Introduction

Welcome to the exciting world of Test Automation with Selenium using Java 8 and Selenium WebDriver 3.13. Throughout this book, we will get up to speed with Selenium and it's surrounding technologies. Let's understand why test automation is needed. Todays agile world needs quick feedback on the code quality. The developers checkin application code. A test automation suite can be configured to run at a particular time in the day. A cut-off time should be provided to the developers say for eg. 6 pm in the evening. The automation suite may be scheduled to run at 7 pm daily. Jenkins is a tool that can be used for this purpose. Advantages of test automation include reducing the burden on the testers doing the manual execution so that they can focus on testing the functional aspects of the application. Generally a smoke, sanity, regression test suite is created for this purpose. The advantage of having the automatic triggering is to facilitate test execution in an unattended mode.

We will be using version 3.13 of Selenium which is the latest version of Selenium when this book is being written. It has developed a lot from early ancestor Selenium-1. Selenium RC, is another tool that lets you write automated web application UI tests in programming languages such as Java, C#, Python, Ruby etc against a HTTP website using any Javascript enabled browser. We will be working with Java 8 in this book and Selenium 3.13. Learning with Java can be fun and at the same time, it can be fast. Till Java 7, we had only object oriented features in Java. Java 8 has added many new features. Some of the features are listed below.

- Lambda expressions and functional interfaces

- default and static methods in interfaces

- forEach() method in Iterable interface

- Java Stream API for bulk data operations on Collections

Dont worry, we will not need all of these. Rather we would mostly focus on the object oriented features.

**What's new in Java-8**

***Lambda expressions and functional interfaces***

Lambda expressions enables functional programming. These donot belong to a class but are methods which exist in isolation. These functions can be treated as values.allow one to express functionality as a method argument or code as data. One particular scenario where lambda expressions can be used is while creating classes which comprise of just one method. Lambda expressions in this case help to replace Anonymous classes, which might prove cumbersome in such a situation. We will briefly see two examples side by side of how we can convert a conventional java snippet to lambda expression

In the code below, we assign a method to a variable called blockofCodeA

blockofCodeA = public void demo() {

System.out.println("Hello World");

}

blockofCodeA = () -> {

System.out.println("Hello World");

}

Remove the name, return type and the modifier and simply add the arrow after the brackets. This is your lambda expression. Functional interfaces are interfaces which contain only one abstract method. These interfaces are used along with lambda expressions. Say for example the method demo is inside an interface Greeting, then this interface should only have one abstract method which is the demo method.

public interface Greeting {

public void demo();

}

Default and Static methods in an Interface

Till Java 1.7, it was not possible to define a method inside an interface. Now 1.8 introduces the default methods through which we can provide implementation for a method inside the interface. Let's see an example below

Interface Phone

{

void dial();

default void text() {

System.out.println("Texting a message");

}

}

Static methods in java indicate that one can invoke the method without creating an object of a class, provided the static method is in a class. In Java 8, one can have static methods defined inside an interface as shown below

interface phone

{

inx x;

void changeRingtone();

static void text() {

System.out.println("Texting");

}

}

public class PhoneDemo

{

public static void main(String[] args) {

phone.text();

}

}

One can invoke the text method directly using the name of the interface.

forEach method for a collection

Starting java 1.8, we can invoke the forEach method on a collection and iterate through the contents of the collection. Lets compare the 1.7 and 1.8 versions of iterating over an array list of strings

List<String> fruits = Arrays.asList(“Apples”,”Oranges”,”Bananas”,”Pears”);

For (int I = 0;i<fruits.size();i++){

System.out.println(fruits.get(i));

}

A second alternative

For (String fruit : fruits ){

System.out.println(fruit);

}

In Java 8

fruits.forEach(I -> System.out.println(i));

Streams in Java 8

As per the Javadoc definition, “Streams are a sequence of elements supporting sequential and parallel aggregate operations”. Imagine a factory in which workers are standing with tools in their hands and machine parts keep moving around so that the individual worker can do his part.

List<String> fruits = Arrays.asList(“Apples”,”Oranges”,”Bananas”,”Pears”);

fruits.stream().forEach(fruit -> System.out.println(fruit));

Understanding Selenium RC

Selenium RC is popular UI automation library for automating browsers. Selenium RC uses generic JavaScript called Selenium Core to perform automation. Hovewer, this should comply with a security policy called Same-Origin policy. The Same-Origin policy is a security measure which prevents a websites scripts from accessing the scripts on other websites. For eg. Javascript present on google cannot access or communicate with javascript present on Yahoo. Three things are checked for the same origin policy protocol, domain and port. If these three things match then only the request can be said as one from the same domain.

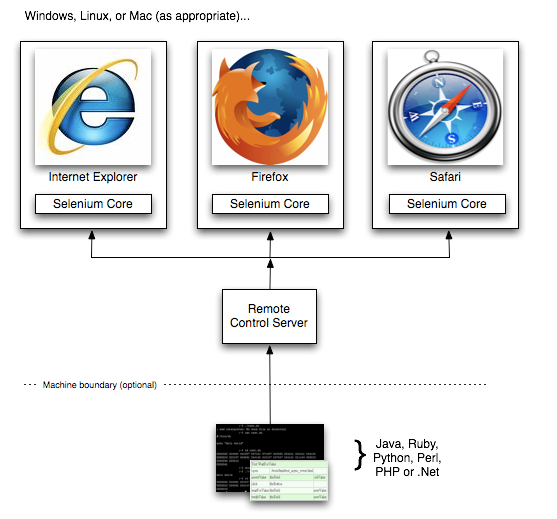
Selenium Core which was the first to be introduced for website automation by Jason Higgins, was a Javascript program. The problem with Selenium Core was that the testers had to install the WebServer containing the web application and Selenium Core in the browser. This method proved non feasible giving birth to Selenium RC.. Paul Hemmant, a Thoughtworks engineer decided to created a Server. This Server acts as an HTTP proxy that makes the browser to believe that Selenium Core and Web Application come from the same domain

What is Cross Site Scripting (XSS)?

Another concept related to same origin policy is Cross Site Scripting. Cross Site Scripting (XSS) is a computer security vulnerability found in Web Applications. XSS enables attackers to inject client side scripts into web pages viewed by other users. It may be used by attackers to bypass access controls such as Same Origin Policy. It is very important for a browser to restrict this security vulnerability. This piece of information is just there for one to know what cross site scripting is. Cross site scripting, as a whole, is a very wide topic and out of scope of this book.

Architecture of Selenium RC

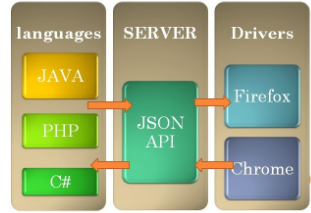
RC consists of two parts 1) Selenium Server 2) Client libraries



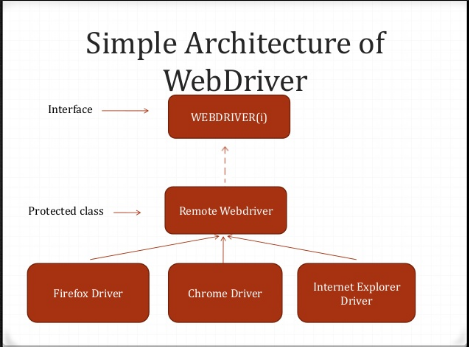
The role of Remote Control Server is to inject the Selenium Core in the respective browser. The client libraries send instructions in the form of request to RC Server and the RC Server communicates this to the browser. After receiving response, this is communicated back to the user by the RC Server.

Introducing Selenium WebDriver

Selenium WebDriver is used for automating Web browsers by using the browser’s internal plugins or dll with the individual browser drivers available for each individual browser.



Class Structure of Selenium WebDriver



What is the Remote Web Driver

Remote Webdriver is the implementation class of WebDriver interface. Apart from WebDriver, it also implements the interfaces viz. TakesScreenShot,findBy,JavaScriptExecutor etc.

Mobile Drivers

All modern webapps have implementations for mobile devices. The two popular operating systems in mobile devices are Android and IOS.Selenium has implementations for Android and IPhone viz. AndroidDriver and IODDriver. Both of these are direct implementations of WebDriver.

Headless Browsers

Headless browsers are those that do not have a Graphical User Interface. Everything runs in the background. When a test is executed with a headless browser, no screen gets displayed to the user. Two popular headless browsers are HTMLUnit and Phantom JS.

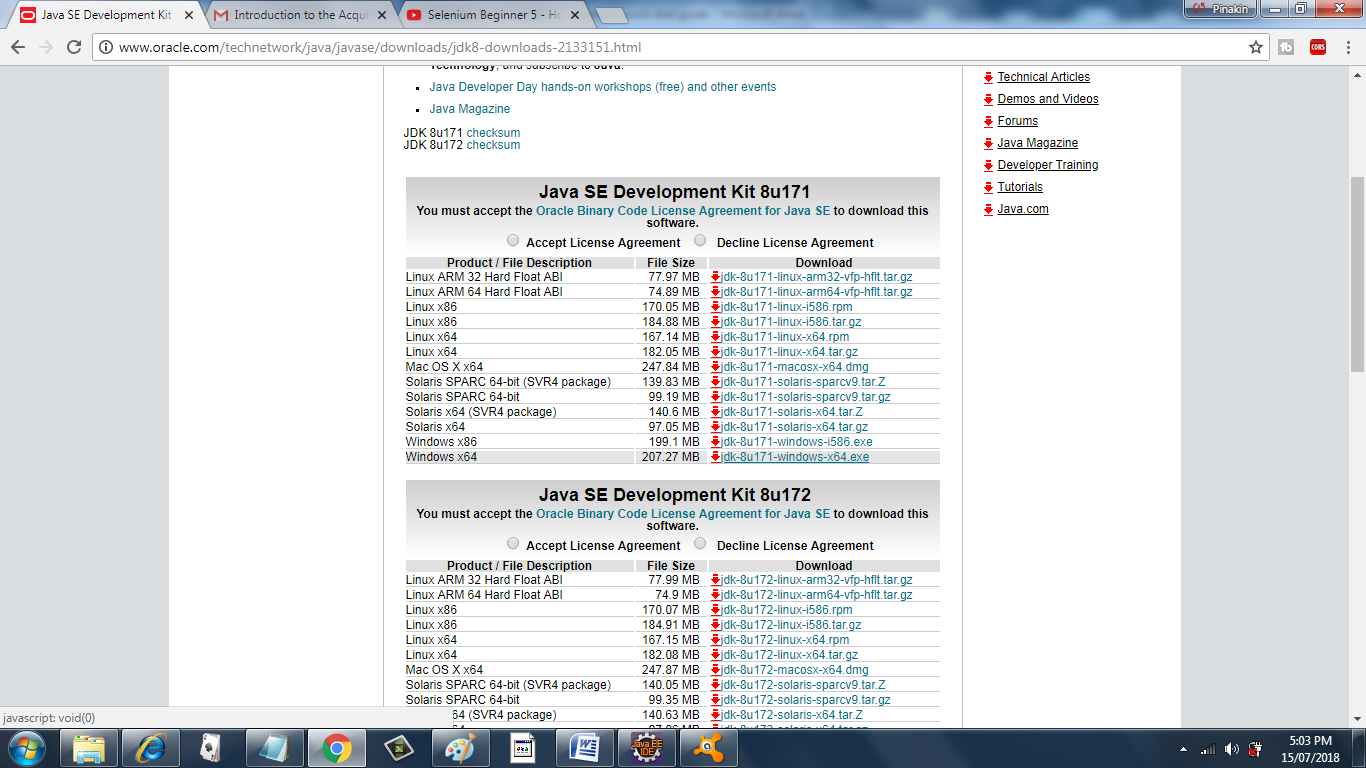
Why do we need them?

Suppose selenium tests have to be executed on a OS which does not have a Graphical User Interface(GUI) like Linux or when multiple browser behavior has to be simulated on just one machine. The advantage of headless browser is that the resources utilized by the test are minimal. A scenario where you can use these browsers is for test data creation. In these situations, there is no special need to display the screen to the user

Preparing for the very first script

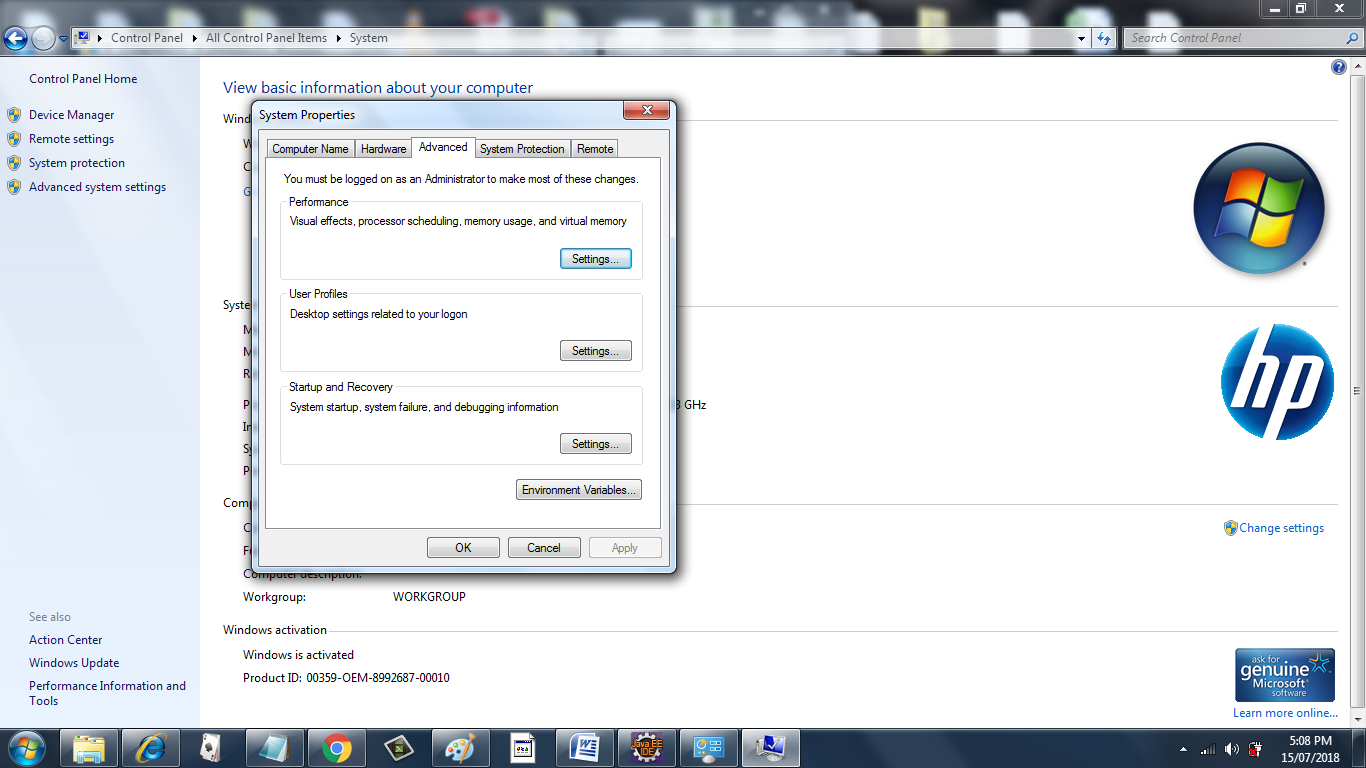
Install Java 8

Go to <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html> and click on the appropriate version. I have selected 64 bit Windows version since mine is a Windows machine



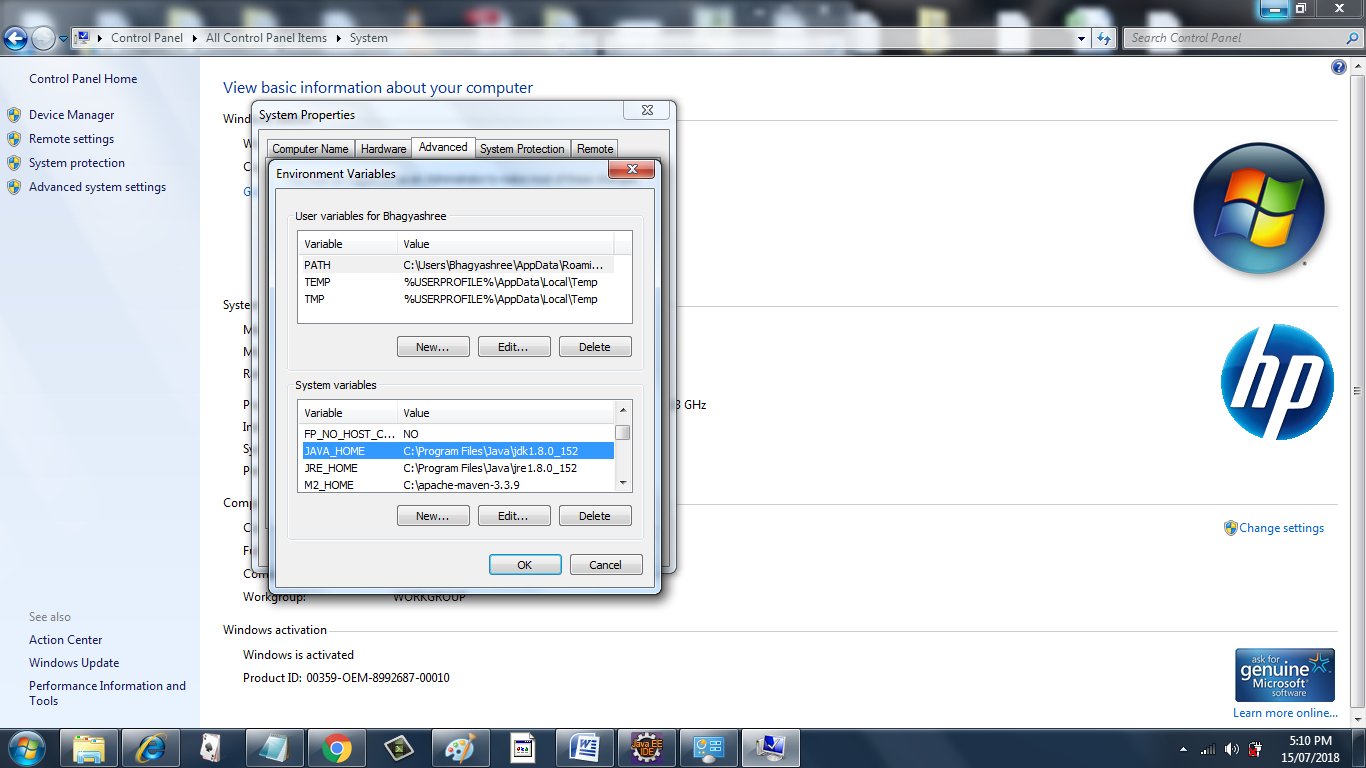
Once this file gets downloaded, run the .exe file. Java will start installing on your machine. Next we have to set two environment variables in order to use Java.

Go to Control Panel and click Advanced System Settings

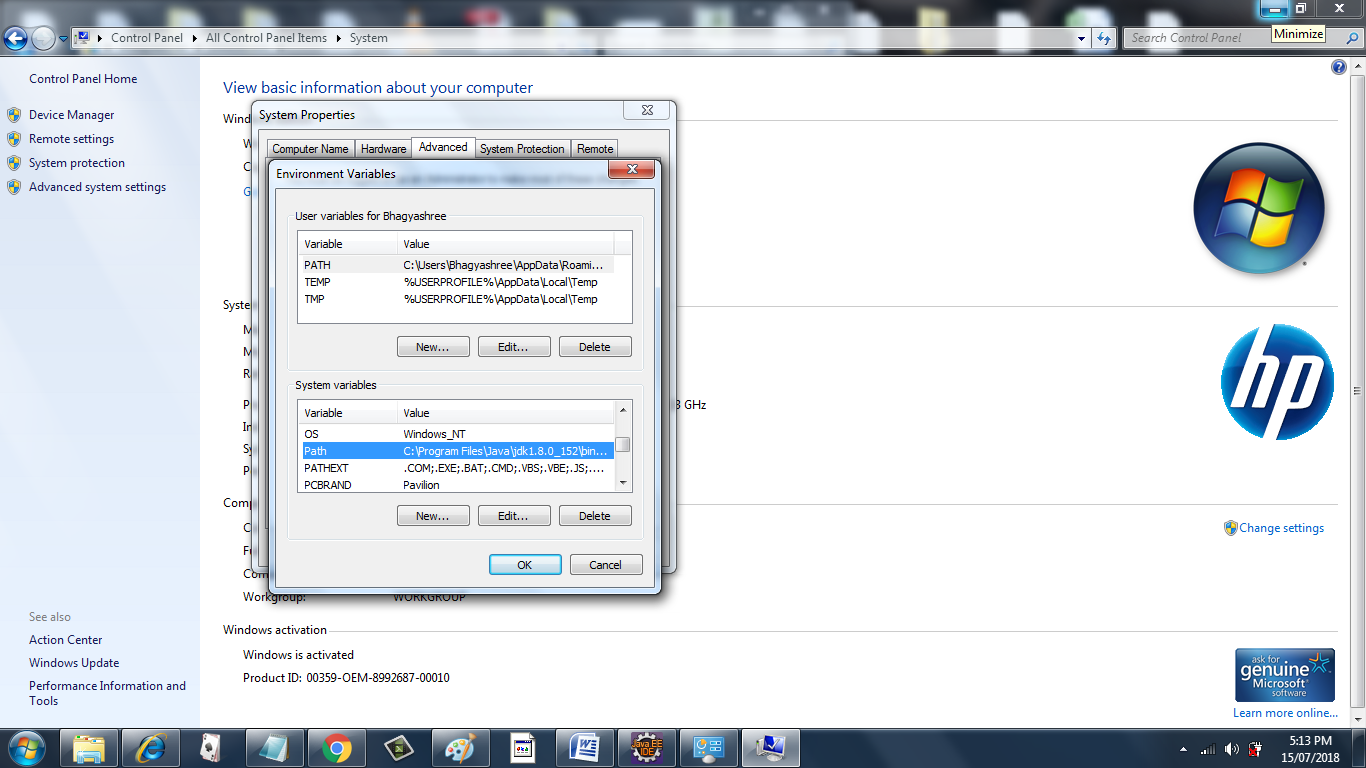


Click on Environment variables and add two System variables

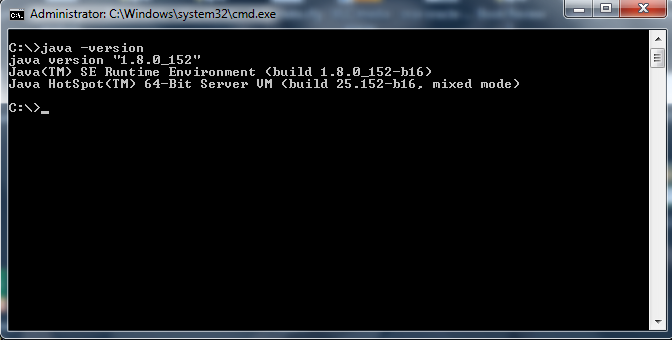
One is JAVA\_HOME. Give the path of root folder where Java is installed. In this case C:\Program Files\Java\jdk1.8.0\_152



And second is the Path variable. Remember this variable has to be appended after adding a ‘;’. Here the path of the bin folder has to be specified. In this case C:\Program Files\Java\jdk1.8.0\_152\bin



Next step is to check our configuration. Open the command prompt and type java –version



If you get the output similar to the one shown above, you are all set to start coding.

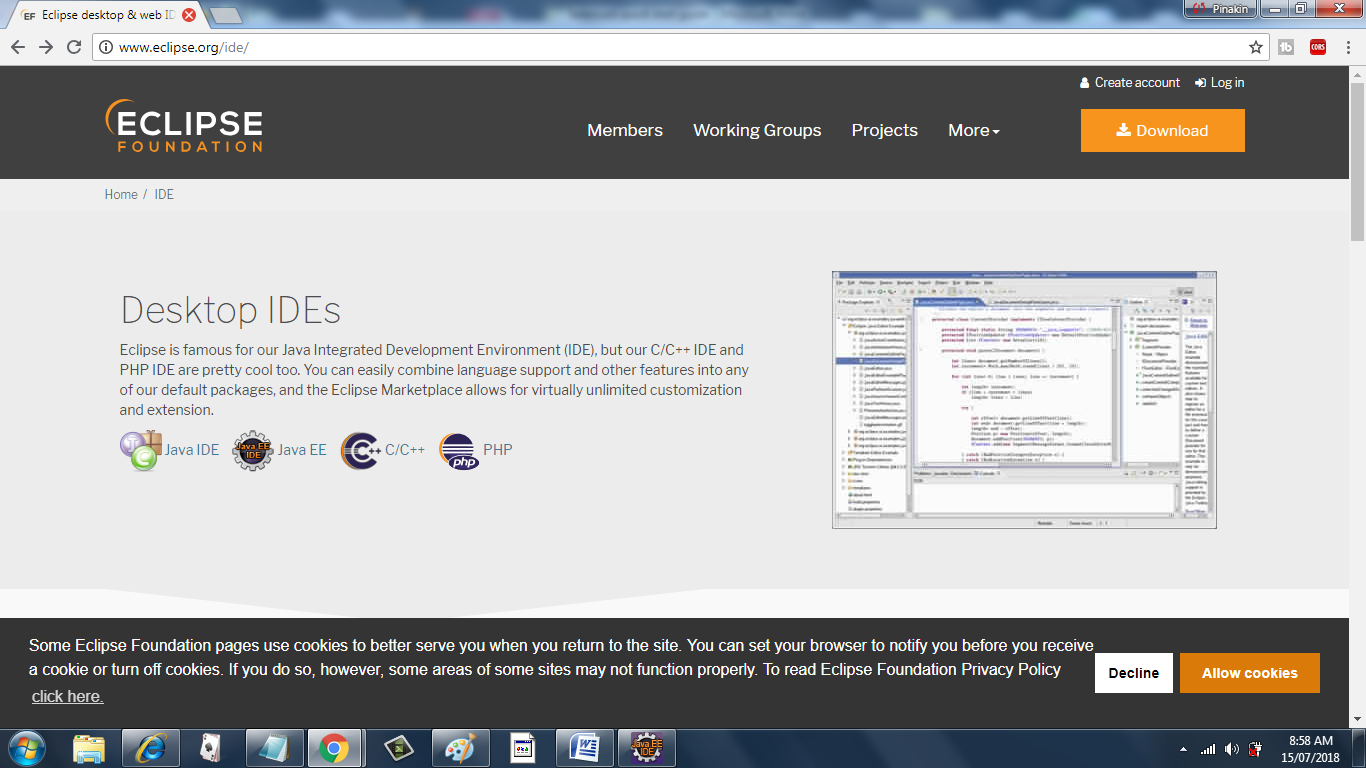
Let’s get our hands dirty

Setting Eclipse

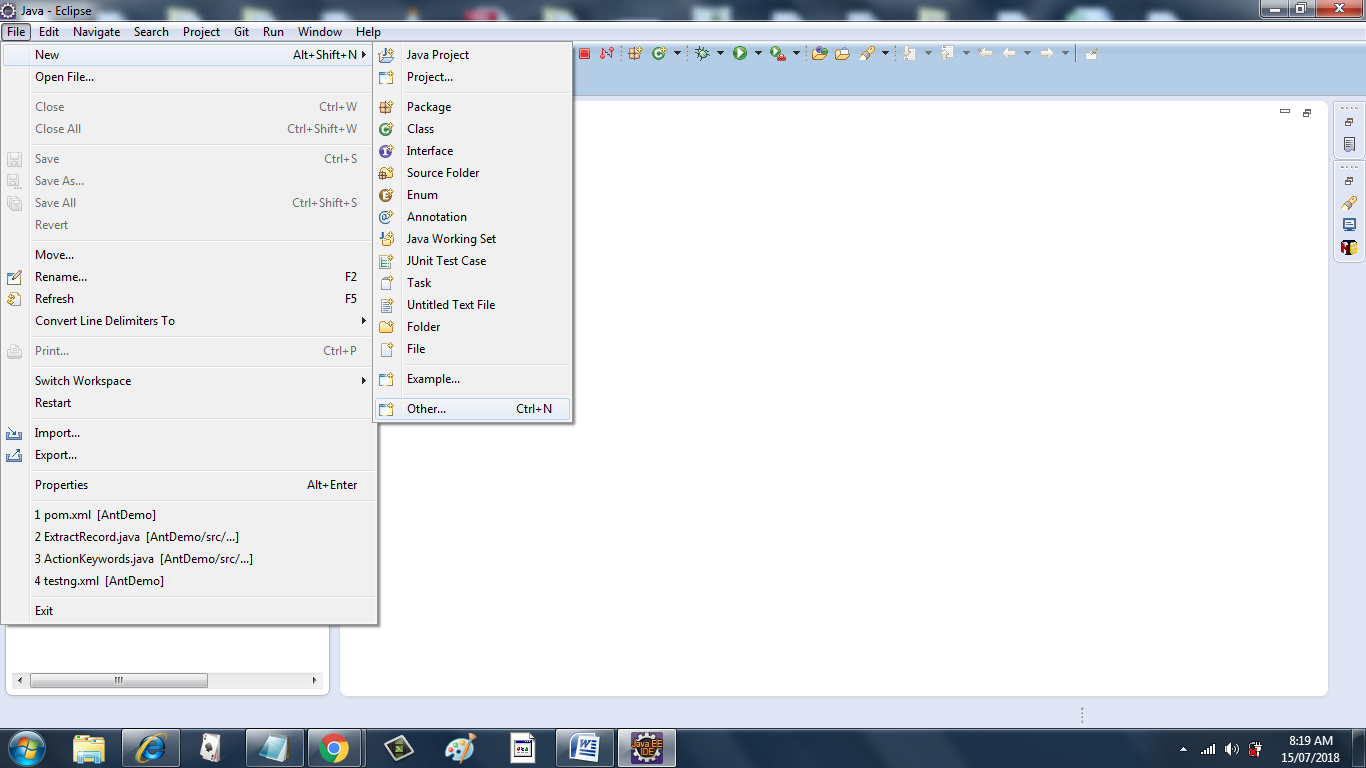
We will be using Eclipse as a IDE for developing Selenium Scripts in this book but you are free to use whichever IDE suits you.

Download Eclipse

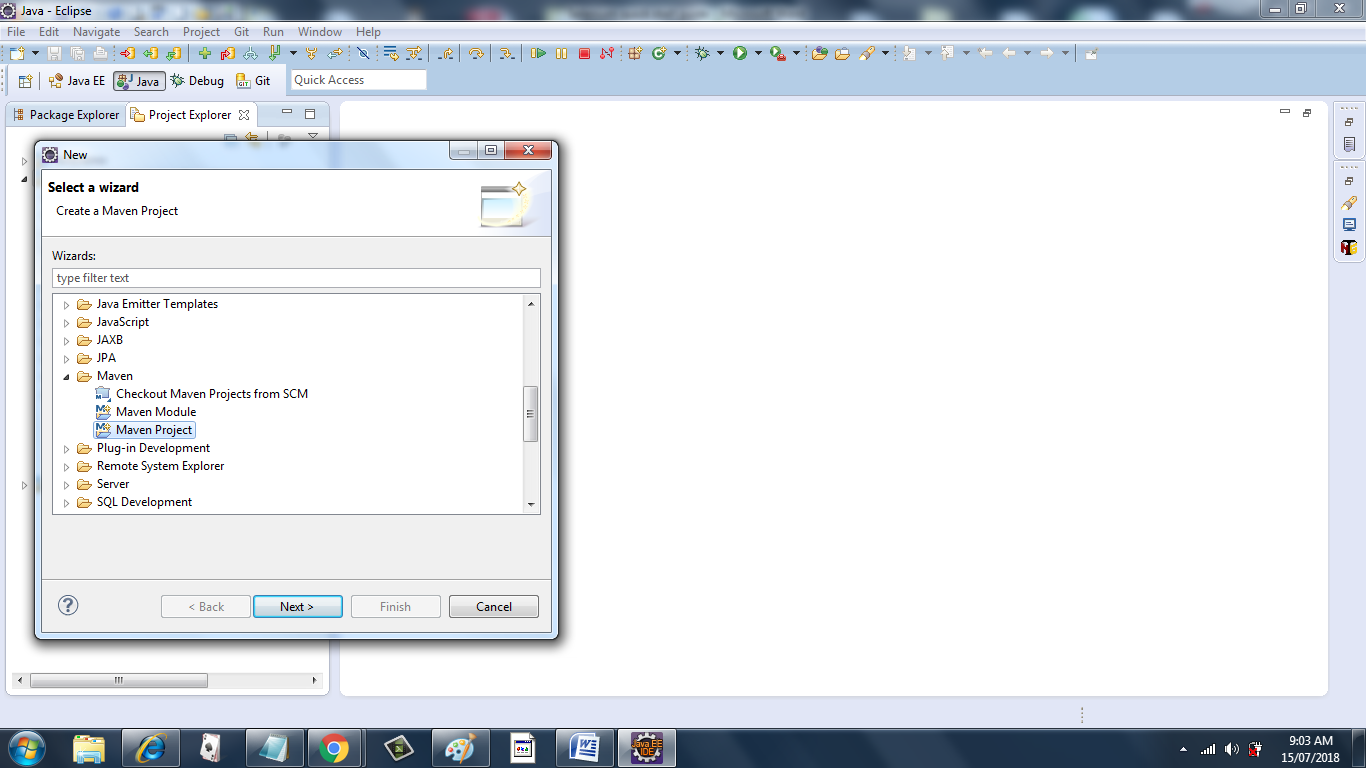
Navigate to the website [www.eclipse.org/ide](http://www.eclipse.org/ide) and click on the Download link. Here you can find very specific instructions about how to install you favourite ide version (Kepler, Neon etc)



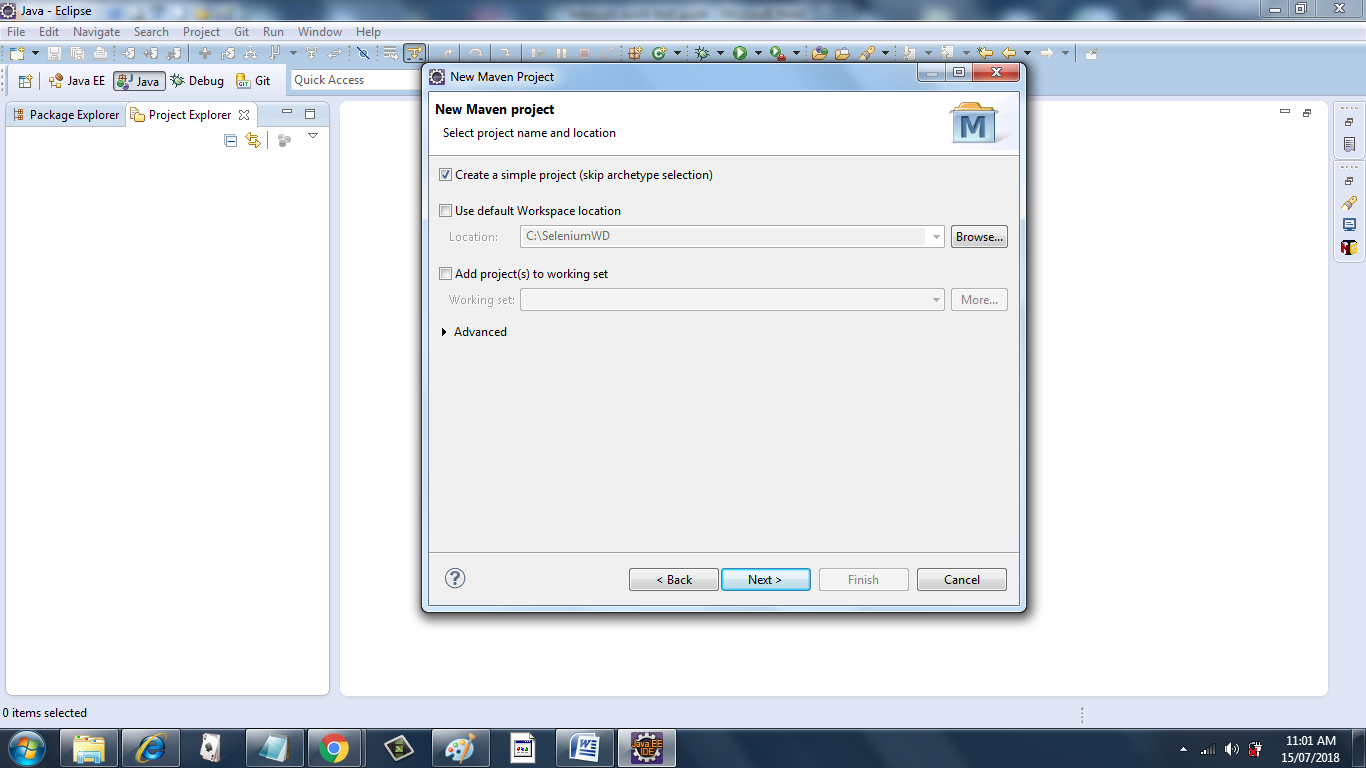
Once the IDE gets installed, double click on the .exe file for Eclipse and go to File->New->Other



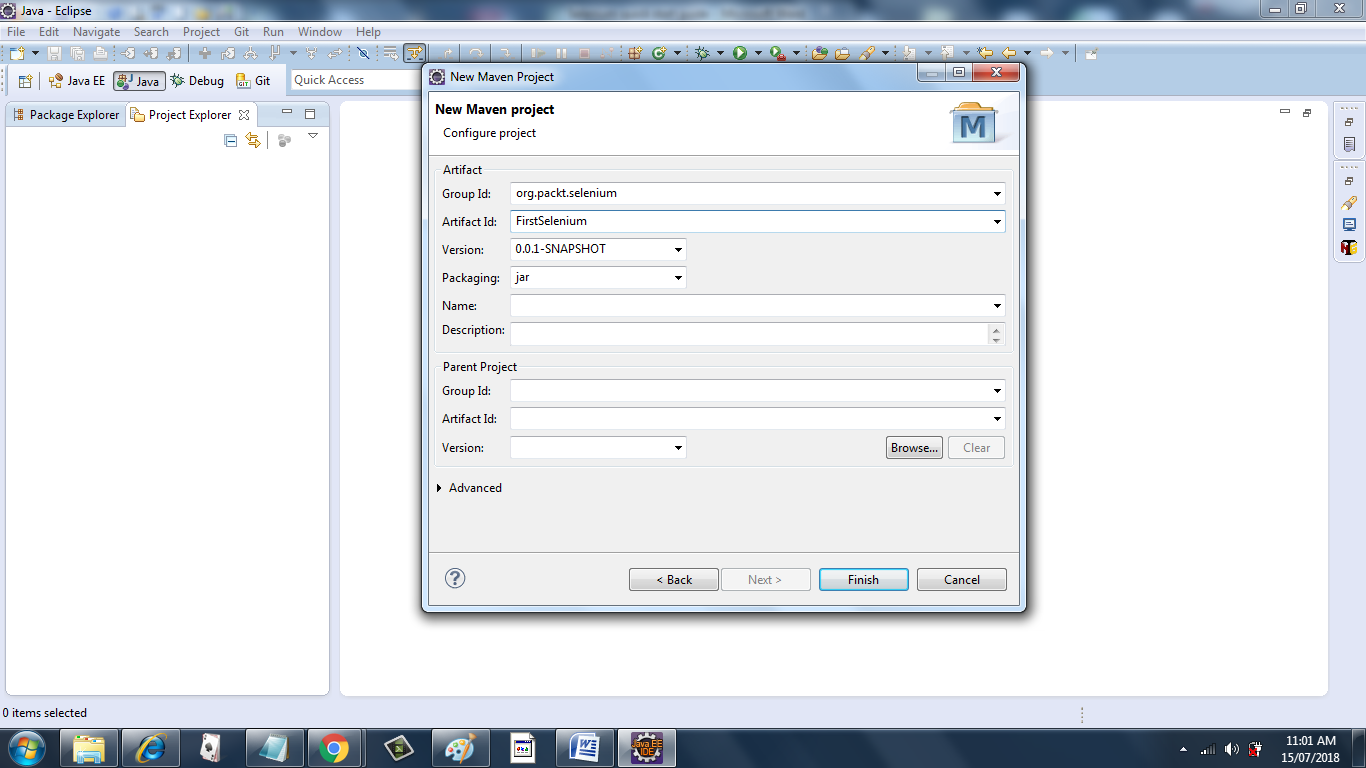
Select Maven Project. Click Next



Click Next

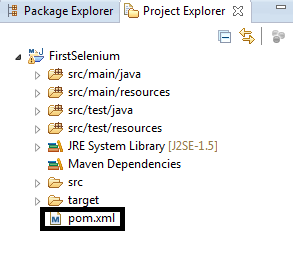


Select the ‘Create a simple project(skip archetype selection)’. Click Next.



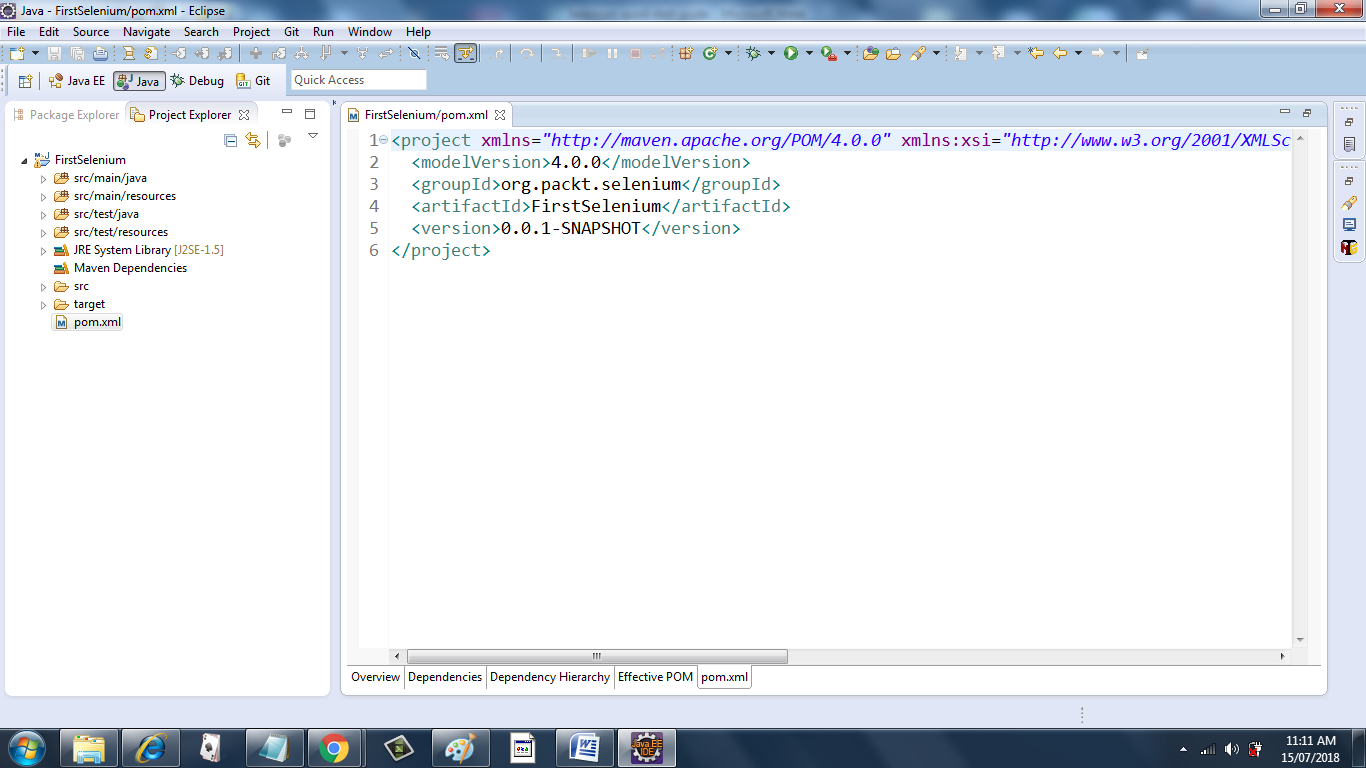
Input the Group ID. This ideally is the package name of the project. The Artifact id corresponds to the name of the JAR file in case you want to create one. Keep the packaging as JAR. Notice the version is 0.0.1-SNAPSHOT. The ‘SNAPSHOT’ part indicates that the project is still under development and has not been released.

Click Finish



This is a snapshot of the Project Explorer. When you create a Maven Project, the following folders get created for you. src/main/java,src/main/resources,src/test/java,src/test/resources. Apart from these, you see a ‘Maven Dependencies folder that is currently empty. Marked with a black box, there is an xml file called ‘pom.xml’. This is the place where you place all the dependencies for your project. By dependencies, I mean dependent JARs. JAR stands for Java Archive.

It’s time to explore pom.xml



This is what pom.xml looks like. The Group ID and Artifact ID that you added in the previous screens has appeared in the file above inside the ‘Project’ tag. In order to work with Selenium, we will need to add Selenium dependencies within the Project tag. Let’s go ahead and add those from the Maven repository

Just go the Maven Repository <https://mvnrepository.com> and grab the dependency shown below

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>3.13.0</version>

</dependency>

Place this dependency inside a dependencies tag as shown in the pom.xml below

<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>org.packt.selenium</groupId>

<artifactId>FirstSelenium</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

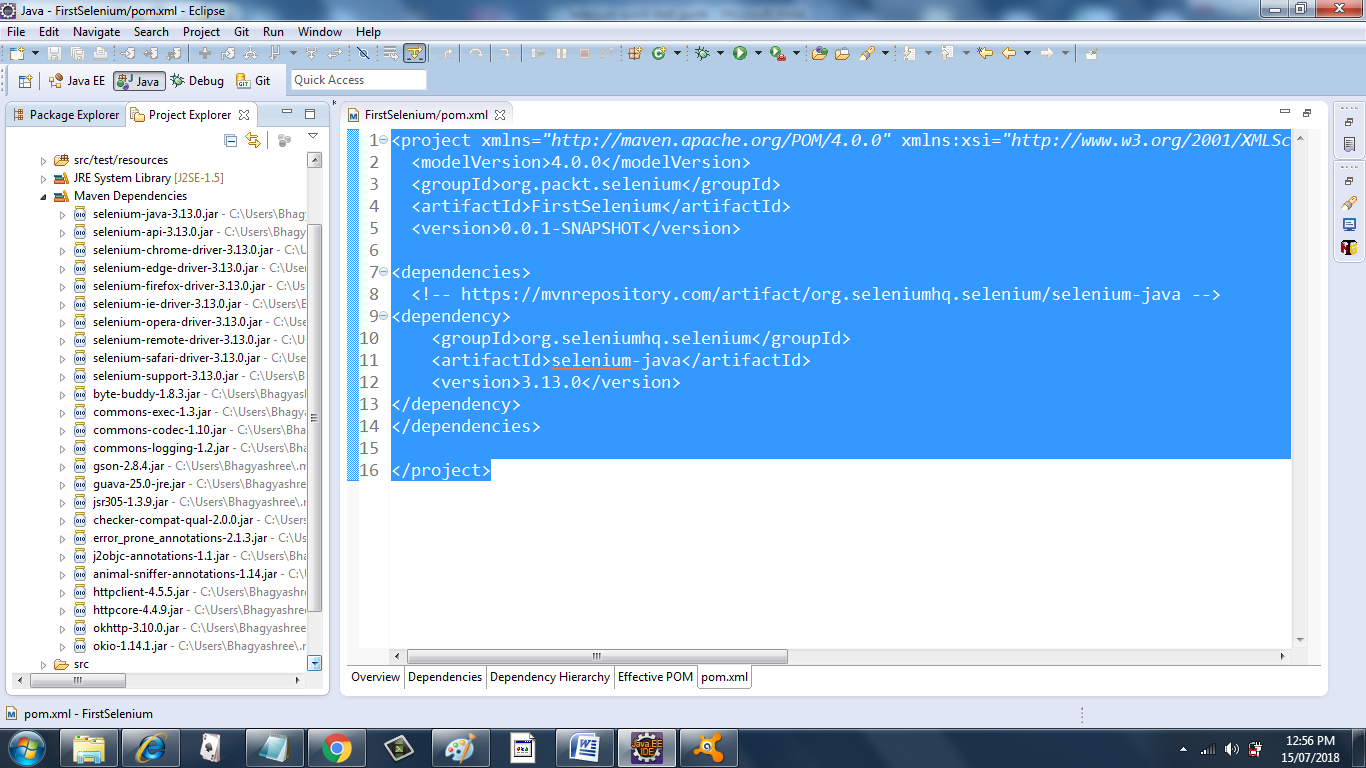
<version>3.13.0</version>

</dependency>

</dependencies>

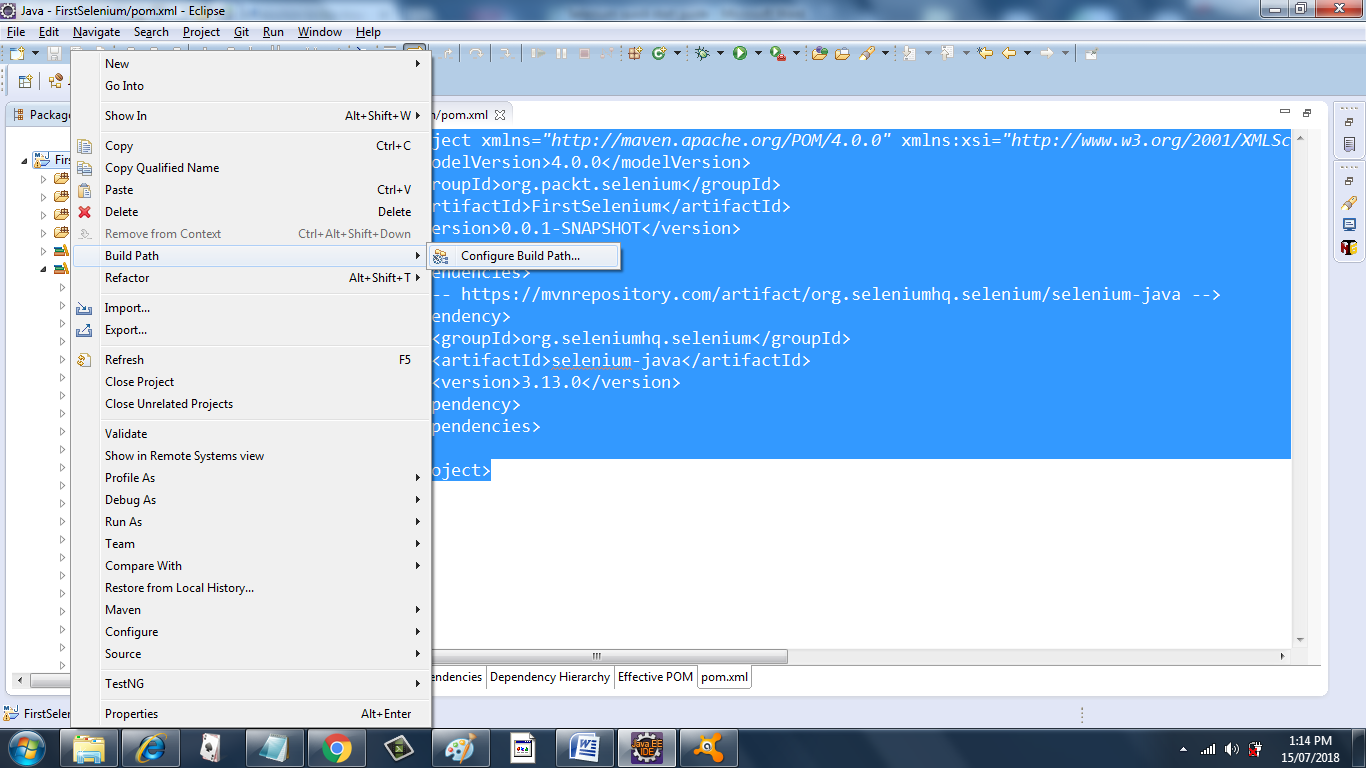
</project>

Save the pom.xml. You will see a small activity on the bottom right corner of Eclipse stating that the project is being built.

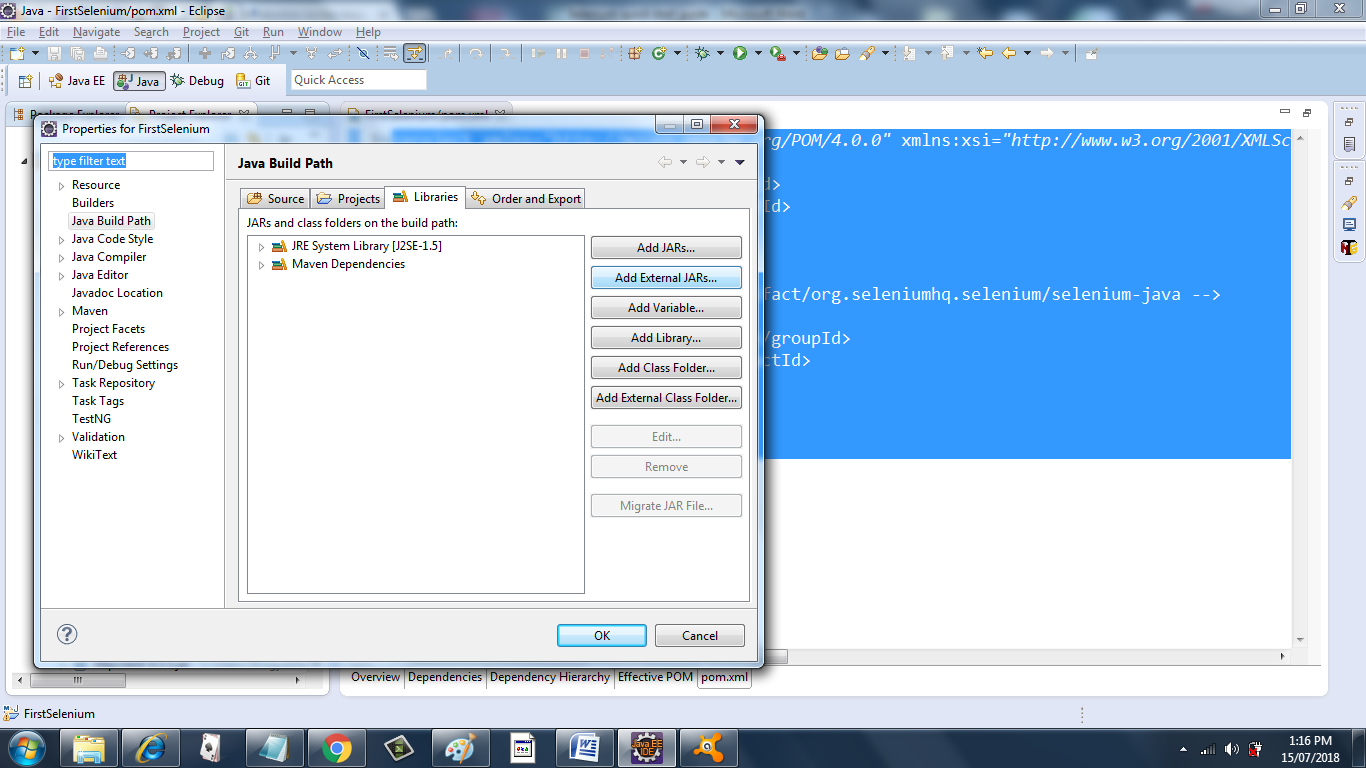


The Maven Dependencies folder now gets populated with all the downloaded Jars as shown above.

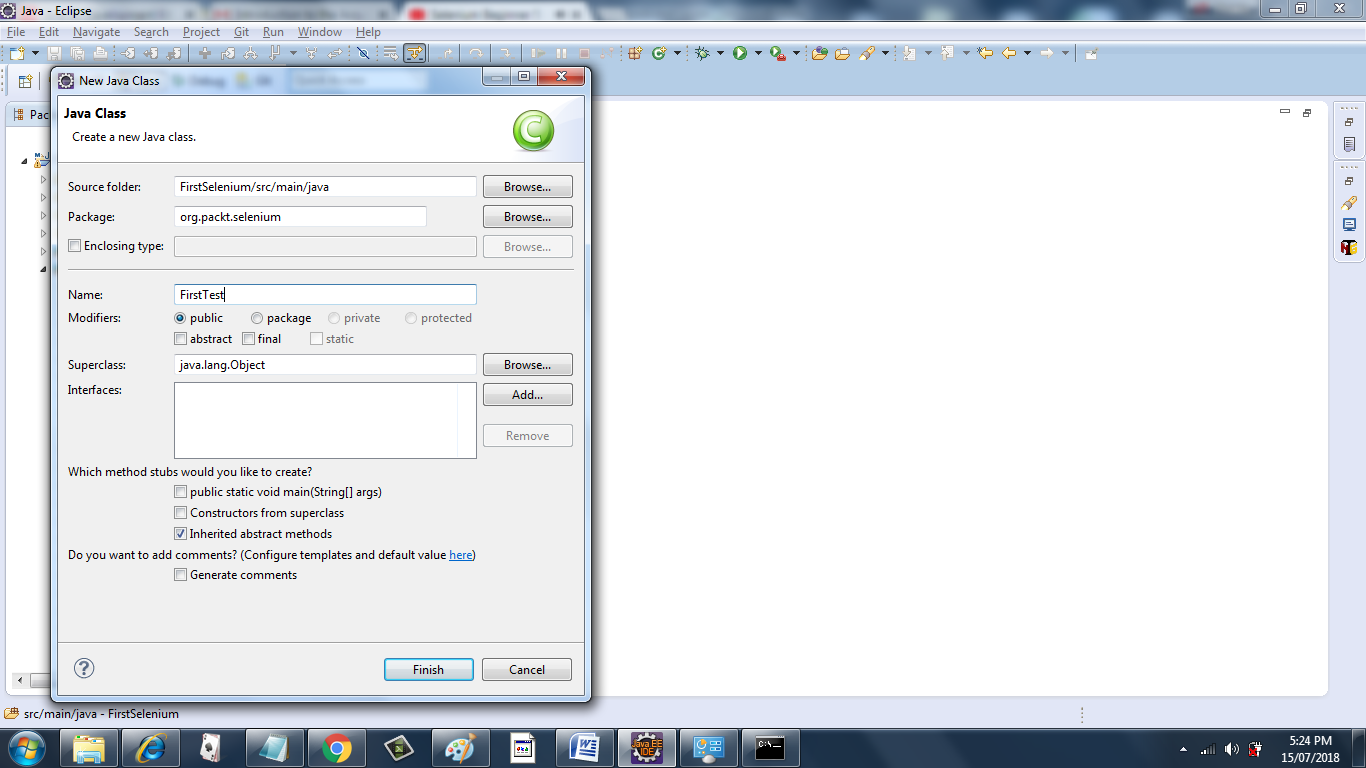
With this we are ready with the basic Eclipse setup for Selenium WebDriver. But we are not done yet. It might happen that under a corporate firewall, you are unable to download the required JARS. In this situation, simply create a plain Java project. Right click on the project in project explorer



Select Build Path -> Configure Build Path



Click on Add External JARs and add the required Jars manually.

Next we will write a very simple script which just opens [www.google.com](http://www.google.com). Right click the Project and select new class

Type the following code

**public** **class** FirstTest {

**public** **static** **void** main(String[] args) {

System.*setProperty*("webdriver.chrome.driver",

"C:\\SeleniumWD\\src\\main\\resources\\chromedriver.exe");

WebDriver driver = **new** ChromeDriver();

driver.get("http://www.google.com");

}

}

Right click the file and click ‘Run as Java Application’ and hurrah!! A chrome browser opens and [www.google.com](http://www.google.com) gets loaded.

You have successfully created your first Selenium Script