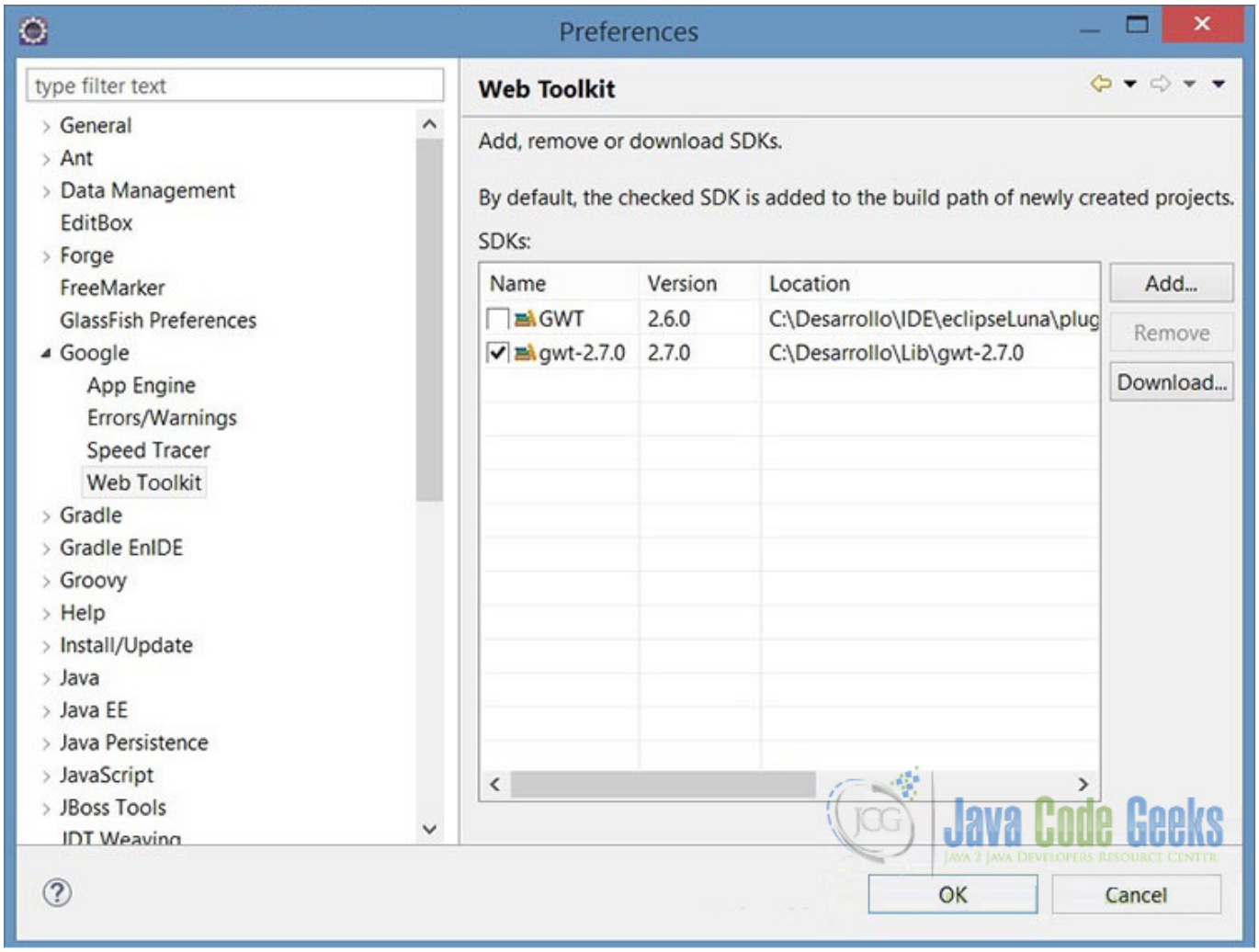

Figure 3.4: GWT SDK Eclipse Setting

With these steps, we are done to work in the integration between Gradle and GWT.

3.4 Creating GWT Project

To start, we need to have already installed the GWT plugin in Eclipse. Then, using the Eclipse Wizard we will create a Google Web Application Project

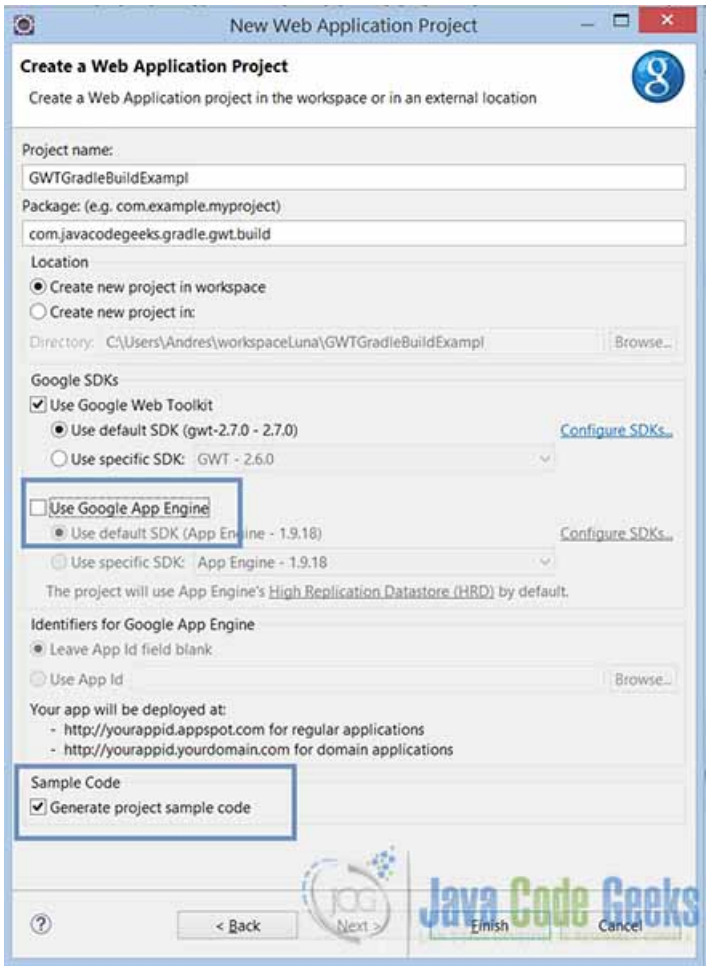

Figure 3.5: GWT Gradle Project Wizard

In this step is necessary to unmark the Google App Engine check because we don't need to publish this example in the Google repositories and to verify that the sample code generation is checked. After that, we have a valid GWT project to build and run with Gradle.

But first, we need to test that the **project runs on Super Dev Mode.**

What is GWT Super Dev Mode?

To run a GWT Application on a Web Explorer, we need to install a gwt plugin for older versions like Firefox 24 or Chrome 21, but if we use this upgraded mode, we can run a GWT application without the annoying plugin. Be aware that the Super Dev Mode is only available GWT 2.5+

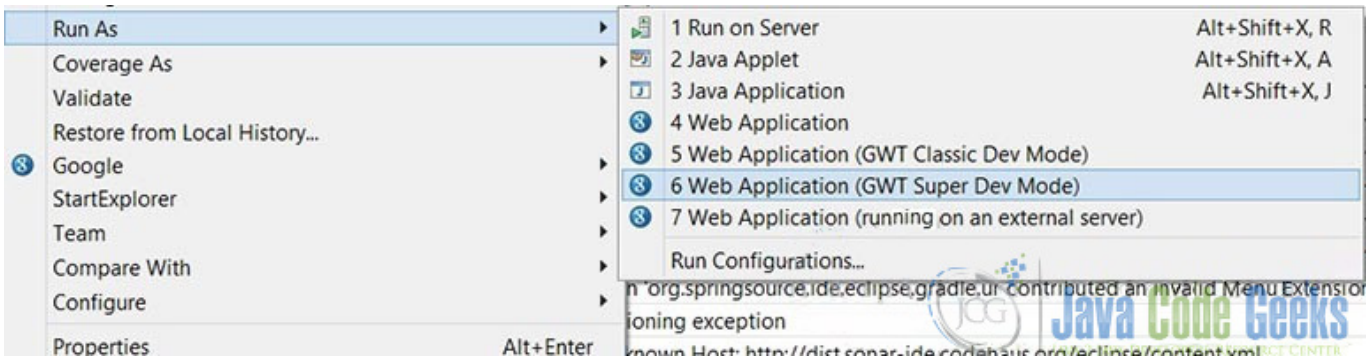

Figure 3.6: GWT Running Super Dev Mode

Then, make a right click on your GWT project > Run As > GWT Super Dev Mode.

Even if you want to run as a classic mode, you can do that only if you have the GWT Plugin installed in your web explorer. The main goal here is to test that the GWT Project runs correctly.

3.5 Setting Gradle in GWT Project

To continue, we have to create the main file, Gradle Build Script, so, make a new File called `build.gradle` in the root of the project.

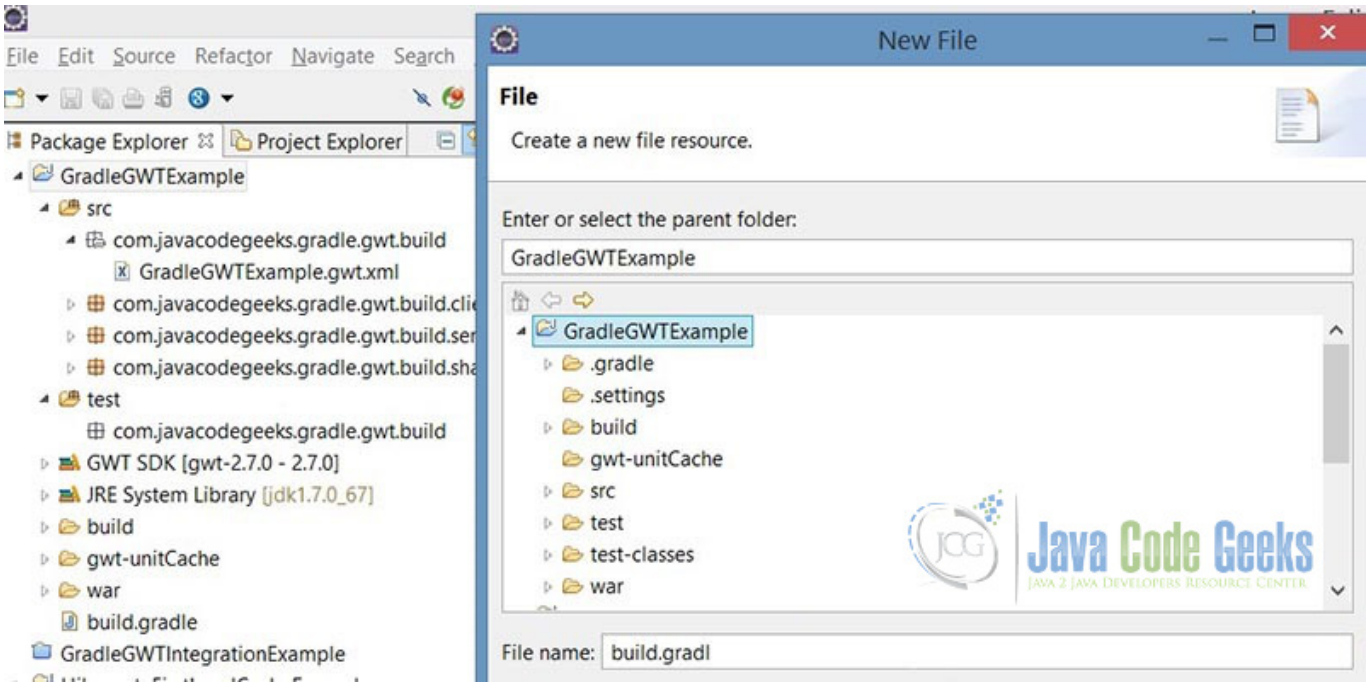

Figure 3.7: Gradle Build File

Until this step, we only have set the workspace, GWT Project and created a `build.gradle` file. To next, we see how develop step by step our Gradle Build Script to make it works.

3.6 GWT Gradle Build Script (Step by step)

build.gradle

```
apply plugin: 'war'
apply plugin: 'java'
apply plugin: 'gwt'
apply plugin: 'eclipse'
apply plugin: 'jetty'

//Java version compatibility to use when compiling Java source.
sourceCompatibility = 1.7
//Java version to generate classes for.
targetCompatibility = 1.7
//Script Version
version = '1.0'

buildscript {
        repositories {
                jcenter() //repository where to fetch gwt gradle plugin
        }
        dependencies {
                classpath 'de.richsource.gradle.plugins:gwt-gradle-plugin:0.6'
        }
}

// central repository to load the GWT library
repositories {
        mavenCentral()
}

compileJava{
        //enable incremental compilation
        options.incremental = true
}

gwt {
        gwtVersion='2.7.0'
        modules 'com.javacodegeeks.gradle.gwt.integration.build. ↩
            GradleGWTIntegrationExampleJCG'

        sourceSets {
                main {
                        java {
                                srcDir 'src'
                        }
                }
        }

        logLevel = 'ERROR'

        minHeapSize = "512M";
        maxHeapSize = "1024M";

        superDev {
                noPrecompile=true
        }

        // The following is only needed if you don't use the Google Plugin for Eclipse.
        eclipse{
                addGwtContainer=false // Default set to true
        }
```

```
        //Specify the deployment Port
        jettyRunWar.httpPort = 8089
}

task jettyDraftWar(type: JettyRunWar) {
        dependsOn draftWar
        dependsOn.remove('war')
        webApp=draftWar.archivePath
}
```

3.6.1 Plugins and Dependencies

The first step is apply the plugins and set GWT Project dependencies to provide all the necessary libs and tasks to make the Build and Deployment.

- `apply plugin:'war'`, basic plugin that adds assembling war tasks; it provides the main task War, that assembles the file itself.
- `apply plugin:'java'`, is the basis from other plugins and provides the capabilities to build and test the projects.
- `apply plugin:'gwt'`, this plugin is provided by the declared dependency in line 19. This plugin facilitates the assembly process of gwt-war, if we don't use them, we would have to do the process of making directories, copy files, compile java files, etc.
- `apply plugin:'eclipse'`, this plugin provides configuration tasks to the eclipse project, is useful if you want to do a full synchronization between the Gradle generated files and the Eclipse Project.
- `apply plugin:'jetty'`, with this plugin we can deploy the war directly with no need run configuration in Eclipse.

The `BuildScript` block, this concept is used to define the external dependencies that are available to the classloader during the execution of Gradle build script. In the declaration of dependencies only be used the method `classpath`, and you can't use the configurations `compile` , `runtime`, `testCompile`, etc. because this block is just to declare external libraries references available to the classpath, if you need to provide any configuration like `compile` or `runtime` you need to do outside of `BuildScript` block.

The `repositories` block (line 23), indicates where Gradle looks for any required dependency, in this case in the Maven central repository. For this example get gwt libraries and automatically fetch gwt dependencies; while the script is executing some lines like these appears in the console.

```
Download https://jcenter.bintray.com/de/richsource/gradle/plugins/gwt-gradle-plugin/0.6/gwt ↩
    -gradle-plugin-0.6.pom
Download https://jcenter.bintray.com/de/richsource/gradle/plugins/gwt-gradle-plugin/0.6/gwt ↩
    -gradle-plugin-0.6.jar
Download https://repo1.maven.org/maven2/com/google/gwt/gwt-codeserver/2.7.0/gwt-codeserver ↩
    -2.7.0.pom
Download https://repo1.maven.org/maven2/com/google/gwt/gwt-codeserver/2.7.0/gwt-codeserver ↩
    -2.7.0.jar
```

3.6.2 Gradle GWT Integration (Building and assembling)

The Main goal of this post is to understand how to integrate Gradle with GWT, and in this part is the key.

In the line 27 set the incremental compilation of java files, why do use this? If your GWT project it's so large, indicates to Gradle exactly which input files were out of date compared to a previous execution and only compile those files, improving the compile performance.

The GWT task defined in line 32, is the main method of the build script, in here we need to define the GWT version that are we using, for this example is 2.7.0 distribution. In line 35, we need to set the name of the GWT Module, how we find them? It's simple

are composed by the package + name of the module in the gwt.xml file, then is com.javacodegeeks.gradle.gwt.integration.build. + GradleGWTIntegrationExampleJCG , look the name in the line 8.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  When updating your version of GWT, you should also update this DTD reference,
  so that your app can take advantage of the latest GWT module capabilities.
-->
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 2.7.0//EN"
  "https://gwtproject.org/doctype/2.7.0/gwt-module.dtd">
<module rename-to='gradlegwtintegrationexamplejcg'>
  <!-- Inherit the core Web Toolkit stuff.                        -->
  <inherits name='com.google.gwt.user.User'/>

  <!-- Inherit the default GWT style sheet.  You can change       -->
  <!-- the theme of your GWT application by uncommenting          -->
  <!-- any one of the following lines.                            -->
  <inherits name='com.google.gwt.user.theme.clean.Clean'/>
  <!-- <inherits name='com.google.gwt.user.theme.standard.Standard'/> -->
  <!-- <inherits name='com.google.gwt.user.theme.chrome.Chrome'/> -->
  <!-- <inherits name='com.google.gwt.user.theme.dark.Dark'/>     -->

  <!-- Other module inherits                                      -->

  <!-- Specify the app entry point class.                         -->
  <entry-point class='com.javacodegeeks.gradle.gwt.integration.build.client. ↩
      GradleGWTIntegrationExampleJCG'/>

  <!-- Specify the paths for translatable code                    -->
  <source path='client'/>
  <source path='shared'/>

  <!-- allow Super Dev Mode -->
  <add-linker name="xsiframe"/>
</module>
```

The `SourceSets` in line 37, are an important part, because here we define the custom directories where are the classes, but we do this, Gradle look for classes in the default directory which is `'src/main/java'`.

The `logLevel` property set the error level to print on the console; `minHeapSize` and `maxHeapSize` set the space of Java Heap, like an jvm args, would do.

The `addGwtContainer` sets if GWT_CONTAINER should be added to the eclipse classpath instead of using the Gradle dependencies, making dependency with GPE (GWT Plugin Eclipse) if is set to true, for this we prefer to set false.

For last property `httpPort` in line 60, allows us to customize deployment port.

The `jettyDraftWar` tasks will be used if you want to run in a draft mode, with a no production quality version. This task is a type of JettyRunWar, i.e. extending and inherits its properties.

3.7 Running the Example

Open a command shell on the root of the GWT Project, where is the Gradle build script.

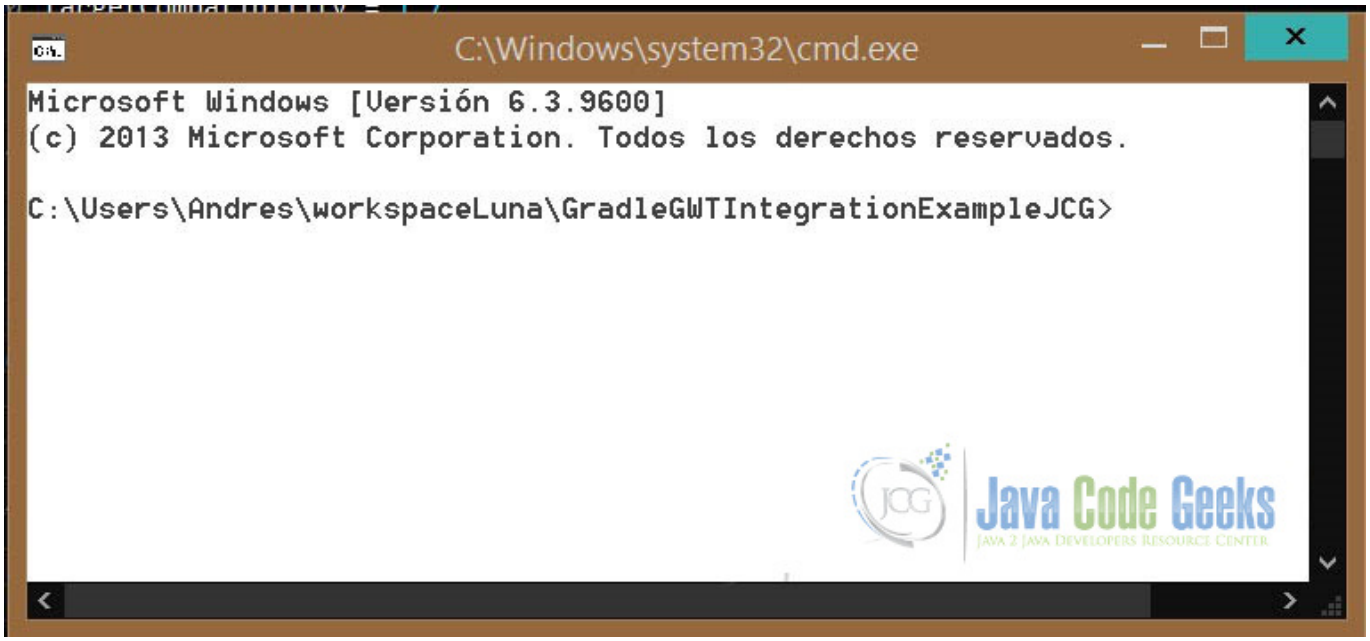

Figure 3.8: Gradle GWT Console

And is as easy as running this command to make all the job. `gradle JettyRunWar`

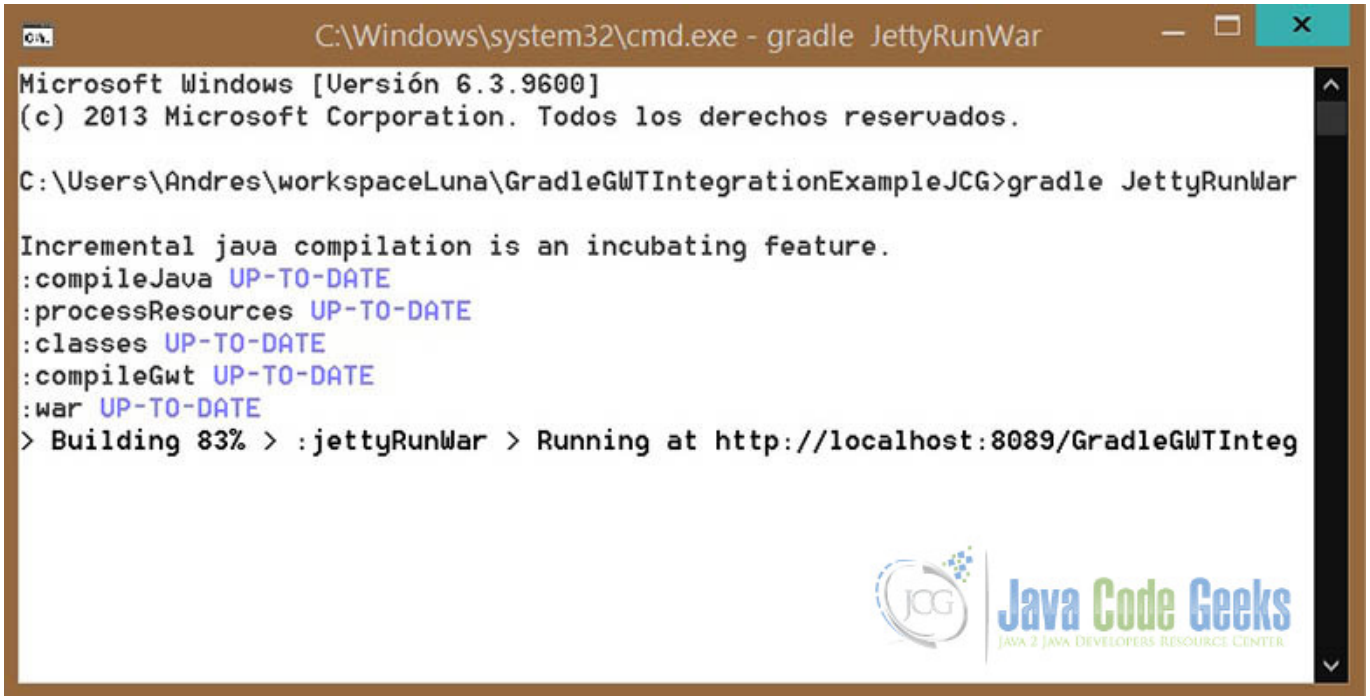

Figure 3.9: Gradle GWT Building and Running

If we open the deployed URL, `https://localhost:8089/GradleGWTIntegrationExampleJCG/`. Look carefully name and port of your application.

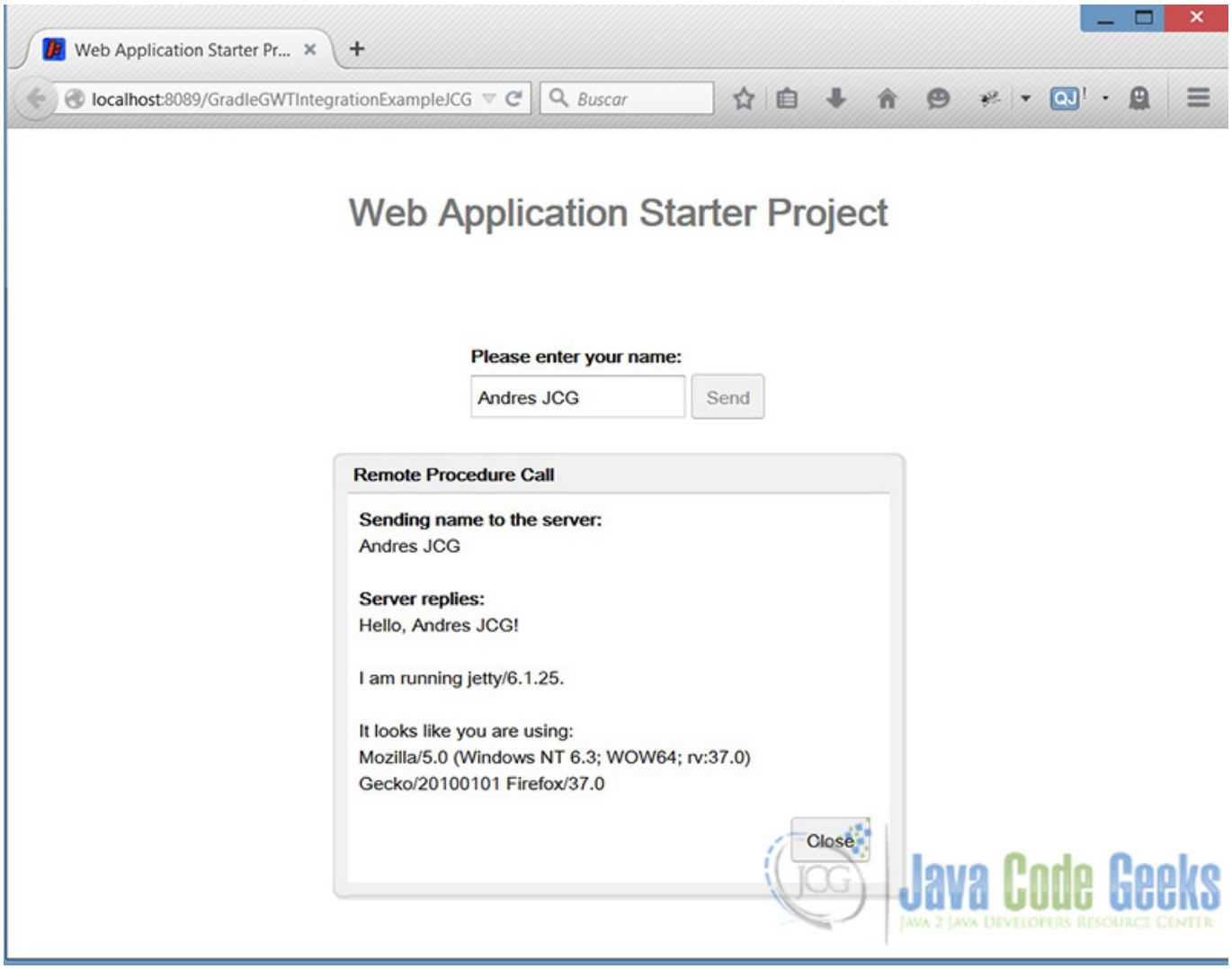

Figure 3.10: Gradle GWT Running

3.8 Key Points

Tips

- You have to distinguish the difference between buildScript block and repositories and dependencies blocks, these are configuration items, that we use to set the classpath external libraries, URL Repositories and set the dependencies with other libraries.
- The use of SourceSets to set the custom directory of Java sources. The default directory in where Gradle search sources is *src/main/java*.
- The dependency of Gradle-Gwt-Plugin developed by Steffen Schaeffer, that provides all GWT compilation Gradle tasks and make so easier the Gradle GWT Integration.
- If you want to make Eclipse Project and GPE compatible, you have to set `addGwtContainer` to true and update the classpath settings in eclipse, it's likely to get some errors in Eclipse GWT Project.
- When we run the task `compileGWT`, the generated resources (cache, draftOut, extra, gen, out, work) are allocated in `root_project/build/gwt/` directory.

- All the libraries are downloaded to the directory: `'C:UsersAndres.gradlecachesmodules-2files-2.1'`, i.e. your Gradle home, `{gradle_home}/caches/modules-x/files-x.x/`
- If you get some errors, check the log in the directory `{gradle_home}/daemon/2.3` and try to fix them. (Prior activation of Gradle Daemon)
- To get more information look in documentation of Gradle GWT Plugin here
- If you want more information about Gradle tasks and basic plugins look in documentation here

3.9 Download the Eclipse Project

Download

You can download the full source code of this example here: **Gradle GWT Integration Example**

Chapter 4

Gradle SourceSets Example

Gradle SourceSets are a key concept for the Gradle Java Plugin which define the structure of Java Source Files. In this example will see how to use this concept, customize them through gradle properties, create a new sourceset, get documentation and assembling them in a JAR

4.1 Introduction to Gradle SourceSets

4.1.1 What is a Gradle SourceSet ?

A SourceSet is a collection of java source files and additional resource files that are compiled and assembled together to be executed. The main idea of sourcesets is to group files with a common meaning for the project, with no need of separate them in another project.

4.2 What do We Need?

- As IDE: Eclipse Luna 4.4
- Eclipse Gradle Plugin
- JDK 1.7_75 or higher
- Gradle 2.3

But the main idea is to edit a `build.gradle` script and you can do this with only a plain text editor, also should have a java project ready to work on it.

4.3 Environment Configuration

Please set your Gradle environment variables and install the Gradle plugin on your IDE. To avoid to be boilerplate visit this previous posts that show how to configure your Gradle Environment. Gradle Hello World Tutorial & Gradle GWT Integration Example

4.4 Creating a Gradle Project

Go to the Eclipse Wizard and then use Gradle Project Wizard.

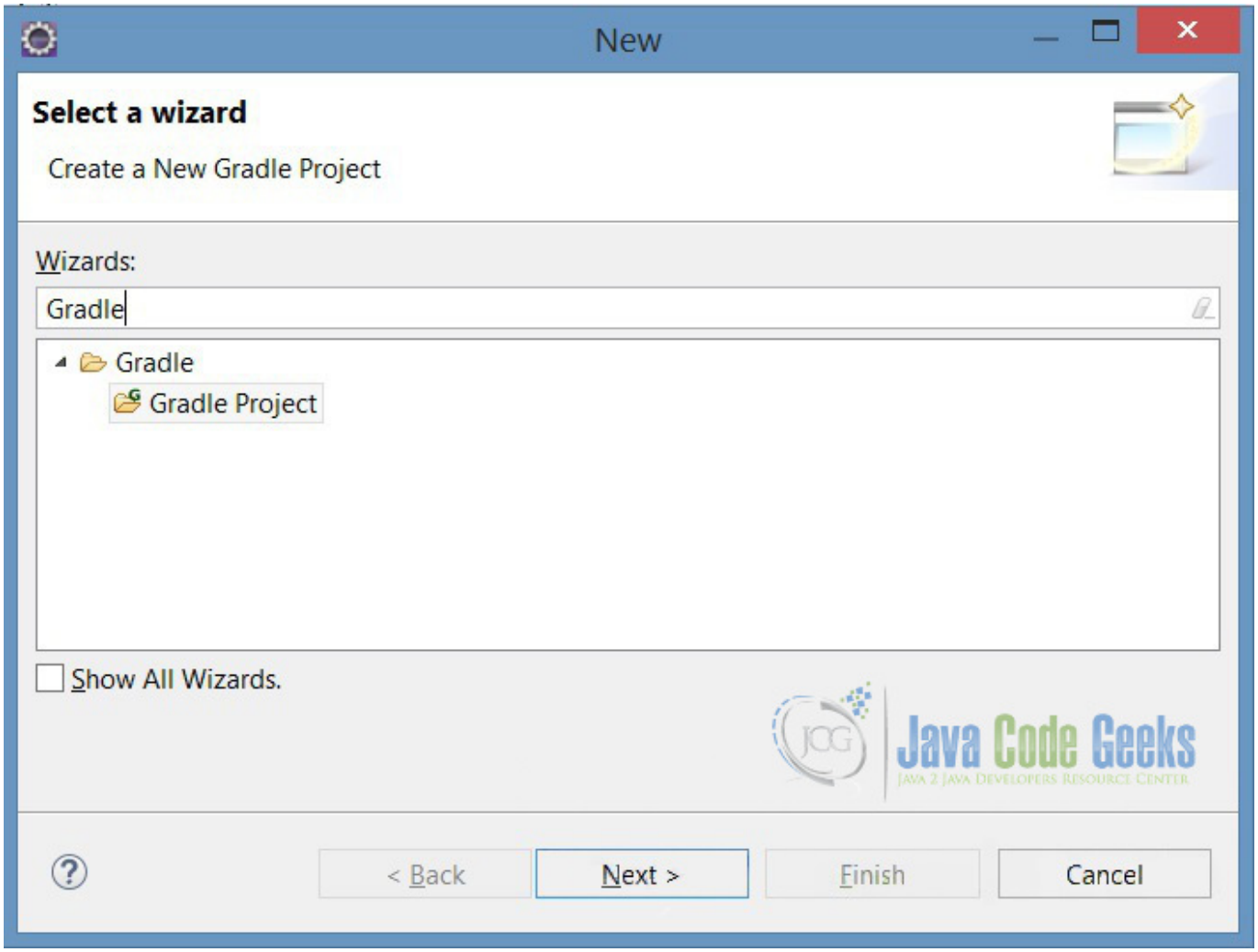

Figure 4.1: Gradle SourceSet Project Wizard

Then, please choose in sample project Java API and Implementation because is most useful for this example, but you can use any other.

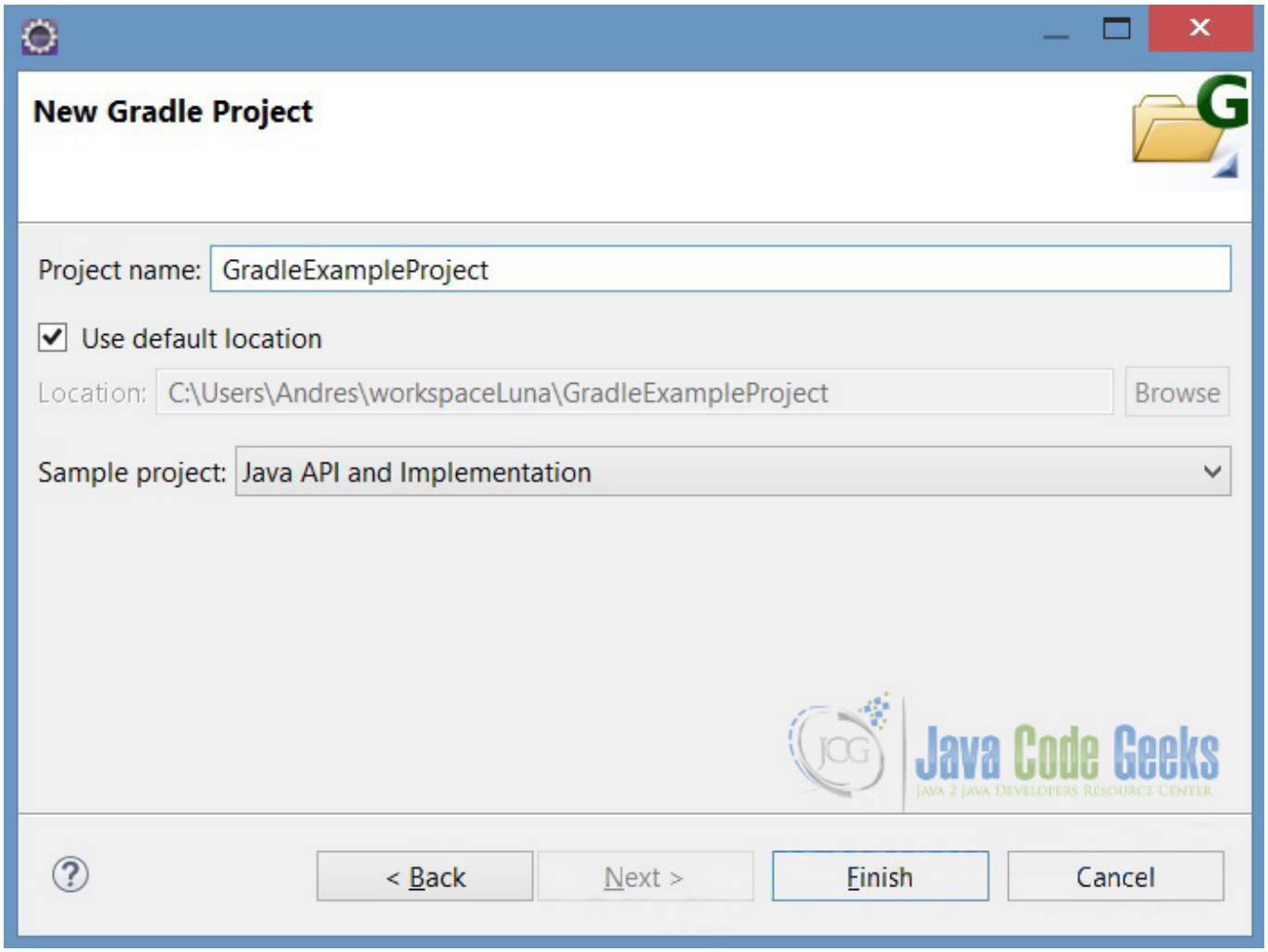

Figure 4.2: Gradle Sourceset Sample Project

4.5 Generated Build.Gradle (New SourceSet Definition)

If we go to the build.gradle file in project's root, we find a script like this:

```
apply plugin: "java"
apply plugin: "maven"

group = "myorg"
version = 1.0

repositories {
    mavenCentral()
}

dependencies {
    apiCompile 'commons-codec:commons-codec:1.5'

    implCompile sourceSets.api.output
    implCompile 'commons-lang:commons-lang:2.6'

    testCompile 'junit:junit:4.9'
```

```
    testCompile sourceSets.api.output
    testCompile sourceSets.impl.output
    runtime configurations.apiRuntime
    runtime configurations.implRuntime
}

sourceSets.all { set ->
    def jarTask = task("${set.name}Jar", type: Jar) {
        baseName = baseName + "-$set.name"
        from set.output
    }

    artifacts {
        archives jarTask
    }
}

sourceSets {
    api
    impl
}

jar {
    from sourceSets.api.output
    from sourceSets.impl.output
}

uploadArchives {
    repositories {
        mavenDeployer {
            repository(url: uri("${buildDir}/repo"))

            addFilter("main") { artifact, file -> artifact.name == project.name }
            ["api", "impl"].each { type ->
                addFilter(type) { artifact, file -> artifact.name.endsWith("-$type") }

                // We now have to map our configurations to the correct maven scope for  ↩
                    each pom
                ["compile", "runtime"].each { scope ->
                    configuration = configurations[type + scope.capitalize()]
                    ["main", type].each { pomName ->
                        pom(pomName).scopeMappings.addMapping 1, configuration, scope
                    }
                }
            }

        }
    }
}
```

So, this script has a some tasks configurations, take a look to each:

- First we apply java and maven plugins to use the tasks defined in them. See `apply plugin` reference.
- The maven repository is referenced to download the libraries with which there are dependencies to compile and run. See `repositories` and `dependencies` reference.
- Then, dynamically it's defined for each SourceSet a `jar` assembly `task`. In the 25 line, we define 2 tasks, called `apiJar` and `implJar` that both are Jar type and what they do is to assemble the jar with the classes contained for the SourceSets.
- In the line 35 we define the new `SourceSets`, **api** and **impl**, that are contained in the src folder, in next steps we see how to set a custom location.

- The `Jar` method in line 40 set the sources that are assembled in the Jar, in this case get all sources from api and impl SourceSets. If we take a look at line 27, for apiJar and implJar tasks, those only has api sources or impl sources but no both, so if them have dependency will occurs compilation or runtime errors using the Jar.
- Last method `uploadArchives` will deploy this jar file to a remote Maven repository. For this example we can delete this.

4.6 SourceSet's Properties

Foremost we will add a task to see all the `properties` of SourceSets, please add it to the end of the script.

```
task sourceSetProperties << {
        sourceSets {
                main {
                        println "java.srcDirs = ${java.srcDirs}"
                        println "resources.srcDirs = ${resources.srcDirs}"
                        println "java.files = ${java.files.name}"
                        println "allJava.files = ${allJava.files.name}"
                        println "resources.files = ${resources.files.name}"
                        println "allSource.files = ${allSource.files.name}"
                        println "output.classesDir = ${output.classesDir}"
                        println "output.resourcesDir = ${output.resourcesDir}"
                        println "output.files = ${output.files}"
                }
        }
}
```

In this tasks we point to the main SourceSet that's set by default. So, then run the task with the command `gradle sSP` (brief invocation) and will see this output:

```
C:\Users\Andres\workspaceLuna\GradleSourceSetProject>gradle sSP
:sourceSetProperties
java.srcDirs = [C:\Users\Andres\workspaceLuna\GradleSourceSetProject\src\main\ja
va]
resources.srcDirs = [C:\Users\Andres\workspaceLuna\GradleSourceSetProject\src\ma
in\resources]
java.files = []
allJava.files = []
resources.files = []
allSource.files = []
output.classesDir = C:\Users\Andres\workspaceLuna\GradleSourceSetProject\build\c
lasses\main
output.resourcesDir = C:\Users\Andres\workspaceLuna\GradleSourceSetProject\build
\resources\main
output.files = [C:\Users\Andres\workspaceLuna\GradleSourceSetProject\build\class
es\main, C:\Users\Andres\workspaceLuna\GradleSourceSetProject\build\resources\ma
in]

BUILD SUCCESSFUL

Total time: 1.169 secs
```

The result is, all *files properties* are empty because the directories point to the default values `src/main/java`. So to make this properties works we can set this task to impl or api SourceSets, or also we can set any as the main SourceSet. If we change `main` for `impl` in 3rd line and run again the task we will see this output.

```
C:\Users\Andres\workspaceLuna\GradleSourceSetProject>gradle sSP
:sourceSetProperties
java.srcDirs = [C:\Users\Andres\workspaceLuna\GradleSourceSetProject\src\impl\ja
va]
resources.srcDirs = [C:\Users\Andres\workspaceLuna\GradleSourceSetProject\src\im
```

```
pl\resources]
java.files = [DoublerImpl.java]
allJava.files = [DoublerImpl.java]
resources.files = []
allSource.files = [DoublerImpl.java]
output.classesDir = C:\Users\Andres\workspaceLuna\GradleSourceSetProject\build\c
lasses\impl
output.resourcesDir = C:\Users\Andres\workspaceLuna\GradleSourceSetProject\build
\resources\impl
output.files = [C:\Users\Andres\workspaceLuna\GradleSourceSetProject\build\class
es\impl, C:\Users\Andres\workspaceLuna\GradleSourceSetProject\build\resources\im
pl]

BUILD SUCCESSFUL

Total time: 1.265 secs
```

Gradle this time whether found java files.

Another strategy is to set the main SourceSet point to src's root. adding this configuration to SourceSets task. So if we run again the `gradle sSP` command will see all Java sources without test clasess. This is how we customize the SourceSet directory.

```
sourceSets {
    api
    impl
        main{
                java {
                        srcDir 'src/api/java'
                        srcDir 'src/impl/java'
                }
        }
        test {
                java {
                        srcDir 'src/test/java'
                }
        }
}
```

```
java.srcDirs = [C:\Users\Andres\workspaceLuna\GradleSourceSetProject\src\main\ja
va, C:\Users\Andres\workspaceLuna\GradleSourceSetProject\src\api\java, C:\Users\
Andres\workspaceLuna\GradleSourceSetProject\src\impl\java]
java.files = [Doubler.java, DoublerImpl.java]
allJava.files = [Doubler.java, DoublerImpl.java]
resources.files = []
allSource.files = [Doubler.java, DoublerImpl.java]
```

We have 3 directories for the main SourceSet, and both classes are contained.

4.7 Assembling SourceSets in JAR Files

So if we want to package the output classes in a new JAR file, it's simple we must define a task of `Jar type` and set a key sentence `from sourceSet.output`. In previous steps we already define them, look at 25 line or Jar configuration in line 40. So if we run the inital dinamic tasks gradle apiJar and gradle implJar, Gradle will generate a JAR with the output sources in the directories defined for the task.

```
C:\Users\Andres\workspaceLuna\GradleSourceSetProject>gradle apiJar
:compileApiJava UP-TO-DATE
:processApiResources UP-TO-DATE
:apiClasses UP-TO-DATE
```

```
:apiJar UP-TO-DATE

BUILD SUCCESSFUL

Total time: 0.997 secs
```

Gradle automatically will create 3 new tasks based on any new SourceSet added to the project, apiClasses, compileApiJava, and processApiResources for this case, but in other case change api for sourceset's name. These tasks have a dependency between them, so what they do in 3 steps is to compile the java files, process resources and assemble the jar copying all the files to it, keeping the project structure.

Then, we assemble the three possible Jar files, so:

- `gradle apiJar` , only contains api output sources
- `gradle impJar`, only contains api output sources.
- `gradle Jar`, contains all output project sources.

So refresh the project and take a look in the `build/libs` directory.

Figure 4.3: Gradle SourceSet Jars

4.8 Creating SourceSet Documentation

So if we want to generate the Javadoc documentation, we must use the javadoc task. This tasks seeks by default the main SourceSet, but with the property `sourceSets.<sourceSet>.allJava` we can add another custom SourceSet. Next, add this task to the build script and we can run the command gradle javadoc; the generated documentation is allocated into `build/docs/javadoc`.

```
javadoc {
        // but the main sourceset contains both api & impl sourceset, javadoc will generate ↩
            all documentation
        source sourceSets.api.allJava
        }
```

```
C:\Users\Andres\workspaceLuna\GradleSourceSetProject>gradle javadoc
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:javadoc UP-TO-DATE

BUILD SUCCESSFUL

Total time: 1.321 secs
```

4.9 Testing

So for run the Unit test classes contained in the test SourceSet, we need to add this configuration to the `test Task` to view the result of execution.

```
test {
        // Print in console the result of test
        afterTest { test, result ->
                println "Executing test ${test.name} [${test.className}] with result: ${ ↩
                    result.resultType}"
        }
}
```

We run the task, `gradle test` and get this output:

```
C:\Users\Andres\workspaceLuna\GradleSourceSetProject>gradle test
:compileApiJava UP-TO-DATE
:processApiResources UP-TO-DATE
:apiClasses UP-TO-DATE
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:compileImplJava UP-TO-DATE
:processImplResources UP-TO-DATE
:implClasses UP-TO-DATE
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test
Executing test testIt [doubler.impl.DoublerImplTest] with result: SUCCESS

BUILD SUCCESSFUL

Total time: 1.596 secs
```

4.10 Final Build.Gradle SourceSet Script

This is the final version of the script.

```
/*
 * Author: Andres Cespedes
 * Date: 01 May 2015
 * Example: Gradle SourceSets Example
 * Site: www.javacodegeeks.com
 * */
apply plugin: "java"

//Script Version
version = 1.0
//Java version compatibility to use when compiling Java source.
sourceCompatibility = 1.7
//Java version to generate classes for.
targetCompatibility = 1.7

//repository where to fetch third party libraries and dependencies
repositories { mavenCentral() }

//Set a Jar task to all of SourceSets.
sourceSets.all { set ->
        def jarTask = task("${set.name}Jar", type: Jar) {
                baseName = baseName + "-$set.name"
                from set.output
        }

        artifacts { archives jarTask }
}

//Print Main Sourceset Properties
task sourceSetProperties << {
        sourceSets {
                main {
                        println &quot;java.srcDirs = ${java.srcDirs}&quot;
                        println &quot;resources.srcDirs = ${resources.srcDirs}&quot;
                        println &quot;java.files = ${java.files.name}&quot;
                        println &quot;allJava.files = ${allJava.files.name}&quot;
                        println &quot;resources.files = ${resources.files.name}&quot;
                        println &quot;allSource.files = ${allSource.files.name}&quot;
                        println &quot;output.classesDir = ${output.classesDir}&quot;
                        println &quot;output.resourcesDir = ${output.resourcesDir}&quot;
                        println &quot;output.files = ${output.files}&quot;
                }
        }
}

// SourceSet&amp;#039;s Configuration
sourceSets {
        api
        impl
        main{
                java {
                        srcDir &amp;#039;src/api/java&amp;#039;
                        srcDir &amp;#039;src/impl/java&amp;#039;
                }
        }
        test {
                java { srcDir &amp;#039;src/test/java&amp;#039; }
        }
```

```
}

// Compile, Test and Run dependencies
dependencies {
        apiCompile &#039;commons-codec:commons-codec:1.5&#039;

        implCompile sourceSets.api.output
        implCompile &#039;commons-lang:commons-lang:2.6&#039;

        testCompile &#039;junit:junit:4.9&#039;
        testCompile sourceSets.api.output
        testCompile sourceSets.impl.output
        runtime configurations.apiRuntime
        runtime configurations.implRuntime
}

// JAR Task Configuration, define output the directories that make up the file.
jar {
        from sourceSets.api.output
        from sourceSets.impl.output
}

javadoc {
        // but the main sourceset contains both api & impl sourceset, javadoc will generate ↩
            all documentation
        source sourceSets.api.allJava
        }

test {
        // Print in console the result of test
        afterTest { test, result ->
                println &quot;Executing test ${test.name} [${test.className}] with result: ↩
                    ${result.resultType}&quot;
        }
}
```

4.11 Key Points

Tips

- The Gradle SourceSet's Properties are made to access to the directories and files that make up the SourceSet.
- Java plugin give us a lot of basic funcionalities to improves the development process.
- Gradle have a lot of default values, you need to set the custom values that adjust to your project
- Every task have so many properties, that can be useful to your needs but to mantain the article more readable we don't mention it here
- Gradle SourceSet concept, is an excellent tool to build a clean structure into your project and make software's components, atomic pieces that you can manage and assemble.

4.12 Download the Eclipse Project

Download

You can download the full source code of this example here **Gradle SourceSet Project**

Chapter 5

Gradle OSGi Plugin Example: BNDTools Bundle Integration

In this example we will talk about how to integrate OSGi frameworks with Gradle build tool. This work consists in build JAR files and customizing Manifest file through Gradle build Script and Deploying them in a OSGi Container like Apache Felix.

5.1 Introduction to Gradle and OSGi Integration

Gradle is an automatic build tool of more use today, so it's almost necessary to integrate the different types of projects with Gradle, to automate the process of build and liberation. In this example we will see how using Gradle can compile, build, install and run a bundle in an OSGi environment. The main goal is to understand Gradle integration with OSGi BND tools.

5.1.1 Basic Concepts

There are some basic concepts that we must check before starting the example.

- **Gradle:** Our tool to build and automate the process.
- **OSGi:** It's the framework to make Java Modular applications and the environment where to deploy.
- **BND Tools:** It's a framework or library pack, that makes OSGi development process more friendly and give to us a lot of capabilities to customize out OSGi bundles. OSGi and BND Tools are required each other.
- **Gradle-Osgi Plugin:** It's a Gradle Plugin developed by Renato Athaydes to make the OSGi integration easier.
- **Apache Felix, Eclipse Equinox:** Are the main OSGi implementations, are containers to deploy OSGi bundles. Think in OSGi bundles like a war file and Apache Felix like Tomcat container.

I want to make this example so simple, so we start.

5.2 What do We Need?

- IDE: Eclipse Luna 4.4, you can download here
- Gradle 2.x or superior, you can download Gradle 2.5 here
- Java JDK 1.7_75 or superior, download here
- BND Tools 2.4, this is a Eclipse Plugin that we can download in Eclipse Marketplace
- Gradle-Osgi Plugin, you can get more detail here

5.3 Environment Configuration

To make simpler the example, we assume that you have already installed your Eclipse, JDK, Gradle and environment variables for both tools (JAVA_HOME and GRADLE_HOME).

How to Install OSGi?

It's the most simple installation ever, you only need to download Eclipse! Eclipse is one the projects that use OSGi for your development and release process, so OSGi, Equinox and Felix frameworks are already installed as Eclipse's plugins.

Next Step, Install BND Tools Eclipse Plugin.

Please download and install from Eclipse Marketplace using the menu: Help > Eclipse Marketplace > type "bndtools".

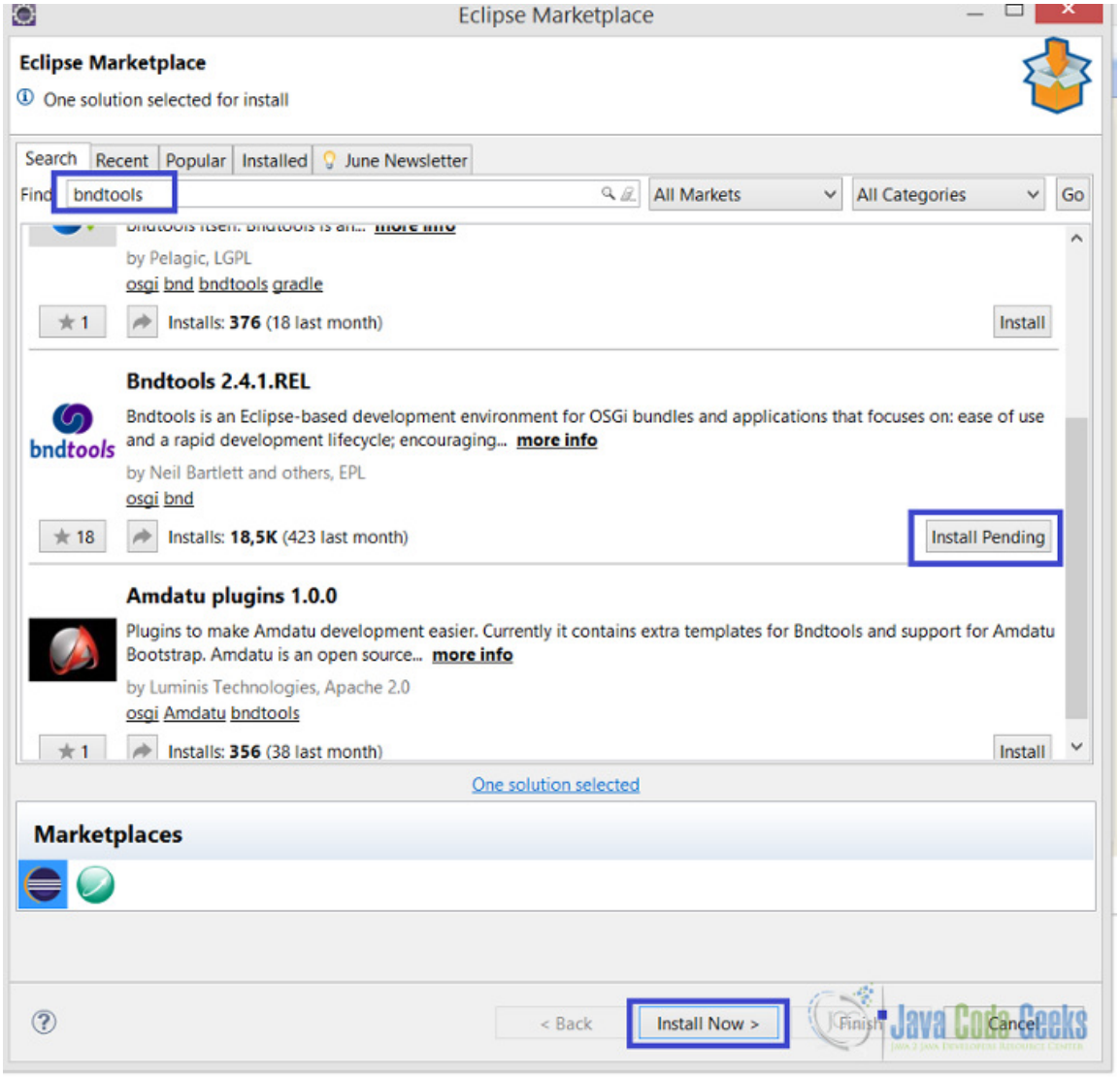

Figure 5.1: BND Tools Eclipse Plugin installation

5.4 Creating OSGi Project

Create a new OSGi bundle project using File > New > Other > Bndtools > Bndtools OSGi project

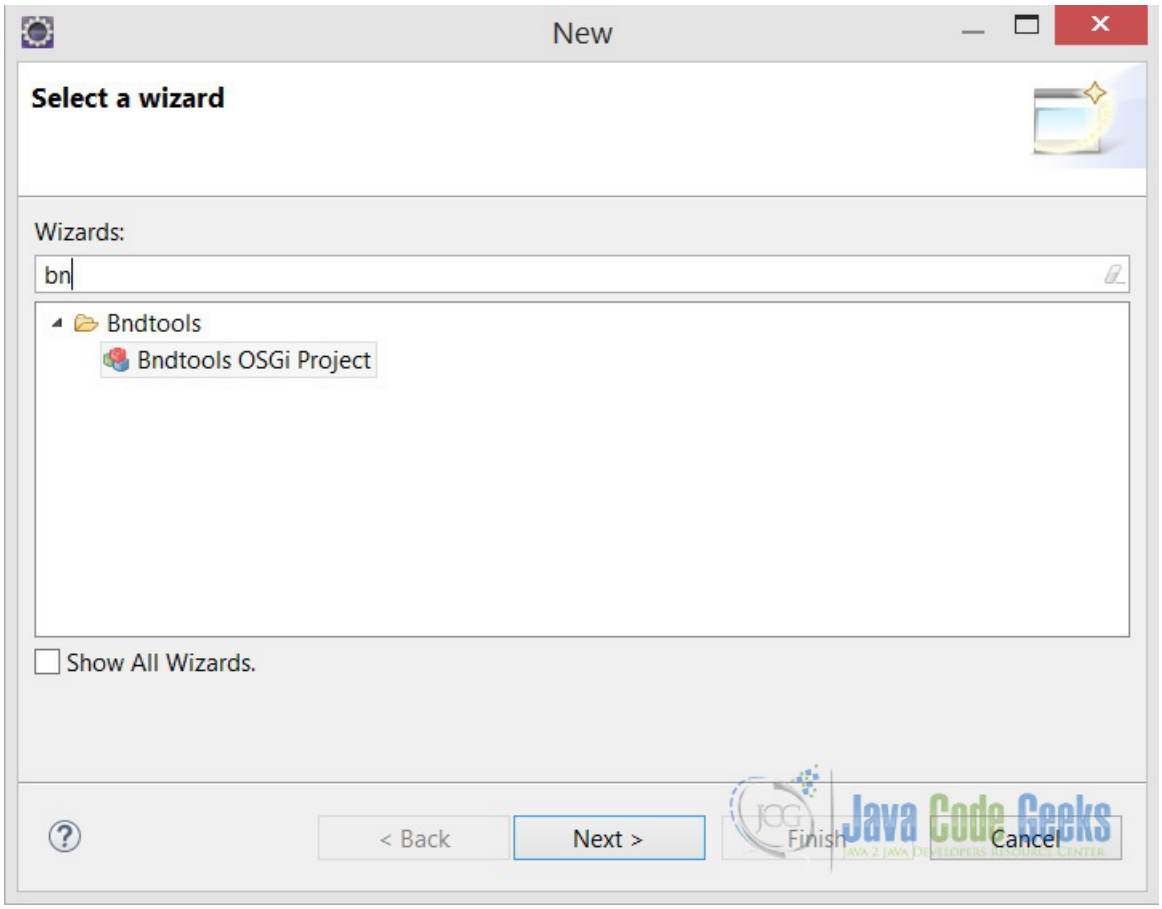

Figure 5.2: BND Tools OSGi Project

Then, type a project's name and keep all the other settings as they are, then click next to choose project template, in this case, we use OSGi component development by default.

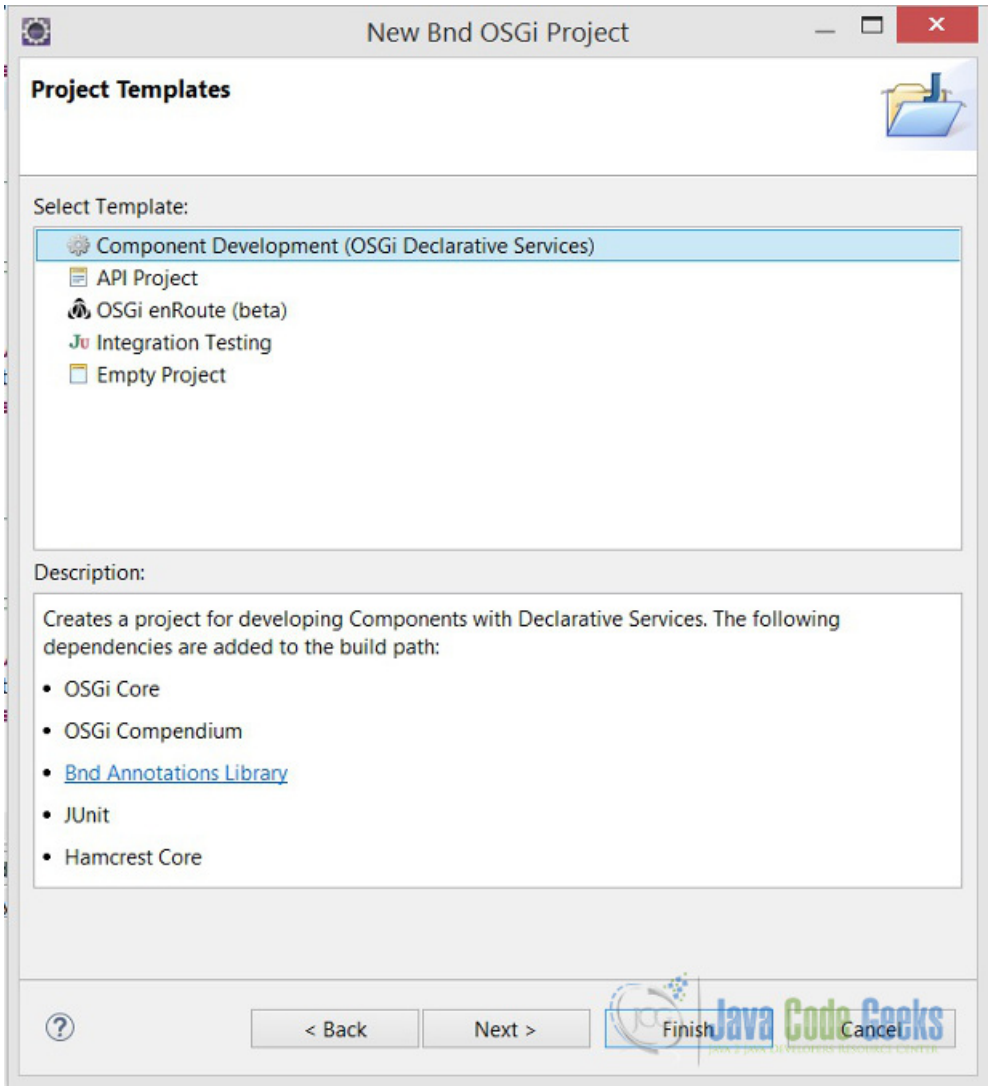

Figure 5.3: OSGi Project Template

So, when we create the first OSGi project, bnd tools ask to create a cnf project, that will store all OSGi configuration and libraries downloaded from external repositories. Only click next and leave all configurations by default, at the end we have 2 projects cnf (bnd tools configuration) and our Gradle OSGi project.

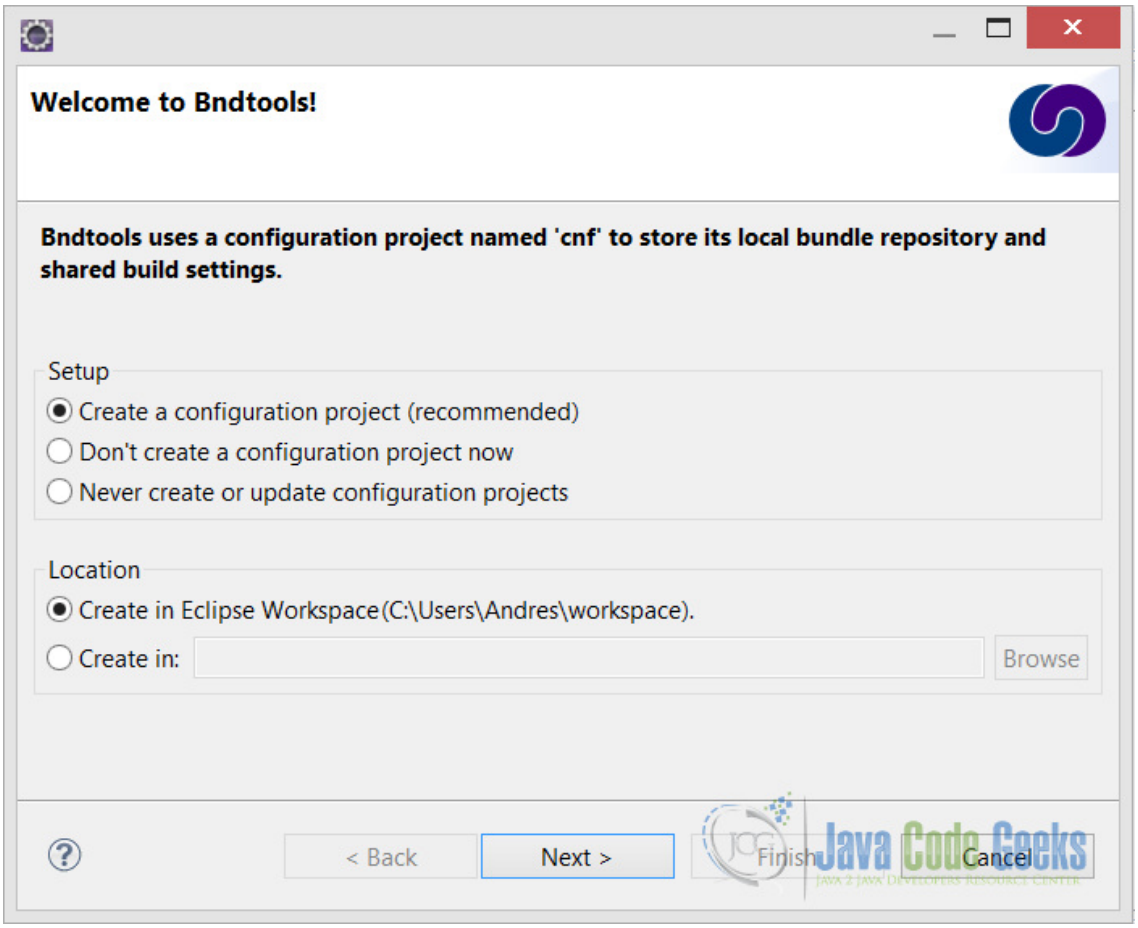

Figure 5.4: BND Tools CNF Project

Then, define a package for the Bundle Activator and create a new class, we call HelloActivator.java

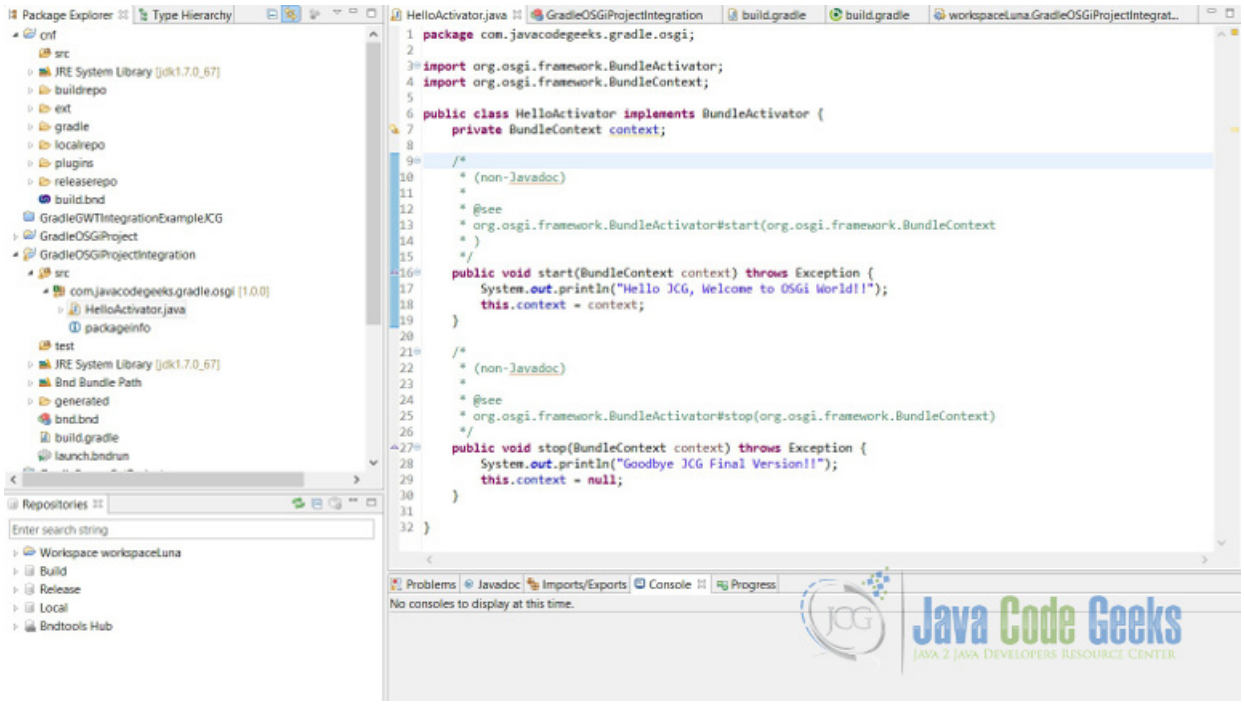

Figure 5.5: Gradle OSGi BundleProject

HelloActivator.java

```
package com.javacodegeeks.gradle.osgi;

import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;

public class HelloActivator implements BundleActivator {
        private BundleContext context;

        public void start(BundleContext context) throws Exception {
                System.out.println("Hello JCG, Welcome to OSGi World!!");
                this.context = context;
        }

        /*
         * (non-Javadoc)
         *
         * @see
         * org.osgi.framework.BundleActivator#stop(org.osgi.framework.BundleContext)
         */
        public void stop(BundleContext context) throws Exception {
                System.out.println("Goodbye JCG");
                this.context = null;
        }

}
```

5.5 Deploying OSGi Project

The last step before injects Gradle is to deploy the Bundle in an OSGi container, in this example we use Apache Felix. To make possible this, we need to configure bnd.bnd file.

Define a version number, choose the Activator that we created before. In the Export Packages please choose the green plus sign, and add the main package that contains the Activator.

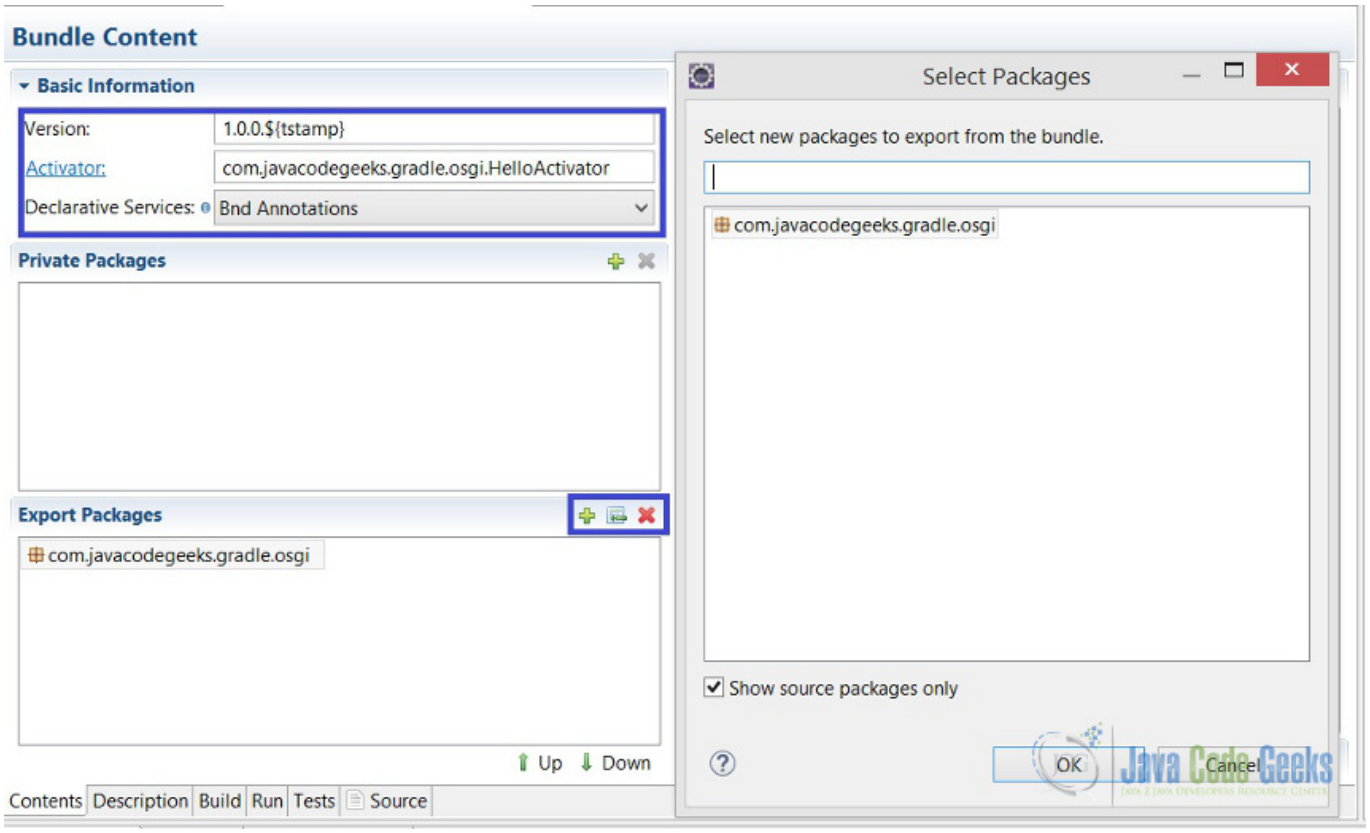

Figure 5.6: OSGi Bundle Content

In the Run Tab, we must configure 3 options.

- Core Runtime, establish the OSGi Framework and JDK to execution, for this case choose Apache Felix 4.0.3 and Java 1.7.
- Run Requirements, add all 4 bundles that appears in the caption, project itself (GradleOSGiProject in this case), osgi.cmpn, felix.command and felix.shell.
- In the bottom of the Run tab, add the same bundles at Run Bundles configuration

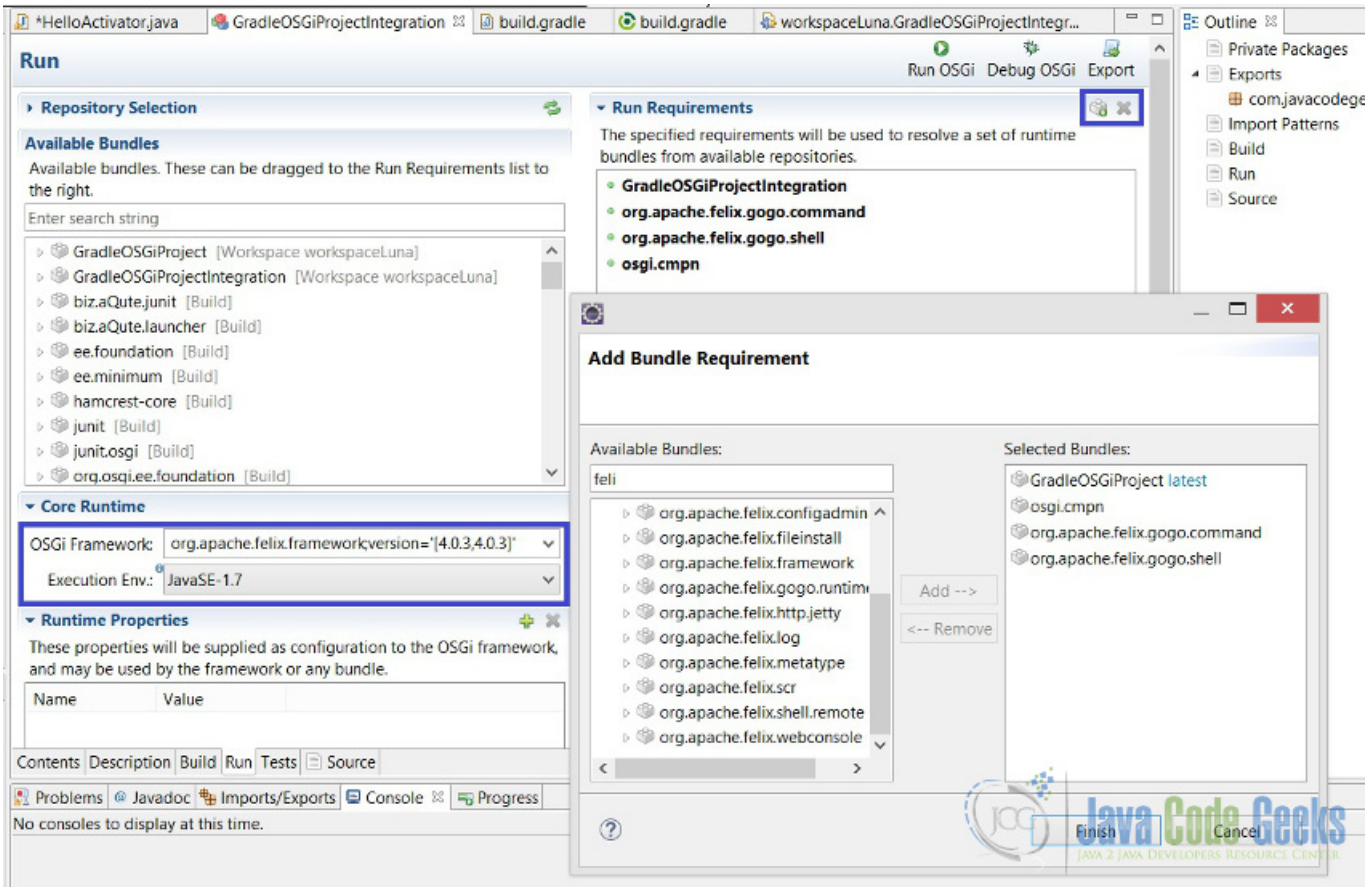

Figure 5.7: OSGi Run BND Tools

Finally, we can run the project using Run OSGi button in Run tab of bnd.bnd file, so this is the output. We can see the installed bundles typing `lb` on Apache Felix Gogo Shell, so when we start or stop OSGiGradle Bundle, `start` and `stop` Activator's methods are executed.

```
Hello JCG, Welcome to OSGi World!!
****************************
Welcome to Apache Felix Gogo

g! lb
START LEVEL 1
   ID|State      |Level|Name
    0|Active     |    0|System Bundle (4.4.1)
    1|Active     |    1|GradleOSGiProjectIntegration (1.0.0.201507130549)
    2|Active     |    1|Apache Felix Gogo Command (0.14.0)
    3|Active     |    1|Apache Felix Gogo Runtime (0.12.1)
    4|Active     |    1|Apache Felix Gogo Shell (0.10.0)
g!
```

5.6 Gradle Integration

To "Gradlify" the OSGi Project we need to create a Gradle Build script and configure it correctly. So, create a new File in root project, New > File > type name "build.gradle".

This is the file, so we'll look in detail each instruction.

build.gradle

```
/*
 * Author: Andres Cespedes
 * Date: 01 July 2015
 * Example: Gradle OSGi Integration Example
 * Site: www.javacodegeeks.com
 * */
buildscript {
        repositories {
                jcenter()
                mavenCentral()
        }
        dependencies {
                classpath "com.athaydes.gradle.osgi:osgi-run-core:1.2"
        }
}

// repositories to download external files, like apache felix
repositories {
        jcenter()
        mavenCentral()
}

// java version source compatibility
sourceCompatibility = 1.7

apply plugin: 'osgi-run'

// osgi-run plugin task that add OSGi subprojects as a bundle files to deploy.
runOsgi {
  bundles += subprojects
}

jar {
    manifest { // Manifest.MF file customization
        instruction 'Private-Package','com.javacodegeeks.gradle.osgi'
        instruction 'Bundle-Vendor', 'JavaCodeGeeks'
        instruction 'Bundle-Description', 'First OSGi Bundle Created By Gradle JCG Tutorial ↩
            '
        instruction 'Bundle-Activator', 'com.javacodegeeks.gradle.osgi.HelloActivator'
        instruction 'Bundle-ManifestVersion', '2'
        instruction 'Bundle-Version', '1.0.0.${tstamp}'
        instruction 'Bundle-Name', 'GradleOSGiProjectIntegration'
        instruction 'Bundle-SymbolicName', 'GradleOSGiProjectIntegration'
        instruction 'Export-Package', 'com.javacodegeeks.gradle.osgi;version="1.0.0";uses ↩
            :="org.osgi.framework"'
    }
}
```

The first part `buildscript`, define the repositories and dependencies to import Gradle Bnd OSGi Plugin. In line 18, we set repositories to download and import external libraries to Gradle, this part it's important as some think they just simply repositories defined in the `buildscript` block, are necessary to get libraries like apache felix, these libraries are downloaded to *%USER_HOME%.gradlecachesmodules-2files-2.1* directory.

In line 20, apply the external OSGi plugin developed by Renato Athaydes, because Gradle's official plugin doesn't add value at all, don't adds tasks more than customize the Manifest.

Line 23 and 27 are the main part of the script, then we add the project to the bundles to deploy and then can configure OSGi Manifest, each Manifest property is added as instruction, you can check OSGi documentation for available properties.

5.7 Running Gradle OSGi Integration

This is how integration works.

Select build.gradle file and press Ctrl+Alt+D to open windows command shell, then execute this Gradle command to run the OSGi environment.`gradle runOsgi` or `gradle rO` as shortened way.

```
Microsoft Windows [VersiÃ$^3$n 6.3.9600]
(c) 2013 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Andres\workspaceLuna\GradleOSGiProject>gradle rO
:GradleOSGiProject:compileJava UP-TO-DATE
:GradleOSGiProject:processResources UP-TO-DATE
:GradleOSGiProject:classes UP-TO-DATE
:GradleOSGiProject:jar
:GradleOSGiProject:createOsgiRuntime
:GradleOSGiProject:runOsgi
****************************
Welcome to Apache Felix Gogo

> Building 83% > :GradleOSGiProject:runOsgilb
g! START LEVEL 1
   ID|State      |Level|Name
    0|Active     |    0|System Bundle (4.4.0)
    1|Active     |    1|Apache Felix Gogo Command (0.14.0)
    2|Active     |    1|Apache Felix Gogo Runtime (0.12.1)
    3|Active     |    1|Apache Felix Gogo Shell (0.10.0)
> Building 83% > :GradleOSGiProject:runOsgiinstall file:../GradleOSGiProject.jar

g! Bundle ID: 4
> Building 83% > :GradleOSGiProject:runOsgi
```

When apache felix gogo shell is active, type `lb` to see all available Bundles, if your bundle don't appears yet, so let install it, how? Execute this simple command `install file:../GradleOSGiProject.jar`, obviously considering the name of your jar; apache gogo shell will tell you the Bundle ID, in this case is 4.

5.8 Testing Gradle OSGi

Gradle already setup OSGi environment, so, executing apache gogo shell commands we can interact in Gradle thread with the OSGi Bundle, updating Manifest or Activator itself.

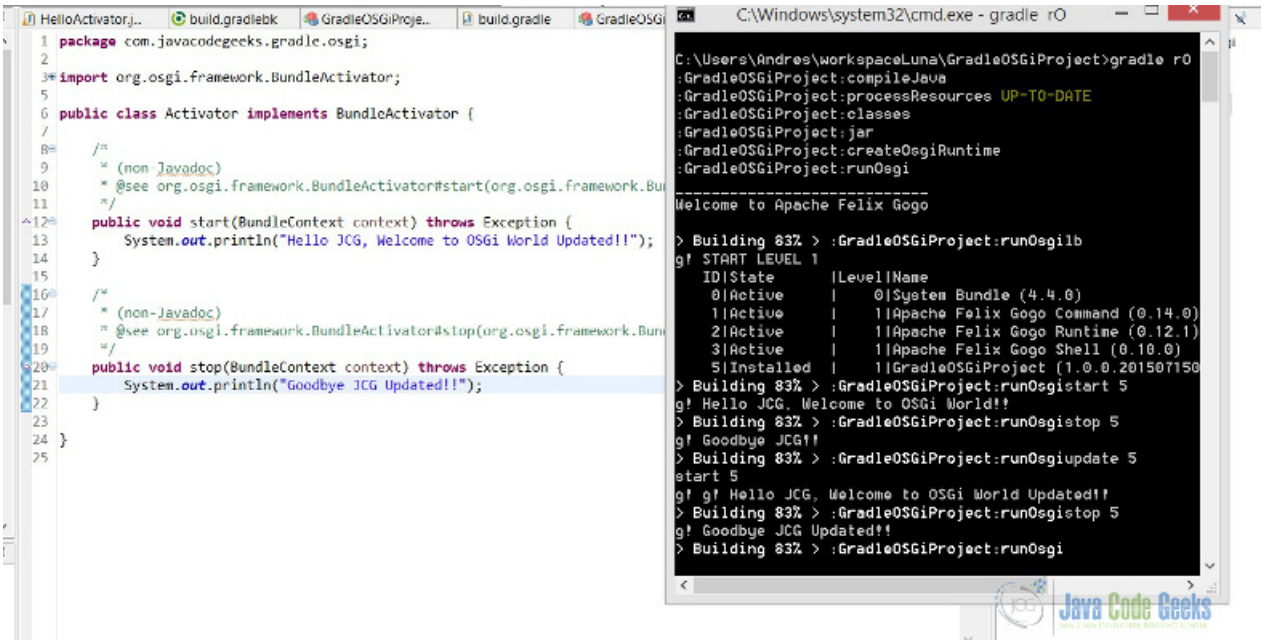

Figure 5.8: Testing Gradle OSGi Plugin

Then, if we modify any artifact or java file only executing `update BundleID` in Gradle OSGI environment shell, in this case `update 5` we get updated Bundle, if we start or stop the Bundle can get updated messages.

For the last part of this example, the Manifest is printed in console executing `headers BundleID` command, `headers 5`, useful to compare the deployed Bundle with Gradle Manifest configuration.

```
> Building 83% > :GradleOSGiProject:runOsgiheaders 5
g!
GradleOSGiProject (5)
**************
Bnd-LastModified = 1436938982133
Bundle-Activator = com.javacodegeeks.gradle.osgi.Activator
Bundle-ManifestVersion = 2
Bundle-Name = GradleOSGiProject
Bundle-SymbolicName = GradleOSGiProject
Bundle-Version = 1.0.0.201507150543
Created-By = 1.7.0_67 (Oracle Corporation)
Export-Package = com.javacodegeeks.gradle.osgi;version="1.0.0";uses:="org.osgi.f
ramework"
Import-Package = org.osgi.framework;version="[1.3,2)"
Manifest-Version = 1.0
Private-Package = com.javacodegeeks.gradle.osgi
Require-Capability = osgi.ee;filter:="(&(osgi.ee=JavaSE)(version=1.7))"
Tool = Bnd-2.4.1.201501161923
> Building 83% > :GradleOSGiProject:runOsgi
```

5.9 Key Points

Tips

- Gradle Official plugin is poor to support OSGi integration
- Key point to Gradle-OSGi integration is to know how to install external bundles

- Unofficial Gradle Run Osgi Bundle plugin facilitates the integration process, thanks to Renato Athaydes for his job.
- Manage OSGi deployment through Gradle enhances the development process by automatization and isolation the execution from Eclipse

5.10 Download the Eclipse Project

This was an example of Gradle OSGi Plugin.

Download

You can download the full source code of this example here **Gradle OSGi Project**

Chapter 6

Gradle Wrapper Example

In this example, we will understand how Gradle can be used to build projects even if the developer machine doesn't have Gradle installed, by using Gradle Wrapper. This is a best practice to unify the Gradle version used by the entire development team.

6.1 What's Gradle Wrapper?

Gradle Wrapper is a type batch or shell script that downloads and automatically configures Gradle to execute tasks. Imagine that you want to run a Gradle build, well you need to download and install Gradle in your computer, so this concept allows is to distribute our project and build configurations with no need to have Gradle installed.

6.2 What we need to start?

This is a simple example, really you only need Gradle to start and to create the base wrapper to distribute to all others, but to make more readable will use Eclipse. So

- As IDE: Eclipse Luna 4.4
- Eclipse Gradle Plugin
- Java JDK 1.7
- Gradle 2.3 or higher

6.3 Environment Configuration

Please set your Gradle environment variables and install the Gradle plugin on your IDE. To avoid to be boilerplate visit this previous posts that show how to configure your Gradle Environment. Gradle Hello World Tutorial

6.4 Creating Wrapper Script

In Eclipse, Create a new Gradle Project and then edit gradle build script.

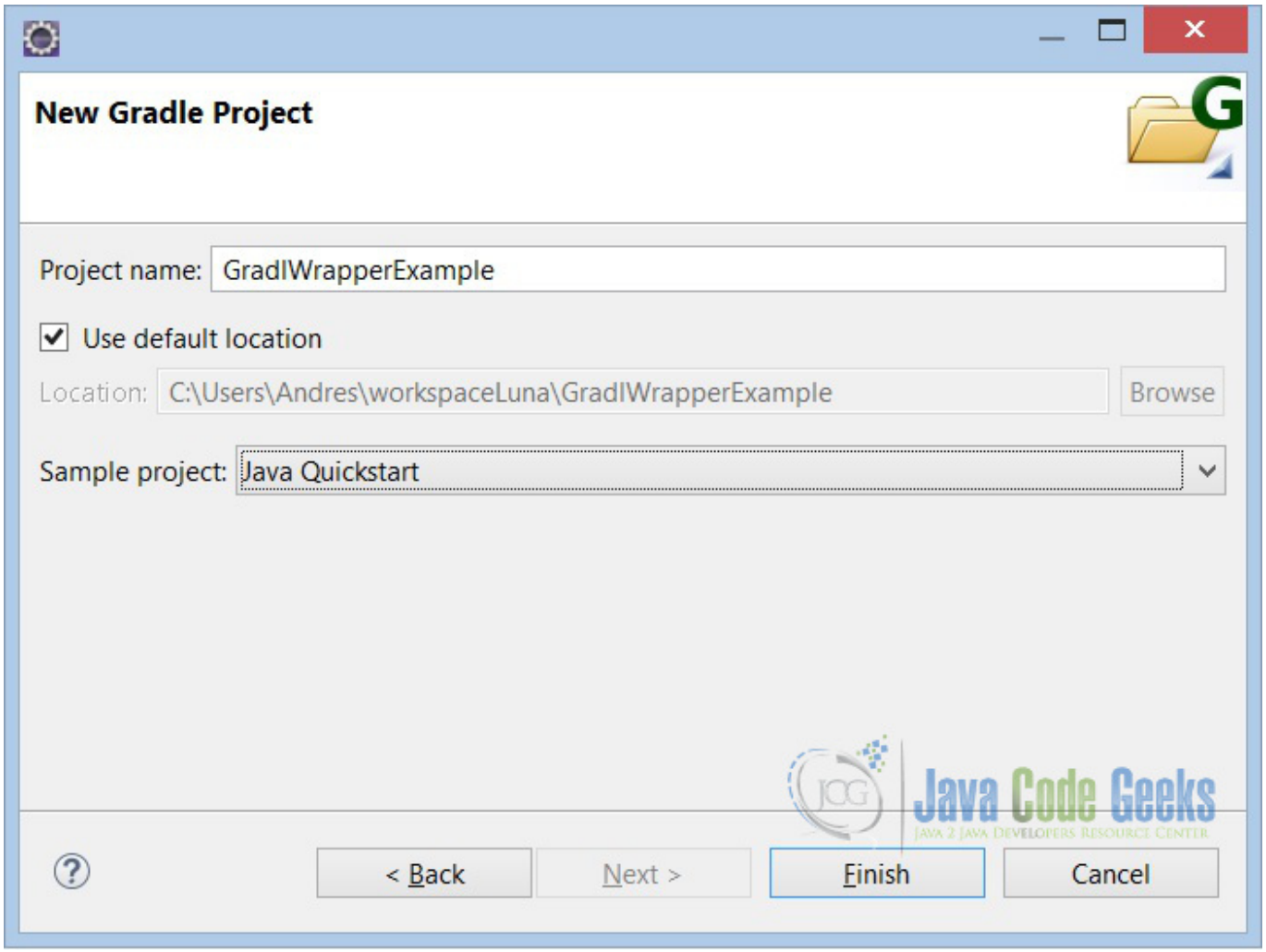

Figure 6.1: Gradle Wrapper Project

Then, in the `build.gradle` script we must add a task of type `org.gradle.api.tasks.wrapper.Wrapper` to customize the configuration of the default `Wrapper` task.

```
task createGradleWrapper(type: Wrapper) {
        gradleVersion = '2.3'
        scriptFile = 'GradleJ' //define a new name for gradle
        jarFile = 'gradle-bin.jar' //jar with files to download and invoke Gradle
        distributionUrl = 'https://services.gradle.org/distributions/gradle-2.3-bin.zip'
}
```

Then, we can execute this task to generate the wrapper files. Execute `gradle createGradleWrapper` or `gradle cGW` in abbreviated form on Windows command shell. This is the output:

```
C:\Users\Andres\workspaceLuna\GradlWrapperExample>gradle cGW
:GradlWrapperExample:createGradleWrapper UP-TO-DATE

BUILD SUCCESSFUL

Total time: 1.044 secs
C:\Users\Andres\workspaceLuna\GradlWrapperExample>
```

6.5 Using Gradle Wrapper

Then, after the execution of the task, two files are generated: `gradlej` and `gradlej.bat` in the root of the project (so refresh it to see them), that contain all the logic and configurations to run Gradle.

This new files are part of the project, so is a good practice add these files to the version control, to able team people that checkout project and build the scripts with gradle and gradlew by default if you don't customize the name) instead of theirs Gradle version.

So, to test Gradle Wrapper we add this simple task and we run with `gradlej` instead of `gradle` command.

```
task helloWrapper << {
        println 'Welcome to JCG Gradle Wrapper Tutorial'
}
```

Execute this command `gradlej helloWrapper` or `gradlej hW` in abbreviated form on Windows command shell. This is the output:

```
C:\Users\Andres\workspaceLuna\GradlWrapperExample>gradlej hW
:GradlWrapperExample:helloWrapper
Welcome to JCG Gradle Wrapper Tutorial

BUILD SUCCESSFUL

Total time: 1.445 secs
C:\Users\Andres\workspaceLuna\GradlWrapperExample>
```

This is how we can use Gradle wrapper to build projects even if other developers don't have Gradle installed.

6.6 Key Points

Tips

- Gradle Wrapper is a good practice to standardize the builds
- Is a good practice too, add Gradle Wrapper in a control version system to distribute to the team
- `distributionUrl` property can used to reference a download URL in your company intranet or a custom fixed Gradle version.
- If you build via Gradle Wrapper, any Gradle version installed in the PC is ignored.
- You will save time on installing and setting Gradle in every developer's machine.

6.7 Download the Eclipse Project

This was an example of Gradle Wrapper.

Download

You can download the full source code of this example here: **Gradle Wrapper Project**

Chapter 7

Gradle NetBeans Example

In this example, we will explain how to integrate Gradle with NetBeans IDE and how to perform basic Gradle tasks.

7.1 Why use Gradle in NetBeans IDE?

Gradle is a powerful tool for building and automation. It has many advantages, enabling the development of build scripts that are cross-platform and cross-IDE. You should have no problems migrating your Gradle-based applications to a different IDE or Continuous Integration server.

7.2 Requirements

- NetBeans IDE 8.0.2
- JDK 1.7.0_67 (or higher)
- Gradle 2.3 (or higher)

7.3 Installing the Gradle Plugin in NetBeans IDE

The tasks that we will perform in this post are simple. To install Gradle in NetBeans IDE, go to Tools > Plugins > Available Plugins.

Then, type "Gradle" and choose "Gradle Support" plugin, of which the author is Attila Kelemen, and click Install.

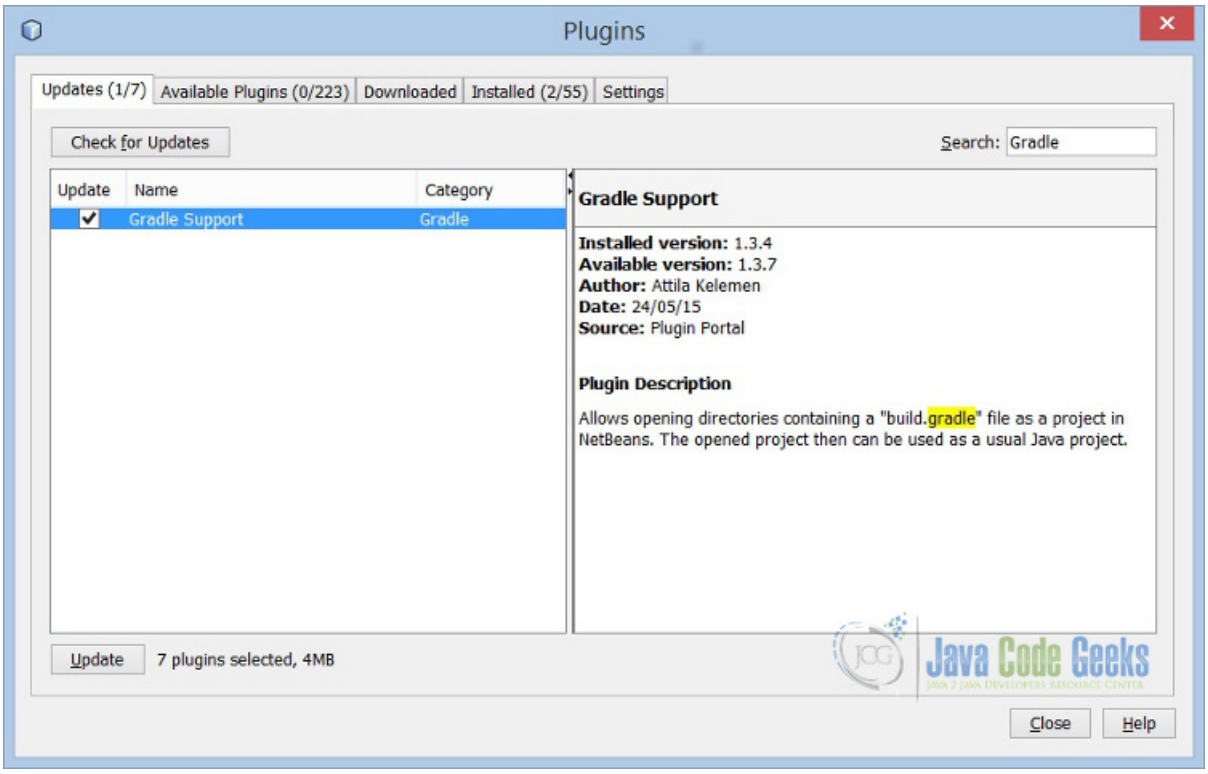

Figure 7.1: Gradle NetBeans Plugin

7.4 How to Start using Gradle in NetBeans?

We will assume that you have installed Gradle in your computer if you didn't have yet, take a look here.

In this step we have to set the default configuration for Gradle's environment.

Then, go to Menu Tools > Options > Miscellaneous > Gradle.

Here we need to set 2 important and critical configurations in the Gradle Installation category.

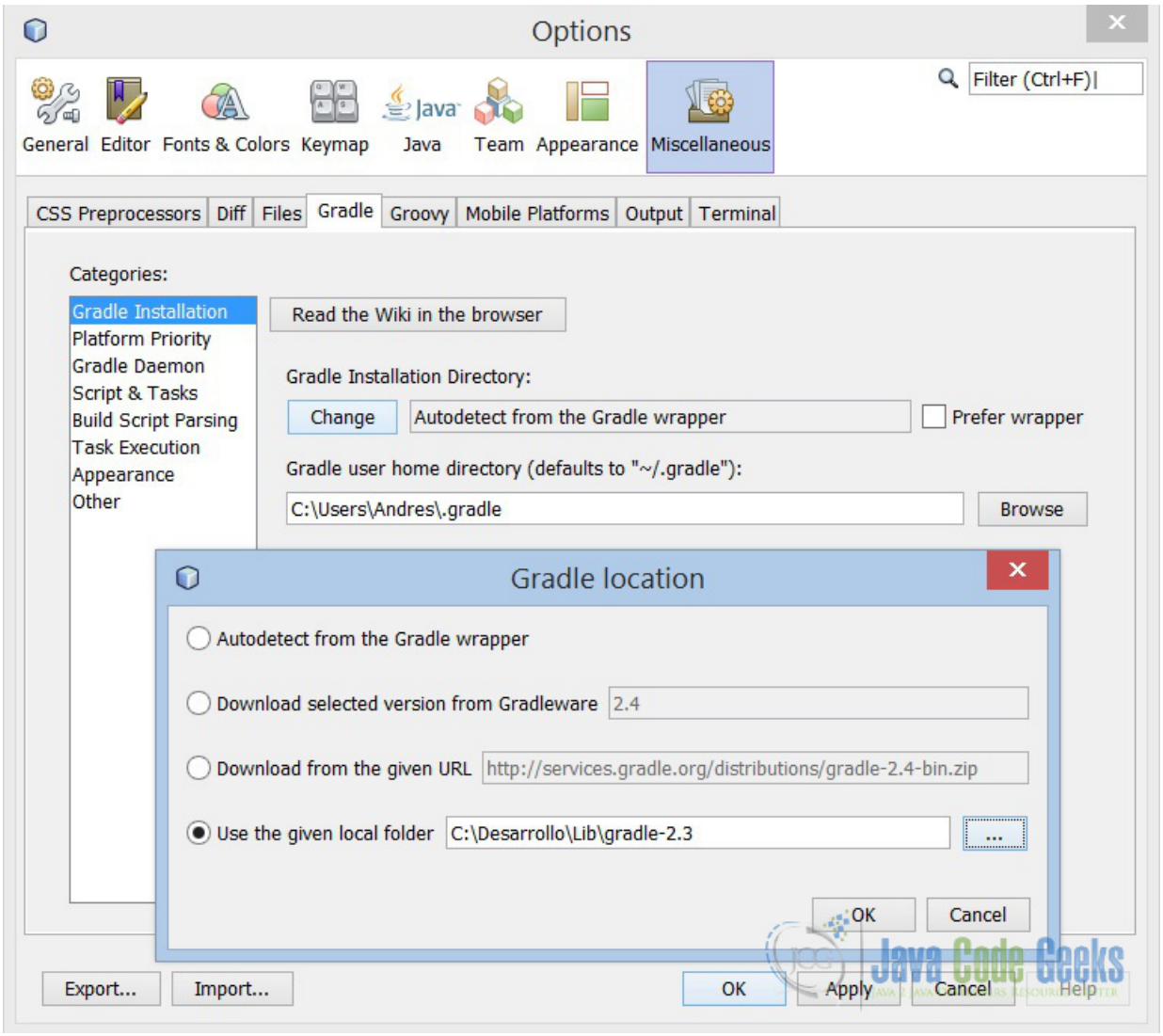

Figure 7.2: Gradle NetBeans Plugin Location

First, make click in change button to set Gradle installation directory by giving a local folder, then do the another configuration, set Gradle user home, if you are a Windows user is on C:Users%USER_NAME%.gradle folder.

7.5 Create your first Gradle Project

In NetBeans IDE we need to set a Root Project to start working, so open new project wizard, and in Gradle category choose Gradle Root Project and click next, give a project's name and Maven Group Id, then click Finish.

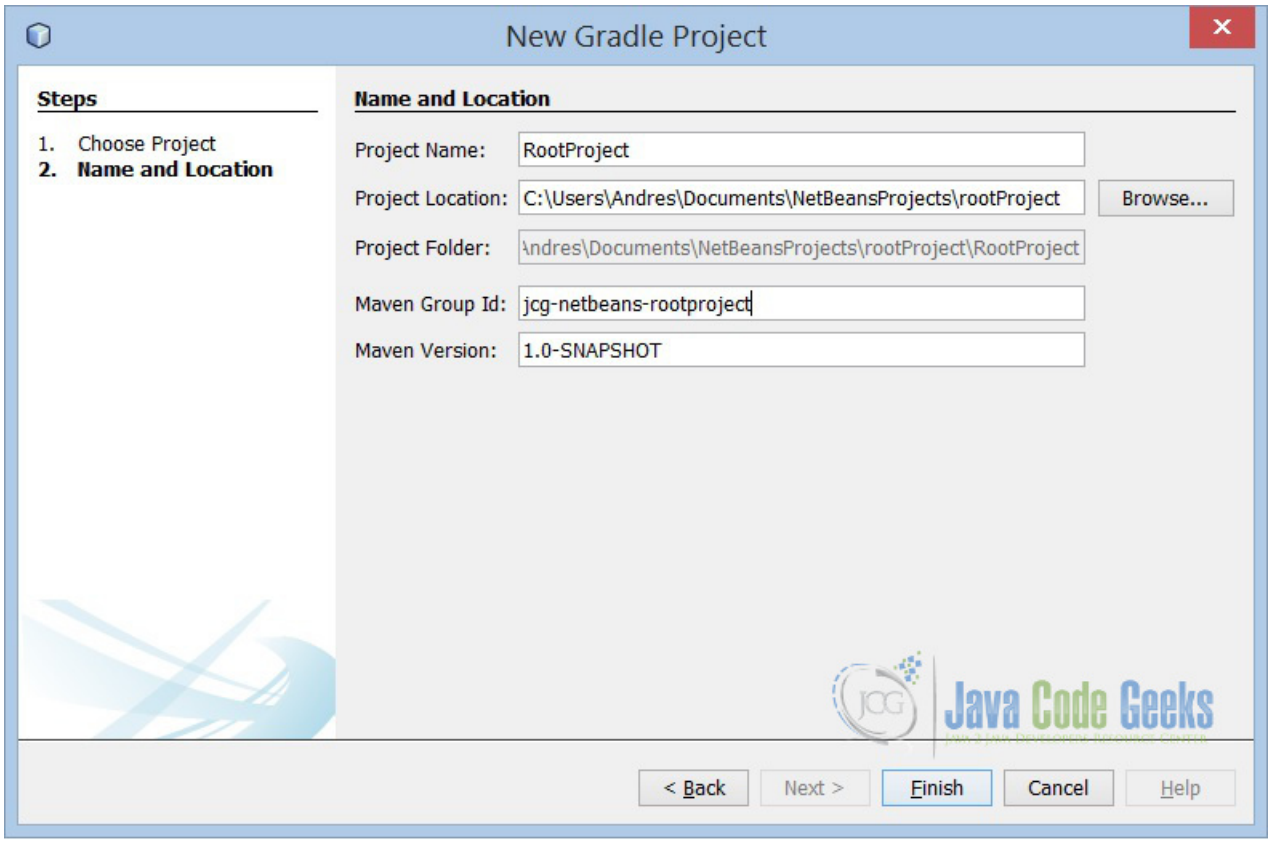

Figure 7.3: Gradle Root Project

Now, we have to add subprojects to the root project and make them run, so repeat the previous step but instead of choose root project, please select Gradle subproject. Then set the location of your root project so:

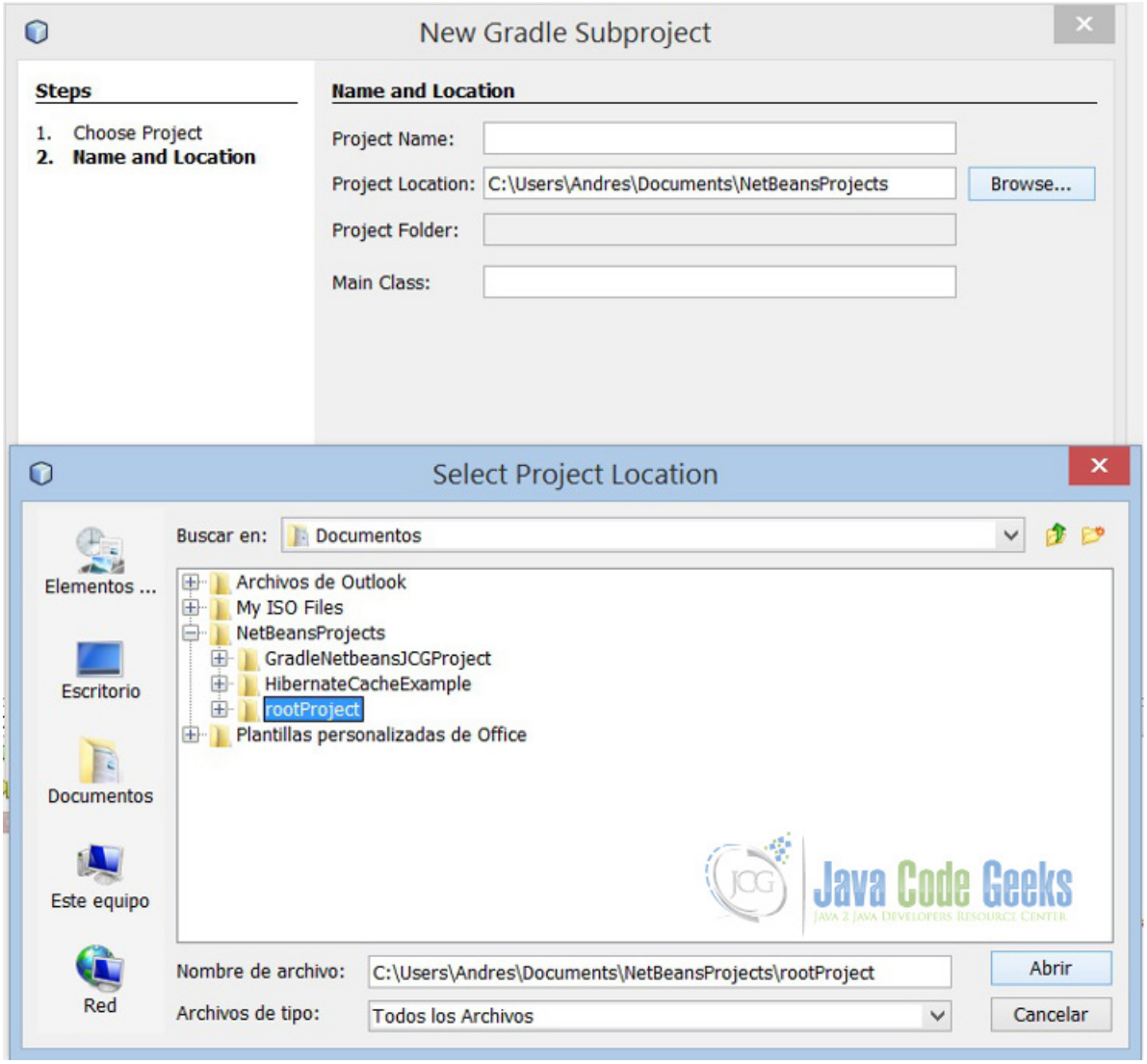

Figure 7.4: Gradle Subproject

After this step, we have to set in `common.gradle` script the property `sourceCompatibility` to the JDK level 1.6, 1.7 or 1.8, for this example we change it to 1.7. Once you have adjusted the property, ensure that both projects (root and subproject) have the same configuration, make right click in the name of both projects and set all properties to Java compilation level 1.7.

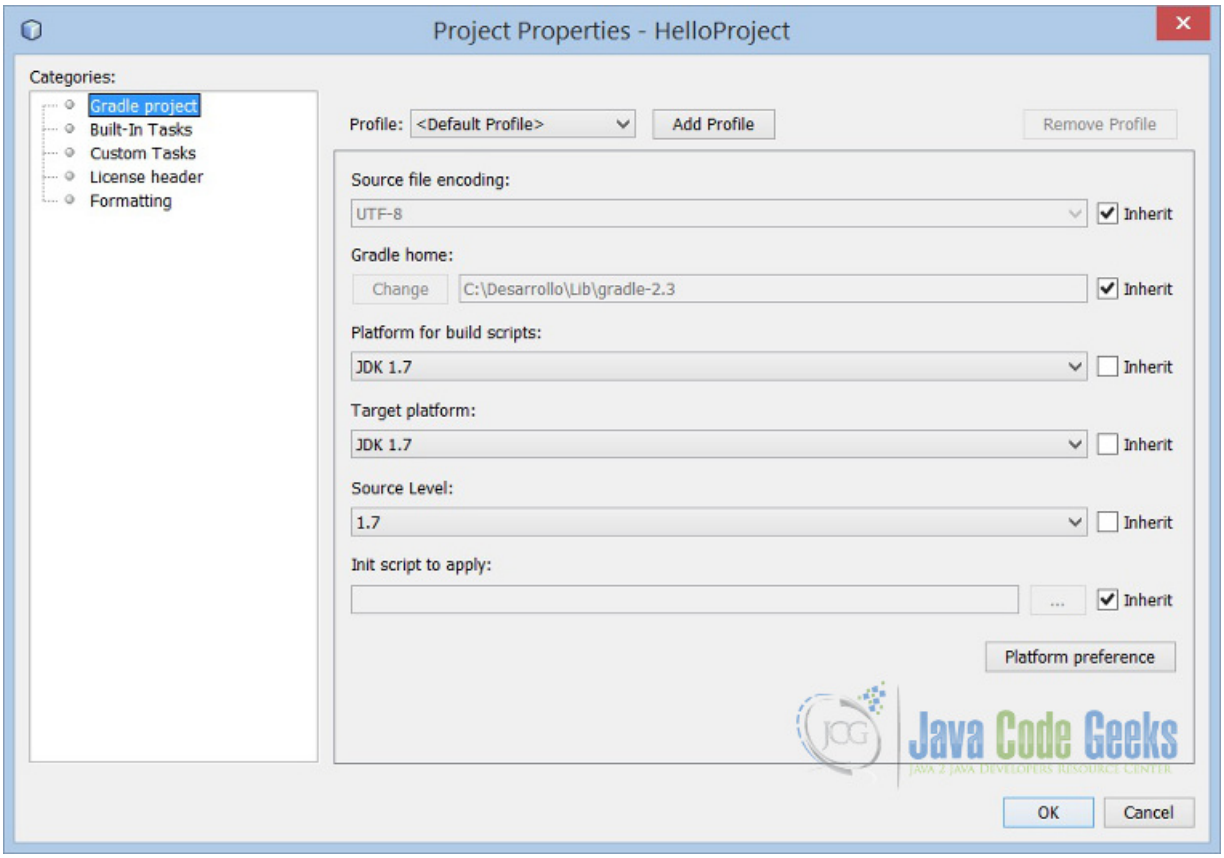

Figure 7.5: Gradle Project Properties

7.6 Testing Gradle Project

Then, we have to add a simple class that executes a print console, so create a package and class on Gradle subproject called Greeting with only a main method.

Greeting.java

```
package com.javacodegeeks.gradle.netbeans;

/**
 *
 * @author Andres Cespedes
 * @since 1.7
 */
public class Greeting {

    public static void main(String[] args) {
        System.out.println("Hello World JCG in Gradle NetBeans");
    }
}
```

The last step is to set this class as Main Class, so in `build.gradle` script of subproject HelloProject type full name without extension. The build.gradle script may looks like this:

```
// Note: "common.gradle" in the root project contains additional initialization
//   for this project. This initialization is applied in the "build.gradle"
//   of the root project.
```

```
// NetBeans will automatically add "run" and "debug" tasks relying on the
// "mainClass" property. You may however define the property prior executing
// tasks by passing a "-PmainClass=" argument.
//
// Note however, that you may define your own "run" and "debug" task if you
// prefer. In this case NetBeans will not add these tasks but you may rely on
// your own implementation.
if (!hasProperty('mainClass')) {
        ext.mainClass = 'com.javacodegeeks.gradle.netbeans.Greeting'
}

dependencies {
        // TODO: Add dependencies here
        //   but note that JUnit should have already been added in parent.gradle.
        //   By default, only the Maven Central Repository is specified in
        //   parent.gradle.
        //
        // You can read more about how to add dependency here:
        //   https://www.gradle.org/docs/current/userguide/dependency_management.html#sec: ↩
            how_to_declare_your_dependencies
}
```

So, make right click on subproject **HelloProject** then `Run`, the output will be this: .NetBeans Gradle Output NetBeans Gradle Output

7.7 Key Points

Tips

- Gradle NetBeans Plugin is unofficial yet, but Kelemen has done a great job.
- In NetBeans Gradle needs a multiproject structure, is mandatory a root project to work.
- All Gradle configuration needs to be done right in both places to ensure correctly operation, gradle build scripts and project properties.

7.8 Download the NetBeans Project

This was an example of Gradle NetBeansPlugin.

Download

You can download the full source code of this example here: **Gradle NetBeans Project**

Chapter 8

Gradle War Plugin (& Tomcat) Example

In this example, we will learn how to package a WAR File and how to deploy it on Tomcat server, using Gradle War Plugin and Gradle Tomcat Plugin.

8.1 Introduction to Gradle War Plugin

This is a basic Gradle plugin that allows packaging web applications in WAR files. This plugin adds a war task for us that we can invoke instead of creating a war task ourselves. Basically, it's a copy file task, that maintains the project's structure on WAR file, but it follows some configurations.

- `src/main/java` default directory where java source files are. We can customize this with sourcesets.
- `src/main/webapp` default web sources directory for the content of the WAR file. We can change the value with the `webAppDirName` property.
- Adds 2 dependency configurations, providedCompile and providedRuntime, any dependencies added to these configurations are not packaged in WAR file, so are not copied to the WEB-INF/lib directory. Tipically libraries declared in these dependecy configurations reside on the container.

8.2 What we need to start?

- As IDE: Eclipse Luna 4.4
- Java JDK 1.7
- Gradle 2.3 or higher, already installed and configured.

But the main idea is to edit a `build.gradle` script and you can do this with only a plain text editor, also you should have a java web project ready to work on it.

8.3 Environment Configuration

Please set your Gradle environment variables and install the Gradle plugin on your IDE. To avoid to be repetitive, visit this previous posts that show how to configure your Gradle Environment. Gradle Hello World Tutorial

8.4 Create Java Web Application

On eclipse create a new Dynamic Web Project, go to File > New > Dynamic Web Project, choose your tomcat container and servlet version (in this example is servlet 3.0).

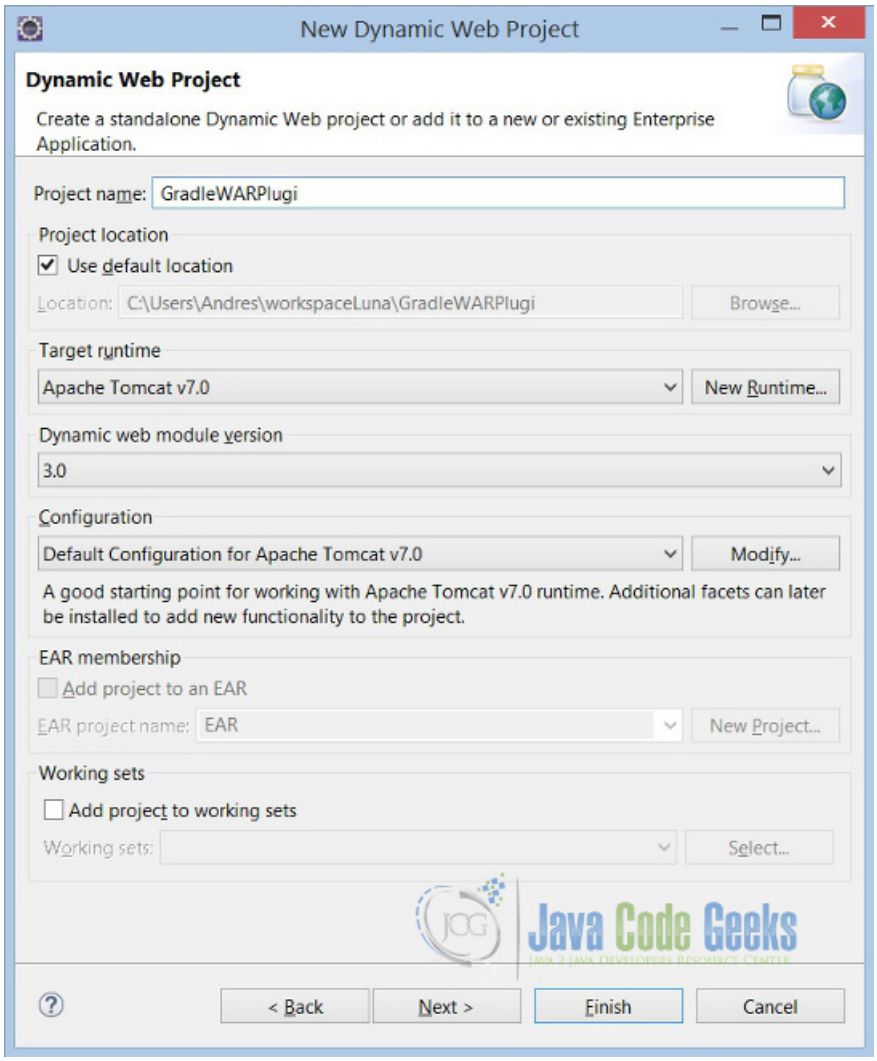

Figure 8.1: Gradle Web Project

Then, create a JSP index file to test the web application. On WebContent folder root create a new index.jsp and edit it with a hello world message.

index.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
        pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "https://www.w3.org/TR/html4 ↩
    /loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>JCG Gradle War Project</title>
</head>
<body>I'm deployed with Gradle on JCG War tutorial!
</body>
```

```
</html>
```

So the `web.xml` file of the project may looks like that:

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance" xmlns="https://java.sun.com ↩
    /xml/ns/javaee" xsi:schemaLocation="https://java.sun.com/xml/ns/javaee https://java.sun. ↩
    com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>GradleWarPlugin</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

The last step is to test the dynamic web application running on Tomcat server, so make right click to the project and Run As "Run On Server" to check the application is on a valid state, hereafter we will use Gradle to do all tasks.

8.5 Using Gradle WAR Plugin

Create a build.gradle file on project's root and so let's configure it.

build.gradle

```
buildscript {
    repositories {
        jcenter()
    }

    dependencies {
        classpath 'com.bmuschko:gradle-tomcat-plugin:2.0'
    }
}

apply plugin: 'java'
apply plugin: 'war'
apply plugin: 'com.bmuschko.tomcat'

// JDK version source compatibility
sourceCompatibility = 1.7
// project version
version = '1.0'
// War file name
war.baseName = 'jcg-gradle-war-example'
// Web directory, this overrides the default value "webapp"
project.webAppDirName = 'WebContent'

repositories {
    mavenLocal()
    mavenCentral()
}

// Set source directory
sourceSets {
        main {
            java {
                srcDir 'src'
            }
        }
```

```
    }

// dependencies to run on tomcat, are mandatory for tomcat plugin
dependencies {
    def tomcatVersion = '7.0.57'
    tomcat "org.apache.tomcat.embed:tomcat-embed-core:${tomcatVersion}",
            "org.apache.tomcat.embed:tomcat-embed-logging-juli:${tomcatVersion}"
    tomcat("org.apache.tomcat.embed:tomcat-embed-jasper:${tomcatVersion}") {
        exclude group: 'org.eclipse.jdt.core.compiler', module: 'ecj'
    }
}

// context where tomcat is deployed, by defautl localhost:8080/
tomcatRun.contextPath = '/'
tomcatRunWar.contextPath = '/'
```

Look at the previous file line by line, in the first part we define the repositories to download all libraries and dependencies. Then we must apply plugins to perform all necessary tasks, java plugin is to compile and copy java files, war plugin is to package all files and resources into WAR file and tomcat Benjamin Muschko plugin is to deploy the generated war on Tomcat server.

In line 22, we customize the webapp directory value that by default point to `webapp` folder. Repositories task is to define where it's all dependencies and libraries will be downloaded.

The last 2 parts are, sourcesets to set the custom directory of which classes are provided, by default is `src/main/java`. And for last, dependencies is to define the tomcat dependencies that are needed by the tomcat plugin.

8.6 Running Example

To test all our job, we will perform these 2 tasks.

First, execute this task `gradle war` command on shell console. This task will perform all compiling and packaging required jobs, after that a WAR file will be created ready to deploy on tomcat.

```
C:\Users\Andres\workspaceLuna\GradleWarPlugin>gradle war
:GradleWarPlugin:compileJava UP-TO-DATE
:GradleWarPlugin:processResources UP-TO-DATE
:GradleWarPlugin:classes UP-TO-DATE
:GradleWarPlugin:war UP-TO-DATE

BUILD SUCCESSFUL

Total time: 5.838 secs
C:\Users\Andres\workspaceLuna\GradleWarPlugin>
```

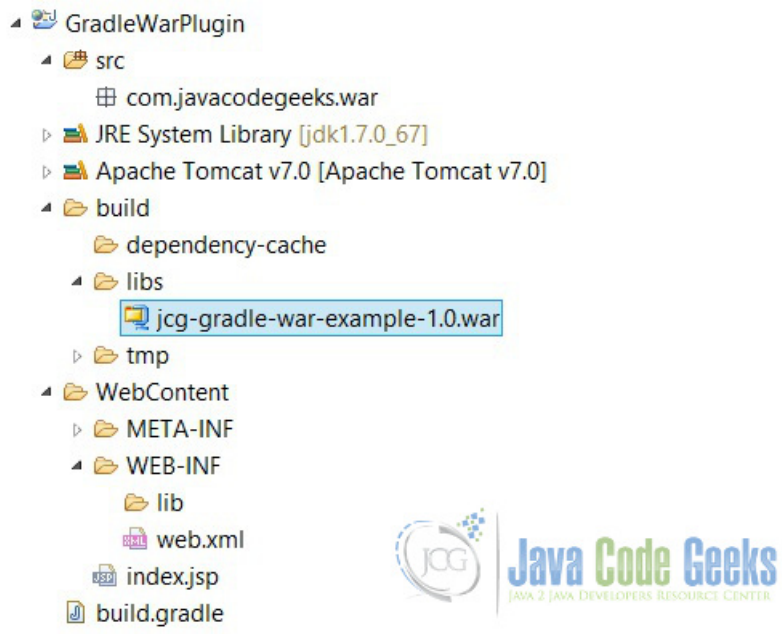

Figure 8.2: Gradle War Project Final Structure

8.7 Deploying WAR File

The last task is to deploy the WAR file on tomcat server so with all previous job only we need to execute this Gradle command, `gradle tomcatRun` or `gradle tR` as abbreviated form on command shell. This will be the output:

```
C:\Users\Andres\workspaceLuna\GradleWarPlugin>gradle tomcatRun
:GradleWarPlugin:compileJava UP-TO-DATE
:GradleWarPlugin:processResources UP-TO-DATE
:GradleWarPlugin:classes UP-TO-DATE
:GradleWarPlugin:tomcatRun
Started Tomcat Server
The Server is running at https://localhost:8080
> Building 75% > :GradleWarPlugin:tomcatRun
```

Only need to access to the deployed URL https://localhost:8080

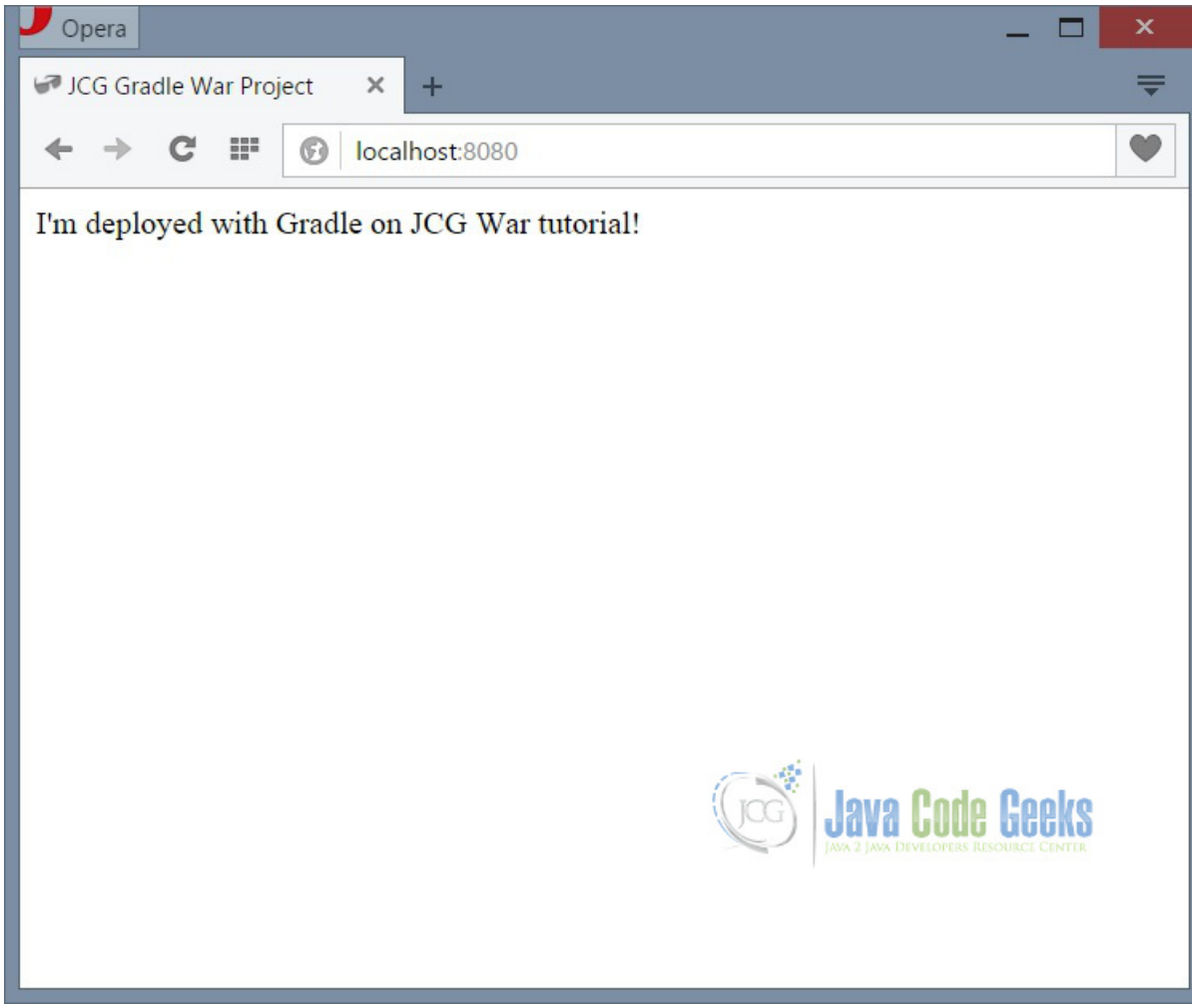

Figure 8.3: Gradle War Deployed on Tomcat

Check that WAR and Tomcat Plugins are powerful, if we update any resource (java files, resources, views) in this case the JSP, only with execute `gradle tomcatRun` task all resources will be updated and packaged, making this task a "one button to production" deploy process.

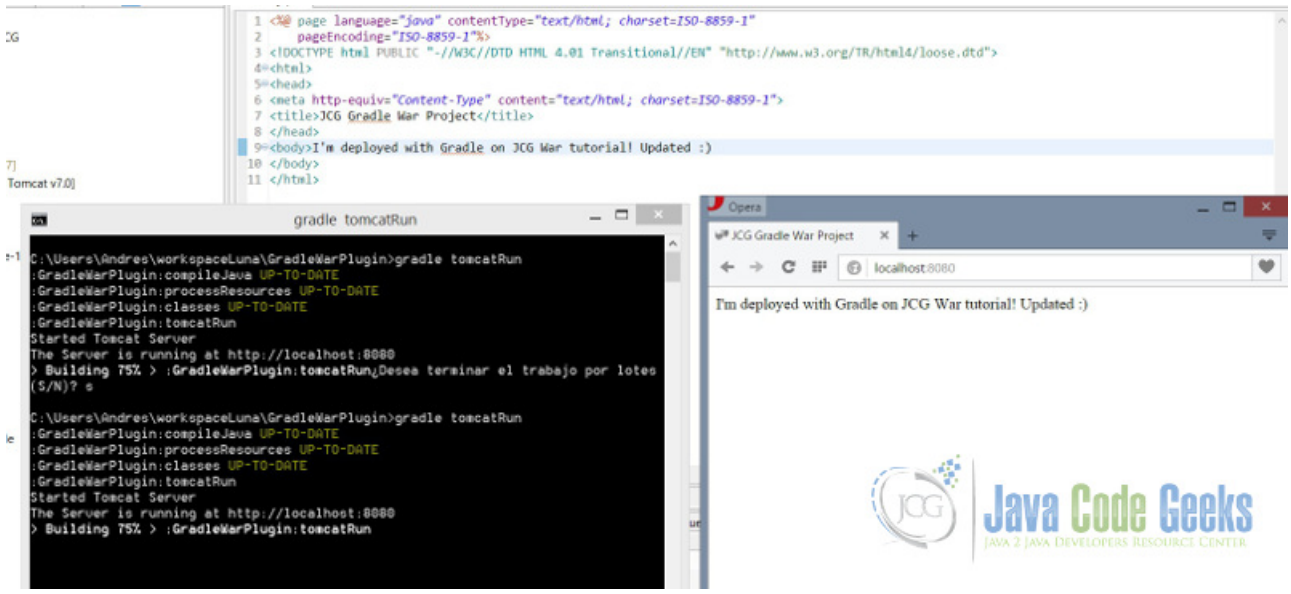

Figure 8.4: Gradle War Deployment Updated

8.8 Key Points

Tips

- Gradle WAR plugin provides a clean tool to package web applications, that isn't IDE dependent.
- It's necessary to set our custom webapp diretory and sourceset values to ensure that WAR packaging works.
- Tomcat Plugin has an embedded Tomcat server that allows to test WAR files. If you want to deploy WAR on remote or external server you need to use another plugin. See cargo plugin.
- With both plugins WAR and Tomcat, the deployment process it becomes a one-button production mode process.

8.9 Download the Eclipse Project

This was an example of Gradle WAR Plugin and how to deploy it using Gradle Tomcat Plugin.

Download

You can download the full source code of this example here: **Gradle WAR and Tomcat Plugin Project**