GWT PROGRAMMING COOKBOOK

Hot Recipes for GWT Development

GWT

Java Code Geeks
Java 2 Java Developers Resource Center

GWT Programming Cookbook

Contents

Preface

Google Web Toolkit, or GWT Web Toolkit, is an open source set of tools that allows web developers to create and maintain complex JavaScript front-end applications in Java. Other than a few native libraries, everything is Java source that can be built on any supported platform with the included GWT Ant build files. It is licensed under the Apache License version 2.0.

GWT emphasizes reusable approaches to common web development tasks, namely asynchronous remote procedure calls, history management, bookmarking, UI abstraction, internationalization, and cross-browser portability. (Source: https://en.wikipedia.org/-wiki/Google_Web_Toolkit)

In this ebook, we provide a compilation of GWT examples that will help you kick-start your own projects. We cover a wide range of topics, from sample applications and interview questions, to Callback functionality and various widgets. With our straightforward tutorials, you will be able to get your own projects up and running in minimum time.

About the Author

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

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

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

Chapter 1

GWT Tutorial for Beginners

1.1 Overview

In this tutorial, we will get to know about *Google Web Toolkit (GWT)*. GWT is a development toolkit for creating optimized web application while programming in Java.

GWT allows you to build a complete web application in Java. *Eclipse IDE* supports its development using plugin *Google Plugin*. GWT compiler compiles Java code into optimized *Java Scripts* compatible for multiple browsers. GWT allows easy development of *AJAX* based web application and provides a rich library of *UI widgets* to support faster development.

GWT Web Application can be customized using *CSS* files. GWT widgets provides the Java APIs for styling widgets. GWT provides its own *RPC (Remote Procedure Call) framework* to communicate between client and server. The implementation of GWT RPC service is based on *Java Servlet* architecture that enables exchange of Java object over *HTTP*. GWT handles *serialization* of the Java objects passing back and forth & the arguments in the method calls and the return value.

Pre-requisite: The readers are expected to know the basics of Java Programming (SE & EE).

1.2 Sample Web Application using GWT

1.2.1 Download Eclipse, install Google plugin and GWT SDK

For the GWT development, Google provides Eclipse plugin and that can be downlodaded from here. If you don't have Eclipse installed yet, go to the Eclipse Download section and get it done. As part of this article, we will be using *Eclipse 4.5 (marsh)*.

1.2.2 Steps to install Eclipse plugin for GWT development

Go to Help → Install New Software...

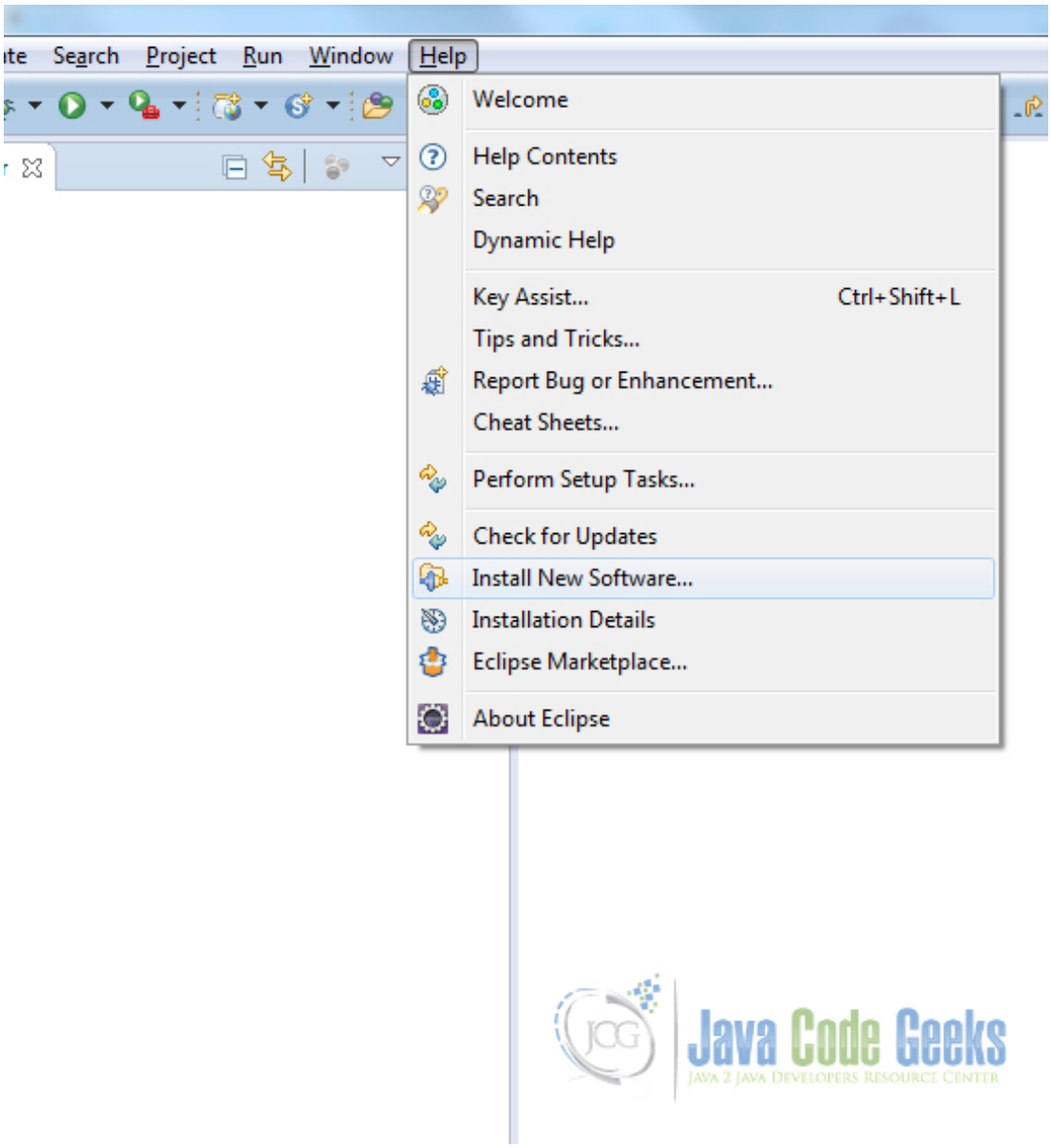

Figure 1.1: Installing Google Plugin - Step 1

Enter the URL to download Google Plugin which one is compatible with your Eclipse version. Here I am using Eclipse 4.5. Click on *Add* button.

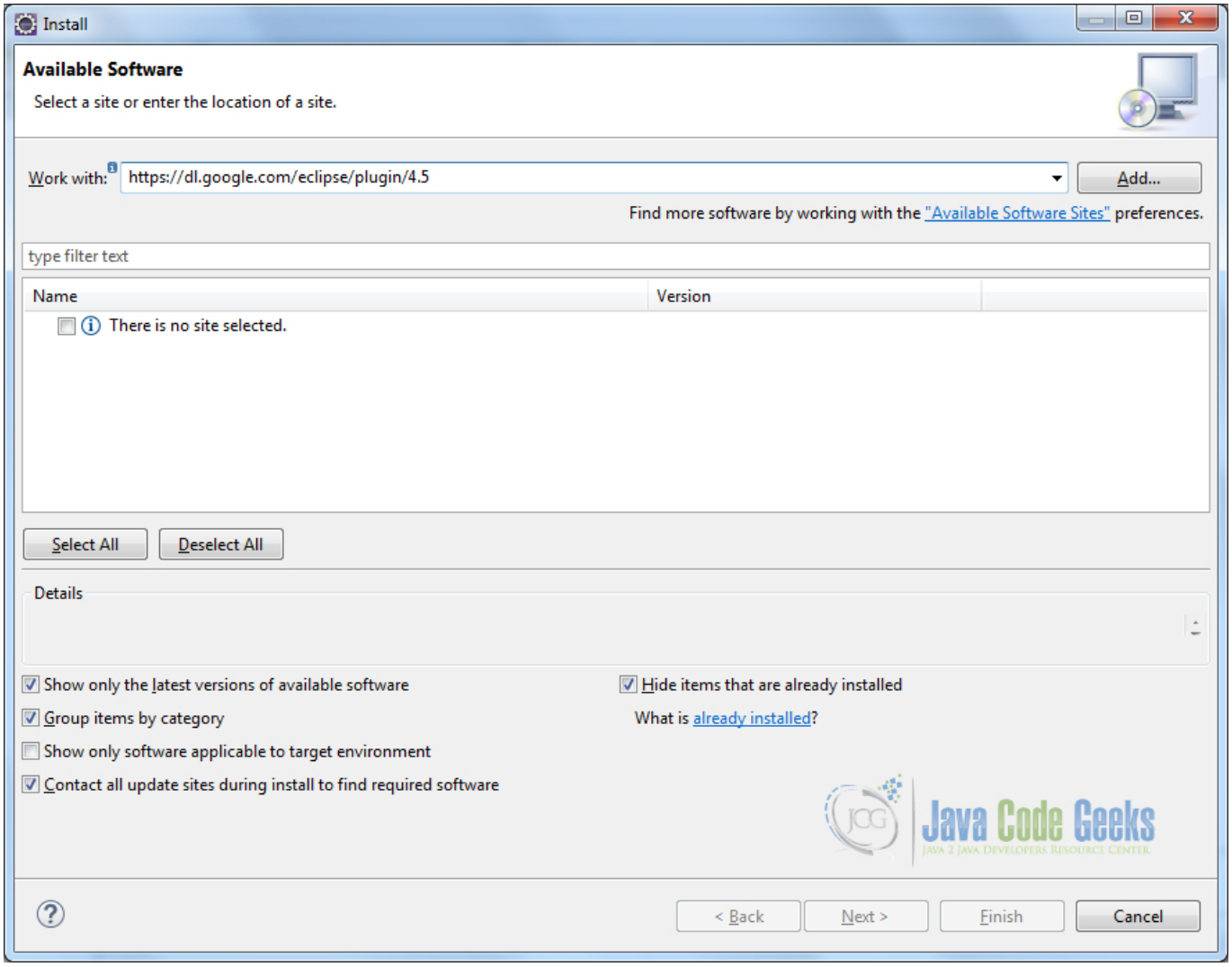

Figure 1.2: Installing Google Plugin - Step 2

A pop-up will appear to Add repository. Provide a relevant name and URL to download Google Plugin. Click on ‘OK’ button.

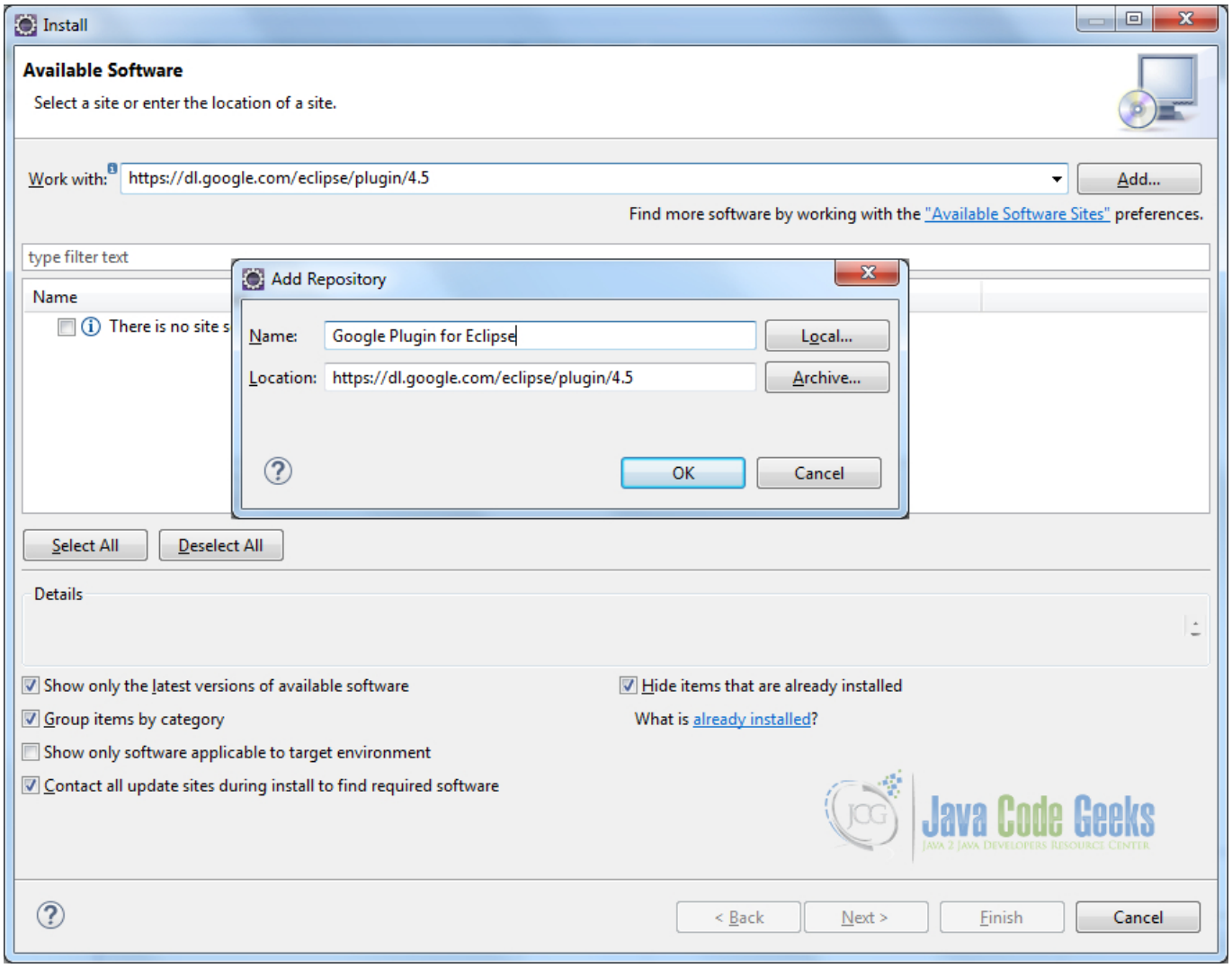

Figure 1.3: Installing Google Plugin - Step 3

It might take few seconds to locate available software within given repository. Once it showsthe available softare under given repository, select the check box for 'Google Plugin for Eclipse', *Developer Tool* and *SDK*. Click on 'Next' button.

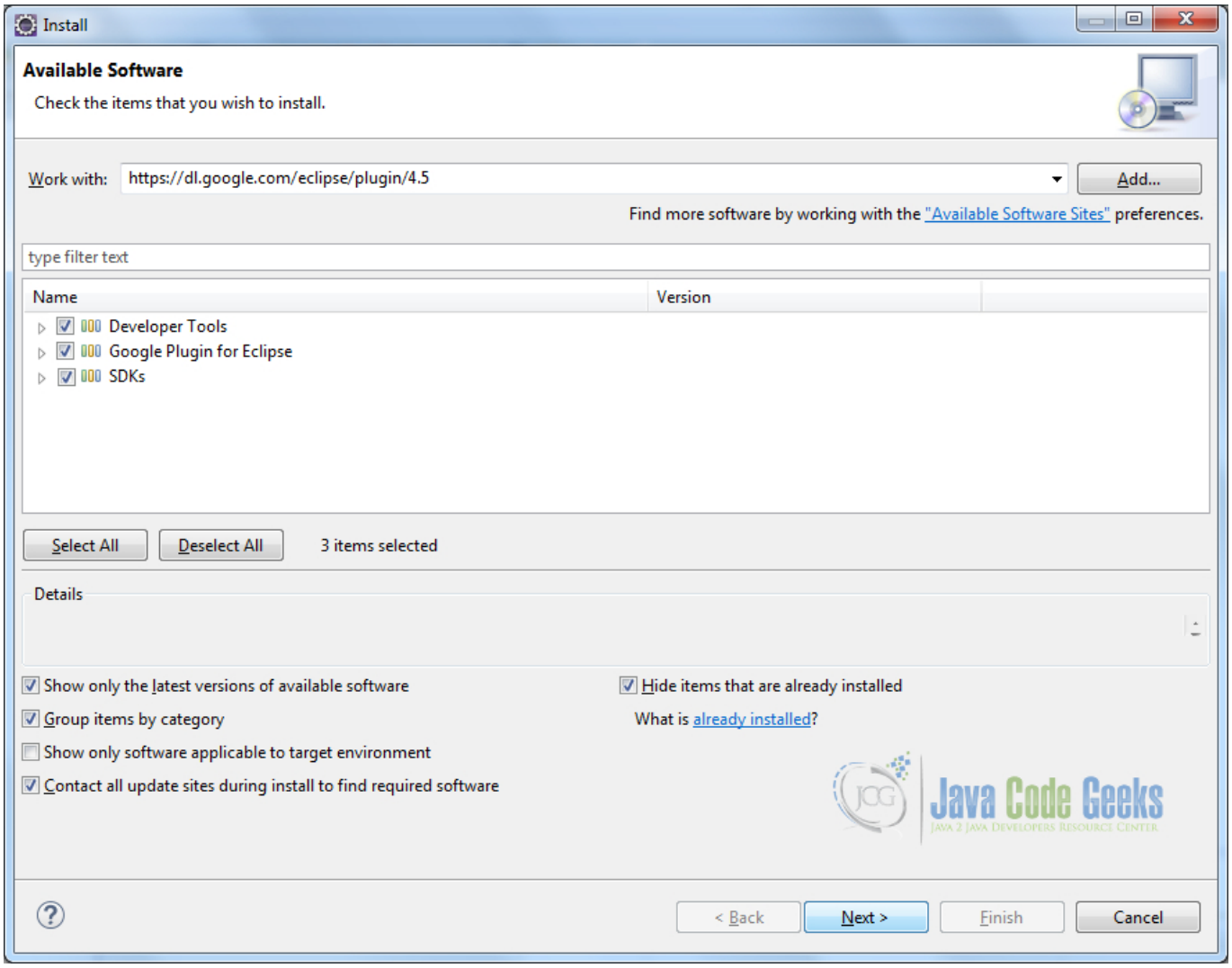

Figure 1.4: Installing Google Plugin - Step 4

It will show plugin details for review just before installing it. Click on *Next* button.

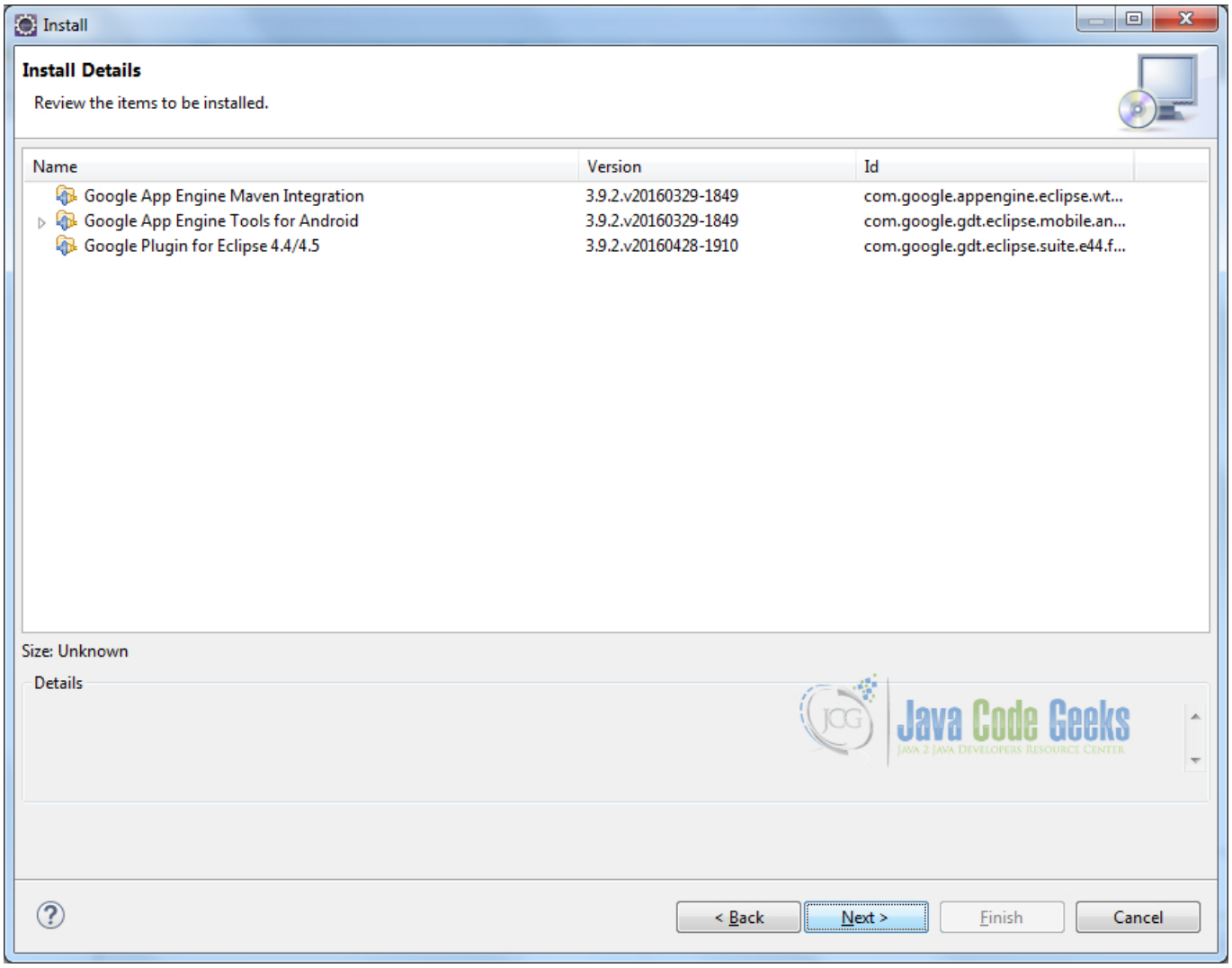

Figure 1.5: Installing Google Plugin - Step 5

After this, accept the license agreement and finish the Google Plugin installation. After Eclipse restart, Goolge Plugin is visible on Eclipse's toolbar.

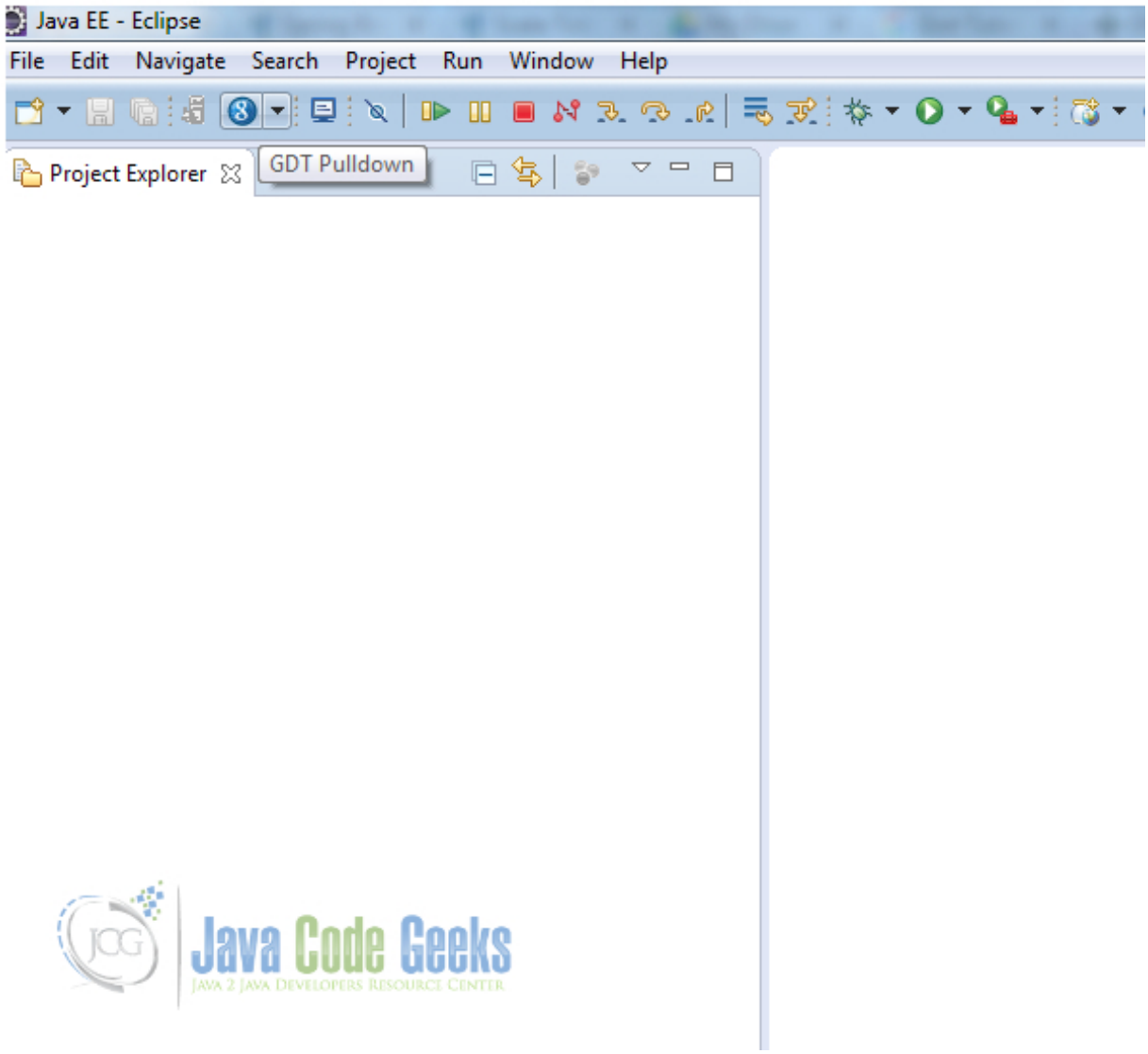

Figure 1.6: Installing Google Plugin - Step Final

1.2.3 Creating Sample Web Application in GWT

1.2.3.1 Create a new project using GWT Development toolkit

Go to File → New → Other. . .

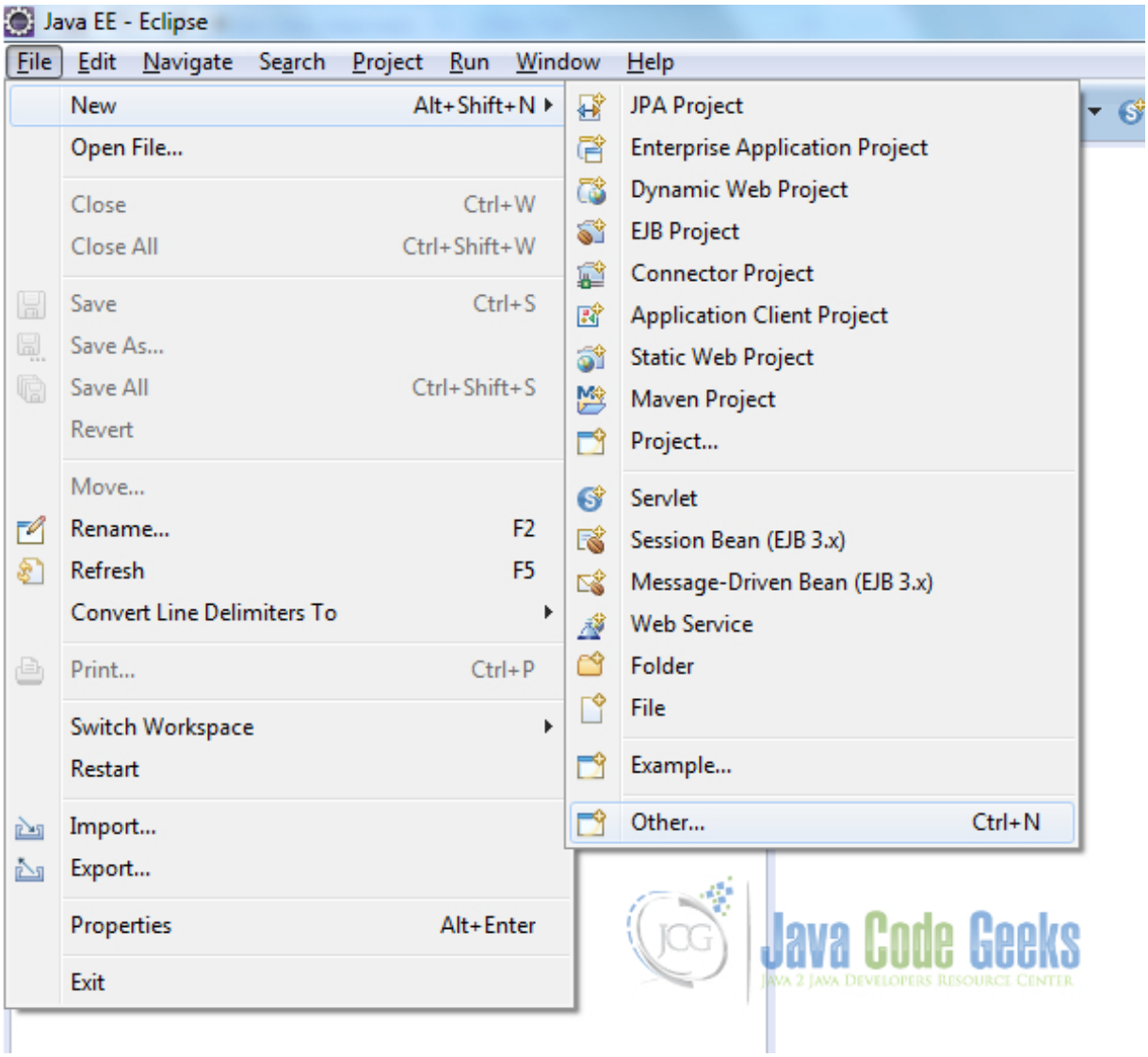

Figure 1.7: Creating GWT Web App Project - Step 1

Select Google Web Application wizard and click on ‘Next’ button.

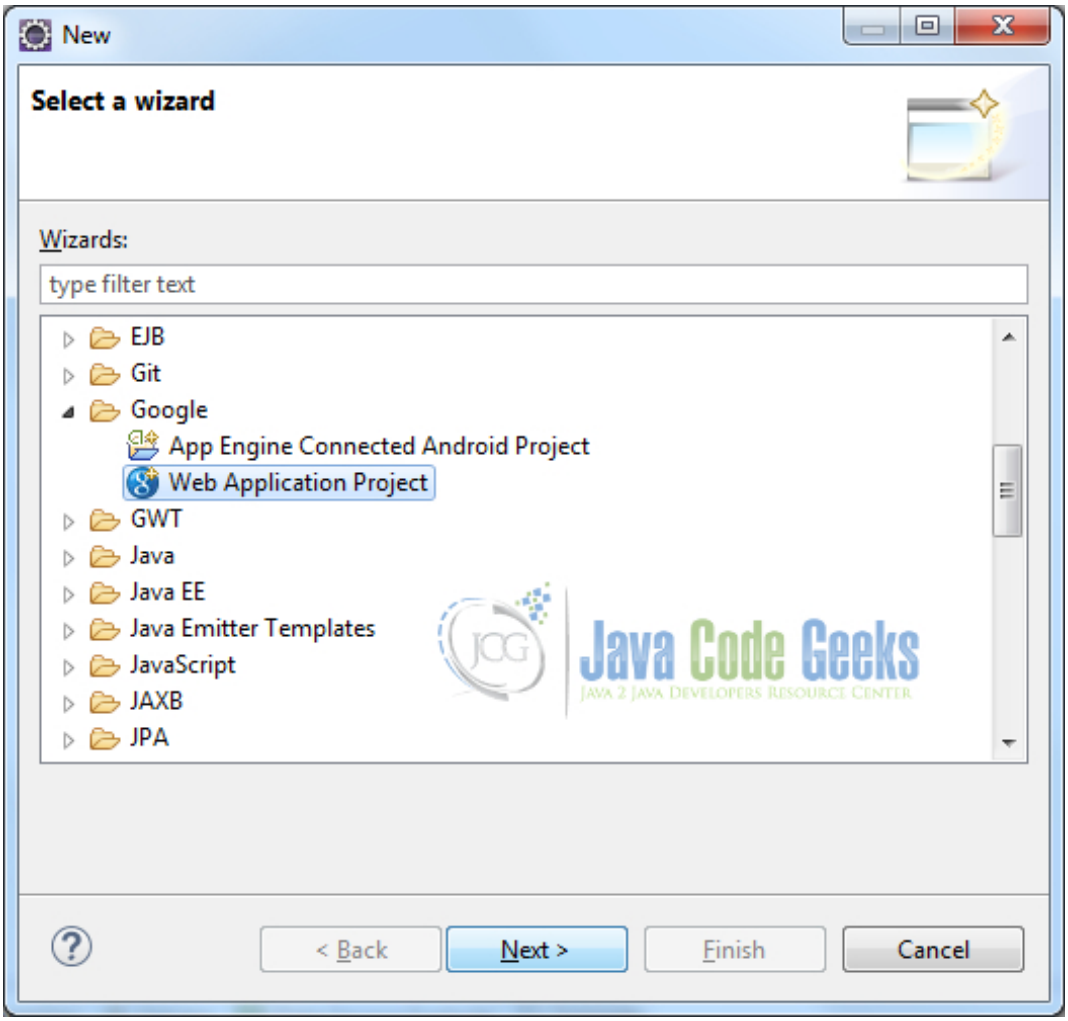

Figure 1.8: Creating GWT Web App Project - Step 2

Provide project name and package name. Make sure that you have selected the checkbox to 'Use GWT'. Here my intention is not using the Google App Engine that may slow down Eclipse significantly. Moreover, if you have created a Google Web Toolkit project you don't necessarily need to deploy it to Google App Engine. For example, you can deploy the web application on Tomcat or Jboss or any other java based web container.

Here the option to generate sample code is remains checked. This basically create entire example project. Although I am not going to use most of its generated files but this will going to help while understanding the GWT Web Application architecture in details. Click on 'Finish' button.

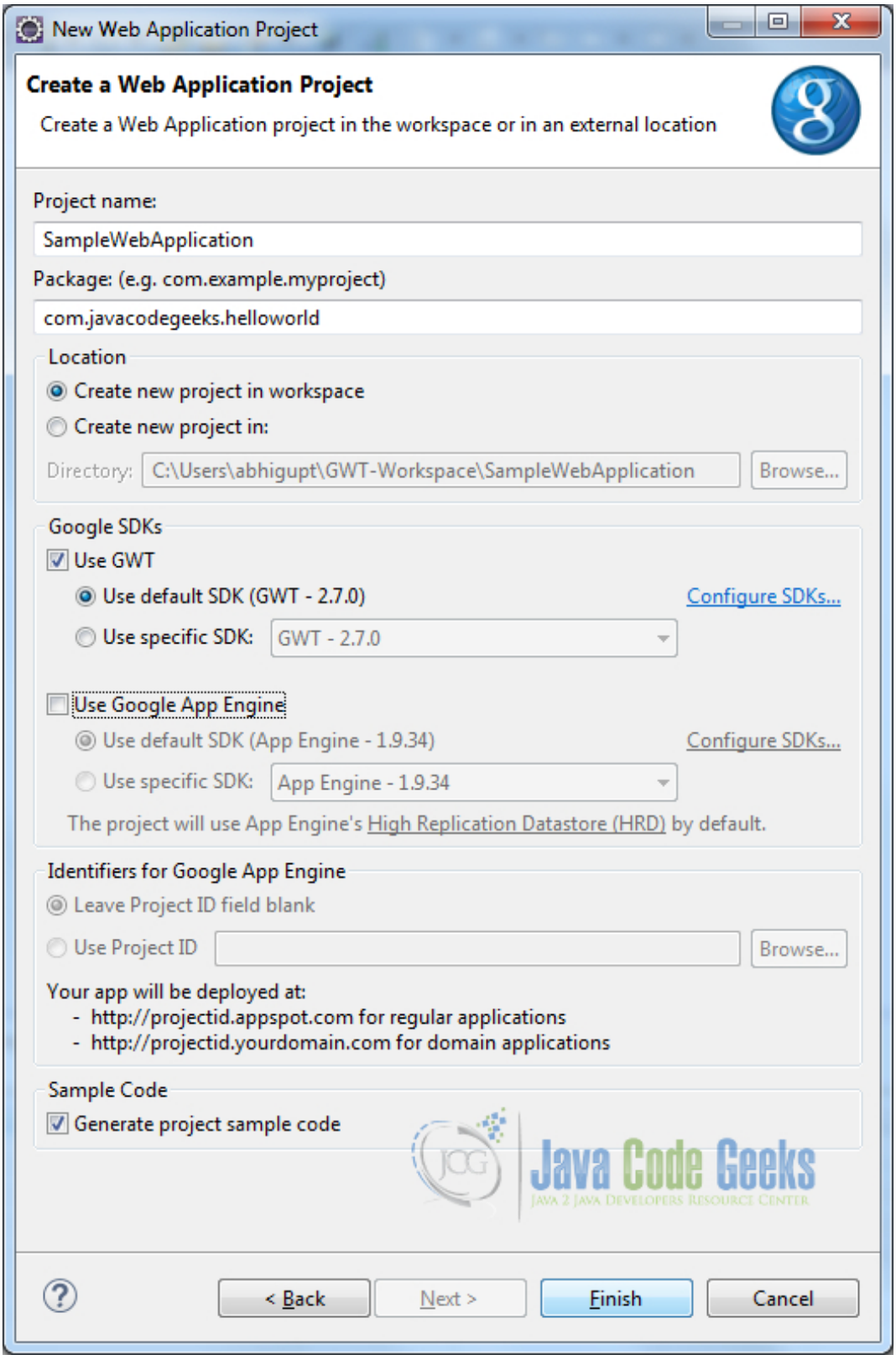

Figure 1.9: Creating GWT Web App Project - Step 3

1.2.3.2 GWT Web Application Project Structure

Client and Source packages

Now when you open the source package you can see a client package that basically contains GUI code, a server package that contains server-side implementation and a shared package which basically for shared classes between various parts of the project.

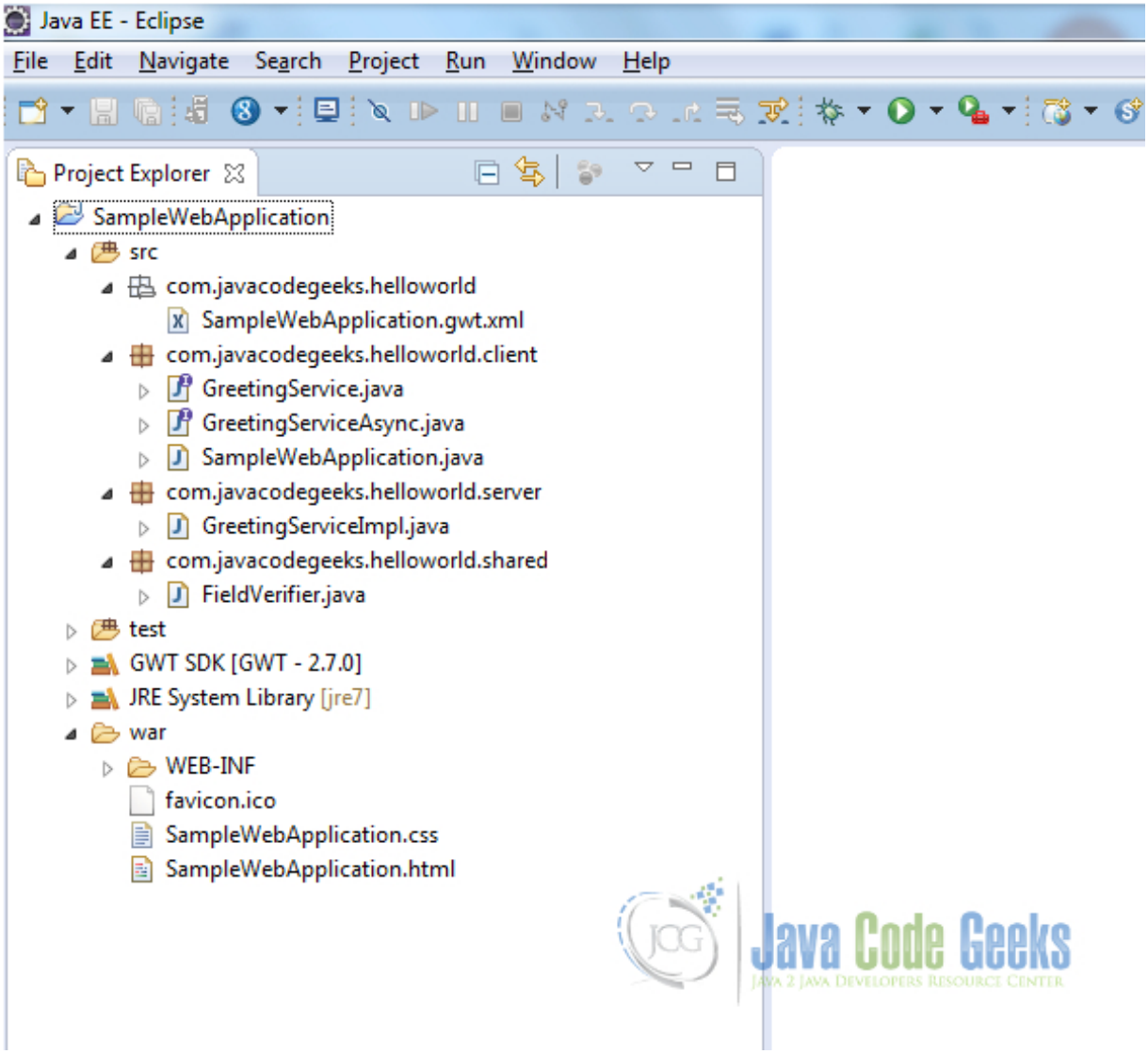

Figure 1.10: Creating GWT Web App Project - Step Final

Entry Point Class

Considering the scope of this tutorial, I am not bothered to server side implementation, so I am going to delete shared package and classes inside server package. There are references to server-side code inside client package named as `GreetingService.java` and `GreetingServiceAsynch.java`, these classes are also need to be deleted. There is a class `SampleWebApplication.java` I am going to keep this file because the is *Entry Point* for a GWT application.

Although this file required to be cleaned as there are many references to server-side code inside it. There is a method `onModuleLoad()`, this is the *Entry Point* of the program or the very first method that gets executed when running the GWT Web Application. This is very similar to `public static void main(String args[])` method in a conventional java program.

SampleWebApplication.java

```
package com.javacodegeeks.helloworld.client;

import com.google.gwt.core.client.EntryPoint;

/**
 * Entry point classes define onModuleLoad()
 */
public class SampleWebApplication implements EntryPoint {

        /**
         * This is the entry point method.
         */
        public void onModuleLoad() {
                // TODO
        }
}
```

Deployment Descriptor

If you are familiar with J2EE programming you will be aware of `web.xml` that is *deployment descriptor* for the *Servlet-based Java Web Application* and used for configuration. As we define *servlet's* entry in `web.xml`, now for this autogenerated code, GWT created servlets are also defined here. As we have deleted server-side code, now these entries in `web.xml` are redundant specification tags. These tags need to be removed as well.

I am going to keep the *welcome file* that is `SampleWebApplication.html`, as this is going to be start page for our GWT Web Application.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://java.sun.com/xml/ns/javaee
              https://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5"
         xmlns="https://java.sun.com/xml/ns/javaee">

  <!-- Servlets -->


  <!-- Default page to serve -->
  <welcome-file-list>
    <welcome-file>SampleWebApplication.html</welcome-file>
  </welcome-file-list>

</web-app>
```

Module Descriptor

File `SampleWebApplication.gwt.xml` under package `com.javacodegeeks.helloworld` is GWT specific configuration file. Let's have a close look into *tags* defined into this file.

Tag `inherits` includes library from core GWT. If we are willing to add any third party library we can add here using tag `inharits`. It references a default style which is based on how GWT control looks.

Tag `entry-point` defines the entry point class that contains the entry point of the GWT Web Application; in this case it is `SampleWebApplication.java`. Besides these, as GWT needs to know which code need to be converted into Java Scripts from Java code, it also contains the references of *client package* and *shared package*.

SampleWebApplication.gwt.xml

```
<?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='samplewebapplication'>
  <!-- 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.helloworld.client.SampleWebApplication'/>

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

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

Welcome file

Now let's have a close look into `SampleWebApplication.html`. We will discuss about several tags and its importance in reference with GWT application.

There is a *CSS* file reference using tag `link` intended for styling. There is `SampleWebApplication.css` file as part of auto-generated files with some default values. The tag `title Web Application Starter Project` where you can mention anything that you want to be displayed as *title* on you web application GUI.

You can see the tag `scripts`. This tag is responsible for including the java scripts code generated from java code after compilation into *HTML* file. When we compile java code using GWT compiler, the java code gets converted into optimized Java Scripts and this tag includes the generated Java Scripts into *HTML* file. If this tag is not here, the GWT code is not going to be included into your web application project.

At the end of the file there is body tag that contains some markups to render HTML file, I am going to delete that.

SampleWebApplication.html

```
<!doctype html>
<!-- The DOCTYPE declaration above will set the     -->
<!-- browser's rendering engine into                -->
<!-- "Standards Mode". Replacing this declaration   -->
<!-- with a "Quirks Mode" doctype is not supported. -->

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">

    <!--                                                               -->
    <!-- Consider inlining CSS to reduce the number of requested files -->
    <!--                                                               -->
    <link type="text/css" rel="stylesheet" href="SampleWebApplication.css">

    <!--                                           -->
```

```
    <!-- Any title is fine                          -->
    <!--                                            -->
    <title>Web Application Starter Project</title>

    <!--                                            -->
    <!-- This script loads your compiled module.    -->
    <!-- If you add any GWT meta tags, they must    -->
    <!-- be added before this line.                 -->
    <!--                                            -->
    <script type="text/javascript" language="javascript" src="samplewebapplication/ ↩
        samplewebapplication.nocache.js"></script>
  </head>

  <!--                                          -->
  <!-- The body can have arbitrary html, or     -->
  <!-- you can leave the body empty if you want -->
  <!-- to create a completely dynamic UI.       -->
  <!--                                          -->
  <body>
    <!-- RECOMMENDED if your web app will not function without JavaScript enabled -->
    <noscript>

        Your web browser must have JavaScript enabled
        in order for this application to display correctly.

    </noscript>


  </body>
</html>
```

Adding UI components into GWT Web Application Project

Before going through this section I will suggest to look into GWT Showcase where you will get familiar with available widgets to develop GUI.

Now we can move to our *Entry Point Class* and will do some coding for UI development. Here we have developed a very basic GWT Web Application GUI that shows a button and a label and that label gets updated on click of the button. Here is the GWT Web Application code that comprises very basic *UI Widgets*: a *GWT Label* and a *GWT Button*. *GWT Label* is getting updated on click of *GWT Button*. All these widgets are added into a *GWT Vertical Panel* and this panel is added into *Root Panel* of *Welcome HTML page*.

SampleWebApplication.java

```
package com.javacodegeeks.helloworld.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.Label;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.VerticalPanel;

/**
 * Entry point classes define `onModuleLoad()`.
 */
public class SampleWebApplication implements EntryPoint {

        /*
         * A vertical panel that hold other components over UI.
         */
        VerticalPanel vPanel;
```

```
        /*
         * A label that gets updated on click of button.
         */
        Label lbl;

        /**
         * This is the entry point method.
         */
        public void onModuleLoad() {

                vPanel = new VerticalPanel();
                lbl = new Label();

                /*
                 * Button and its click handlar.
                 */
                Button btn = new Button("GWT Button");
                btn.addClickHandler(new ClickHandler() {

                        @Override
                        public void onClick(ClickEvent event) {
                                lbl.setText("You clicked GWT Button!");
                        }
                });

                /*
                 * adding label and button into Vertical Panel.
                 */
                vPanel.add(lbl);
                vPanel.add(btn);

                /*
                 * Adding vertical panel into HTML page.
                 */
                RootPanel.get().add(vPanel);

        }
}
```

Running GWT Web Application

GWT Web Application can run into two modes, Development mode and Production mode. In development mode java code runs on JVM whereas in production mode GWT compiler compiles java code and creates java scripts that run on browser.

Here we run our GWT Web Application into 'GWT Super Dev Mode' that compiles java code at run time and this runs java scripts on browser and that can be debugged at browser level. Here I recommend using Google Chrome that provides debug at ease.

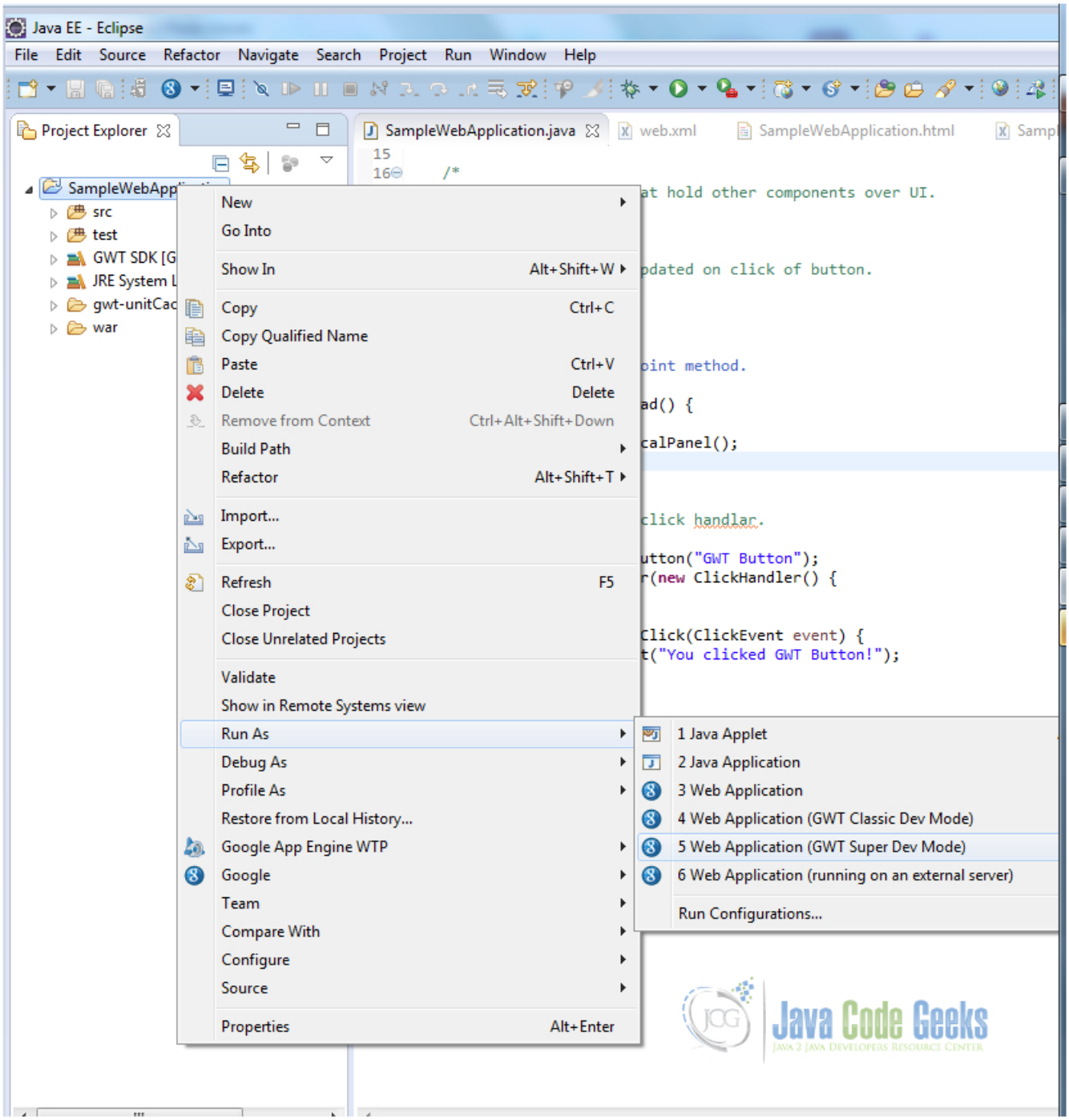

Figure 1.11: Running GWT App - Step 1

Now copy the URL.

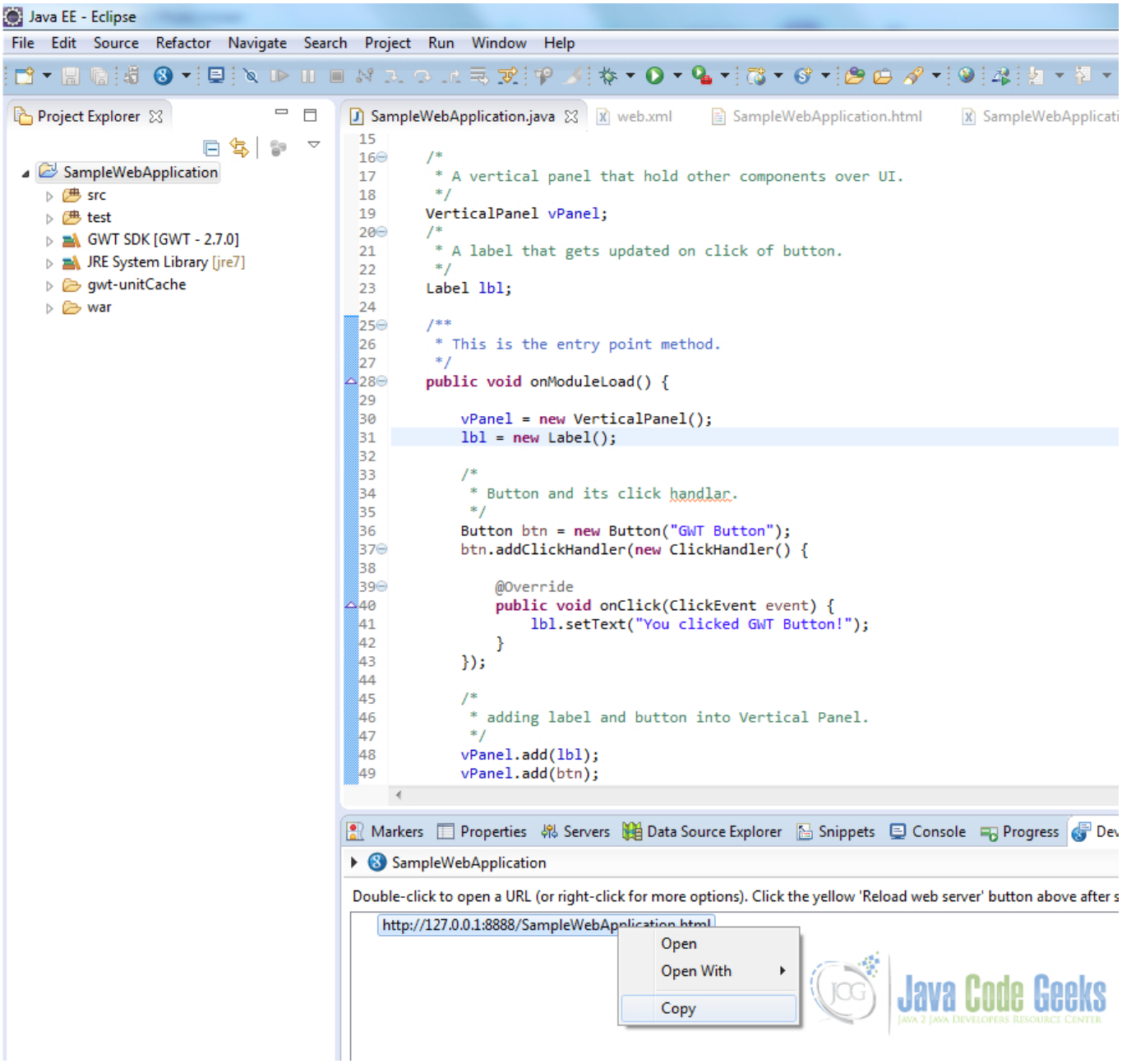

Figure 1.12: Running GWT Web App - Step 2

Paste the *URL* into browser. You can see GWT Web Application is compiling on browser and finally it is running on browser.

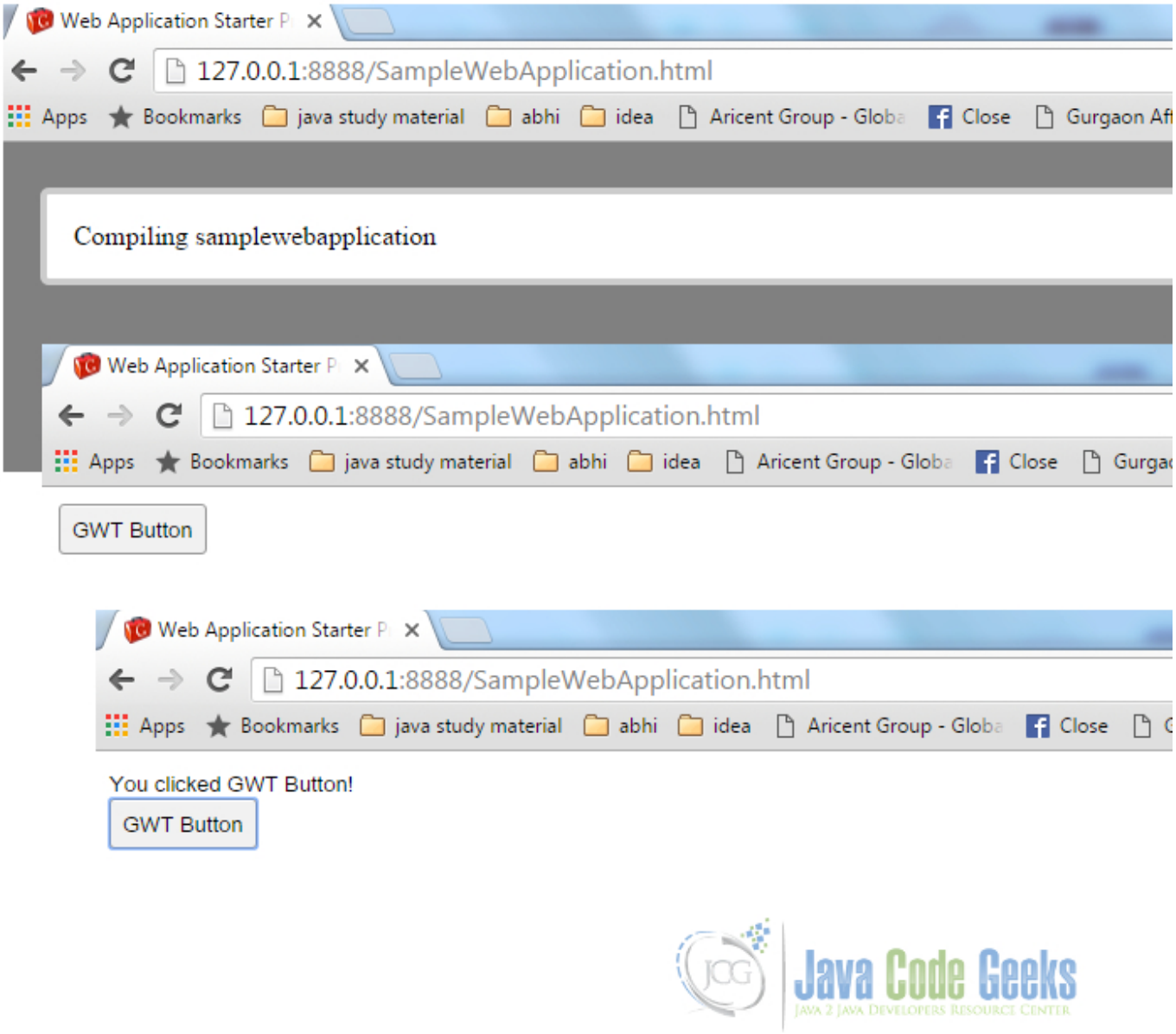

Figure 1.13: Running GWT App - Step Final

1.3 Debugging GWT Web Application

With newer Chrome version, you will not able to run GWT Web Application in classic development mode as its support is now closed from Google. This is because of newer chrome version provides support for 'Super Dev Mode' and enable user to debug using asserts, console logging and error messages.

From GWT Website, *GWT Development Mode will no longer be available for Chrome sometime in 2014, so we improved alternate ways of debugging. There are improvements to Super Dev Mode, asserts, console logging, and error messages.*

1.4 Project References

GWT Overview

GWT Getting Started

GWT Latest Tutorial

GWT Product Release Notes

GWT Sample Showcase

1.5 Conclusion

This tutorial covers the development of Client side of a GWT Web Application. I would suggest to explore more about UI development and hands on GWT Widgets.

We can explore more on Server communication in following tutorial.

1.6 Download Eclipse Project

Download

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

Chapter 2

GWT Sample Application Example

In this example we will learn how to build a simple GWT application from scratch. Google Web Toolkit is a development framework for creating Ajax-enabled web applications in Java. It's open source, completely free. Tools and technologies used in this example are Java 1.8, Eclipse Luna 4.4.2, Eclipse GWT Plugin 2.6

2.1 Introduction

The GWT SDK provides a set of core Java APIs and Widgets. These allow us to write AJAX applications in Java and then compile the source to highly optimized JavaScript that runs across all browsers, including mobile browsers for Android and the iPhone. The GWT SDK contains the Java API libraries, compiler, and development server. It lets us write client-side applications in Java and deploy them as JavaScript.

Constructing AJAX applications in this manner is more productive thanks to a higher level of abstraction on top of common concepts like DOM manipulation and XHR communication. We aren't limited to pre-canned widgets either. Anything we can do with the browser's DOM and JavaScript can be done in GWT, including interacting with hand-written JavaScript.

We can debug AJAX applications in our favorite IDE just like we would a desktop application, and in our favorite browser just like we would if you were coding JavaScript. The GWT developer plugin spans the gap between Java bytecode in the debugger and the browser's JavaScript. Thanks to the GWT developer plugin, there's no compiling of code to JavaScript to view it in the browser. We can use the same edit-refresh-view cycle we're used to with JavaScript, while at the same time inspect variables, set breakpoints, and utilize all the other debugger tools available to us with Java. And because GWT's development mode is now in the browser itself, we can use tools like Firebug and Inspector as we code in Java.

GWT contains two powerful tools for creating optimized web applications. The GWT compiler performs comprehensive optimizations across your codebase - in-lining methods, removing dead code, optimizing strings, and more. By setting split-points in the code, it can also segment your download into multiple JavaScript fragments, splitting up large applications for faster startup time. Performance bottlenecks aren't limited to JavaScript. Browser layout and CSS often behave in strange ways that are hard to diagnose. Speed Tracer is a new Chrome Extension in GWT that enables you to diagnose performance problems in the browser.

When you're ready to deploy, GWT compiles your Java source code into optimized, stand-alone JavaScript files that automatically run on all major browsers, as well as mobile browsers for Android and the iPhone.

2.2 GWT SDK

With the GWT SDK, you write your AJAX front-end in the Java programming language which GWT then cross-compiles into optimized JavaScript that automatically works across all major browsers. During development, you can iterate quickly in the same "edit - refresh - view" cycle you're accustomed to with JavaScript, with the added benefit of being able to debug and step through your Java code line by line. When you're ready to deploy, the GWT compiler compiles your Java source code into optimized, standalone JavaScript files.

Unlike JavaScript minifiers that work only at a textual level, the GWT compiler performs comprehensive static analysis and optimizations across your entire GWT codebase, often producing JavaScript that loads and executes faster than equivalent hand-written JavaScript. For example, the GWT compiler safely eliminates dead code - aggressively pruning unused classes, methods, fields, and even method parameters - to ensure that your compiled script is the smallest it can possibly be. Another example: the GWT compiler selectively inlines methods, eliminating the performance overhead of method calls.

2.3 Installing Eclipse GWT Plugin

You can install the Google Plugin for Eclipse using the software update feature of Eclipse. Below we describe the steps for installing this plugin:

- Start Eclipse
- Select Help > Install New Software... In the dialog that appears, enter the update site URL into the Work with text box: *https://dl.google.com/eclipse/plugin/4.4*.Press the Enter key.

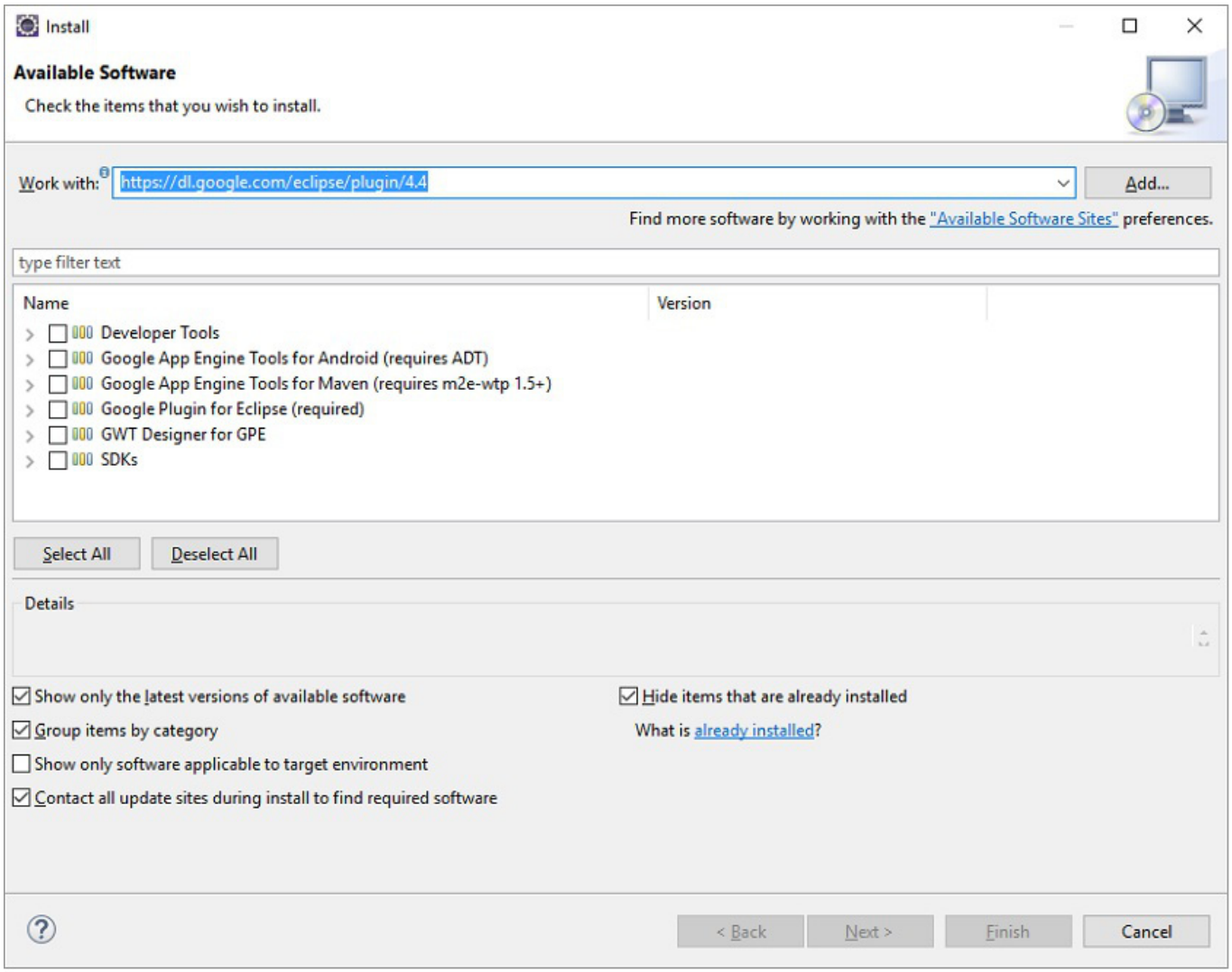

Figure 2.1: GWT Plugin

- The required component is Google Plugin for Eclipse. Select the checkbox next to Google Plugin for Eclipse(required). Click Next.
- Review the features that you are about to install. Click Next.
- Read the license agreements and then select I accept the terms of the license agreements. Click Finish.
- Click OK on the Security Warning.
- You will then be asked be asked if you would like to restart Eclipse. Click Restart Now.

2.4 Creating GWT project

In this section we will learn how to create a new GWT project using Eclipse plugin. To test that the project is configured correctly we will run the application in development mode before deploying it in the reals application server. One of the benefits of using

GWT is that we can leverage the tools, such as refactoring, code completion, and debugging, available in a Java IDE. Below are the steps needed to create a new GWT project using Eclipse.

- Open Eclipse. Click File⇒New⇒Web Application Project. If we don't find *Web Application Project* option in the list, click on *Other* and in the Wizards search box write *Web Application Project*.
- In the Project name text box enter the name of the project (GWTApplication). In the Package text box enter the package name (com.javacodegeeks).

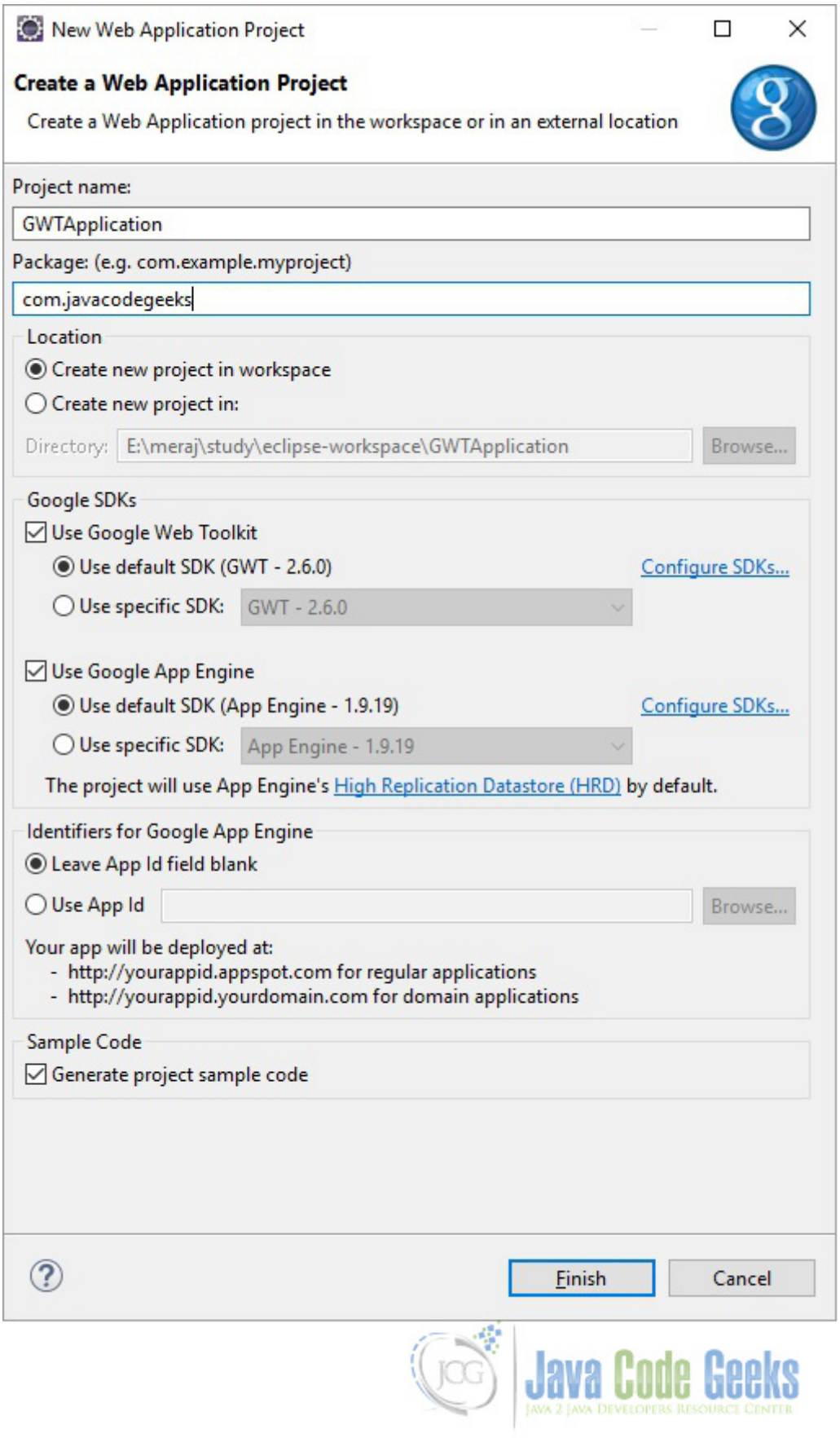

Figure 2.2: Project Setup

- Ensure that the *Use default SDK {$GWT-version}* option is selected.
- (Optional) If you are using Google App Engine, make sure Use Google App Engine is checked and that Use default SDK (App Engine) is selected.
- If you did not install the SDKs when you installed the Google Plugin for Eclipse, you should click Configure SDKs... to specify the directory where GWT (and the App Engine SDK if necessary) was unzipped.
- Make sure the check box under Sample Code (Generate project sample code) is selected.
- Click *Finish* button.

2.5 Development Mode

We can run the GWT application in a development mode which is a very useful feature. This feature can be used to diagnose any UI issues. To start a GWT application in development mode Right click on the GWT project and choose *Debug As*⇒'Web Application'. This creates a Web Application launch configuration for you and launches it. The web application launch configuration will start a local web server and GWT development mode server. You will find a Web Application view next to the console window. Inside you will find the URL for the development mode server. Paste this URL into Firefox, Internet Explorer, Chrome, or Safari. If this is your first time using that browser with the development mode server, it will prompt you to install the GWT Developer Plugin. Follow the instructions in the browser to install.

Once the application is running in a development mode you can make some changes (client-side) and can immediately see the result when refreshing the browser page.

2.6 Testing the default project configuration

In this section we will see how we can test that the project setup is done correctly. To check that all the project components were created, run the starter application in development mode. In development mode, you can interact with the application in a browser just as you would when it's eventually deployed.

- In the Package Explorer window right click on the Project.
- Select Run As⇒Web Application (GWT Classic Dev Mode)
- Copy the URL displayed in the Development Mode window and paste it in the browser and press Enter.

Below is what gets output in the Console window when we run the application:

```
Initializing App Engine server
Feb 26, 2016 12:23:59 PM com.google.apphosting.utils.config.AppEngineWebXmlReader  ↩
    readAppEngineWebXml
INFO: Successfully processed E:\meraj\study\eclipse-workspace\GWTApplication\war\WEB-INF/ ↩
    appengine-web.xml
Feb 26, 2016 12:23:59 PM com.google.apphosting.utils.config.AbstractConfigXmlReader  ↩
    readConfigXml
INFO: Successfully processed E:\meraj\study\eclipse-workspace\GWTApplication\war\WEB-INF/ ↩
    web.xml
Feb 26, 2016 12:23:59 PM com.google.appengine.tools.development.SystemPropertiesManager  ↩
    setSystemProperties
INFO: Overwriting system property key 'java.util.logging.config.file', value 'E:\meraj\ ↩
    study\eclipse\plugins\com.google.appengine.eclipse.sdkbundle_1.9.19\appengine-java-sdk ↩
    -1.9.19\config\sdk\logging.properties' with value 'WEB-INF/logging.properties' from 'E:\ ↩
    meraj\study\eclipse-workspace\GWTApplication\war\WEB-INF\appengine-web.xml'
Feb 26, 2016 12:24:00 PM com.google.apphosting.utils.jetty.JettyLogger info
INFO: Logging to JettyLogger(null) via com.google.apphosting.utils.jetty.JettyLogger
Feb 26, 2016 12:24:00 PM com.google.appengine.tools.development.DevAppServerImpl  ↩
    setServerTimeZone
```

```
WARNING: Unable to set the TimeZone to UTC (this is expected if running on JDK 8)
Feb 26, 2016 12:24:00 PM com.google.apphosting.utils.jetty.JettyLogger info
INFO: jetty-6.1.x
Feb 26, 2016 12:24:02 PM com.google.apphosting.utils.jetty.JettyLogger info
INFO: Started SelectChannelConnector@0.0.0.0:8888
Feb 26, 2016 12:24:02 PM com.google.appengine.tools.development.AbstractModule startup
INFO: Module instance default is running at https://localhost:8888/
Feb 26, 2016 12:24:02 PM com.google.appengine.tools.development.AbstractModule startup
INFO: The admin console is running at https://localhost:8888/_ah/admin
Feb 26, 2016 12:24:02 PM com.google.appengine.tools.development.DevAppServerImpl doStart
INFO: Dev App Server is now running
```

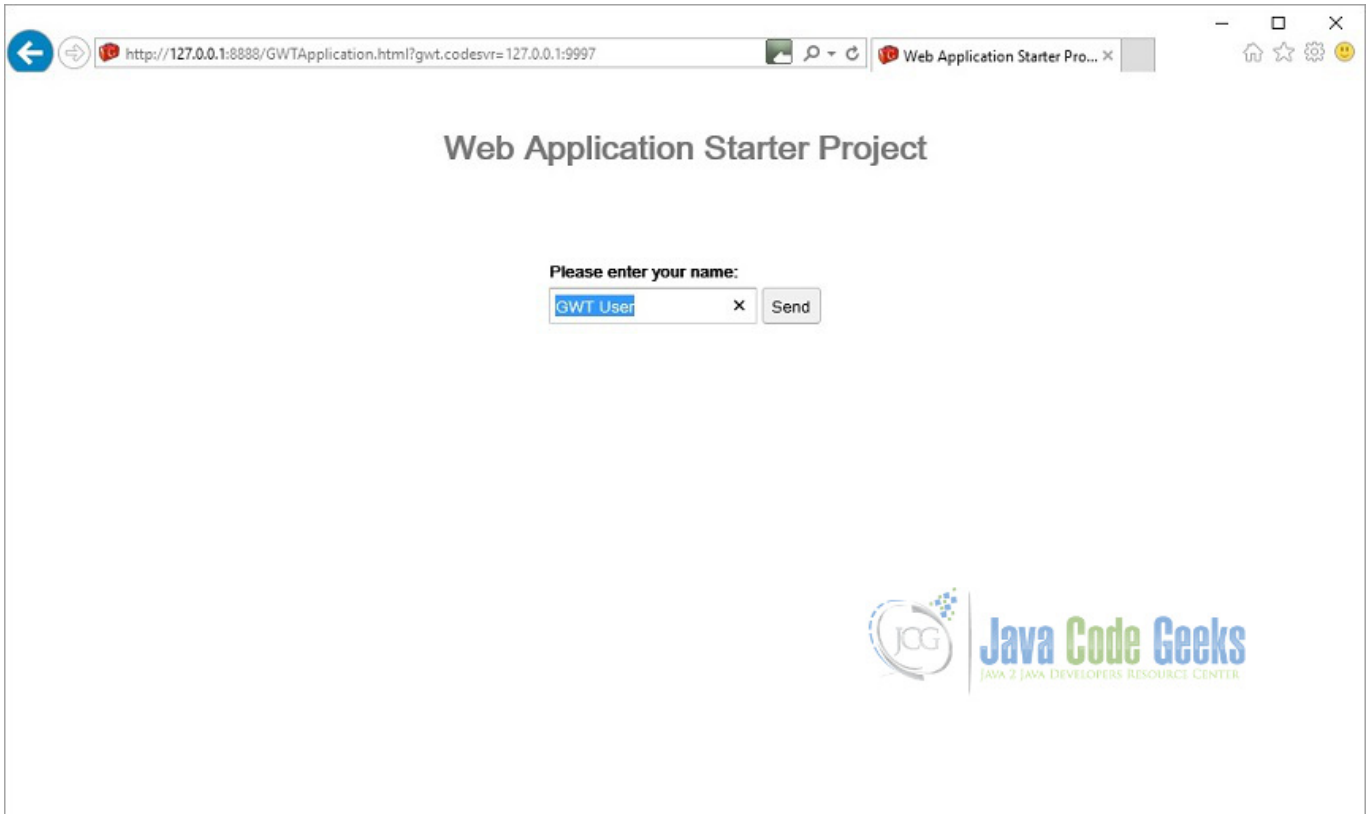

Figure 2.3: Output

Once you have started the development mode and entered the URL into the browser, the browser will attempt to connect. If this is your first time running a GWT application in development mode, you may be prompted to install the Google Web Toolkit Developer Plugin. Follow the instructions on the page to install the plugin, then restart the browser and return to the same URL.

2.7 Project components

Let's examine some of the generated files and see how they fit together to form a GWT project.

2.7.1 GWT Configuration file

The module file is located at src/com/javacodegeeks/GWTApplication.gwt.xml. It contains the definition of the GWT module, the collection of resources that comprise a GWT application or a shared package. By default, it inherits the core GWT functionality required for every project. Optionally, you can specify other GWT modules to inherit from.

```
<?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 2.6.0//EN"
"https://google-web-toolkit.googlecode.com/svn/tags/2.6.0/distro-source/core/src/gwt-module ↩
    .dtd">
<module rename-to='gwtapplication'>
  <!-- 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.client.GWTApplication'/>

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

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

In the module XML file, you specify your application's entry point class. In order to compile, a GWT module must specify an entry point. If a GWT module has no entry point, then it can only be inherited by other modules. It is possible to include other modules that have entry points specified in their module XML files. If so, then your module would have multiple entry points. Each entry point is executed in sequence.

By default, the application uses two style sheets: the default GWT style sheet, standard.css (which is referenced via the inherited theme), and the application style sheet, GSTApplication.css which was generated by Eclipse GWT plugin.

2.7.2 Landing page

The code for a web application executes within an HTML document. In GWT, we call this the host page. For example, the host page for the GWTApplication project is GWTApplication.html. The host page references the application style sheet, GWTApplication.css. The host page references the path of JavaScript source code (generated by GWT) responsible for the dynamic elements on the page. The contents of the entire body element can be generated dynamically, for example, as it is with starter application. However, when you implement the StockWatcher application, you will use a mix of static and dynamic elements. You'll create an HTML element to use as placeholder for the dynamically generated portions of the page.

To provide better cross-browser compatibility, GWT sets the doctype declaration to HTML 4.01 Transitional. This, in turn, sets the browser's rendering engine to "Quirks Mode". If you instead want to render the application in "Standards Mode", there are a number of other doctypes you can use to force the browser to this render mode. In general, GWT applications will work in "Standards Mode" just as well as "Quirks Mode", but in some cases using widgets like panels and such may not render correctly. This problem has been greatly improved since GWT 1.5, and more work is being done to solve this problem once and for all.

GWT provides a mechanism for helping your application meet users' expectations of a web page, specifically in their ability to use the browser's back button in such situations as a multi-page wizard or a shopping cart/checkout scenario. The host page contains the iframe tag necessary for incorporating history support in your GWT application.

2.7.3 Stylesheet

A style sheet is associated with each project. By default, the application style sheet, GWTApplication.css, contains style rules for the starter application. Just as for any web page, you can specify multiple style sheets. List multiple style sheets in their order of inheritance; that is, with the most specific style rules in the last style sheet listed.

2.7.4 Java code

Currently, GWTApplication.java contains the Java source for the starter application. The GWTApplication class implements the GWT interface `EntryPoint`. It contains the method `onModuleLoad()`. Because the GWTApplication class is specified as the entry point class in GWTApplication's module definition, when you launch GWTApplication the `onModuleLoad()` method is called.

The GWTApplication class inherits functionality via other GWT modules you included in GWTApplication's module definition (GWTApplication.gwt.xml). For example, when building the user interface, you'll be able to include types and resources from the package `com.google.gwt.user.client.ui` because it is part of the GWT core functionality included in the GWT module `com.google.gwt.user.User`.

2.8 Download the source file

This was an example of developing a GWT application.

Download

You can download the full source code of this example here: **GWT Application Example**. Please note that to save space the jar files from the lib folder have been removed.

Chapter 3

GWT Interview Questions and Answers

In this article we will learn about the most common interview questions asked in GWT domain. We will start with the basic ones then move on to more tricky ones.

3.1 What is GWT?

Google Web Toolkit (GWT) is a development toolkit for building ajax applications using Java. The programmer writes code in Java then GWT compiler converts this code to JavaScript. With GWT, we can develop and debug AJAX applications in the Java language using the Java development tools of our choice

GWT provides two modes:

- *Development Mode*: allows to debug the Java code of the application directly via the standard Java debugger.
- *Web mode*: the application is translated into HTML and Javascript code and can be deployed to a webserver.

3.2 What is a module descriptor in GWT application?

A module descriptor is a configuration file used to set-up a GWT application.

3.3 What is a GWT Module?

A GWT module is simply an encapsulation of functionality. It shares some similarities with a Java package but is not the same thing. A GWT module is named similarly to a Java package in that it follows the usual dotted-path naming convention. For example, most of the standard GWT modules are located underneath "com.google.gwt" However, the similarity between GWT modules and Java packages ends with this naming convention.

A module is defined by an XML descriptor file ending with the extension ".gwt.xml", and the name of that file determines the name of the module. For example, if we have a file named `src/com/mycompany/apps/MyApplication.gwt.xml`, then that will create a GWT module named `com.mycompany.apps.MyApplication`. The contents of the .gwt.xml file specify the precise list of Java classes and other resources that are included in the GWT module.

3.4 What is an entry point class?

A module entry-point is any class that is assignable to `EntryPoint` and that can be constructed without parameters. When a module is loaded, every entry point class is instantiated and its `EntryPoint.onModuleLoad()` method gets called.

3.5 Which method of the Entry point class is called when the GWT application is loaded? What happens if there are multiple Entry point classes?

`onModuleLoad()`. If there are more than one Entry point classes then each of then gets called in the sequence in which they are defined in the configuration file.

3.6 How do I enable assertions?

The GWT compiler recognizes the -ea flag to generate code for assertions in the compiled JavaScript. Only use assertions for debugging purposes, not production logic because assertions will only work under GWT's development mode. By default, they are compiled away by the GWT compiler so do not have any effect in production mode unless we explicitly enable them.

3.7 What is the default style name of any GWT widget?

By default, the class name for each component is gwt-<classname>. For example, the `Button` widget has a default style of `gwt-Button` and similar way `TextBox` widgest has a default style of `gwt-TextBox`.

3.8 What is internationalization?

Internationalization is changing the language of the text based on the locale. For example the browser should display the website content in Hindi for a user sitting in India and in French for the user accessing the website from France.

3.9 What is the difference between `TextResource` and `ExternalTextResource`

The related resource types `TextResource` and `ExternalTextResource` provide access to static text content. The main difference between these two types is that the former interns the text into the compiled JavaScript, while the latter bundles related text resources into a single file, which is accessed asynchronously.

3.10 How can you set Browser targeted Compilation in GWT?

To reduce the compilation time, choose favorite browser and add the `user.agent` property in the module configuration file.

3.11 Why doesn't GWT provide a synchronous server connection option?

GWT's network operations are all asynchronous, or non-blocking. That is, they return immediately as soon as called, and require the user to use a callback method to handle the results when they are eventually returned from the server. Though in some cases asynchronous operations are less convenient to use than synchronous operations, GWT does not provide synchronous operations.

The reason is that most browsers' JavaScript engines are single-threaded. As a result, blocking on a call to `XMLHTTPRequest` also blocks the UI thread, making the browser appear to freeze for the duration of the connection to the server. Some browsers provide a way around this, but there is no universal solution. GWT does not implement a synchronous network connection because to do so would be to introduce a feature that does not work on all browsers, violating GWT's commitment to no-compromise, cross-browser AJAX. It would also introduce complexity for developers, who would have to maintain two different versions of their communications code in order to handle all browsers.

3.12 What is GWT `ClientBundle`?

The resources in a deployed GWT application can be roughly categorized into resources to never cache (.nocache.js), to cache forever (.cache.html), and everything else (myapp.css). The `ClientBundle` interface moves entries from the everything-else category into the cache-forever category.

3.13 What is `DataResource` in GWT?

A `DataResource` is the simplest of the resource types, offering a URL by which the contents of a file can be retrieved at runtime. The main optimization offered is to automatically rename files based on their contents in order to make the resulting URL strongly-cacheable by the browser. Very small files may be converted into data: URLs on those browsers that support them.

3.14 How to create custom widgets in GWT?

There are three general strategies to follow:

Create a widget that is a composite of existing widgets. The most effective way to create new widgets is to extend the `Composite` class. A composite is a specialized widget that can contain another component (typically, a `Panel`) but behaves as if it were its contained widget. We can easily combine groups of existing widgets into a composite that is itself a reusable widget. Some of the UI components provided in GWT are composites: for example, the `TabPanel` (a composite of a `TabBar` and a `DeckPanel`) and the `SuggestBox`. Rather than create complex widgets by subclassing `Panel` or another Widget type, it's better to create a composite because a composite usually wants to control which methods are publicly accessible without exposing those methods that it would inherit from its `Panel` superclass.

Create an entirely new widget written in the Java language. It is also possible to create a widget from scratch, although it is trickier since we have to write code at a lower level. Many of the basic widgets are written this way, such as `Button` and `TextBox`.

Create a widget that wraps JavaScript using JSNI methods. When implementing a custom widget that derives directly from the `Widget` base class, we may also write some of the widget's methods using JavaScript. This should generally only be done as a last resort, as it becomes necessary to consider the cross-browser implications of the native methods that we write, and also becomes more difficult to debug.

3.15 What is a `UiBinder`?

`UiBinder` provides a declarative way of defining User Interface. It helps to separate the programming logic from the UI.

3.16 What is the Same Origin Policy, and how does it affect GWT?

Modern browsers implement a security model known as the Same Origin Policy (SOP). Conceptually, it is very simple, but the limitations it applies to JavaScript applications can be quite subtle. Simply stated, the SOP states that JavaScript code running on a web page may not interact with any resource not originating from the same web site. The reason this security policy exists is to prevent malicious web coders from creating pages that steal web users' information or compromise their privacy. While very necessary, this policy also has the side effect of making web developers' lives difficult.

It's important to note that the SOP issues are not specific to GWT; they are true of any AJAX application or framework.

3.17 Which class is the superclass of all UI widgets?

`com.google.gwt.user.client.ui.UIObject`

3.18 What is GWT RPC

The GWT RPC framework makes it easy for the client and server components of web application to exchange Java objects over HTTP. The server-side code that gets invoked from the client is often referred to as a service. The implementation of the GWT RPC service is based on a Servlet architecture. Within a client code, we will use a automatically generated proxy class to make calls to the service. GWT will handle serialization of the Java objects. GWT RPC service is different from SOAP and REST.

3.19 What are Layout Panels?

Layout Panels can contain other widgets. These panels controls the way widget is displayed on User Interface. Every Panel widget inherits properties from `Panel` class which in turn inherits properties from `Widget` class and which in turn inherits properties from `UIObject` class.

3.20 How is GWT different from other frameworks?

GWT provides a set of ready-to-use user interface widgets that we can immediately utilize to create new applications. It also provides a simple way to create innovative widgets by combining the existing ones. We can use IDE to create, debug, and unit-test our AJAX applications. We can build RPC services to provide certain functionalities that can be accessed asynchronously from the web applications easily using the GWT RPC framework.

GWT enables us to integrate easily with servers written in other languages, so we can quickly enhance our applications to provide a much better user experience by utilizing the AJAX framework. GWT has the Java-to-JavaScript compiler to distill our application into a set of JavaScript and HTML files that we can serve with any web server. This gives us a great feature browser compatibility.

3.21 What are the features of GWT

Google Web Toolkit (GWT) is a development toolkit to create RICH Internet Application. GWT provides developers option to write client side application in Java. Application written in GWT is cross-browser compliant. GWT automatically generates javascript code suitable for each browser

3.22 What can I do to make images and borders appear to load more quickly the first time they are used?

Use `Image.prefetch()`

3.23 What is Deferred Binding?

Deferred Binding is GWT's answer to Java reflection. Every web browser has its own idiosyncrasies, usually lots of them. The standard Java way of dealing with idiosyncrasies would be to encapsulate the custom code into subclasses, with one subclass for each supported browser. At runtime, the application would use reflection and dynamic classloading to select the appropriate subclass for the current environment, load the class, create an instance, and then use that instance as the service provider for the duration of the program.

This is indeed what GWT does. However, the JavaScript environment in which GWT applications ultimately run simply does not support dynamic classloading (also known as dynamic binding.) Because dynamic binding is unavailable as a technique to GWT, GWT instead uses deferred binding. One way to think of this is as "dynamic class-loading that occurs at compile time instead of execution time." When the GWT Compiler compiles the Java application, it determines all the different "idiosyncrasies" that it

must support, and generates a separate, tightly streamlined version of the application for that specific configuration. For example, it generates a different version of the application file for Firefox than it does for Opera.

The GWT Compiler uses Deferred Binding to generate a completely separate version of the application for each language.

3.24 How do I create an app that fills the page vertically when the browser window resizes?

As of GWT 2.0, creating an application that fills the browser is easy using Layout Panels. `LayoutPanels` such as `DockLayoutPanel` and `SplitLayoutPanel` automatically resize to the size of the window when the browser resizes.

3.25 How do you make a call to the server if you are not using GWT RPC?

To communicate with the server from the browser without using GWT RPC:

- Create a connection to the server, using the browser's `XMLHTTPRequest` feature.
- Construct a payload, convert it to a string, and send it to the server over the connection.
- Receive the server's response payload, and parse it according to the protocol.

3.26 How can I dynamically fetch JSON feeds from other web domains?

Like all AJAX tools, GWT's HTTP client and RPC libraries are restricted to only accessing data from the same site where the application was loaded, due to the browser Same Origin Policy. If the application is using JSON, there is a work around to this limitation using a <script> tag (aka JSON-P).

First, we need an external JSON service which can invoke user defined callback functions with the JSON data as argument. An example of such a service is GData's "alt=json-in-script& callback=myCallback" support. Then, we can use `JsonpRequestBuilder` to make our call, in a way similar to a `RequestBuilder` when we're not making a cross-site request.

3.27 Conclusion

In this article we saw some of the GWT related questions which are quite popular in Interviews. GWT is open source, completely free, and used by thousands of enthusiastic developers around the world. Its goal is to enable productive development of high-performance web applications without the developer having to be an expert in browser quirks, XMLHttpRequest, and JavaScript. The GWT SDK provides a set of core Java APIs and Widgets. These allow you to write AJAX applications in Java and then compile the source to highly optimized JavaScript that runs across all browsers, including mobile browsers for Android and the iPhone. Most of the time the interviewer is more interested to know whether the person understands the concept of GWT, very rarely they are interested to know about the APIs.

Chapter 4

GWT AsyncCallback Example

In previous GWT tutorials we have seen how to setup basic project, how to create GUI using GWT Widgets and few more GUI related chapters. As part of this tutorial we are going to look into how GWT Web Application interacts with a backend server.

GWT provides a couple of different ways to communicate with a server via HTTP. We can use the GWT Remote Procedure Call (RPC) framework to transparently make calls to Java servlets. *GWT AsynchCAllback* is the primary interface that a caller must implement to receive a response from a RPC.

Here we are using GWT 2.7 integrated with Eclipse Mars 4.5.

4.1 Introduction

Typically Client communicates with server uses RPC (Remote Procedure Call). RPC is essentially a way of invoking a method in a class, however the only difference is that the class is located on a server, not actually the part of client program you are running. There is a problem with RPC, as *Javascripts* runs in web browser and the RPC call from browser hangs browser until the response is received. To avoid the browser hanging, GWT RPC call is made "Asynchronous" and the browser does not hang while waiting for the response.

4.2 GWT RPC Mechanism

The implementation of *GWT RPC* is based on the *Java Servlet* technology. *GWT* allows *Java Objects* to communicate between the client and the server; which are automatically serialized by the framework. The server-side code that gets invoked from the client is usually referred to as a service and the remote procedure call is referred as invoking a service. Below diagram shows the RPC implementation in a GWT application.

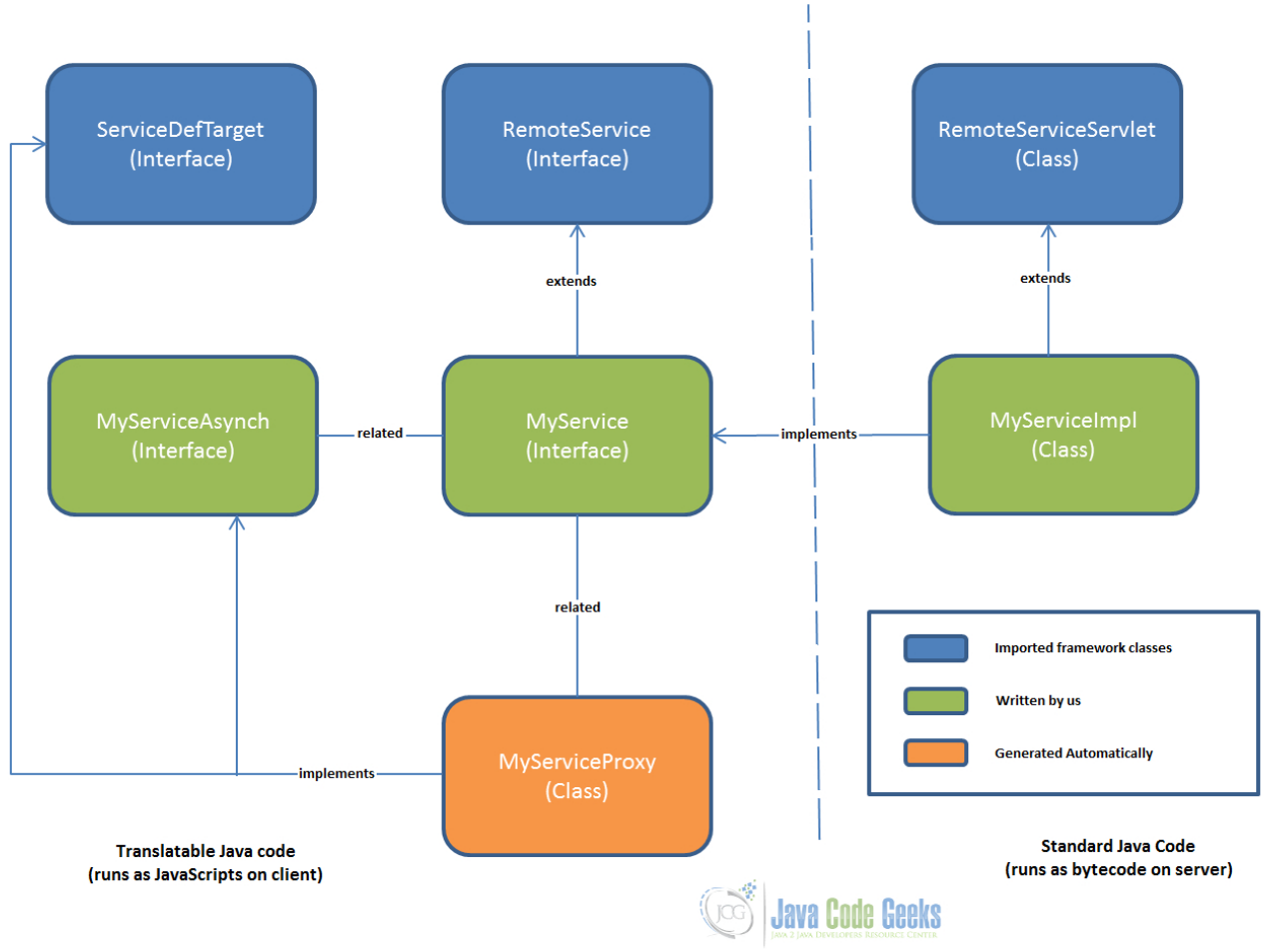

Figure 4.1: GWT RPC Mechanism

4.3 Creating Service

An interface at client-side that defines all service methods and the only way to communicate between client and server. The service is available at client-side, therefore it must be placed in the client package.

4.3.1 Define service Interface

To develop a new service interface, we will be creating a client-side Java interface that extends the `RemoteService` interface.

SampleService.java

```
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;

@RemoteServiceRelativePath("sampleservice")
public interface SampleService extends RemoteService{

        String sayHello(String name);
}
```

4.3.2 Define Async Service Interface

The service will be erroneous until we define the same service inside Async interface with return type `void` and the callback object for the Async service. The name of this interface must be the service interface name concatenated with "Async".

SampleServiceAsync.java

```
import com.google.gwt.user.client.rpc.AsyncCallback;

public interface SampleServiceAsync {

        void sayHello(String name, AsyncCallback callback);

}
```

4.3.3 Implementing AsynchCallback and handling its Failure

The interface AsyncCallback defines two methods *OnSuccess* and *OnFailure*. A class needs to be implemented to receive a callback from server and provide functionality on failure/success in the communication.

SampleCallback.java

```
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.rpc.AsyncCallback;

/**
 * Class which handles the asynchronous callback from the server
 *
 * Need to react on server communication failure and success
 */
public class SampleCallback implements AsyncCallback {

        @Override
        public void onFailure(Throwable caught) {
                // handle failure from server.
                Window.alert("Not able to process client reuest. Exception occured at  ↩
                    server: " + caught);
        }

        @Override
        public void onSuccess(String result) {
                // handle the successful scenario.
                Window.alert("Client request processed sucessfully. Result from server: " + ↩
                     result);
        }
}
```

4.4 Implementing Service

Services are responsible to perform some processing to respond to client requests. Such server-side processing occurs in the service implementation, which is based on the well-known servlet architecture.

4.4.1 Define Service Interface Implementation

GWT service implementation must extend `RemoteServiceServlet` and must implement the associated service interface. Every service implementation is ultimately a servlet. However it extends `RemoteServiceServlet` instead of `HttpServ`

let. RemoteServiceServlet automatically handles serialization of the data being passed between the client and the server and invoking the intended method in your service implementation.

SampleServiceImpl.java

```
import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import com.javacodegeeks.helloworld.client.service.SampleService;

public class SampleServiceImpl extends RemoteServiceServlet implements SampleService {

        @Override
        public String sayHello(String name) {
                return "Hello " + name;
        }

}
```

4.4.2 Update entry of Service inside web.xml

Define the servlet and map the URL to particular service by using the short-cute name of the service or fully qualified name of the service.

web.xml

```
<!-- Servlets -->
<servlet>
    <servlet-name>sampleServlet</servlet-name>
    <servlet-class<com.javacodegeeks.helloworld.server.SampleServiceImpl</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>sampleServlet>/servlet-name>
    <url-pattern>/samplewebapplication/sampleservice</url-pattern>
</servlet-mapping>
</pre>
```

4.5 Example

This shows that the user enters value inside and test box and clicks on button and a client request goes to server and response is getting handled on GUI. On successful completion, the label gets updated otherwise an alert windows apeears.

SampleWebApplication.java

```
/**
 * Entry point classes define `onModuleLoad()`.
 */
public class SampleWebApplication implements EntryPoint, ClickHandler{

        /**
         * Instantiates service.
         */
        SampleServiceAsync sampleServiceAsync = GWT.create(SampleService.class);
        /**
         * Label & Text Box.
         */
        Label lbl; TextBox textBox;
        /**
         * This is the entry point method.
         */
        public void onModuleLoad() {
```

```
                VerticalPanel verticalPanel = new VerticalPanel();
                verticalPanel.setSize("100%", "100%");
                verticalPanel.setVerticalAlignment(HasVerticalAlignment.ALIGN_MIDDLE);
                verticalPanel.setHorizontalAlignment(HasHorizontalAlignment.ALIGN_CENTER);
                textBox = new TextBox();
                Button btn = new Button("Get Update from Server"); btn.addClickHandler(this ↩
                    );
                lbl = new Label("The text will be updated here.");
                Image image = new Image();
                image.setUrl("https://www.javacodegeeks.com/wp-content/uploads/2012/12/ ↩
                    JavaCodeGeeks-logo.png");

                verticalPanel.add(textBox); verticalPanel.add(btn); verticalPanel.add(lbl);
                verticalPanel.add(image);

                RootLayoutPanel.get().add(verticalPanel);
        }

        @Override
        public void onClick(ClickEvent event) {
                sampleServiceAsync.sayHello(textBox.getText(), new AsyncCallback() {

                        @Override
                        public void onFailure(Throwable caught) {
                                // handle failure from server.
                                Window.alert("Exception Received from server.");
                        }

                        @Override
                        public void onSuccess(String result) {
                                lbl.setText(result);
                        }
                });
        }
}
```

Output:

Check Video Output

4.6 Project References

Using GWT RPC

GWT API Reference

4.7 Download Eclipse Project

Download

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

Chapter 5

GWT Panel Example

5.1 Overview

In this tutorial, we will get to know about Google Web Toolkit (GWT) Panels and we shall see how to use GWT Panels in a GWT Web Application to design user interface. In our previous tutorial GWT Tutorial for Beginners , we explained how to create a GWT Web Application project using eclipse and we have seen the basic steps to develop user interface using widgets. In this tutorial we shall see how to use different type of *Panels* to develop user interface.

5.2 Introduction

Panels in a *GWT Web Application* are used to set the layout of the Application. *GWT Panels* use *HTMP element* such as *DIV* and *TABLE* to layout their child *Widgets*. It is similar to use *LayoutManagers* in desktop *Java Application*. *Panels* contain *Widgets* and other *Panels*. They are used to define the layout of the user interface in the browser.

Here we are using GWT 2.7 integrated with Eclipse Mars 4.5.

5.3 Layout of a GWT Web Application UI

Layout design of user interface can be controlled via *HTML* and *Java*. A typical user interface comprises of Top-level panel and simple panels. Top-level panels are usually `DeckLayoutPanel`, `TabLaypitPanel`, `StacklayoutPanel` etc. Simple Panels or Basic Panels are `FlowPanel`, `HTMLPanel`, `FormPanel`, `ScrollPanel`, `Grid`, `FlexTable` etc. Each panel can contain other panels.

5.4 Basic Panels

5.4.1 RootPanel

RootPanel is the top most Panel to which all other *Widgets* are ultimately attached. *RootPanels* are never created directly. On the other hand, they are accessed via `RootPanel.get()`. It returns a *singleton panel* that wraps the GWT Web Application's welcome *HTML* page's *body element*. To achieve more control over user interface via *HTML*, we can use `RootPanel.get(String argument)` that returns a panel for any other *element* on the *HTML* page against provided argument.

Frequently used methods:

Method Name	Description
public static RootPanel.get()	Gets the default root panel. This panel wraps the body of the browser's document.
public static RootPanel.get(String id)	Gets the root panel associated with a given browser element against the provided element's id.

Refer RootPanel Javadoc for detailed api description.

SampleRootPanel.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {

        Label lbl = new Label("This Label is added to Root Panel.");
        /*
         * Adding label into HTML page.
         */
        RootPanel.get().add(lbl);
}
```

Output:

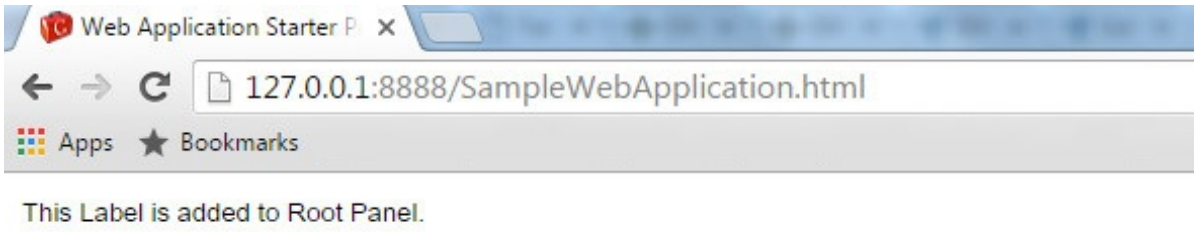

Figure 5.1: Example RootPanel.

5.4.2 FlowPanel

A FlowPanel is the simplest panel that formats its child widgets using the default *HTML* layout behavior. A `FlowPanel` is rendered as an *HTML div* and attaches children directly to it without modification. Use it in cases where you want the natural *HTML* flow to determine the layout of child widgets.

Frequently used methods:

Method Name	Description
public void add(Widget w)	Adds a new child widget to the panel.

Refer FlowPanel Javadoc for detailed api description.

SampleFlowPanel.java

```
 /**
  * This is the entry point method.
  */
public void onModuleLoad() {

        FlowPanel flowPanel = new FlowPanel();
        // Add buttons to flow Panel
        for(int i = 1;  i <= 8; i++){
                Button btn = new Button("Button " + i);
                flowPanel.add(btn);
        }

        // Add the Flow Panel to the root panel.
        RootPanel.get().add(flowPanel);
}
```

Output:

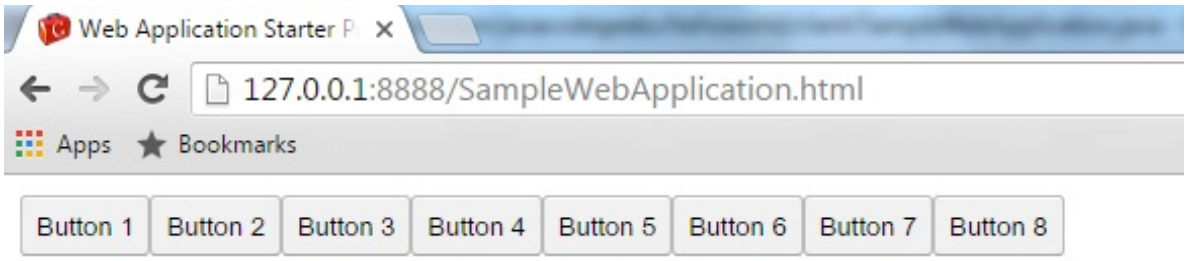

Figure 5.2: Example FlowPanel

5.4.3 HTMLPanel

An `HTMLPanel` rendered with the specified *HTML* contents. Child widgets can be added into identified elements within that *HTML* contents.

Frequently used methods:

Method Name	Description
public HTMLPanel(String html)	Creates an HTML panel with the specified HTML contents inside a DIV element.

Refer HTMLPanel Javadoc for detailed api description.

SampleHTMLPanel.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad(){

        // Add buttons to html Panel
        String htmlString = "This HTMLPanel contains"
                        +" html contents. This shows sample text inside HTMLPanel.";
        HTMLPanel htmlPanel = new HTMLPanel(htmlString);

        // Add the HTML Panel to the root panel.
        RootPanel.get().add(htmlPanel);
}
```

Output:

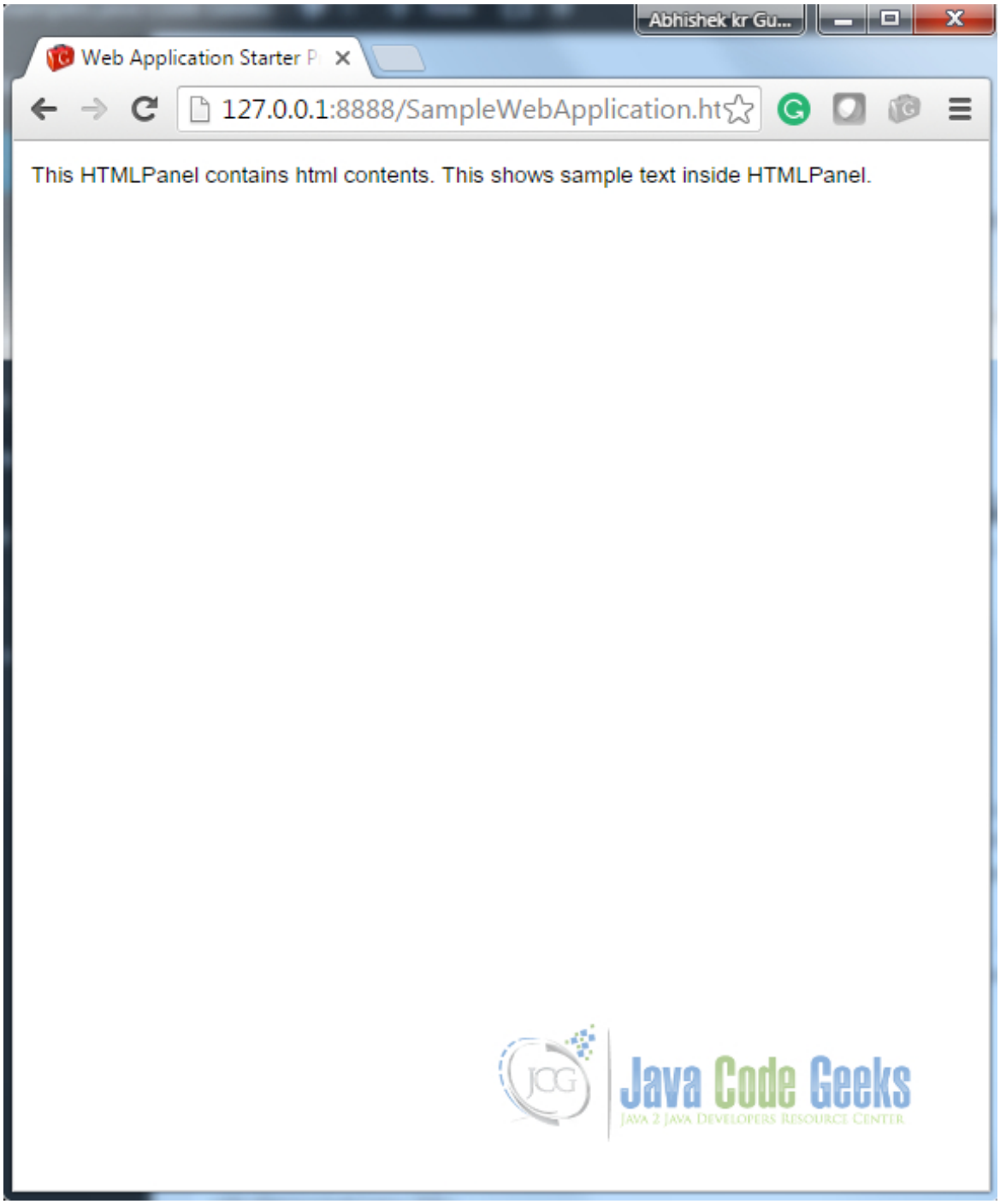

Figure 5.3: Example HTMLPanel

5.4.4 FormPanel

The panel provides the behavior of an *HTML FORM* element. Any widgets added to this panel will be wrapped inside *HTML form element*. The panel can be used to achieve interoperability with servers that accept traditional *HTML form* encoding.

Frequently used methods:

Method Name	Description

public HandlerRegistration addSubmitCompleteHandler(SubmitCompleteHandler handler)	Adds a SubmitCompleteEvent handler.
public HandlerRegistration addSubmitHandler(SubmitHandler handler)	Adds a SubmitEvent handler.

Refer FormPanel Javadoc for detailed api description.

SampleFormPanel.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
        // Create a FormPanel and point it at a service.
        final FormPanel form = new FormPanel();
        form.setAction("/myFormHandler");

        // Because we're going to add a FileUpload widget, we'll need to set the
        // form to use the POST method, and multipart MIME encoding.
        form.setEncoding(FormPanel.ENCODING_MULTIPART);
        form.setMethod(FormPanel.METHOD_POST);

        // Create a panel to contains all of the form widgets.
        VerticalPanel panel = new VerticalPanel();
        panel.setBorderWidth(1);
        panel.setSpacing(4);
        form.setWidget(panel);

        // Create a TextBox, giving it a name so that it will be submitted.
        final TextBox tb = new TextBox();
        tb.setName("textBoxForm");
        panel.add(tb);

        // Create a ListBox, giving it a name and some values to be associated with
        // its options.
        ListBox lb = new ListBox();
        lb.setName("listBoxForm");
        lb.addItem("list1", "List1 Value");
        lb.addItem("list2", "List2 Value");
        lb.addItem("list3", "List3 Value");
        panel.add(lb);

        // Create a FileUpload widget.
        FileUpload upload = new FileUpload();
        upload.setName("uploadForm");
        panel.add(upload);

        // Adding a submit button.
        panel.add(new Button("Submit", new ClickHandler() {
                @Override
                public void onClick(ClickEvent event) {
                        form.submit();
                }
        }));

        // Adding an event handler to the form.
        form.addSubmitHandler(new FormPanel.SubmitHandler() {
                public void onSubmit(SubmitEvent event) {
                        // This event is fired just before the form is submitted.
                        // this provides opportunity to perform validation.
                        if (tb.getText().length() == 0) {
```

```
                                Window.alert("Text box must not be empty");
                                event.cancel();
                        }
                }
        });
        form.addSubmitCompleteHandler(new FormPanel.SubmitCompleteHandler() {
                public void onSubmitComplete(SubmitCompleteEvent event) {
                        // This event fired after the form submission is successfully ↩
                            completed.
                        // Assuming the service returned a response of type text/html,
                        // we can get the result text here.
                        Window.alert(event.getResults());
                }
        });

        RootPanel.get().add(form);
}
```

Output:

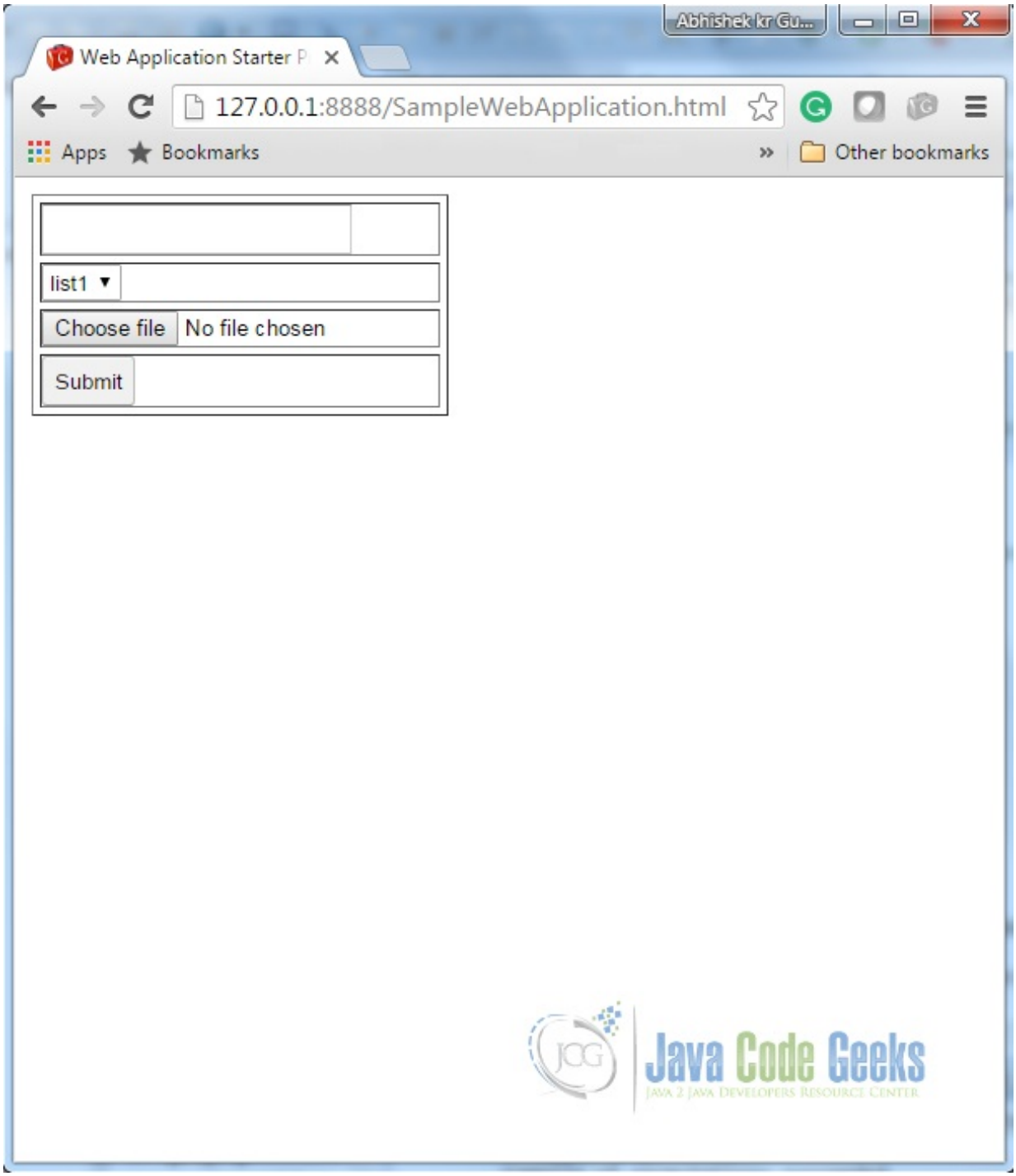

Figure 5.4: Example FormPanel

5.4.5 ScrollPanel

A Simple panel that wraps its contents into a scrollable area. Using constructor `scrollPanle()` and `scrolPanle(Widget w)` we can create empty scroll panel and scroll panel with given widget.

Frequently used methods:

Method Name	Description

public ScrollPanel(Widget child)	Creates a new scroll panel with the given child widget.
public void setSize(String width, String height)	Sets the object's size.

Refer ScrollPanel Javadoc for detailed api description.

SampleScrollPanel.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
        // scrollable text
        HTML htmlString = new HTML("This *HTMLPanel* contains"
                        +"This is sample text inside the scrollable panel. "
                        + "This content should be big enough to enable the scrolling. "
                        + "We added the same content here again and again to make the "
                        + "content large enough. This is text inside the scrollable panel."
                        + " This content should be big enough to enable the scrolling."
                        + " This is text inside the scrollable panel. This content should "
                        + "be big enough to enable the scrolling. This is text inside the "
                        + "scrollable panel. This content should be big enough to enable"
                        + " the scrolling. This is text inside the scrollable panel."
                        + " This content should be big enough to enable the scrolling."
                        + " This is text inside the scrollable panel. This content "
                        + "should be big enough to enable the scrolling.");

        // scrollpanel with text
        ScrollPanel scrollPanel = new ScrollPanel(htmlString);
        scrollPanel.setSize("400px", "150px");

        // Adding the scroll panel to the root panel.
        RootPanel.get().add(scrollPanel);
}
```

Output:

Figure 5.5: Example ScrollPanel

5.4.6 Grid

A Grid is used to create traditional *HTML table*. It extends `HTMLTable`. `Grid` which can contain text, *HTML*, or a child Widget within its cells. It can be configured as per required number of rows and columns.

Frequently used methods:

Method Name	Description
public Grid(int rows, int columns)	Constructs a grid with the requested size.

public void setWidget(int row, int column, Widget widget)	Sets the widget within the specified cell. It belongs to it's parent class HTMLTable.

Refer Grid Javadoc for detailed api description.

SampleGrid.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
        // Create a grid
        Grid grid = new Grid(2, 2);
        grid.setBorderWidth(1);

        // Add buttons, checkboxes to the grid
        int rows = grid.getRowCount();
        int columns = grid.getColumnCount();
        for (int row = 0; row < rows; row++) {
                for (int col = 0; col < columns; col++) {
                        if (row == 0) {
                                grid.setWidget(row, col, new Button("Button " + row + col)) ↩
                                    ;
                        } else {
                                grid.setWidget(row, col, new CheckBox("CheckBox " + row +  ↩
                                    col));
                        }
                }
        }

        // Adding grid to the root panel.
        RootPanel.get().add(grid);
}
```

Output:

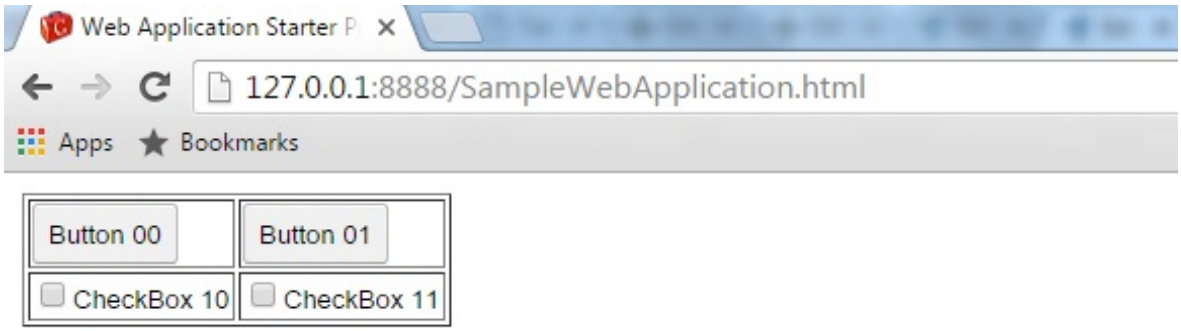

Figure 5.6: Example Grid

5.4.7 FlexTable

`FlexTable` also extends `HTMLTable` like `Grid`. This table creates cells on demand. Individual cells inside the table can be set to span multiple rows or columns based on indexes.

Frequently used methods:

Method Name	Description
public void setWidget(int row, int column, Widget widget)	Sets the widget within the specified cell. It belongs to it's parent class HTMLTable.

public void removeRow(int row)	Removes the specified row from the table.

Refer FlexTable Javadoc for detailed api description.

SampleFlexTable.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {

        // Create a Flex Table
        final FlexTable flexTable = new FlexTable();
        // Add button that will add more rows to the table
        Button addBtn = new Button(" Add Button ", new ClickHandler() {

                @Override
                public void onClick(ClickEvent event) {
                        addRow(flexTable);
                }
        });
        addBtn.setWidth("120px");
        // Remove button that will add more rows to the table
        Button removeBtn = new Button("Remove Button", new ClickHandler() {

                @Override
                public void onClick(ClickEvent event) {
                        removeRow(flexTable);
                }
        });
        removeBtn.setWidth("120px");
        VerticalPanel buttonPanel = new VerticalPanel();
        buttonPanel.add(addBtn);
        buttonPanel.add(removeBtn);
        flexTable.setWidget(0, 2, buttonPanel);

        // Add two rows to start
        addRow(flexTable);
        addRow(flexTable);
        RootPanel.get().add(flexTable);
}

/**
 * Add a row to the flex table.
 */
private void addRow(FlexTable flexTable) {
        int numRows = flexTable.getRowCount();
        flexTable.setWidget(numRows, 0, new Button("Button at column " + "0"));
        flexTable.setWidget(numRows, 1, new Button("Button at column " + "1"));
}

/**
 * Remove a row from the flex table.
 */
private void removeRow(FlexTable flexTable) {
        int numRows = flexTable.getRowCount();
        if (numRows > 1) {
                flexTable.removeRow(numRows - 1);
        }
}
```

Output:

Check Video Output

5.5 LayoutPanels

LayoutPanel is the most general approach to design layout. Mostly other layouts are built upon it. *LayoutPanel* always tries to fill all the available size in the window, so the content inside of the panel will perfectly fit the browser window size. *LayoutPanel* follows the size of the browser window, and try to resizes child elements accordingly. Whereas *Panel's* child widgets are not automatically resized when the browser window resizes.

It is notable that this panel works in standard mode, which requires that the HTML page in which it is run has an explicit *!DOCTYPE* declaration.

5.5.1 RootLayoutPanel

A singleton implementation of LayoutPanel always attaches itself to the element of *GWT Web Application Welcome HTML Page*. You can't choose which *HTML* element on the welcome page will become a *starting point* unlike `RootPanel`. This panel automatically calls `RequiresResize.onResize()` on itself when initially created, and whenever the window is resized.

Frequently used methods:

Method Name	Description
public static RootLayoutPanel get()	Gets the singleton instance of RootLayoutPanel. This instance will always be attached to the document body.

Refer RootLayoutPanel Javadoc for detailed api description.

SampleRootLayoutPanel.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
        // Attach two child widgets to a LayoutPanel, laying them out horizontally,
        // splitting at 50%.
        Widget childOne = new HTML("left");
        Widget childTwo = new HTML("right");
        LayoutPanel p = new LayoutPanel();
        p.add(childOne);
        p.add(childTwo);

        p.setWidgetLeftWidth(childOne, 0, Unit.PCT, 50, Unit.PCT);
        p.setWidgetRightWidth(childTwo, 0, Unit.PCT, 50, Unit.PCT);

        // Attach the LayoutPanel to the RootLayoutPanel.
        RootLayoutPanel.get().add(p);
}
```

Output:

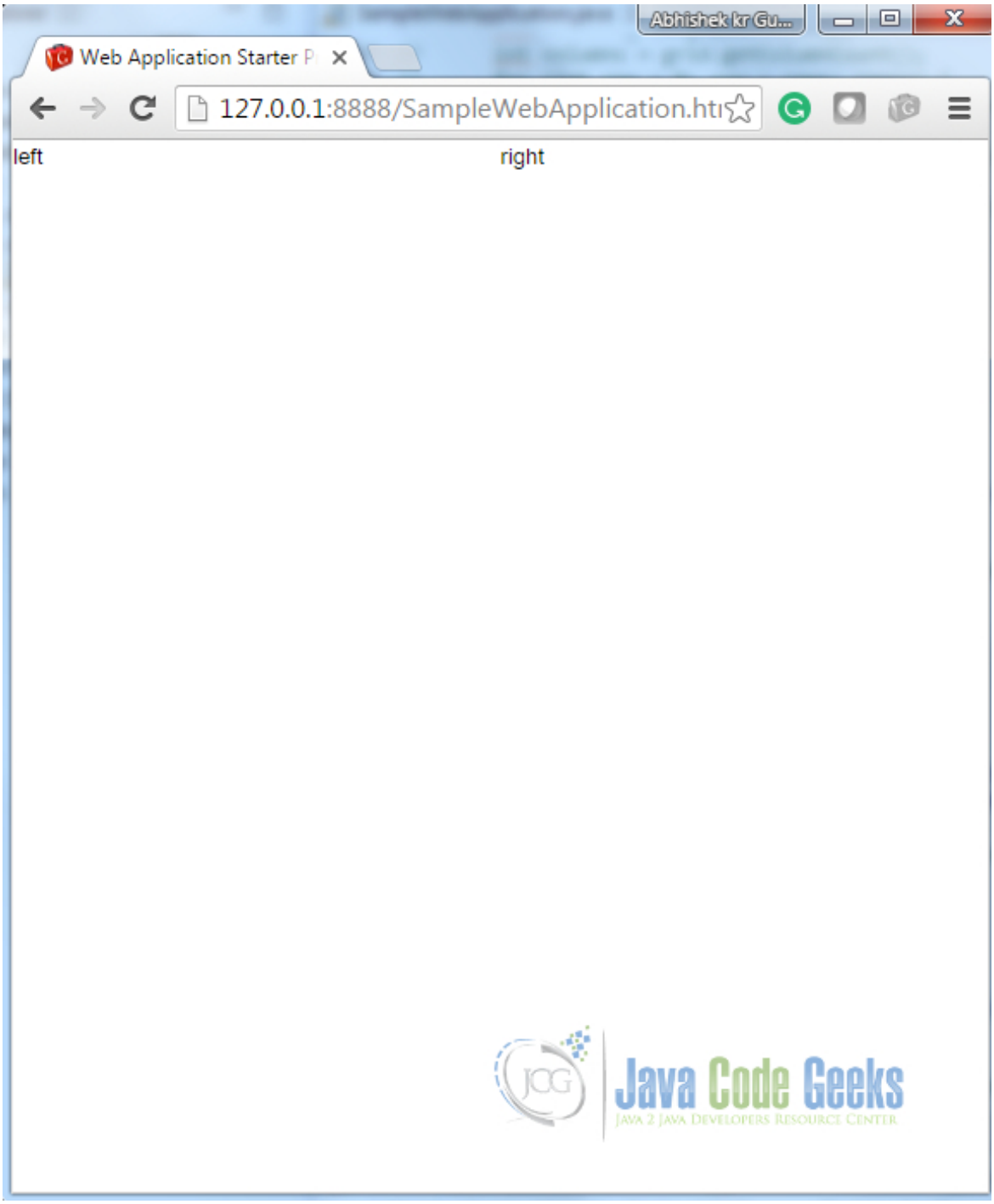

Figure 5.7: Example RootLayoutPanel

5.5.2 DockLayoutPanel

A panel that lays its child widgets at its outer edges, and allows its last widget to take up the remaining space in its center.This widget will also work only in standards mode, which requires that GWT Web Application welcome *HTML* page contains *!DOCTYPE* declaration.

Frequently used methods:

Method Name	Description
public DockLayoutPanel(Unit unit)	Creates an empty dock panel. Provide the unit to be used for layout.
public void add(Widget widget)	Adds a widget at the center of the dock.
public void addEast(Widget widget, double size)	Adds a widget to the east edge of the dock.
public void addNorth(Widget widget, double size)	Adds a widget to the north edge of the dock.
public void addSouth(Widget widget, double size)	Adds a widget to the south edge of the dock.
public void addWest(Widget widget, double size)	Adds a widget to the west edge of the dock.

Refer DockLayoutPanel Javadoc for detailed api description.

SampleDockLayoutPanel.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
        // Attach five widgets to a DockLayoutPanel, one in each direction. Lay
        // them out in 'em' units.
        DockLayoutPanel p = new DockLayoutPanel(Unit.EM);
        p.addNorth(new HTML("north"), 8);
        p.addSouth(new HTML("south"), 8);
        p.addEast(new HTML("east"),8);
        p.addWest(new HTML("west"), 8);
        p.add(new HTML("center"));

        // Attach the DockLayoutPanel to the RootLayoutPanel.
        RootLayoutPanel rp = RootLayoutPanel.get();
        rp.add(p);
}
```

Output:

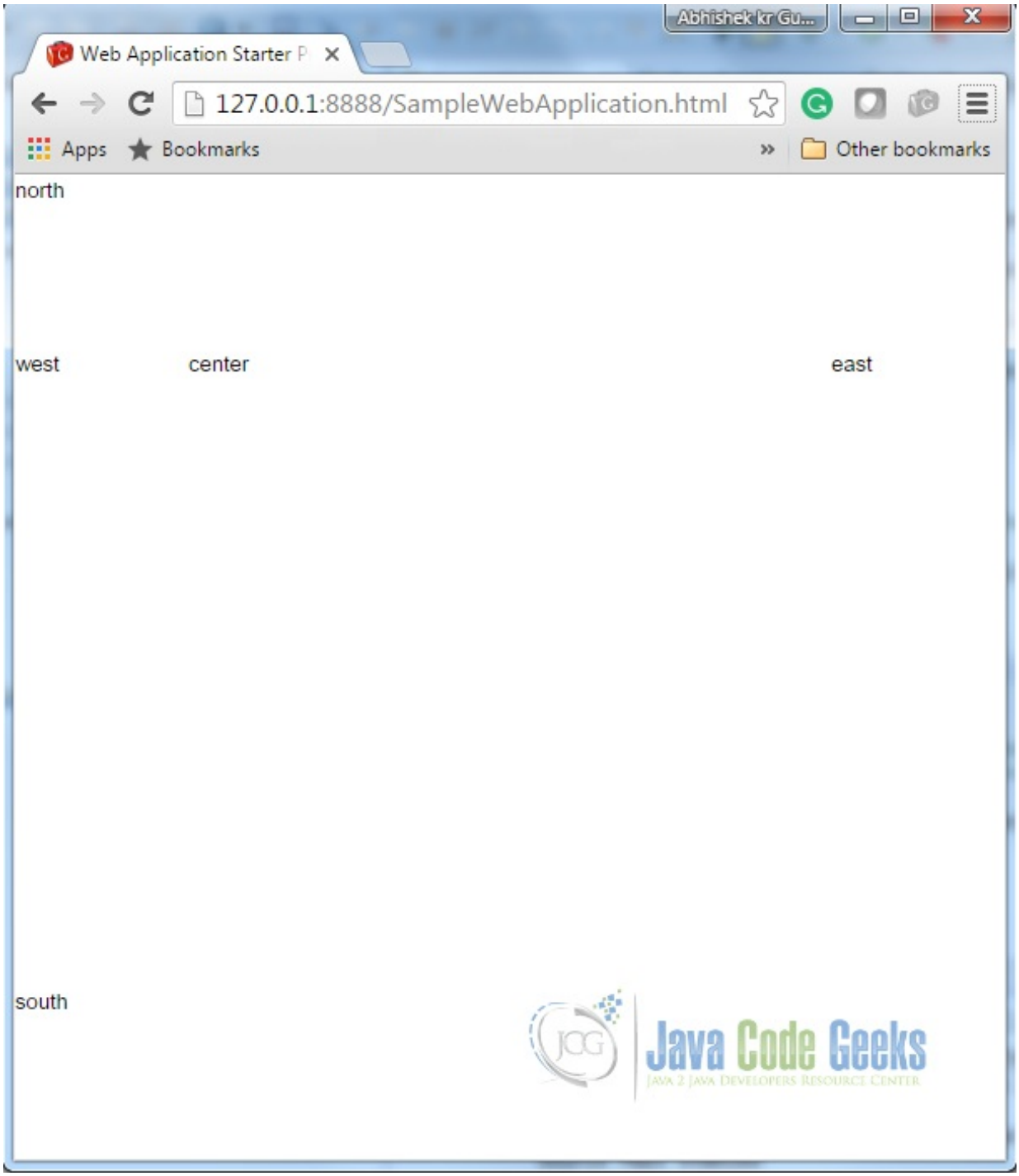

Figure 5.8: Example DockLayoutPanel

5.5.3 SplitLayoutPanel

This panel `extends DockLayoutPanel`. The panel is used in the same way as `DockLayoutPanel`, except that its children's sizes are always specified in absolute value, and each pair of child widget has a splitter between them, that the user can drag.

Frequently used methods:

Method Name	Description
public SplitLayoutPanel()	Construct a new SplitLayoutPanel with the default splitter size of 8px.

Refer SplitLayoutPanel Javadoc for detailed api description.

SampleSplitLayoutPanel.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
        // Create a three-pane layout with splitters.
        SplitLayoutPanel p = new SplitLayoutPanel();
        p.addWest(new HTML("Navigation Tree"), 128);
        p.addNorth(new HTML("Panel 1"), 384);
        p.add(new HTML("Panel 2"));

        // Attach the LayoutPanel to the RootLayoutPanel.
        RootLayoutPanel rp = RootLayoutPanel.get();
        rp.add(p);
}
```

Output:

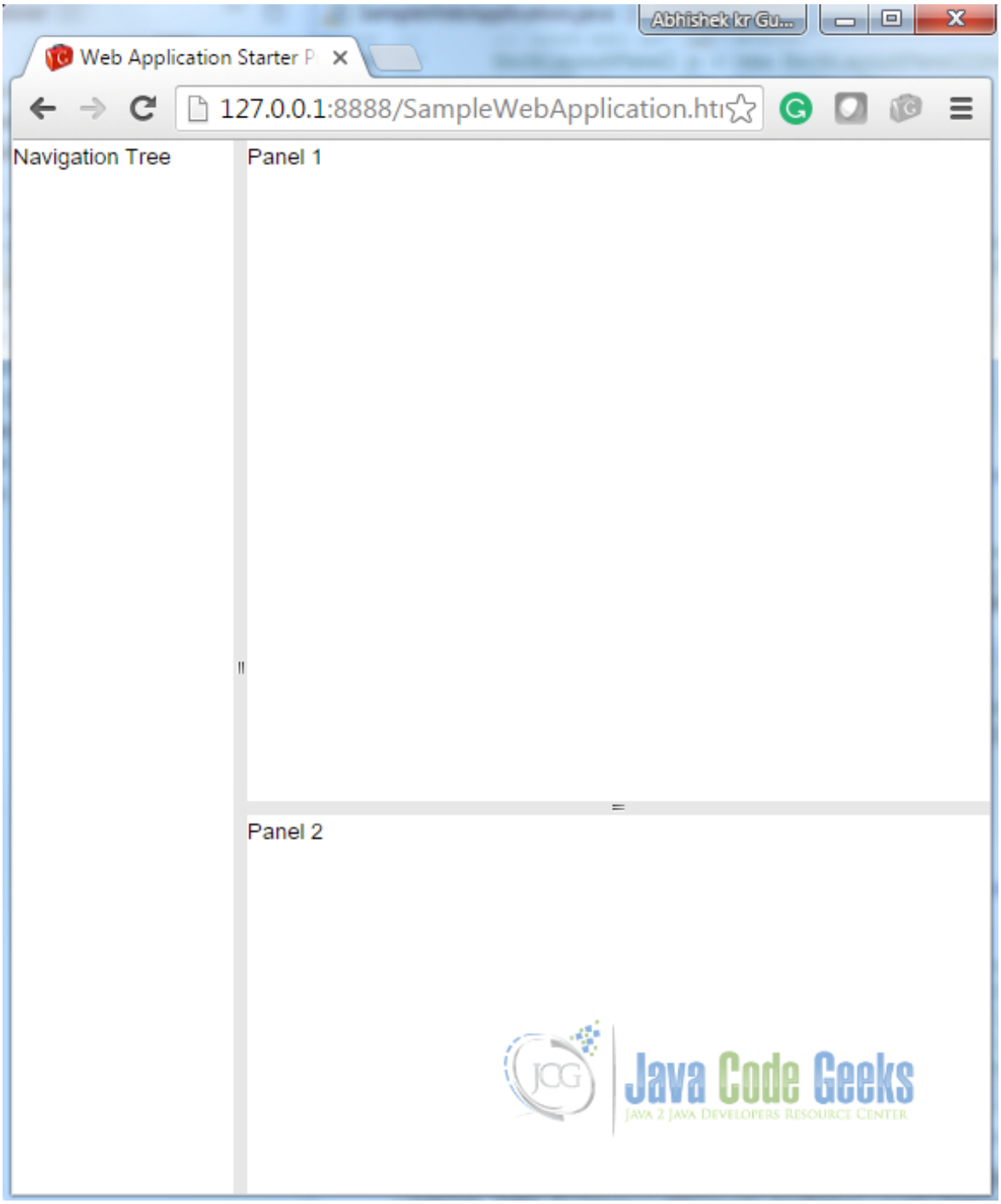

Figure 5.9: Example SplitLayoutPanel

5.5.4 StackLayoutPanel

The panel stacks its children vertically, displaying only one at a time, with a header for each child which the user can click to display.

This widget will only work in standards mode as well.

Frequently used methods:

Method Name	Description
public StackLayoutPanel(Unit unit)	Creates an empty stack panel. Provide the unit to be used for layout.
public void add(final Widget widget, SafeHtml header, double headerSize)	Adds a child widget to this stack, along with a widget representing the stack header.

Refer StackLayoutPanel Javadoc for detailed api description.

SampleStackLayoutPanel.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
        // Create a three-item stack, with headers sized in EMs.
        StackLayoutPanel p = new StackLayoutPanel(Unit.EM);
        p.add(new HTML("this"), new HTML("[this]"), 4);
        p.add(new HTML("that"), new HTML("[that]"), 4);
        p.add(new HTML("the other"), new HTML("[the other]"), 4);

        // Attach the StackLayoutPanelto the RootLayoutPanel.
        RootLayoutPanel rp = RootLayoutPanel.get();
        rp.add(p);
}
```

Output:

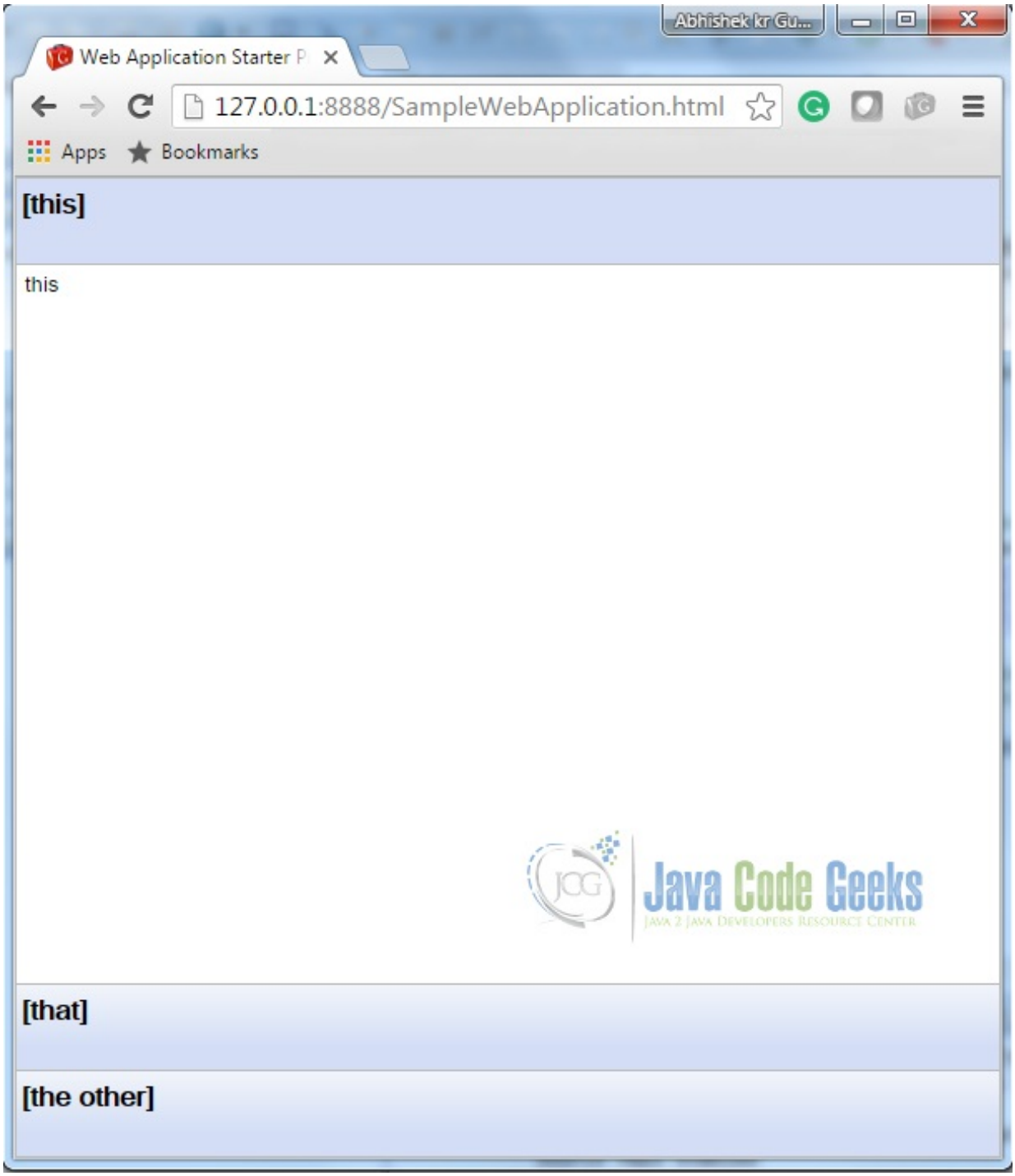

Figure 5.10: Example StackLayoutPanel

5.5.5 TabLayoutPanel

A panel represents a tabbed set of pages, each of which contains another widget. It's child widgets are shown as the user selects the various tabs associated with them. The tabs can contain arbitrary text, *HTML*, or widgets. This widget will only work in standards mode as well.

Frequently used methods:

Method Name	Description
public TabLayoutPanel(double barHeight, Unit barUnit)	Creates an empty tab panel.
public void add(Widget child, String text)	Adds a widget to the panel. If the Widget is already attached, it will be moved to the right-most index.

Refer TabLayoutPanel Javadoc for detailed api description.

SampleTabLayoutPanel.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
        // Create a three-item tab panel, with the tab area 1.5em tall.
        TabLayoutPanel p = new TabLayoutPanel(1.5, Unit.EM);
        p.add(new HTML("tab1 content"), "TAB1");
        p.add(new HTML("tab2 content"), "TAB2");
        p.add(new HTML("tab3 content"), "TAB3");

        // Attach the TabLayoutPanel to the RootLayoutPanel.
        RootLayoutPanel rp = RootLayoutPanel.get();
        rp.add(p);
}
```

Output:

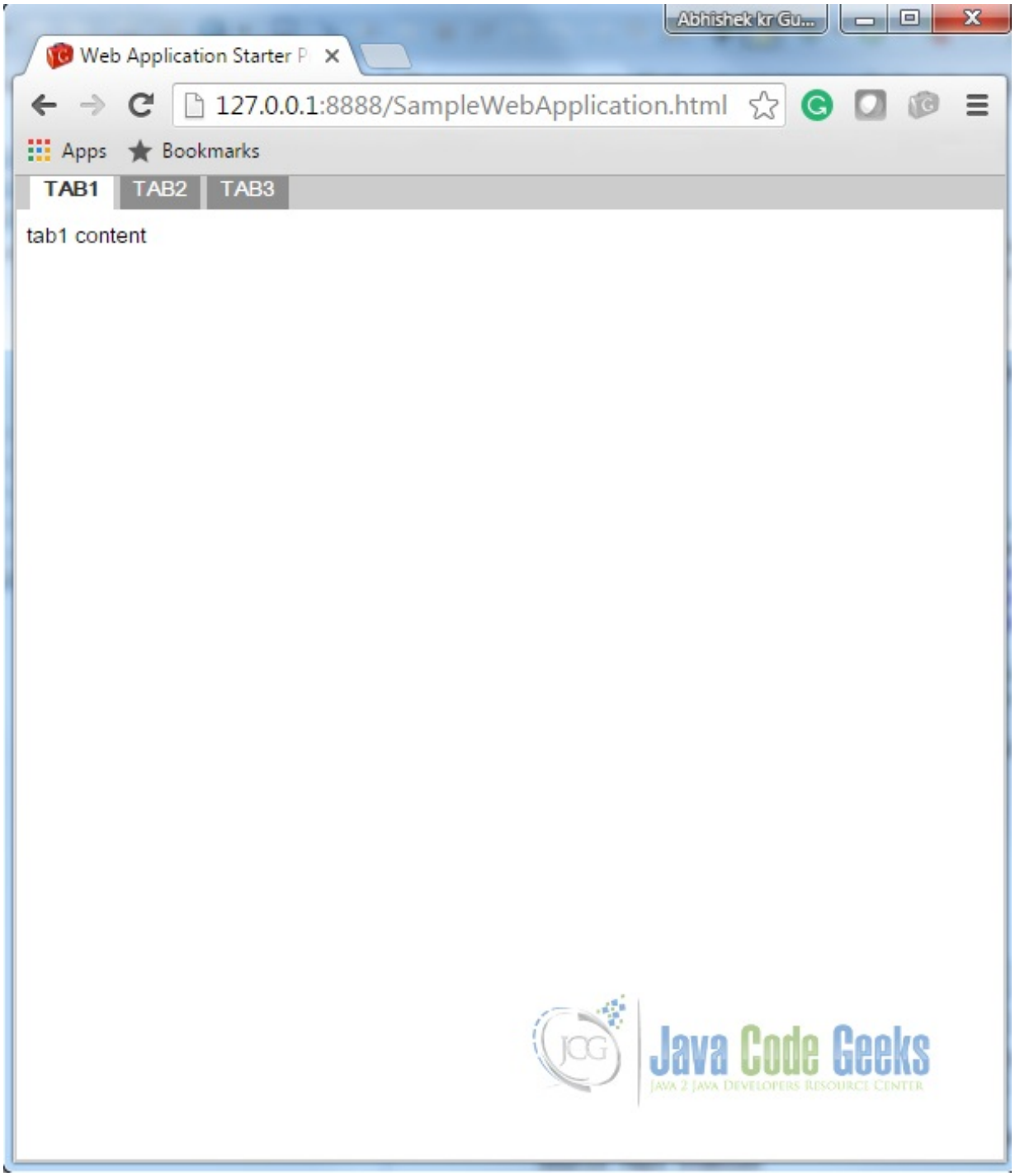

Figure 5.11: Example TabLayoutPanel

5.6 Project References

GWT UIPanels

GWT API Reference

GWT Showcase of Features

5.7 Download Eclipse Project

Download

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

Chapter 6

GWT HTMLPanel Example

In this tutorial, we will learn ins and out of the *Google Web Toolkit (GWT) HTML Panel*. In our previous tutorial GWT Tutorial for Beginners, we explained how to create a GWT Web Application project using eclipse and we have seen the basic steps to develop user interface using widgets. In this tutorial, we will focus on GWT HTML panel and it's usage to develop user interface.

Here we are using GWT 2.7 integrated with Eclipse Mars 4.5.

6.1 Introduction

Panels in a *GWT Web Application* are used to set the layout of the Application. *GWT Panels* use *HTML* element such as `DIV` and `TABLE` to layout their child *Widgets*. Panels may contain Widgets and other Panels. They are used to define the layout of the user interface in the browser. An `HTMLPanel` rendered with the specified *HTML* contents. Child widgets can be added into identified elements within that *HTML* contents.

6.2 Class Declaration

HTMLPanel.java

```
public class HTMLPanel extends ComplexPanel {
   ......
}
```

Here `ComplexPanel` is an *abstract base class* for `HTMLPanel` that can contain multiple child widgets. `ComplexPanel` extends `Panel` which is *abstract base class* for all panels.

6.3 Constructors

6.3.1 HTMLPanel(String html)

Creates an HTMLPanel with the specified *HTML* contents inside a `DIV` element.

SampleWebApplication.java

```
/**
 * This is the entry point method.
 *
 */
public void onModuleLoad(){
```

```
        // Creating HTML String.
        String htmlString ="Example shows HTML Panel constructed through HTML String<br><br ↩
            >"
                        + "|======================="
                        + ""
                        + "<th>FirstName</th><th>LastName</th><th>Age</th>"
                        + ""
                        + ""
                        + "|Bob|Sen|68"
                        + ""
                        + "|=======================";
        HTMLPanel htmlPanel = new HTMLPanel(htmlString);

        // Add the HTML Panel to the root panel.
        RootPanel.get().add(htmlPanel);
}
```

Output:

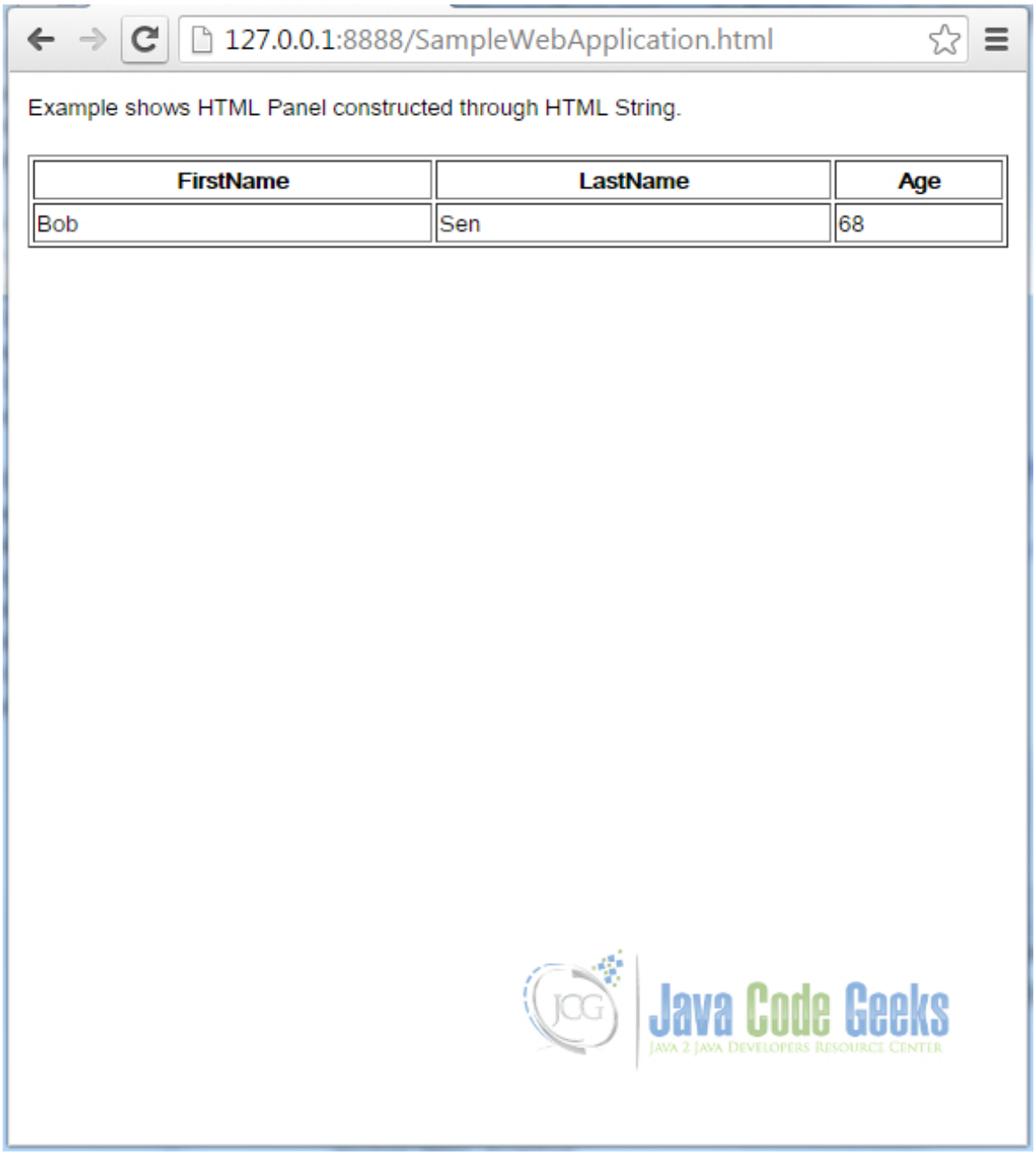

Figure 6.1: HTMP Panel: HTMLPanel(String html)

6.3.2 HTMLPanel(SafeHtml safeHtml)

Initializes the panel's HTML from a given SafeHtml object. Similar to HTMLPanel(String).

SampleWebApplication.java

```
/**
 * This is the entry point method.
 *
 */
public void onModuleLoad(){

        // Creating HTML String.
```

```
        String safeHtml= SafeHtmlUtils.fromSafeConstant(
                                        "Example shows HTML Panel constructed through ↩
                                            Safe HTML.<br><br>"
                + "|========================"
                + ""
                + "<th>FirstName</th><th>LastName</th><th>Age</th>"
                + ""
                + ""
                + "|Bob|Sen|68"
                + ""
                + "|========================");
        HTMLPanel htmlPanel = new HTMLPanel(safeHtml);

        // Add the HTML Panel to the root panel.
        RootPanel.get().add(htmlPanel);
}
```

Output:

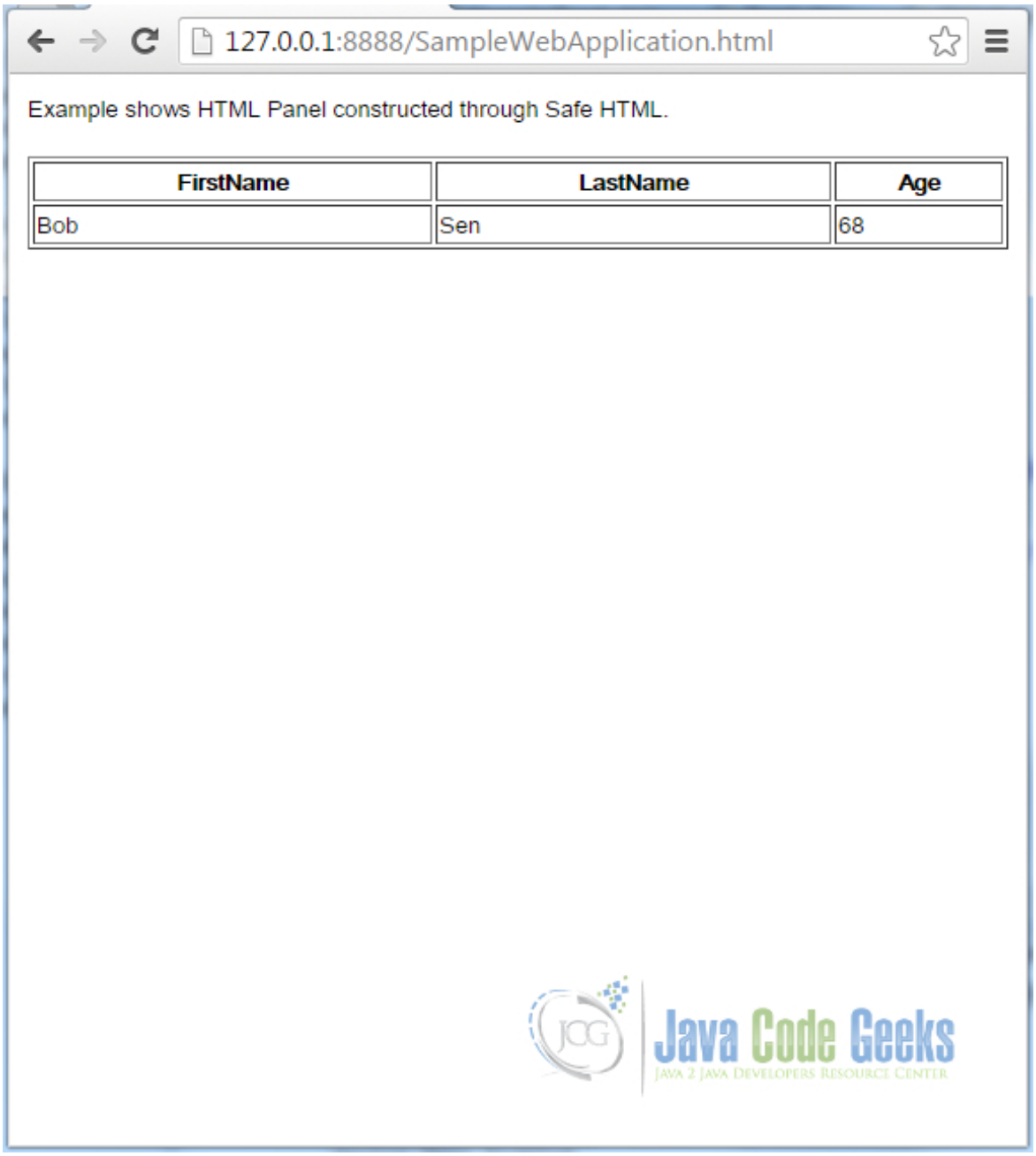

Figure 6.2: HTMP Panel: HTMLPanel(SafeHtml safeHtml)

6.3.3 HTMLPanel(String tag, String html)

Creates an `HTMLPanel` whose root element has the given tag, and with the specified *HTML* contents. The arguments passed inside the constructor are Tag of the root element and the panel's HTML content.

SampleWebApplication.java

```
/**
 * This is the entry point method.
 *
 */
public void onModuleLoad(){
```

```
        // Create HTML Panel with given tag and its HTML value.
        HTMLPanel htmlPanelH1 = new HTMLPanel("h1", "Heading using HTML tag: h1");
        HTMLPanel htmlPanelH2 = new HTMLPanel("h2", "Heading using HTML tag: h2");
        HTMLPanel htmlPanelH3 = new HTMLPanel("h3", "Heading using HTML tag: h3");

        VerticalPanel vp = new VerticalPanel();
        vp.setSize("100%", "100%");vp.setHorizontalAlignment(HasHorizontalAlignment. ↩
            ALIGN_CENTER);
        vp.add(htmlPanelH1);
        vp.add(htmlPanelH2);
        vp.add(htmlPanelH3);
        // Add the HTML Panel to the root       panel.
        RootPanel.get().add(vp);
}
```

Output:

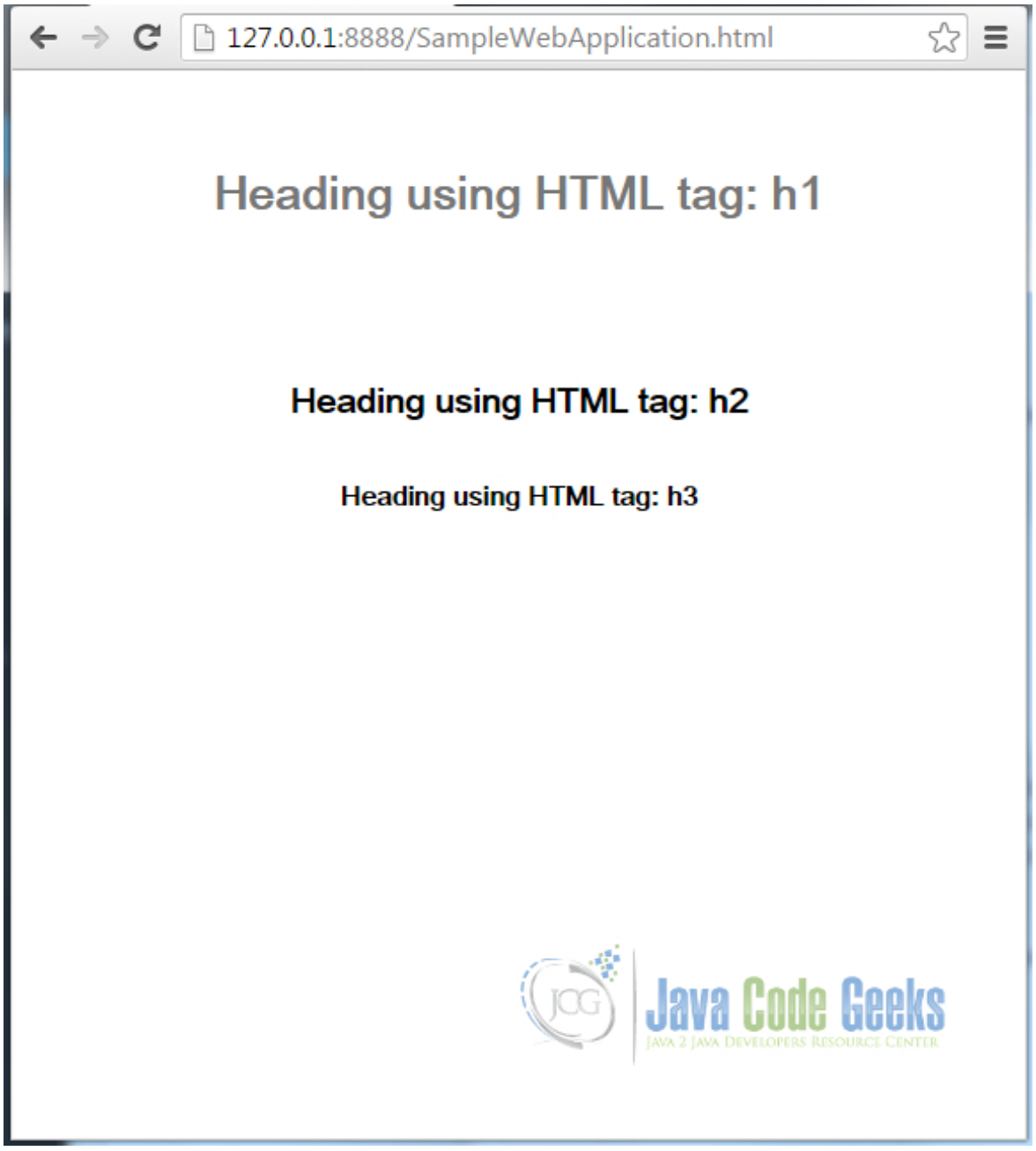

Figure 6.3: HTMLPanel: HTMLPanel(String tag, String html)

6.4 Method Summary

Method Signature	Description
public void add(Widget widget)	Adds a child widget to the panel.
public void add(Widget widget, String id)	Adds a child widget to the panel, contained within the HTML element specified by a given id.
public void add(Widget widget, Element elem)	Adds a child widget to the panel, contained within an HTML element.
public final void addAndReplaceElement(Widget widget, Element toReplace)	Adds a child widget to the panel, replacing the HTML element.

public void addAndReplaceElement(Widget widget, String id)	Adds a child widget to the panel, replacing the HTML element specified by a given id.
public Element getElementById(String id)	Finds an Element within this panel by its id.

6.5 Examples

6.5.1 Login Page using HTMLPanel

Here we designed the login page using `HTMLPanel`. User enters Username/Password and validations can be performed on click of submit button.

SampleWebApplication.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {

        String html =
                        "" +
                          "" +
                                "<label>UserName<br/>" +
                          "" +
                          "" +
                                "<label>Password<br/>" +
                          "" +
                          "" +
                          "" +
                        "";

        HTMLPanel htmlPanel = new HTMLPanel(html);

        // The username field
        TextBox user = new TextBox();
        user.getElement().setId("user_name");
        htmlPanel.add(user, "uname");

        // The password field
        TextBox password = new PasswordTextBox();
        password.getElement().setId("user_password");
        htmlPanel.add(password, "password");

        // The log in button
        Button submit = new Button("Submit");
        submit.getElement().setId("submit");
        htmlPanel.add(submit, "submit");

        submit.addClickHandler(new ClickHandler() {

                          @Override
           public void onClick(ClickEvent event) {
                        // Perform Validations
                                error("|======================<th>"
                                 + "ErrorType</th><th>Error "
                  + "Description</th>|Fatal|"
                                 + "Incorrect Password|======================");
                }
        });
        /*
         * Add panel to the page
```

```
         */
        RootPanel.get().add(htmlPanel);
}
```

Output:

Figure 6.4: Example1 HTMLPanel

6.5.2 Error Dialog Page using HTMLPanel

Error Dialog Page is using `HTMLPanel` and capable to display error message. The error message can be customized using *HTML* tags. This example is an extension of Login page example where Error Dialog Page pops up on click of submit button.

SampleWebApplication.java

```
/**
 * Custom Error Dialog Page.
 * @param err error message text
 */
public void error(String err) {
        final DialogBox dialog = new DialogBox();dialog.center();
        dialog.setSize("80%", "80%");dialog.setText("Error");

        VerticalPanel panel = new VerticalPanel();panel.setSize("100%", "100%");
        HTMLPanel html = new HTMLPanel(err);html.setSize("100%", "100%");
        panel.add(html);

        Button ok = new Button("OK");
        VerticalPanel buttonPanel = new VerticalPanel(); buttonPanel.setSpacing(3);
        buttonPanel.add(ok);
        panel.add(buttonPanel);

        dialog.setWidget(panel);

        ok.addClickHandler(new ClickHandler() {

                public void onClick(ClickEvent arg0) {
                        dialog.hide();
                }

        });
        dialog.show();
}
```

Output

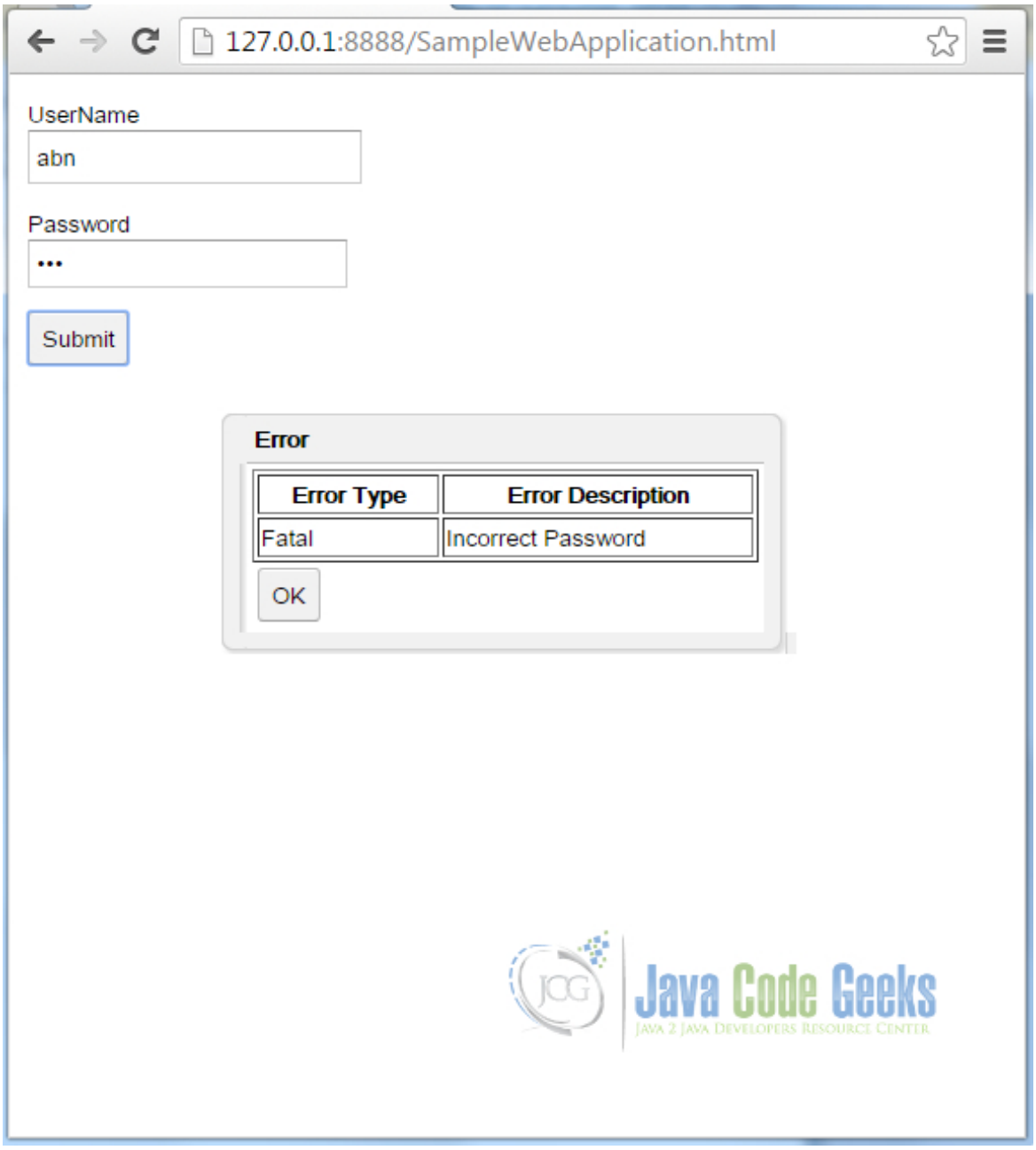

Figure 6.5: Example2 HTMLPanel

6.6 Project References

GWT UIPanels

GWT API Reference

6.7 Download Eclipse Project

Download

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

Chapter 7

GWT Scroll Panel Example

In this example we will learn how to use Scroll Panel widget in GWT. Google Web Toolkit is a development framework for creating Ajax-enabled web applications in Java. Tools and technologies used in this example are Java 1.8, Eclipse Luna 4.4.2, Eclipse GWT Plugin 2.6

7.1 Introduction

When you wish to create a scrollable area within another panel, you should use a `ScrollPanel`. This panel works well in layout panels, which provide it with the explicit size it needs to scroll properly. `ScrollPanel` class extends the `com.google.gwt.user.client.ui.SimplePanel` class and implements four interfaces. It is a simple panel that wraps its contents in a scrollable area.

```
public class ScrollPanel extends SimplePanel implements SourcesScrollEvents, RequiresResize ↩
    , ProvidesResize, HasScrolling
```

7.1.1 Constructors

Below we see the constructors for this class.

`public ScrollPanel()`

Creates an empty scroll panel.

`public ScrollPanel(Widget child)`

Creates a new scroll panel with the given child widget.

Parameters: *child* - the widget to be wrapped by the scroll panel

`protected ScrollPanel(Element root, Element scrollable, Element container)`

Creates an empty scroll panel using the specified root, scrollable, and container elements.

Parameters:

- *root* - the root element of the Widget
- *scrollable* - the scrollable element, which can be the same as the root element
- *container* - the container element that holds the child

7.2 Creating GWT project

To create a new GWT project go to File→New→Other, then type ‘Web App’. Choose ‘Web Application Project’ under ‘Google’.

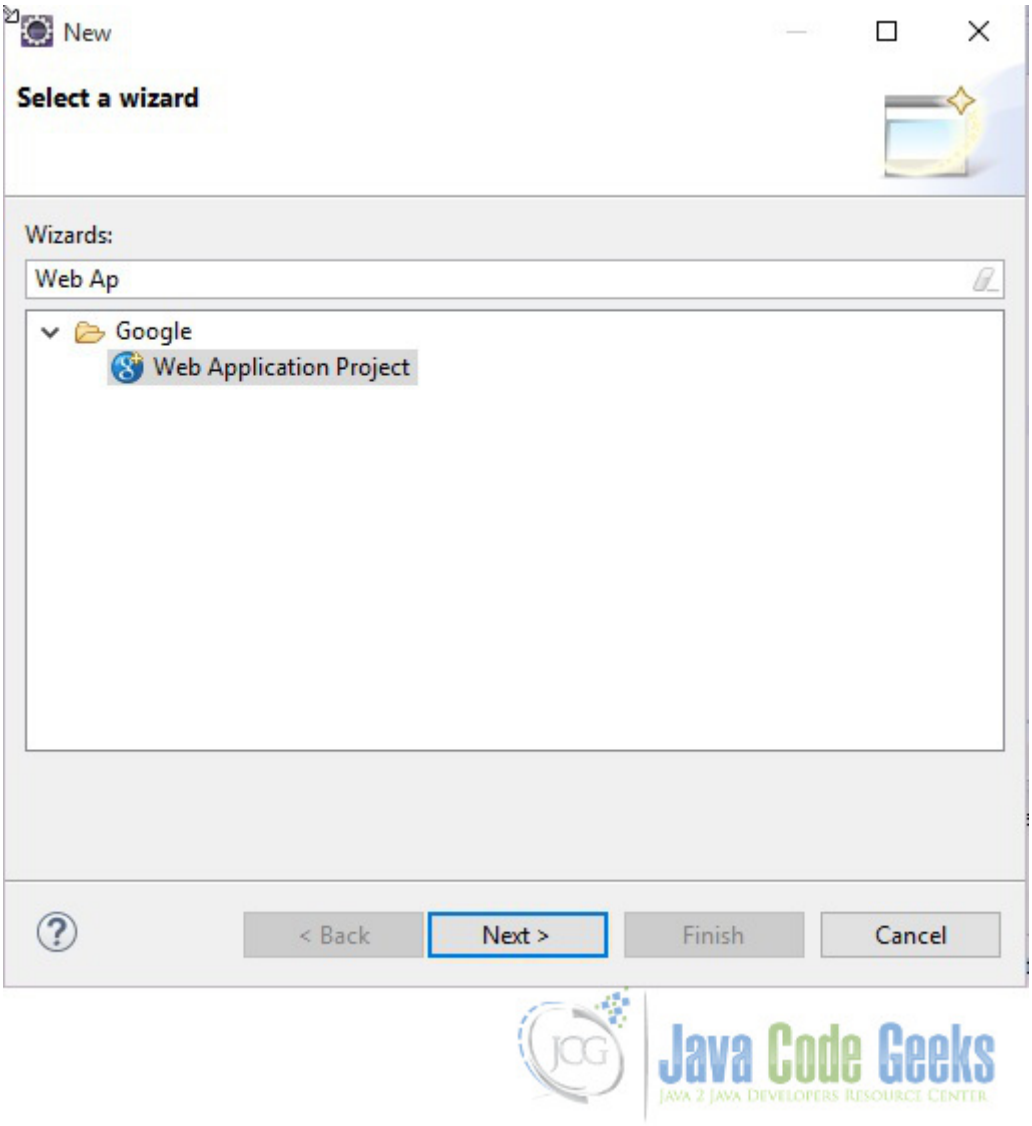

Figure 7.1: Create New Web Application

On the next window enter the Project name (‘GWTScrollPanel’) and the Package (com.javacodegeeks). Leave the other details as it is and click on ‘Finish’. Eclipse will generate some files automatically for you.

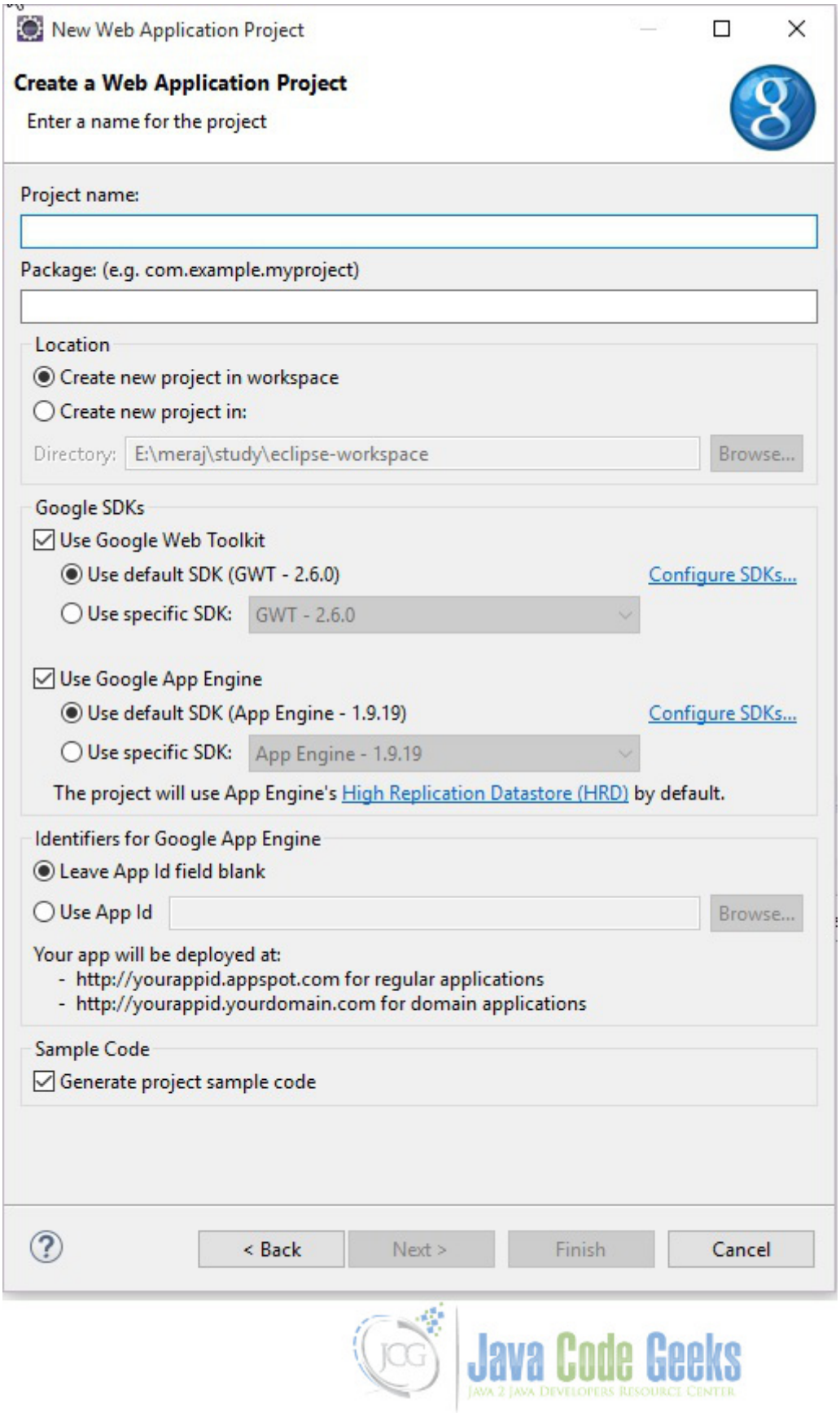

Figure 7.2: Create Project

7.3 Entry point class

GWTScrollPanel class is our entry point class.

GWTScrollPanel.java

```
package com.javacodegeeks.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.DecoratorPanel;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.ScrollPanel;

/**
* Entry point classes define <code>onModuleLoad()</code>.
*/
public class GWTScrollPanel implements EntryPoint {

  /**
  * This is the entry point method.
  */
  public void onModuleLoad() {
    HTML contents = new HTML("This is an example of GWT scroll panel. ScrollPanel class ↩
        extends the "
      + "com.google.gwt.user.client.ui.SimplePanel class and implements four interfaces. It ↩
          is a simple panel"
      + " that wraps its contents in a scrollable area. GWT also allows users to create ↩
        their own custom"
      + " scrollable panels using the CustomScrollPanel class. This class extends the ↩
        ScrollPanel class."
      + " This class allows users to provide their own scrollbars. The position of the ↩
        scrollbar in the"
      + " CustomScrollPanel differs from that of the native scrollable element. In the ↩
        native element,"
      + " scrollbars appear adjacent to the content,shrinking the content client height and ↩
         width when they"
      + " appear. CustomeScrollPanel overlays scrollbars on top of the content, so the ↩
        content does not change"
      + " the size when the scrollbars appear. If the scrollbars obscure the content, you ↩
        can set the padding-top"
      + " and padding-bottom attribute.");

    ScrollPanel scrollPanel = new ScrollPanel(contents);
    scrollPanel.setSize("450px", "200px");

    DecoratorPanel decoratorPanel = new DecoratorPanel();
    decoratorPanel.add(scrollPanel);

    RootPanel.get("container").add(decoratorPanel);
  }
}
```

7.4 Compile

To compile the application right click on the project and select 'Google' ⟹ 'GWT Compile'. You will get a pop-up showing the project name. Click on the 'Compile' button. GWT will start compiling the project.

7.5 Running the application

To run the application right click on the project and select *Run As ⇒ Web Application (GWT Classic Dev Mode)*. Eclipse will use the inbuild Jetty server to load the application. Once the application is up and running the focus will transfer to the *Development Mode* tab where a URL will be displayed - https://127.0.0.1:8888/GWTScrollPanel.html?gwt.codesvr=127.0.0.1:9997. Copy the URL and paste in your favorite browser. Remove the part after .html and press enter. You will see a screen like below.

Figure 7.3: Run

7.6 Custom Scroll Panel

GWT also allows users to create their own custom scrollable panels using the `CustomScrollPanel` class. This class extends the `ScrollPanel` class. This class allows users to provide their own scrollbars. The position of the scrollbar in the `CustomScrollPanel` differs from that of the native scrollable element. In the native element, scrollbars appear adjacent to the content,shrinking the content client height and width when they appear. `CustomeScrollPanel` overlays scrollbars on top of the content, so the content does not change the size when the scrollbars appear. If the scrollbars obscure the content, you can set the `padding-top` and `padding-bottom` attribute.

Unlike `ScrollPanel`, which implements `RequiresResize` but doesn't really require it, `CustomScrollPanel` actually does require resize and should only be added to a panel that implements `ProvidesResize`, such as most layout panels and `ResizeLayoutPanel`.

7.7 Download the source file

This was an example of `GWT Scroll Panel`.

Download

You can download the full source code of this example here : **GWT Scroll Panel Example**

Chapter 8

GWT Calendar Example

In this example we will learn how to use Calendar in GWT. The Google Web Toolkit is a development framework for creating Ajax-enabled web applications in Java. Tools and technologies used in this example are Java 1.8, Eclipse Luna 4.4.2, Eclipse GWT Plugin 2.6

8.1 Creating GWT project

To create a new GWT project go to File→New→Other, then type 'Web App'. Choose 'Web Application Project' under 'Google'.

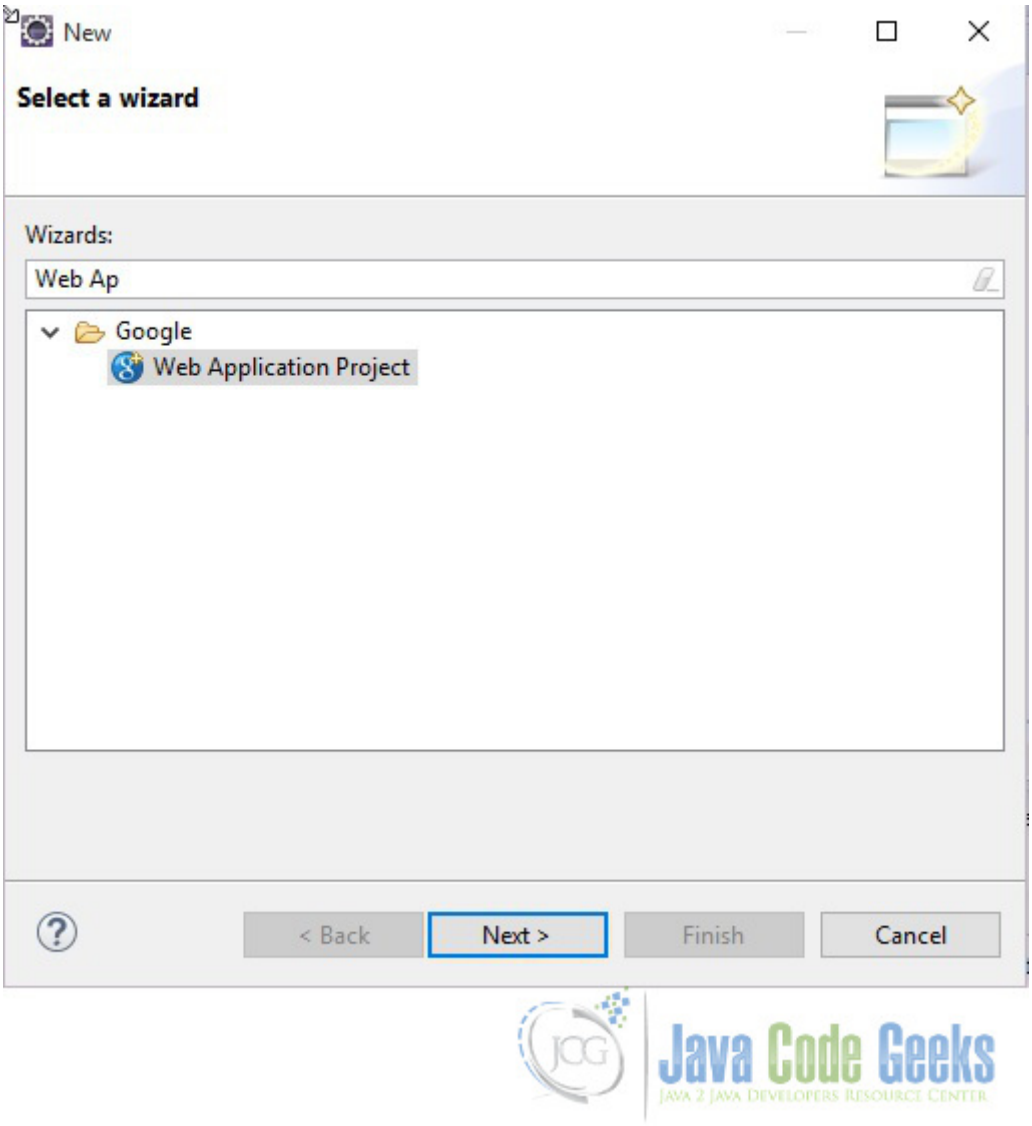

Figure 8.1: Create New Web Application

On the next window enter the Project name (‘GWTCalendar’) and the Package (com.javacodegeeks). Leave the other details as it is and click on ‘Finish’. Eclipse will generate some files automatically for you.

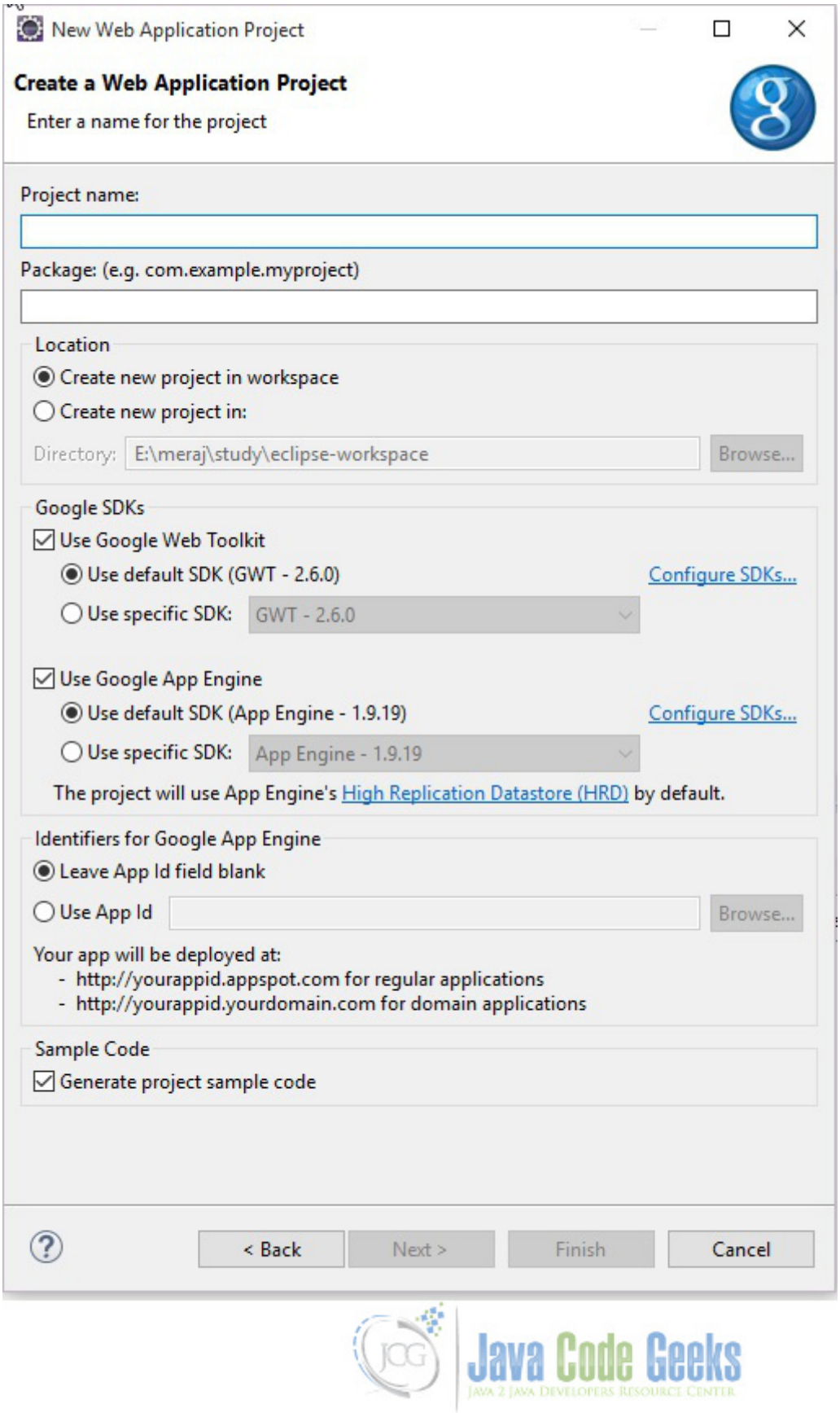

Figure 8.2: Create Project

8.2 Setup

Add the gwt-cal.jar file to the project's build path. Right-click on the project node in the Package Explorer and select *Build Path* > *Configure Build Path* > *Add External JARs*. Specify the downloaded gwt-cal-<version>.jar. Modify GWTCalendar.gwt.xml to inherit the gwt-cal module and theme:

```
<inherits name='com.bradrydzewski.gwt.calendar.Calendar' />
<inherits name='com.bradrydzewski.gwt.calendar.theme.google.Google' />
```

Add the gwt-dnd jar as well.

```
<inherits name='com.allen_sauer.gwt.dnd.gwt-dnd'/>
```

Below is the GWT configuration file:

GWTCalendar.gwt.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit 2.6.0//EN"
"https://google-web-toolkit.googlecode.com/svn/tags/2.6.0/distro-source/core/src/gwt-module ↩
    .dtd">
<module rename-to='gwtcalendar'>
  <inherits name='com.google.gwt.user.User'/>
  <inherits name='com.bradrydzewski.gwt.calendar.Calendar' />
  <inherits name='com.bradrydzewski.gwt.calendar.theme.google.Google' />
  <inherits name='com.allen_sauer.gwt.dnd.gwt-dnd'/>

  <set-property name="user.agent" value="safari"/>
  <inherits name='com.google.gwt.user.theme.clean.Clean'/>

  <!-- Specify the app entry point class. -->
  <entry-point class='com.javacodegeeks.client.GWTCalendar'/>

  <source path='client'/>
  <source path='shared'/>

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

8.3 Add widget

To the the Calendar widget modify the GWTCalendar class to add the code below:

```
Calendar calendar = new Calendar();
calendar.setDate(new Date());
calendar.setDays(5); //number of days displayed at a time
calendar.setWidth("400px");
calendar.setHeight("400px");
RootPanel.get("calendarContainer").add(calendar);
```

Below is the Entry class:

GWTCalendar.java

```
package com.javacodegeeks.client;

import java.util.Date;
```

```
import com.bradrydzewski.gwt.calendar.client.Calendar;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.RootPanel;

/**
* Entry point classes define <code>onModuleLoad()</code>.
*/
public class GWTCalendar implements EntryPoint {

    /**
    * This is the entry point method.
    */
    public void onModuleLoad() {
        Calendar calendar = new Calendar();
        calendar.setDate(new Date());
        calendar.setDays(5); //number of days displayed at a time
        calendar.setWidth("400px");
        calendar.setHeight("400px");
        RootPanel.get("calendarContainer").add(calendar);
    }
}
```

8.4 Compile

To compile the application right click on the project and select 'Google' ⟹ 'GWT Compile'. You will get a pop-up showing the project name. Click on the 'Compile' button. GWT will start compiling the project. To reduce the number of permutation you can add the below property in your GWTCalendar.gwt.xml:

```
<set-property name="user.agent" value="safari"/>
```

Your permutations will decrease from 55 to 11.

8.5 Running the application

To run the application right click on the project and select "Run As" ⟹ "Web Application (Classic Dev Mode)". Eclipse will display a URL in the "Development Mode" tab. Copy this URL and paste it on you favourite browser. Remove the part after ".html" and click enter.

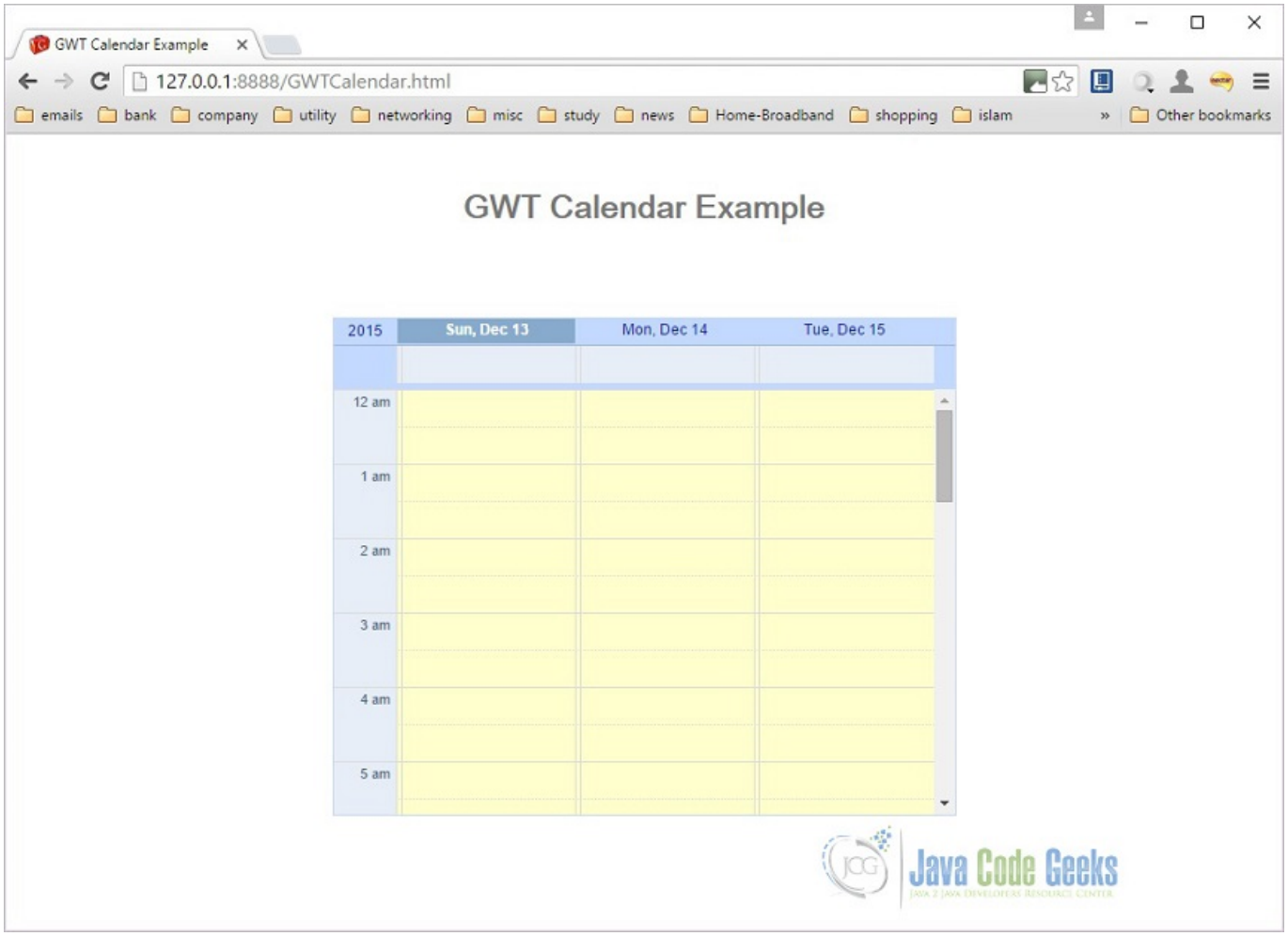

Figure 8.3: Run

8.6 Download the source file

This was an example of `GWT Calendar`.

Download

You can download the full source code of this example here: GWTCalendar. Please note that the jar files from the lib folder has been removed to save some space.

Chapter 9

GWT Dialogbox Example

In this tutorial, we will look into the details of *Google Web Toolkit (GWT) Dialog Box*. In our previous tutorials GWT Tutorial for Beginners, we explained how to create a GWT Web Application project using eclipse and we have seen the basic steps to develop user interface using widgets. GWT Panel Example and GWT HTMLPanel Example are related tutorial that covers GWT Panels in details. In this tutorial, we will focus on GWT DialogBox and it's usage to develop user interface.

Here we are using GWT 2.7 integrated with Eclipse Mars 4.5.

9.1 Introduction

As per the *GWT Javadoc*, *"GWT Dialogbox is a form of popup that has a caption area at the top and can be dragged by the user. Unlike a PopupPanel, calls to PopupPanel.setWidth(String) and PopupPanel.setHeight(String) will set the width and height of the dialog box itself, even if a widget has not been added as yet." GWT Dialogbox* provides a way to show more interactive pop-up through which the user can provide input to the application.

To learn how `DialogBox` can be used in application, we will create customised widgets using *GWT Dialogbox*.

9.2 Class Declaration

DialogBox.class

```
public class DialogBox extends DecoratedPopupPanel implements HasHTML,
    HasSafeHtml, MouseListener {

...
}
```

`DialogBox` inherits the property of `DecoratedPopupPanel` and `PopupPanel` consecutively. The `PopupPanel` overlays the browser's client area and pop-up over other widgets.

9.3 Constructors

9.3.1 DialogBox()

Creates an empty dialog box with auto-hide property set as *false*. This means the dialog should not be automatically hidden when the user clicks outside of it.

9.3.2 DialogBox(boolean autoHide)

Creates an empty dialog box specifying its *auto-hide* property.

9.3.3 DialogBox(Caption captionWidget)

Creates an empty dialog box specifying its caption. Caption is the widget that renders as the DialogBox's header.

9.3.4 DialogBox(boolean autoHide, boolean modal)

Creates an empty dialog box specifying its *auto-hide* and *modal* properties. If modal property is defines as *true*, keyboard and mouse events for widgets will not be contained by the dialog and should be ignored.

9.3.5 DialogBox(boolean autoHide, boolean modal, Caption captionWidget)

Creates an empty dialog box specifying its *auto-hide*, *modal* properties and an custom *Caption* widget.

9.4 Method Summary

Method Signature	Description
public void show()	Shows the popup and attach it to the page. It must has a child widget before this method is called.
public void hide(boolean autoClosed)	Hides the popup and detaches it from the page. This has no effect if it is not currently showing.
public void setWidget(Widget w)	Sets this panel's widget. Any existing child widget will be removed.
public void setAnimationEnabled(boolean enable)	Enable or disable animations(instead of immediate).
public void setPopupPosition(int left, int top)	Sets the popup's position relative to the browser's client area.
public void setText(String text)	Sets the text inside the caption.
public void setHTML(String html)	Sets the html string inside the caption.
public Caption getCaption()	Provides access to the dialog's caption..

9.5 Examples

9.5.1 Custom Dialogbox Example 1

Custom *Dialogbox* appears on click of button "Dialogbox". Auto-hide property of the dialogbox is based on the first parameter passed while initializing the dialog box. The *Close* button on *Dialogbox* is visible only if the auto-hide property is disabled.

SampleWebApplication.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
        VerticalPanel verticalPanel = new VerticalPanel();
        verticalPanel.setSpacing(10);
        verticalPanel.setBorderWidth(1);
        verticalPanel.setSize("100%", "100%");
        verticalPanel.setHorizontalAlignment(HasHorizontalAlignment.ALIGN_CENTER);
        verticalPanel.setVerticalAlignment(HasVerticalAlignment.ALIGN_MIDDLE);
```

```
        // The log in button
        Button submit = new Button("DialogBox");
        verticalPanel.add(submit);
        submit.addClickHandler(new ClickHandler() {
                @Override
                public void onClick(ClickEvent event) {
                        // Add validation
                        showCustomDialog();
                }
        });

        // Add our panel to the page
        RootLayoutPanel.get().add(verticalPanel);
}
/**
 * Draws Custom Dialog box.
 * @return DialogBox
 */
private DialogBox showCustomDialog() {

       final DialogBox dialog = new DialogBox(false, true);
       // final DialogBox dialog = new DialogBox(true, true);
       // Set caption
       dialog.setText("DialogBox Caption");
       // Setcontent
        Label content = new Label("This is sample text message inside "
                + "Customized Dialogbox. In this example a Label is "
                + "added inside the Dialogbox, whereas any custom widget "
                + "can be added inside Dialogbox as per application'ser's need. ");
       if (dialog.isAutoHideEnabled())  {
           dialog.setWidget(content);
        } else {
        VerticalPanel vPanel = new VerticalPanel();vPanel.setSpacing(2);
        vPanel.add(content);vPanel.add(new Label("\n"));
        vPanel.add(new Button("Close", new ClickHandler() {
        public void onClick(ClickEvent event) {
                dialog.hide();
                }
        }));
        dialog.setWidget(vPanel);
        }
        dialog.setPopupPosition(100, 150);
        dialog.show();
        return dialog;
}
```

Output:

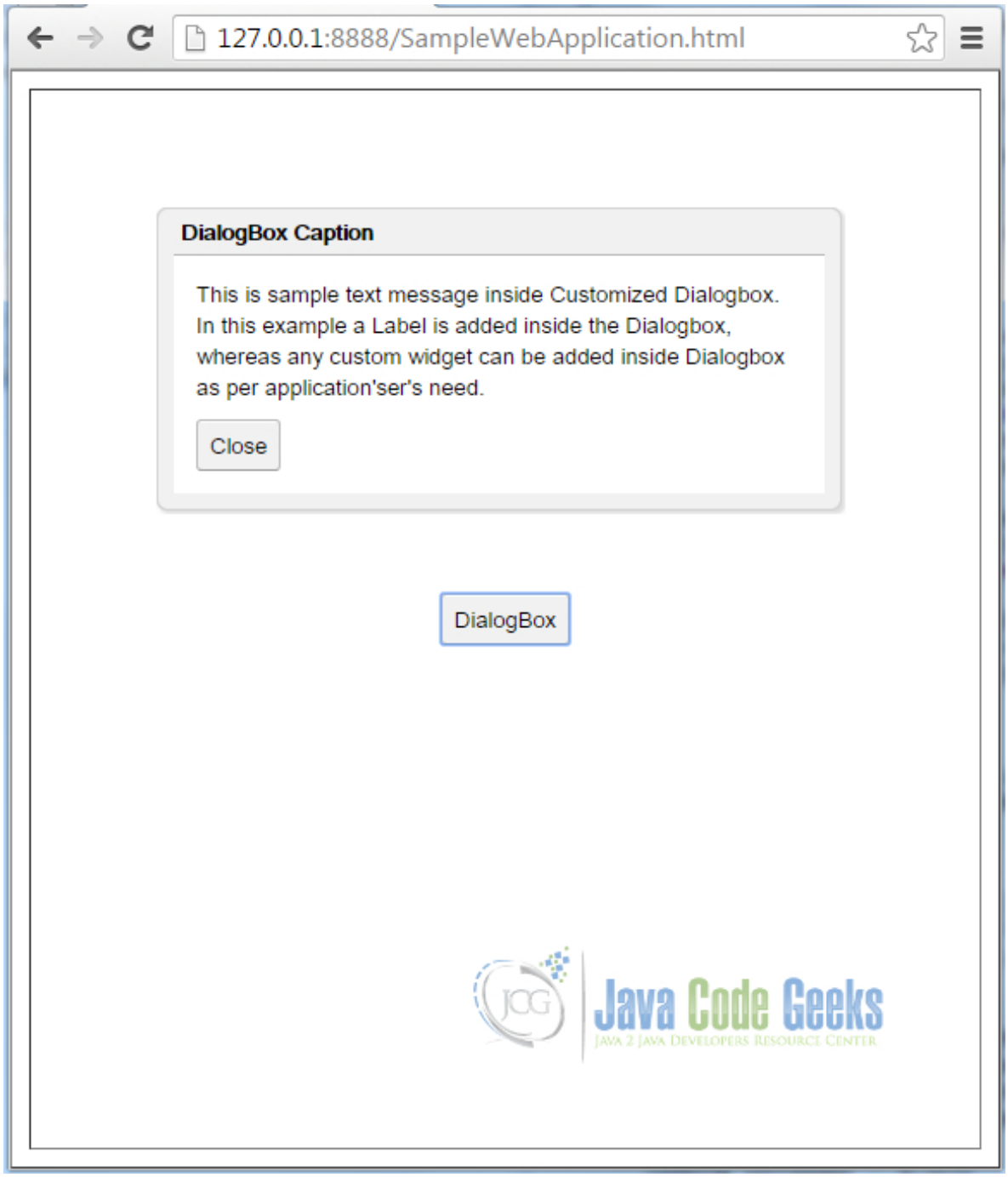

Figure 9.1: Custom Dialogbox Example

After enabling auto-hide property for Dialogbox (Refer line no. 32 in Custom Dialogbox Example 1).

Output:

Figure 9.2: Custom Dialogbox Example Auto-hide Enabled

9.5.2 Custom Dialogbox Example 2

In this example, a Dialogbox is customised by adding `DockPanel` as its child widget.

SampleWebApplication.java

```
/**
 * This is the entry point method.
 */
public void onModuleLoad() {
```

```
        Button btn= new Button("Dialogbox", new ClickHandler() {

                @Override
                public void onClick(ClickEvent event) {
                        DialogBox dlg = new CustomDialog();
                        dlg.center();
                }
        });
        RootPanel.get().add(btn);
}
/**
 * CustomDialog adds DockPanel as its child widget.
 */
class CustomDialog extends DialogBox implements ClickHandler {
        public CustomDialog() {
                super(true);
                setText("Sample DialogBox");

                Button closeButton = new Button("Close", this);
                HTML msg = new HTML("A Custom dialog box.",true);

                DockPanel dock = new DockPanel();
                dock.setSpacing(6);
                Image image = new Image();
                image.setUrl("https://www.javacodegeeks.com/wp-content/uploads/2012/12/ ↩
                    JavaCodeGeeks-logo.png");
                dock.add(image, DockPanel.CENTER);
                dock.add(closeButton, DockPanel.SOUTH);
                dock.add(msg, DockPanel.NORTH);

                dock.setCellHorizontalAlignment(closeButton, DockPanel.ALIGN_CENTER);
                dock.setWidth("100%");
                setWidget(dock);
        }

        @Override
        public void onClick(ClickEvent event) {
                hide();
        }
}
```

Output:

Check Video Output

9.6 Project References

GWT UIPanels

GWT API Reference

9.7 Download Eclipse Project

Download

You can download the full source code of this example here: **GWT DialogBox Examples**

Chapter 10

GWT Dialogbox Example

In this example we will learn how to use Tables in GWT. Google Web Toolkit is a development framework for creating Ajax-enabled web applications in Java. A `CellTable` represents a tabular view that supports paging and columns. A `FlexTable` on the other hand allows user to create cell on demand. It can be jagged (that is, each row can contain a different number of cells) and individual cells can be set to span multiple rows or columns. Tools and technologies used in this example are Java 1.8, Eclipse Luna 4.4.2, Eclipse GWT Plugin 2.6

10.1 Introduction

A cell table (data presentation table) provides high-performance rendering of large data sets in a tabular view. A `CellTable` is used to represent a data in tabular format. The `Column` class defines the `Cell` used to render a column. Implement `Column.getValue(Object)` to retrieve the field value from the row object that will be rendered in the Cell. A Header can be placed at the top (header) or bottom (footer) of the `CellTable`. You can specify a header as text using `AbstractCellTable.addColumn(Column, String)`, or you can create a custom Header that can change with the value of the cells, such as a column total. The Header will be rendered every time the row data changes or the table is redrawn. If you pass the same header instance (==) into adjacent columns, the header will span the columns.

The FlexTable class extends the HTMLTable. `public class FlexTable extends HTMLTable`

10.2 Creating GWT project

To create a new GWT project go to File→New→Other, then type 'Web App'. Choose 'Web Application Project' under 'Google'.

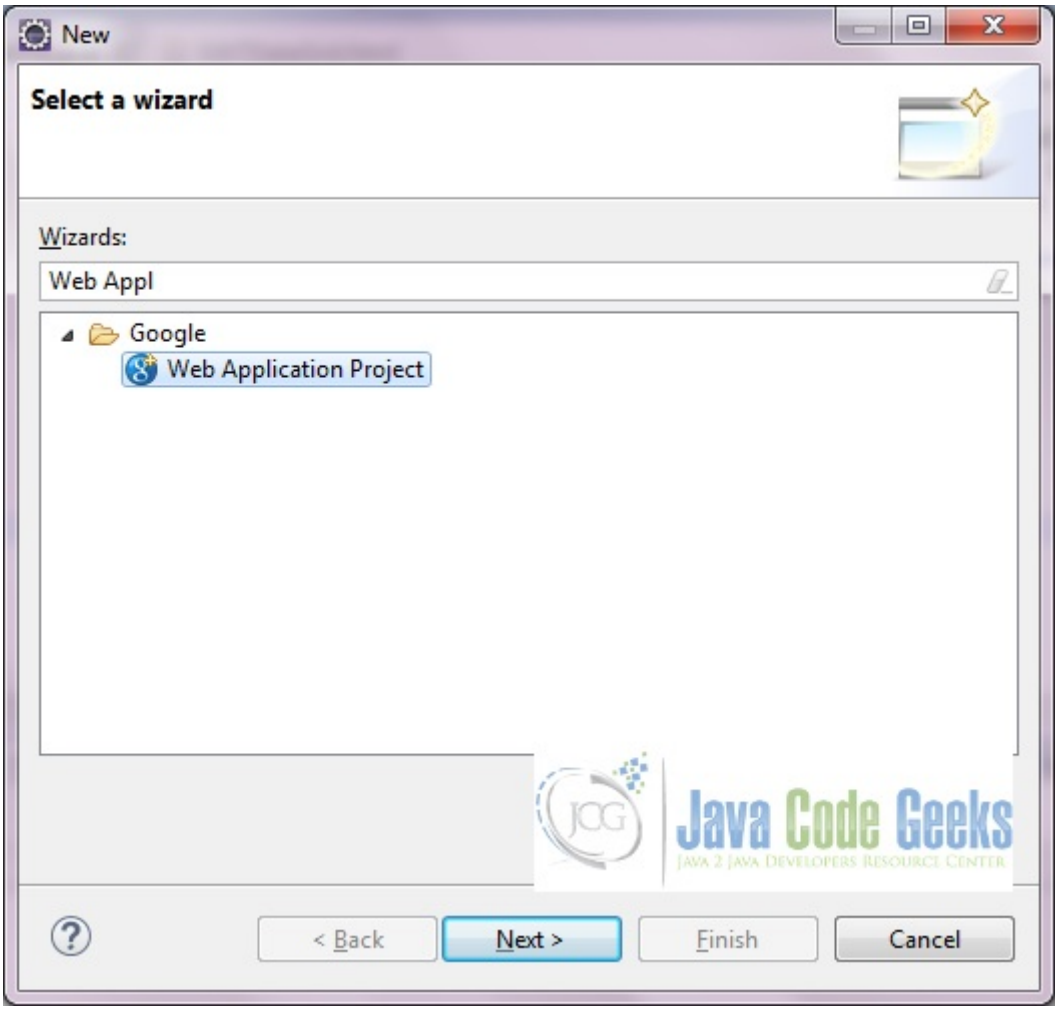

Figure 10.1: Create new Web Application Project

On the next window enter the Project name (‘GWTTable’) and the Package (com.javacodegeeks). Leave the other details as it is and click on ‘Finish’. Eclipse will generate some files automatically for you.

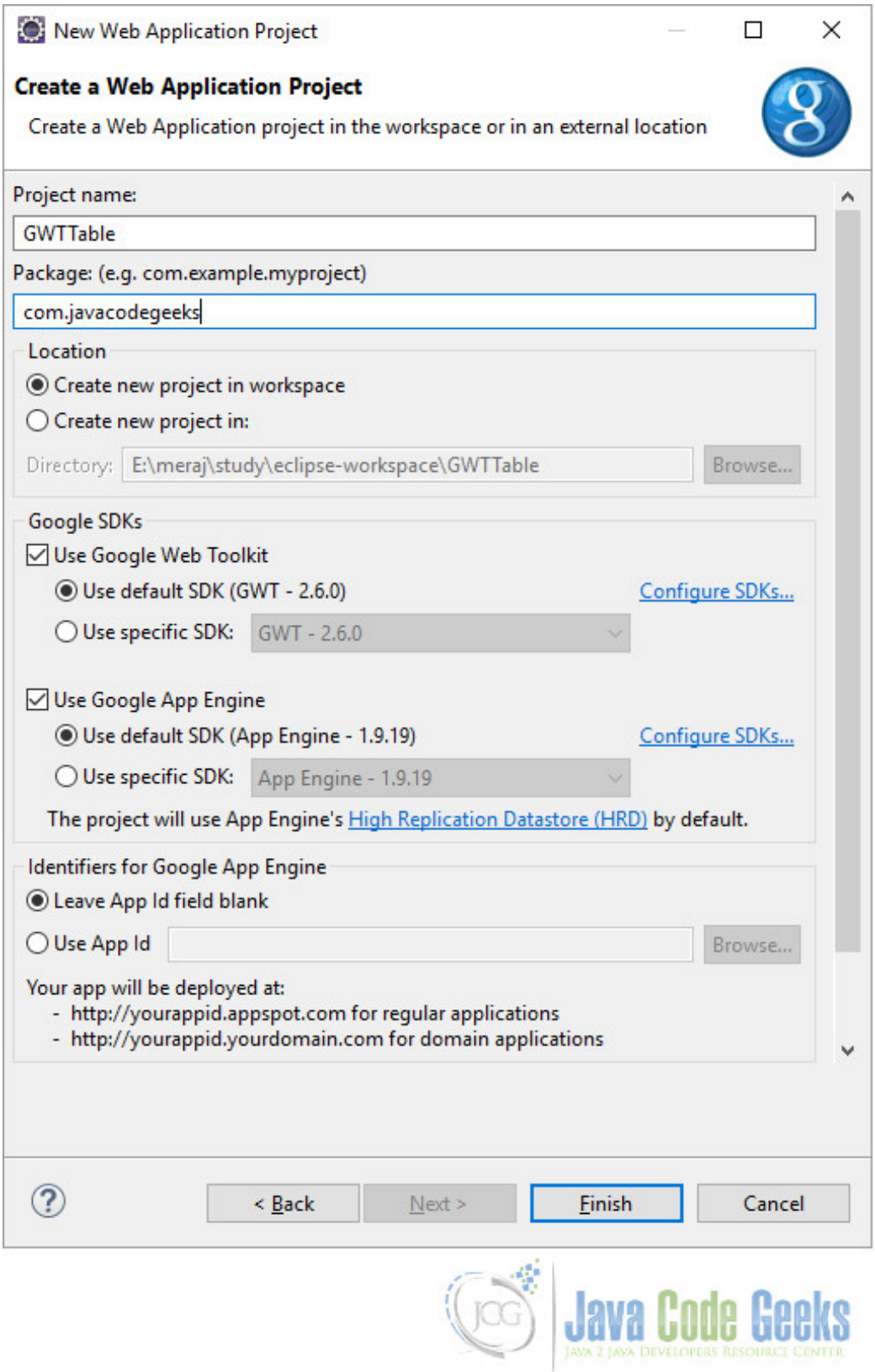

Figure 10.2: Create Project

For this example we don't need to change the GWT configuration file. We only need to change the Entry point class - 'GWT-Table.java'.

10.3 Java classes

First we show the model class which we are using for populating the data in the `CellTable`. It's a simple POJO representation of address data.

Address.java

```
package com.javacodegeeks.client;

public class Address {

  private String firstLine;
  private String secondLine;
  private String town;
  private String country;

  public Address(String firstLine, String secondLine, String town, String country) {
    this.firstLine = firstLine;
    this.secondLine = secondLine;
    this.town = town;
    this.country = country;
}

  public String getFirstLineOfAddress() {
    return this.firstLine;
  }

  public String getSecondLineOfAddress() {
    return this.secondLine;
  }

  public String getTown() {
    return this.town;
  }

  public String getCountry() {
    return this.country;
  }
}
```

Now we will see the Entry point class.

GWTTable.java

```
package com.javacodegeeks.client;
import java.util.ArrayList;
import java.util.List;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.cellview.client.CellTable;
import com.google.gwt.user.cellview.client.HasKeyboardSelectionPolicy. ↩
    KeyboardSelectionPolicy;
import com.google.gwt.user.cellview.client.TextColumn;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.ui.FlexTable;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.view.client.SelectionChangeEvent;
```

```
import com.google.gwt.view.client.SingleSelectionModel;

/**
* Entry point classes define <code>onModuleLoad()</code>.
*/
public class GWTTable implements EntryPoint {

  /**
  * This is the entry point method.
  */
  public void onModuleLoad() {
    FlexTable flexTable = createFlexTable();

    CellTable<Address> cellTableOfAddress = new CellTable<Address>();
    // The policy that determines how keyboard selection will work. Keyboard
    // selection is enabled.
    cellTableOfAddress.setKeyboardSelectionPolicy(KeyboardSelectionPolicy.ENABLED);

    // Add a text columns to show the details.
    TextColumn<Address> columnFirstLine = new TextColumn<Address>() {
      @Override
      public String getValue(Address object) {
        return object.getFirstLineOfAddress();
      }
    };
    cellTableOfAddress.addColumn(columnFirstLine, "First line");

    TextColumn<Address> columnSecondLine = new TextColumn<Address>() {
      @Override
      public String getValue(Address object) {
        return object.getSecondLineOfAddress();
      }
    };
    cellTableOfAddress.addColumn(columnSecondLine, "Second line");

    TextColumn<Address> townColumn = new TextColumn<Address>() {
      @Override
      public String getValue(Address object) {
        return object.getTown();
      }
    };
    cellTableOfAddress.addColumn(townColumn, "Town");

    TextColumn<Address> countryColumn = new TextColumn<Address>() {
      @Override
      public String getValue(Address object) {
        return object.getCountry();
      }
    };
    cellTableOfAddress.addColumn(countryColumn, "Country");

    final SingleSelectionModel<Address> selectionModel = new SingleSelectionModel<Address ↩
        >();
    cellTableOfAddress.setSelectionModel(selectionModel);
    selectionModel.addSelectionChangeHandler(new SelectionChangeEvent.Handler() {

      public void onSelectionChange(SelectionChangeEvent event) {

        Address selectedAddress = selectionModel.getSelectedObject();
        if (selectedAddress != null) {
          Window.alert("Selected: First line: " + selectedAddress.getFirstLineOfAddress() + ↩
              ", Second line: " + selectedAddress.getSecondLineOfAddress());
```

```
        }
      }
    });

    List<Address> addresses = new ArrayList<Address>() {
      {
        add(new Address("Cell Table", "First line", "Oxford", "UK"));
        add(new Address("Cell Table", "Second line", "Cambrige", "UK"));
      }
    };

    cellTableOfAddress.setRowCount(addresses.size(), true);
    cellTableOfAddress.setRowData(0, addresses);

    VerticalPanel vp = new VerticalPanel();
    vp.setBorderWidth(1);
    vp.add(flexTable);
    vp.add(cellTableOfAddress);

    RootPanel.get("container").add(vp);
}

  private FlexTable createFlexTable() {
    FlexTable flexTable = new FlexTable();
    flexTable.setBorderWidth(1);
    flexTable.setText(0, 0, "This is an example of flextable");
    flexTable.setText(2, 2, "This is also an example of flextable");
    flexTable.getFlexCellFormatter().setColSpan(1, 0, 3);
    return flexTable;
  }
}
```

10.4 Difference

Here we will discuss the differences between these two GWT table types.

`CellTable` always has the same number of rows and/or columns while a `FlexTable` can have different rows per column and different columns per row (This is made possible due to the html properties rowspan and colspan). Thus when you need flexibale rows or columns you can use `FlexTable` and otherwise `CellTable`. However, `FlexTable` is extremely slow in Internet Explorer, due to slow DOM methods that are used to create a the table. Therefore, avoid FlexTable if you can or only use it create a simple layout (although in that case the `DockPanel` might be somewhat easier to use).

10.5 Compile

To compile the application right click on the project and select 'Google' ⟹ 'GWT Compile'. A popu-up will be displayed. Click the *Compile* button. GWT will start compiling your projects for different permutations. Below is the result of compilation which you see in the *Console* window.

```
Compiling module com.javacodegeeks.GWTTable
  Compiling 5 permutations
    Compiling permutation 0...
    Compiling permutation 1...
    Compiling permutation 2...
    Compiling permutation 3...
    Compiling permutation 4...
  Compile of permutations succeeded
Linking into E:\meraj\study\eclipse-workspace\GWTTable\war\gwttable
```

```
Link succeeded
Compilation succeeded -- 95.073s
```

10.6 Running the application

To run the application right click on the project and select "Run As" ⟹ "Web Application (Classic Dev Mode)". Eclipse will display a URL in the "Development Mode" tab. Copy this URL and paste it on you favourite browser. Remove the part after ".html" and click enter.

Figure 10.3: Output

10.7 Download the source file

This was an example of `GWT Tables`.

Download

You can download the full source code of this example here: **GWT Table**. Please note that to save space the jar files from the lib directly have been removed.