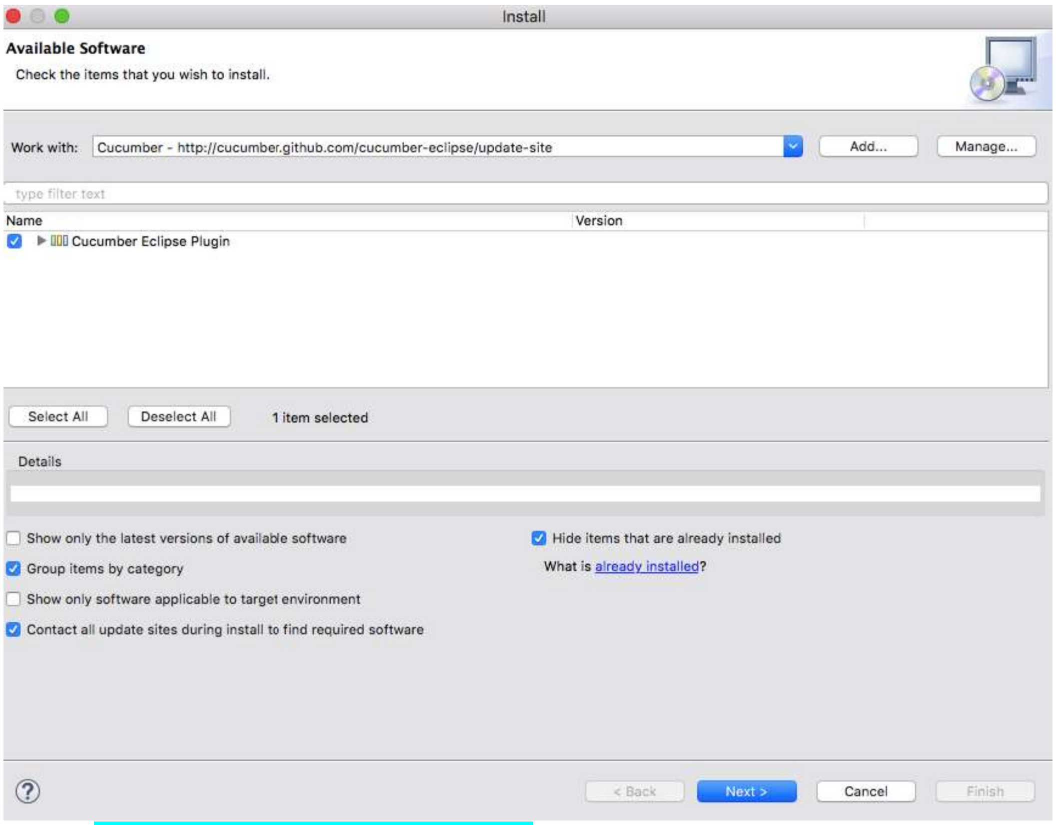
Setup For Selenium With Cucumber Using Maven

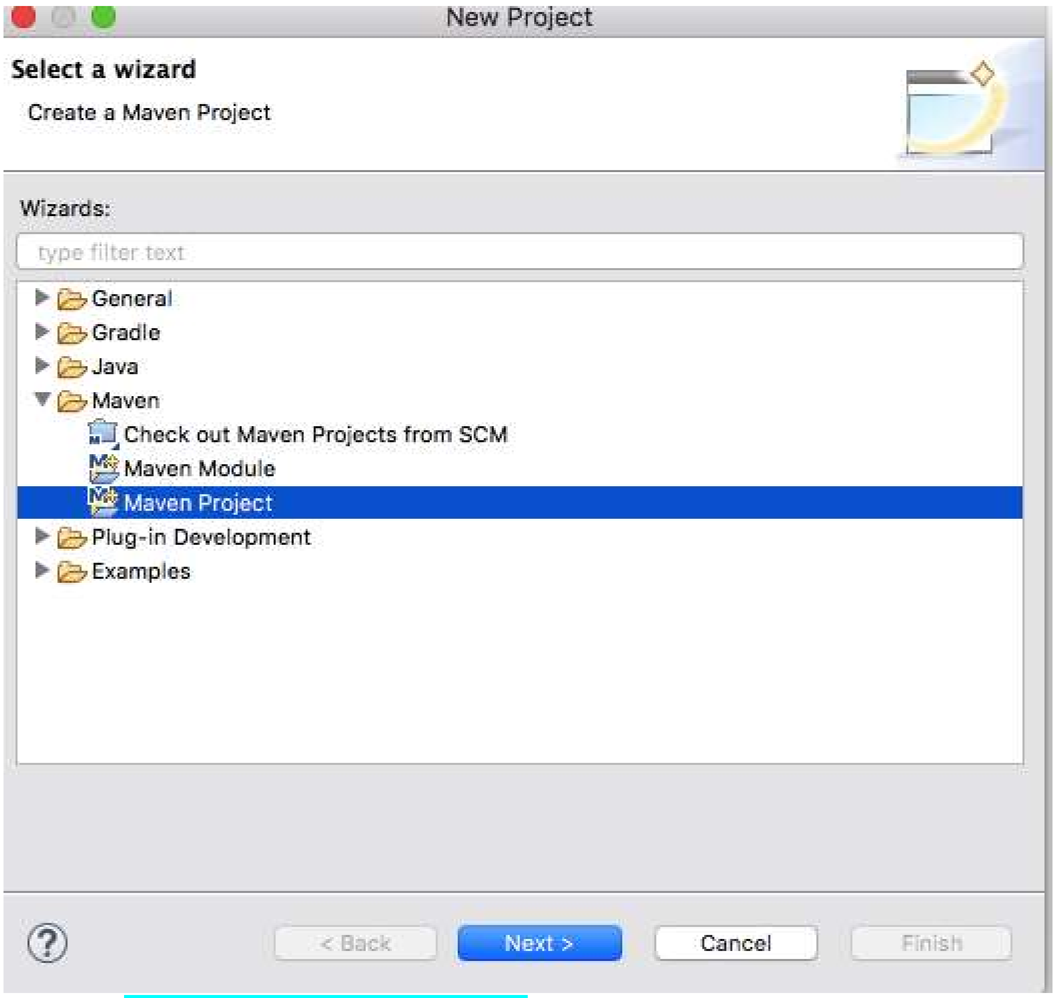
Why Use Cucumber-Selenium?
Cucumber is an open source tool that supports Behavior Driven Development (BDD) framework. It provides the facility to write tests in a human readable language called Gherkin. The Selenium-Cucumber framework supports programming languages such as Perl, PHP, Python, .NET, Java, etc.
In this blog, we will focus on how to set up Selenium with Cucumber using Maven, and also learn to write feature files using Gherkin, execution, and generating HTML reports.
Prerequisites To Set Up Cucumber
In order to install Cucumber on your system, you would need some basic installations on your system:

1. Set up JDK on your system (JDK 1.8 or the latest version)
2. Install Eclipse (Eclipse OXYGEN or the latest version)
3. Install Cucumber plugin:
   - In Eclipse, go to *Help → Install new software*
   - On the Available Software popup, enter the URL "http://cucumber.github.com/cucumber-eclipse/update-site" in the *Work with* field.

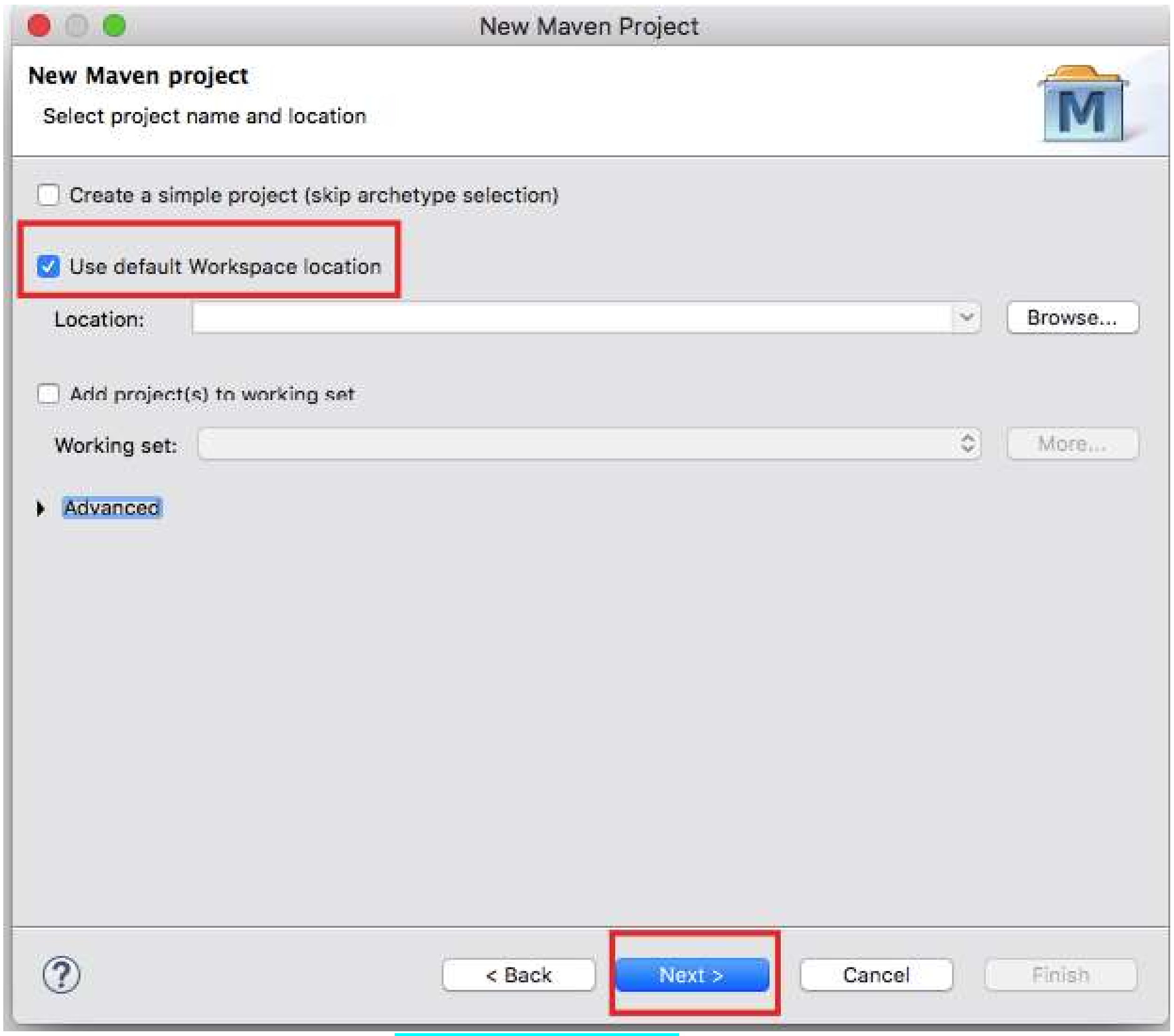

Install
Available Software
Check the items that you wish to install.
Work with: Cucumber - http://cucumber.github.com/cucumber-eclipse/update-site
Name | Version
[✓] Cucumber Eclipse Plugin
1 item selected
Details
[ ] Show only the latest versions of available software
[✓] Group items by category
[ ] Show only software applicable to target environment
[✓] Contact all update sites during install to find required software
[✓] Hide items that are already installed
What is already installed?

   - You will see "Cucumber Eclipse Plugin" displayed in the filter; select the checkbox and click Next, and you will navigate to the Install Details popup. Click Next to proceed further.
   - Accept the license in the Review License pop-up and click Finish.

Why Maven?
Maven is a automation build tool and is widely used for Java projects. It is mainly used in managing dependencies through pom.xml. Suppose you want to upgrade the JAR files and in your project you are using version 1.25 for Cucumber-Java dependency. You need to upgrade to the latest version. With the use of Maven, it's easy to upgrade the version.
Set Up Your Maven Project
Step 1: To create a Maven Project in Eclipse, click on *New → Project → In the wizard, select Maven Project*.



Step 2: On the new Maven Project pop-up, select the checkbox to create your project at the default location OR you can also browse and set a new location of your choice. Click on Next to proceed.


New Maven Project

New Maven project

Select project name and location

[ ] Create a simple project (skip archetype selection)

[✓] Use default Workspace location

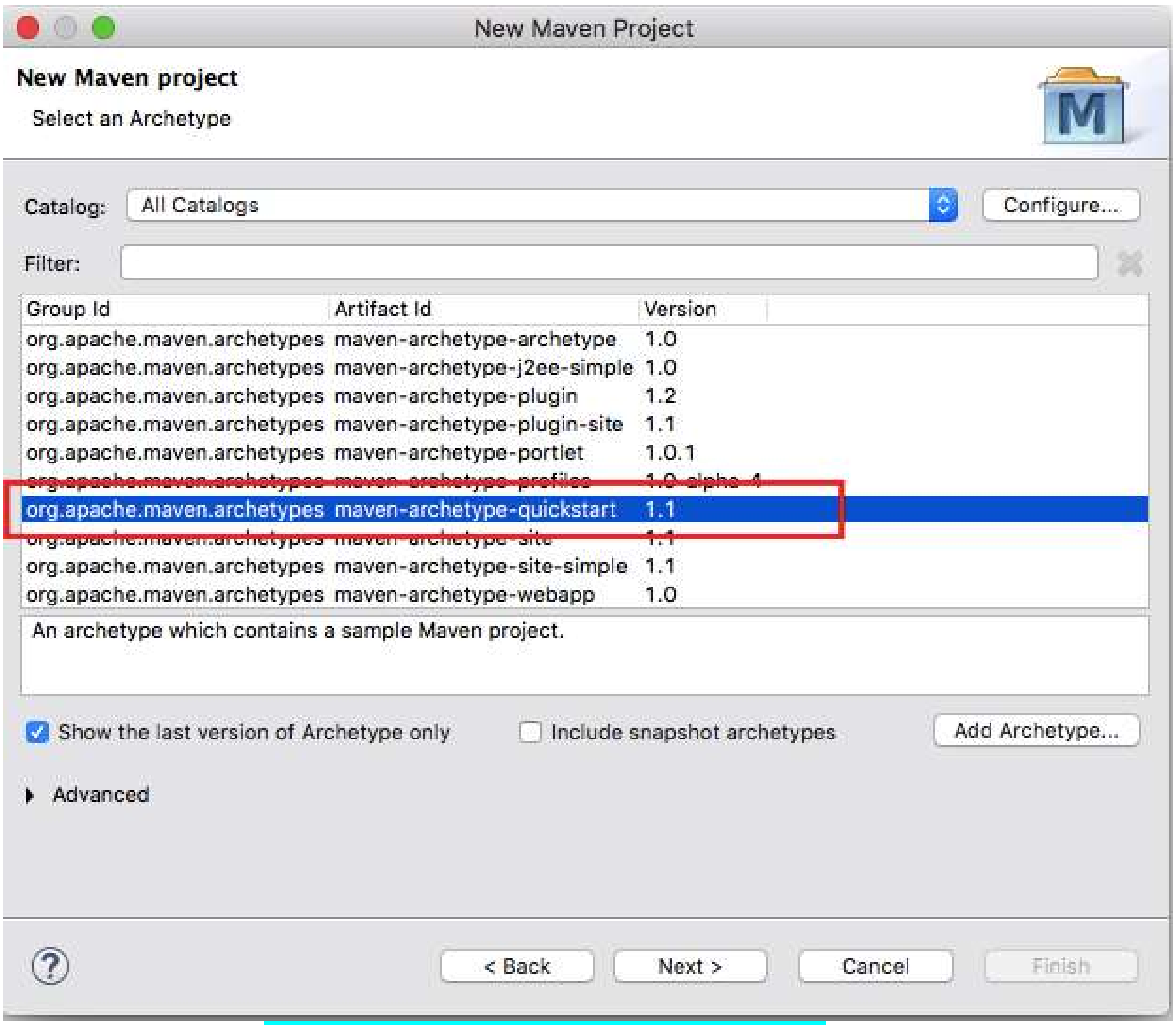
Location:

[ ] Add project(s) to working set

Working set:

Advanced

Step 3: On the next screen, by default the Group ID and Artifact ID org.apache.maven.archetypes maven-archetypes-quickstart 1.1 is selected. Click on Next to proceed.



Catalog: All Catalogs

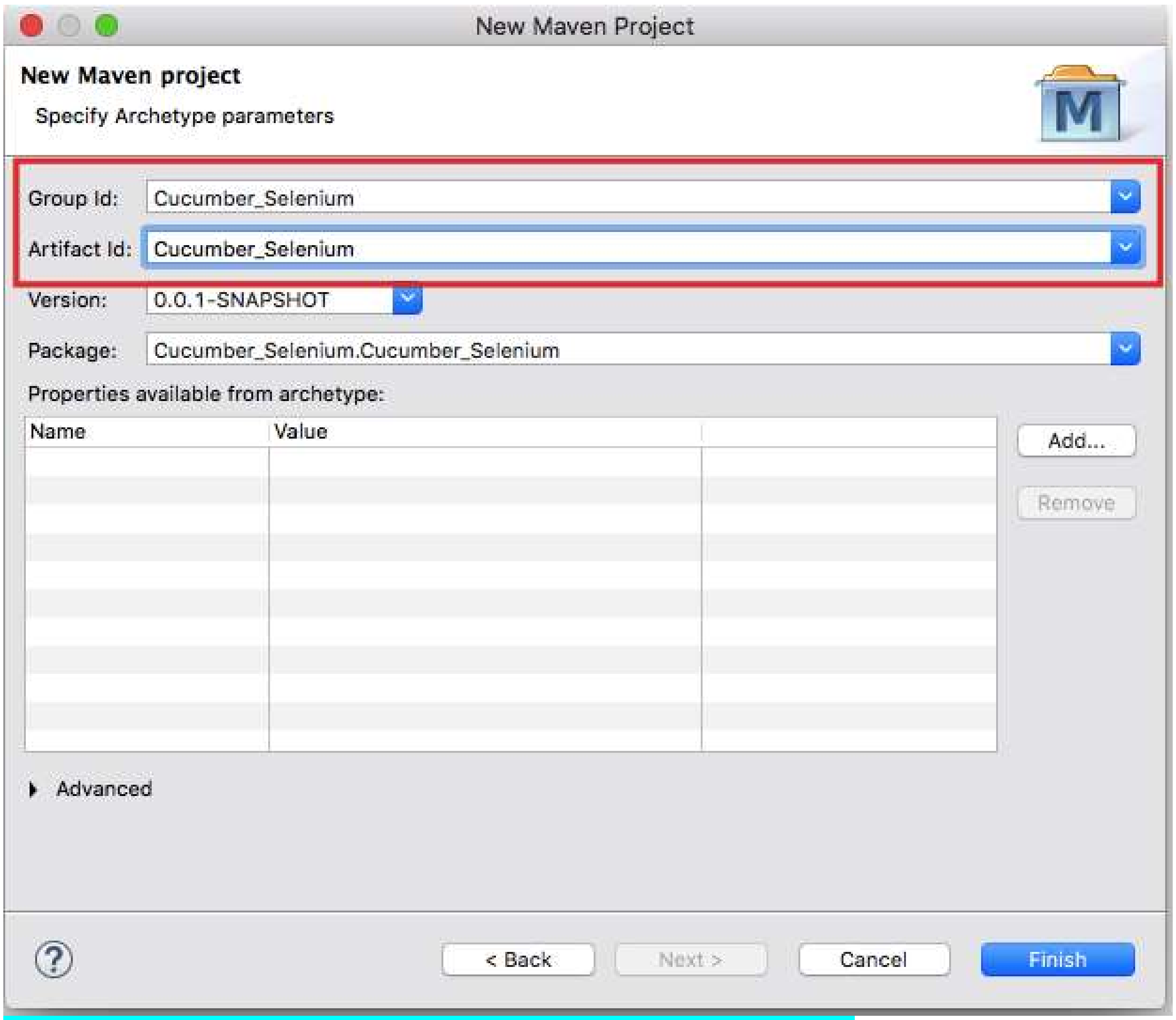
Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-archetype	1.0
org.apache.maven.archetypes	maven-archetype-j2ee-simple	1.0
org.apache.maven.archetypes	maven-archetype-plugin	1.2
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven project.

☑ Show the last version of Archetype only ☐ Include snapshot archetypes

Step 4: In the next screen, you will have to mention a Group ID and Artifact ID of your own choice; this is the name of your Maven project. Once you click the Finish button, a Maven project will be created in Eclipse.


New Maven Project

New Maven project

Specify Archetype parameters

Group Id: Cucumber_Selenium

Artifact Id: Cucumber_Selenium

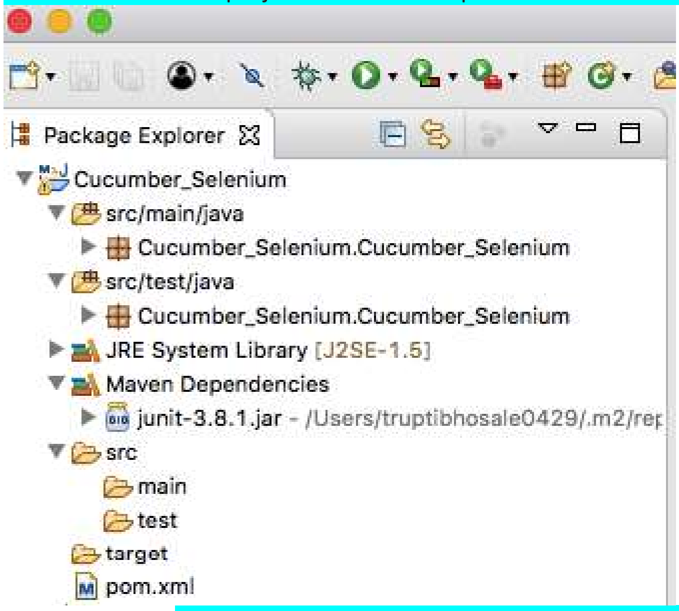
Version: 0.0.1-SNAPSHOT

Package: Cucumber_Selenium.Cucumber_Selenium

Properties available from archetype:

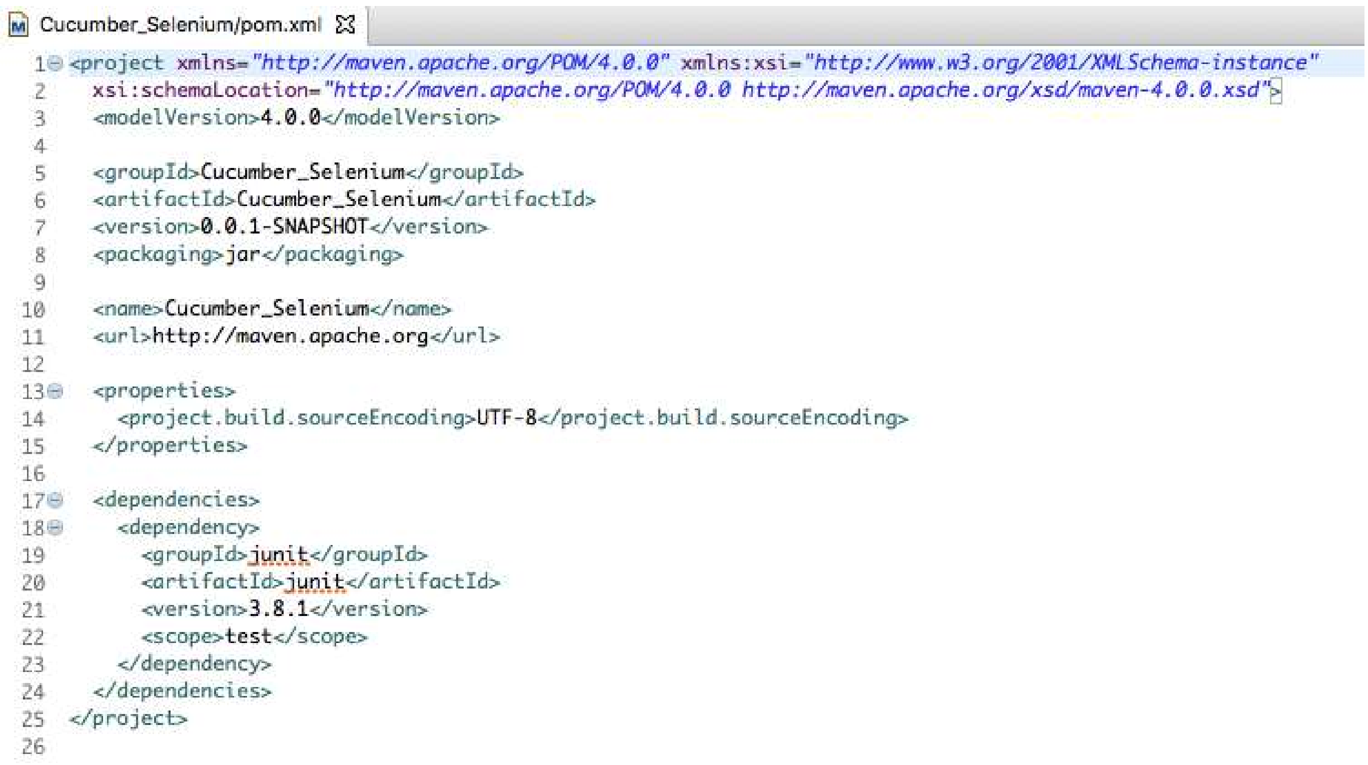
Name	Value

The structure of the project created in Eclipse will be similar to the following image.



```
Cucumber_Selenium
  src/main/java
    Cucumber_Selenium.Cucumber_Selenium
  src/test/java
    Cucumber_Selenium.Cucumber_Selenium
  JRE System Library [J2SE-1.5]
  Maven Dependencies
    junit-3.8.1.jar - /Users/truptibhosale0429/.m2/rep
  src
    main
    test
  target
  pom.xml
```

As you can see, there is a pom.xml file created in your Maven project. This file consists of the Group ID and Artifact ID you entered, and by default it consists of dependency for JUnit. Refer to the screenshot below.


Cucumber_Selenium/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>Cucumber_Selenium</groupId>
  <artifactId>Cucumber_Selenium</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Cucumber_Selenium</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Step 5: Now, in order to build a Selenium-Cucumber framework for us to work with, we need to add dependency for Selenium and Cucumber in pom.xml, which is somewhat similar to adding JAR files. We will be needing dependencies of the following:

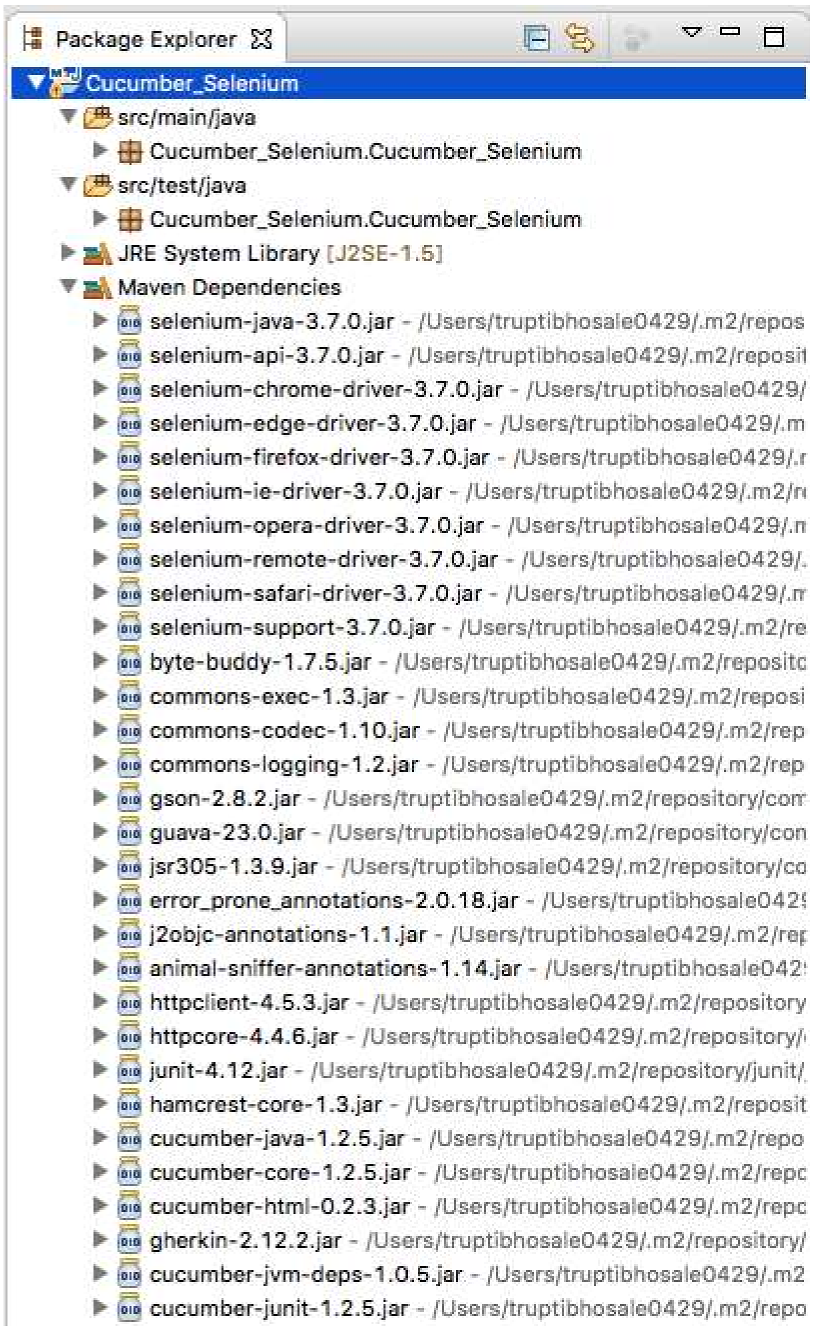
1. *Selenium-java*
2. *Cobertura*
3. *Cucumber-jvm-deps*
4. *Cucumber-reporting*
5. *Gherkin*
6. *Junit*
7. *Mockito-all-1.10.19*
8. *Cucumber-core*
9. *Cucumber-java*
10. *Cucumber-junit*

Note: Make sure the versions on Cucumber-java, Cucumber -junit and Cucumber-core are the same, i.e., if you are using Cucumber-java-1.2.5 make sure the versions of the other two dependencies are the same.
Step 6: To add dependencies for the above, you should refer to https://mvnrepository.com/. After adding dependencies for Cucumber and Selenium, the pom.xml file will look like this:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>Cucumber_Selenium</groupId>
<artifactId>Cucumber_Selenium</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>Cucumber_Selenium</name>
<url>http://maven.apache.org</url>
<properties>
```

```
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
<dependencies>
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>3.7.1</version>
</dependency>

<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.12</version>
<scope>test</scope>
</dependency>

<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>3.7.0</version>
</dependency>
<dependency>
<groupId>info.cukes</groupId>
<artifactId>cucumber-java</artifactId>
<version>1.2.5</version>
</dependency>
<dependency>
<groupId>info.cukes</groupId>
<artifactId>cucumber-jvm-deps</artifactId>
<version>1.0.5</version>
<scope>provided</scope>
</dependency>
<dependency>
<groupId>info.cukes</groupId>
<artifactId>cucumber-junit</artifactId>
<version>1.2.5</version>
<scope>test</scope>
</dependency>
<dependency>
<groupId>com.vimalselvam</groupId>
<artifactId>cucumber-extentsreport</artifactId>
<version>3.0.2</version>
</dependency>
<dependency>
<groupId>com.aventstack</groupId>
<artifactId>extentreports</artifactId>
<version>3.1.2</version>
</dependency>
</dependencies>
</project>
```

Step 7: Make sure to update the project after adding dependencies to pom.xml; you can do that by right clicking *Project* → *Maven* → *Update Project*. Once you update the project, you will see that many JAR files are added to the Maven Dependencies folder in your project.



```
Package Explorer
▼ Cucumber_Selenium
  ▼ src/main/java
    ▶ Cucumber_Selenium.Cucumber_Selenium
  ▼ src/test/java
    ▶ Cucumber_Selenium.Cucumber_Selenium
  ▶ JRE System Library [J2SE-1.5]
  ▼ Maven Dependencies
    ▶ selenium-java-3.7.0.jar - /Users/truptibhosale0429/.m2/repos
    ▶ selenium-api-3.7.0.jar - /Users/truptibhosale0429/.m2/reposit
    ▶ selenium-chrome-driver-3.7.0.jar - /Users/truptibhosale0429/
    ▶ selenium-edge-driver-3.7.0.jar - /Users/truptibhosale0429/.m
    ▶ selenium-firefox-driver-3.7.0.jar - /Users/truptibhosale0429/.r
    ▶ selenium-ie-driver-3.7.0.jar - /Users/truptibhosale0429/.m2/r
    ▶ selenium-opera-driver-3.7.0.jar - /Users/truptibhosale0429/.n
    ▶ selenium-remote-driver-3.7.0.jar - /Users/truptibhosale0429/.
    ▶ selenium-safari-driver-3.7.0.jar - /Users/truptibhosale0429/.m
    ▶ selenium-support-3.7.0.jar - /Users/truptibhosale0429/.m2/re
    ▶ byte-buddy-1.7.5.jar - /Users/truptibhosale0429/.m2/reposito
    ▶ commons-exec-1.3.jar - /Users/truptibhosale0429/.m2/reposi
    ▶ commons-codec-1.10.jar - /Users/truptibhosale0429/.m2/rep
    ▶ commons-logging-1.2.jar - /Users/truptibhosale0429/.m2/rep
    ▶ gson-2.8.2.jar - /Users/truptibhosale0429/.m2/repository/com
    ▶ guava-23.0.jar - /Users/truptibhosale0429/.m2/repository/con
    ▶ jsr305-1.3.9.jar - /Users/truptibhosale0429/.m2/repository/co
    ▶ error_prone_annotations-2.0.18.jar - /Users/truptibhosale042
    ▶ j2objc-annotations-1.1.jar - /Users/truptibhosale0429/.m2/rep
    ▶ animal-sniffer-annotations-1.14.jar - /Users/truptibhosale042
    ▶ httpclient-4.5.3.jar - /Users/truptibhosale0429/.m2/repository
    ▶ httpcore-4.4.6.jar - /Users/truptibhosale0429/.m2/repository/
    ▶ junit-4.12.jar - /Users/truptibhosale0429/.m2/repository/junit/
    ▶ hamcrest-core-1.3.jar - /Users/truptibhosale0429/.m2/reposit
    ▶ cucumber-java-1.2.5.jar - /Users/truptibhosale0429/.m2/repo
    ▶ cucumber-core-1.2.5.jar - /Users/truptibhosale0429/.m2/repo
    ▶ cucumber-html-0.2.3.jar - /Users/truptibhosale0429/.m2/repo
    ▶ gherkin-2.12.2.jar - /Users/truptibhosale0429/.m2/repository/
    ▶ cucumber-jvm-deps-1.0.5.jar - /Users/truptibhosale0429/.m2
    ▶ cucumber-junit-1.2.5.jar - /Users/truptibhosale0429/.m2/repo
```

Step 8: To proceed with Cucumber implementation, we need to create three packages to store the feature files, step definition code and testrunner code. Let us create three packages: features, seleniumgluecode and runner. To create a new package in src/test/java, right click the folder → *New* → *Package*.
Step 9: Now create the feature file in the Features package. *Right click* → *New* → *File* → *Enter name test.feature.*
Note: If you don't find 'File', then click on 'Others' and then select the 'File' option.
Step 10: Create a class test.java to write the gluecode for the features written. *Right click seleniumgluecode* → *New* → *Class* → *enter name as test and save.*
Step 11: To run the feature files and their respective code, we need to write a JUnit runner class. *Right click runner* → *New* → *Class* → *enter name as testrunner.*
The basic structure to write and execute code is ready and you are almost set to write and execute Cucumber scripts.
Basic Scenarios
Let us consider a login scenario where the user needs to enter a username and password, and confirm if he can log in. We need to write a basic scenario in the feature file test.feature which we created.
Feature: Login Feature

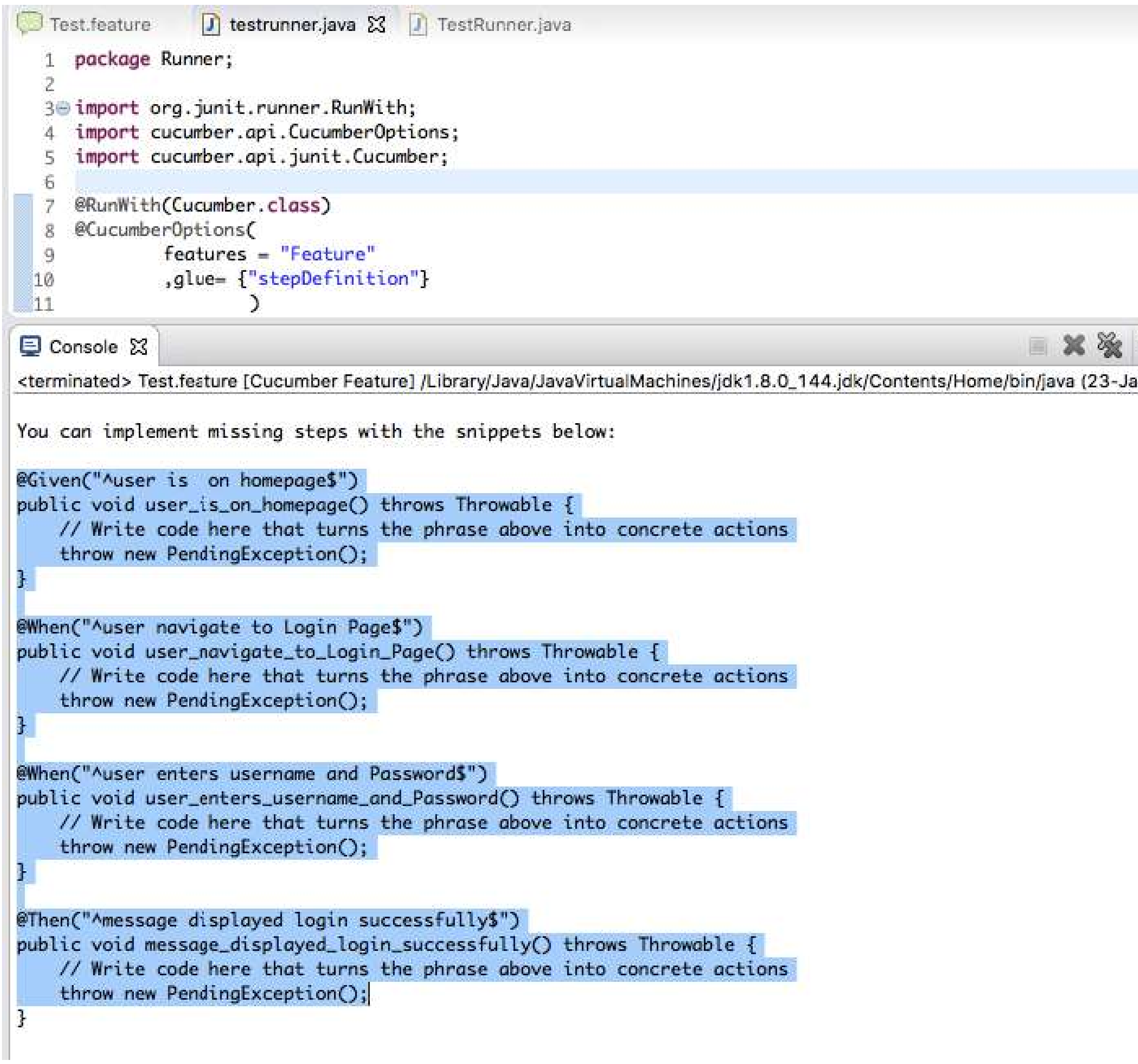
```
  Verify if user is able to Login in to the site

  Scenario: Login as a authenticated user
    Given user is  on homepage
    When user navigates to Login Page
    And user enters username and Password
    Then success message is displayed
```

The basic scenario to test the login functionality is ready. Now, we need to write the JUnit test runner class in order to execute the feature file. Add the code given below to the testrunner.java class.

```
package Runner;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(
features = "src/test/javaFeatures"
,glue= {"seleniumgluecode"}
)

public class testrunner {

}
```

Execution

To execute the code above, right click ***test.feature file → Run As → Cucumber feature***. On executing the test.feature file, you will notice that in the console it mentions the implementation of missing steps. This is because we have not defined the code to execute the steps.



```
package Runner;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(
        features = "Feature"
        ,glue= {"stepDefinition"}
            )
```

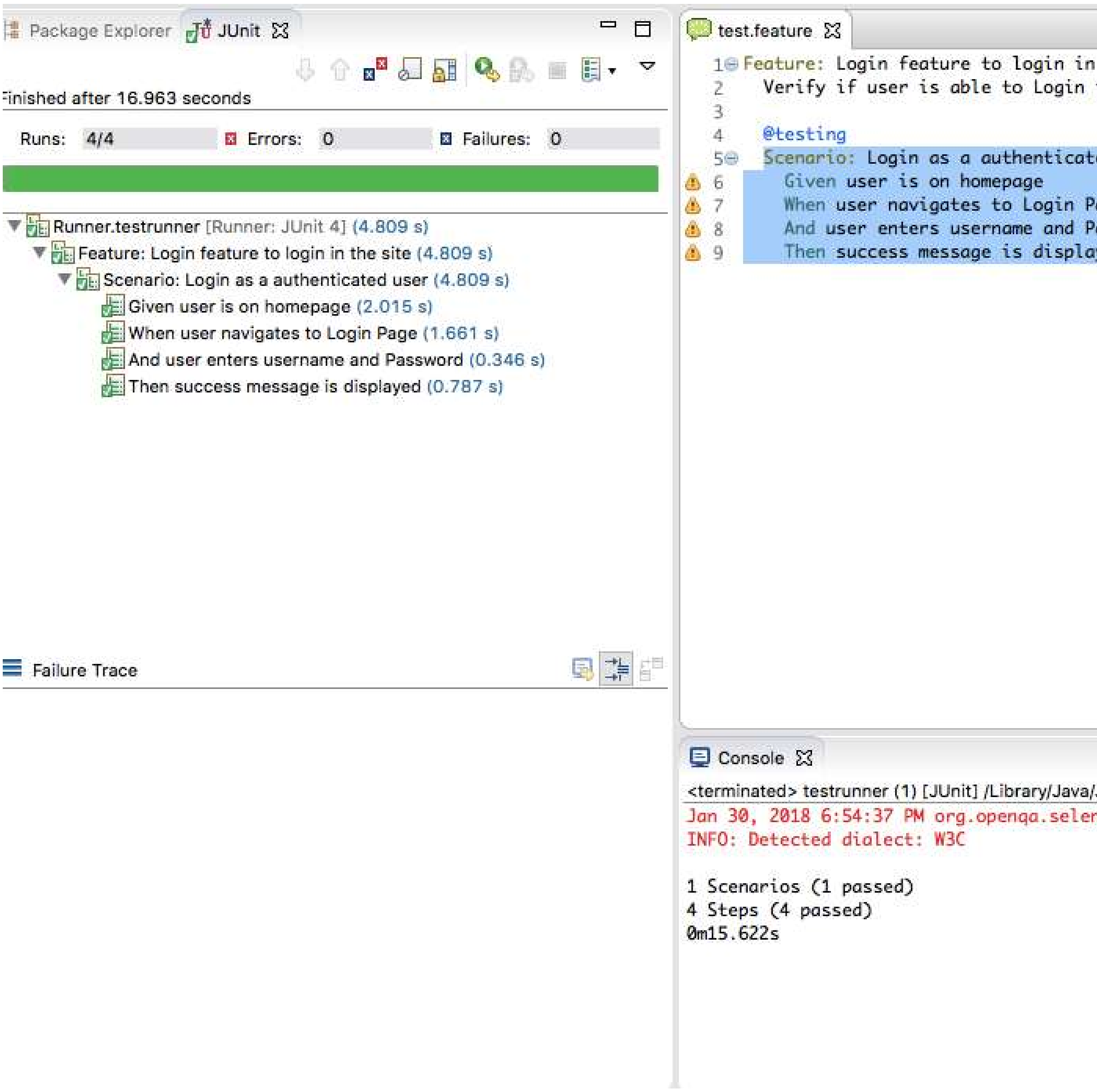
```
<terminated> Test.feature [Cucumber Feature] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (23-Ja

You can implement missing steps with the snippets below:

@Given("^user is  on homepage$")
public void user_is_on_homepage() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^user navigate to Login Page$")
public void user_navigate_to_Login_Page() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@When("^user enters username and Password$")
public void user_enters_username_and_Password() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}

@Then("^message displayed login successfully$")
public void message_displayed_login_successfully() throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

To begin writing the glue code for the steps, copy the snippets from the console and paste them into the test.java class which we created under seleniumgluecode package.
Once you copy the snippet, remove the “throw new PendingException()” and write appropriate code for the steps.

```
package seleniumgluecode;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;
import org.junit.Assert;

public class test {
   public static WebDriver driver;
   @Given("^user is on homepage$")
```

```
    public void user_is_on_homepage() throws Throwable {
                System.setProperty("webdriver.gecko.driver","/Users/Documents/geckodriver");
        driver = new FirefoxDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
        driver.get("http://automationpractice.com/index.php");
    }

    @When("^user navigates to Login Page$")
    public void user_navigates_to_Login_Page() throws Throwable {
        driver.findElement(By.linkText("Sign in")).click();
    }

    @When("^user enters username and Password$")
    public void user_enters_username_and_Password() throws Throwable {
                driver.findElement(By.id("email")).sendKeys("blog.cucumber@gmail.com");
        driver.findElement(By.id("passwd")).sendKeys("Cucumber@blog");
        driver.findElement(By.id("SubmitLogin")).click();
    }

    @Then("^success message is displayed$")
    public void success_message_is_displayed() throws Throwable {
                String exp_message = "Welcome to your account. Here you can manage all   of your personal
information and orders.";
                Stringactual = driver.findElement(By.cssSelector(".info-account")).getText();
        Assert.assertEquals(exp_message, actual);
        driver.quit();
    }
}
```

We are almost ready to execute the feature file, but in order to execute the code on Firefox or Chrome we need to add a very small piece of code in the existing code. Follow the steps in the section below to understand the execution of the code on different browsers.



```
Finished after 16.963 seconds
Runs: 4/4    Errors: 0    Failures: 0

Runner.testrunner [Runner: JUnit 4] (4.809 s)
  Feature: Login feature to login in the site (4.809 s)
    Scenario: Login as a authenticated user (4.809 s)
      Given user is on homepage (2.015 s)
      When user navigates to Login Page (1.661 s)
      And user enters username and Password (0.346 s)
      Then success message is displayed (0.787 s)
```

```
Feature: Login feature to login in
  Verify if user is able to Login

  @testing
  Scenario: Login as a authenticat
    Given user is on homepage
    When user navigates to Login P
    And user enters username and P
    Then success message is displa
```

```
<terminated> testrunner (1) [JUnit] /Library/Java/
Jan 30, 2018 6:54:37 PM org.openqa.seler
INFO: Detected dialect: W3C

1 Scenarios (1 passed)
4 Steps (4 passed)
0m15.622s
```

Execution With Different Browsers
In Selenium 3, to execute the code on the Firefox or Chrome browsers, we need to use an external driver.
To execute the code on Firefox, we need GeckoDriver.
To use this, download Geckdriver.exe to your system, and in the test.java, before initiating the Firefox browser, set the system property as webdriver.gecko.driver.
System.setProperty("webdriver.gecko.driver","Mentionthepath\geckodriver.exe");

To use Chrome, we need to use ChromeDriver. As with Firefox, here also we need to set the system property as webdriver.chrome.driver.
System.setProperty("webdriver.chrome.driver","MentionPathtothisdriver/chromedriver");

We are all set to execute our first scenario for login functionality on Firefox/Chrome. Add the system property for the desired browser to the function, and then execute the test.feature file, and your browser will open and the login functionality test will be carried out.

Create HTML Reports In Cucumber

Imagine that you have to share the test reports with your client and senior management; in that case you will need a shareable HTML report which you can share after executing your tests.
You can achieve this by following some very simple steps.

Create An HTML Report By Adding A Plugin To Testrunner.Java Class

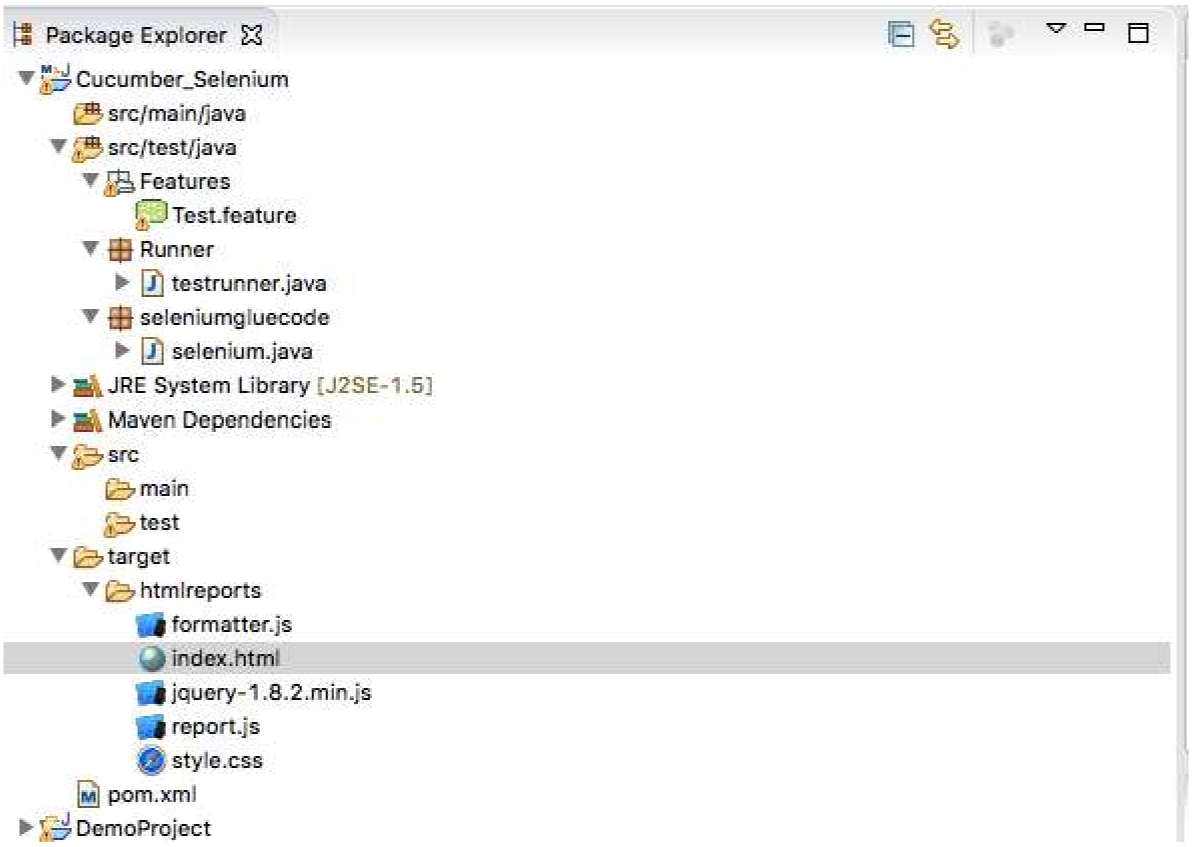
Step 1: In your testrunner.java class, add a plugin inside @CucumberOptions to format your test results into the HTML format.

plugin = { "pretty", "html:target/htmlreports" }

In order to set the path for the reports, we have to give a path in the project. To make this easier, the path is target/htmlreports.

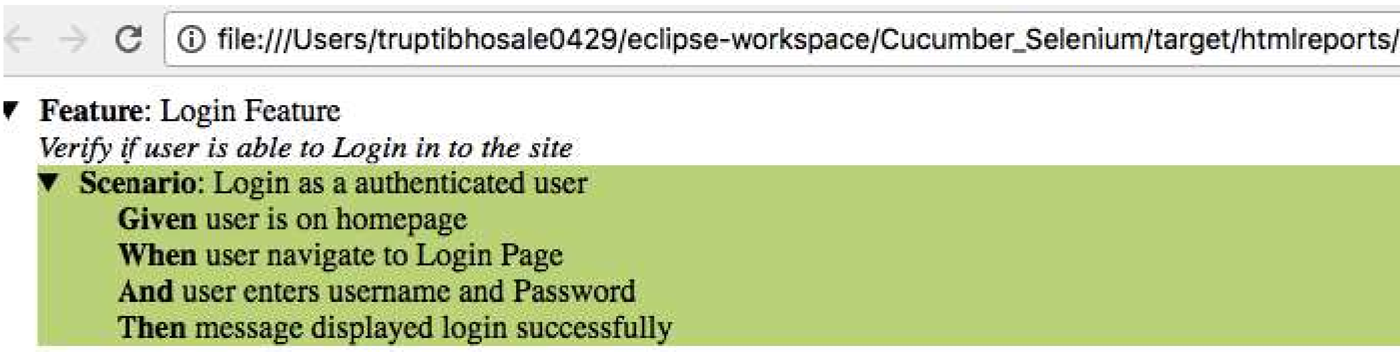
Step 2: Now save the testrunner.java class and execute it. On execution, you will see that the folder htmlreports is created inside the target folder.

Step 3: Access the folder and look for the index.html file; that is the file which contains the test results in HTML format.



```
Package Explorer
▼ Cucumber_Selenium
    src/main/java
  ▼ src/test/java
    ▼ Features
        Test.feature
    ▼ Runner
      ▶ testrunner.java
    ▼ seleniumgluecode
      ▶ selenium.java
  ▶ JRE System Library [J2SE-1.5]
  ▶ Maven Dependencies
  ▼ src
      main
      test
  ▼ target
    ▼ htmlreports
        formatter.js
        index.html
        jquery-1.8.2.min.js
        report.js
        style.css
    pom.xml
▶ DemoProject
```

Step 4: Open the index.html to view the report. The report created would be similar to the image below.


file:///Users/truptibhosale0429/eclipse-workspace/Cucumber_Selenium/target/htmlreports/

Feature: Login Feature
Verify if user is able to Login in to the site
Scenario: Login as a authenticated user
Given user is on homepage
When user navigate to Login Page
And user enters username and Password
Then message displayed login successfully

Command Line Execution
Executing your Cucumber tests from Eclipse is very easy, but you can also execute them through the command line. The steps to execute the tests through the command line are as follows:

1. Open the terminal in your system and navigate to your project directory.
2. Since we have already added a Maven dependency through pom.xml, we can execute the test using the simple command ***mvn test***.
3. In case you have a large number of feature files added to your project, and you only want to execute a smoketest.feature file, you can use the command ***mvn test -Dcucumber.options="src/test/java/Features/smoketest.feature***.