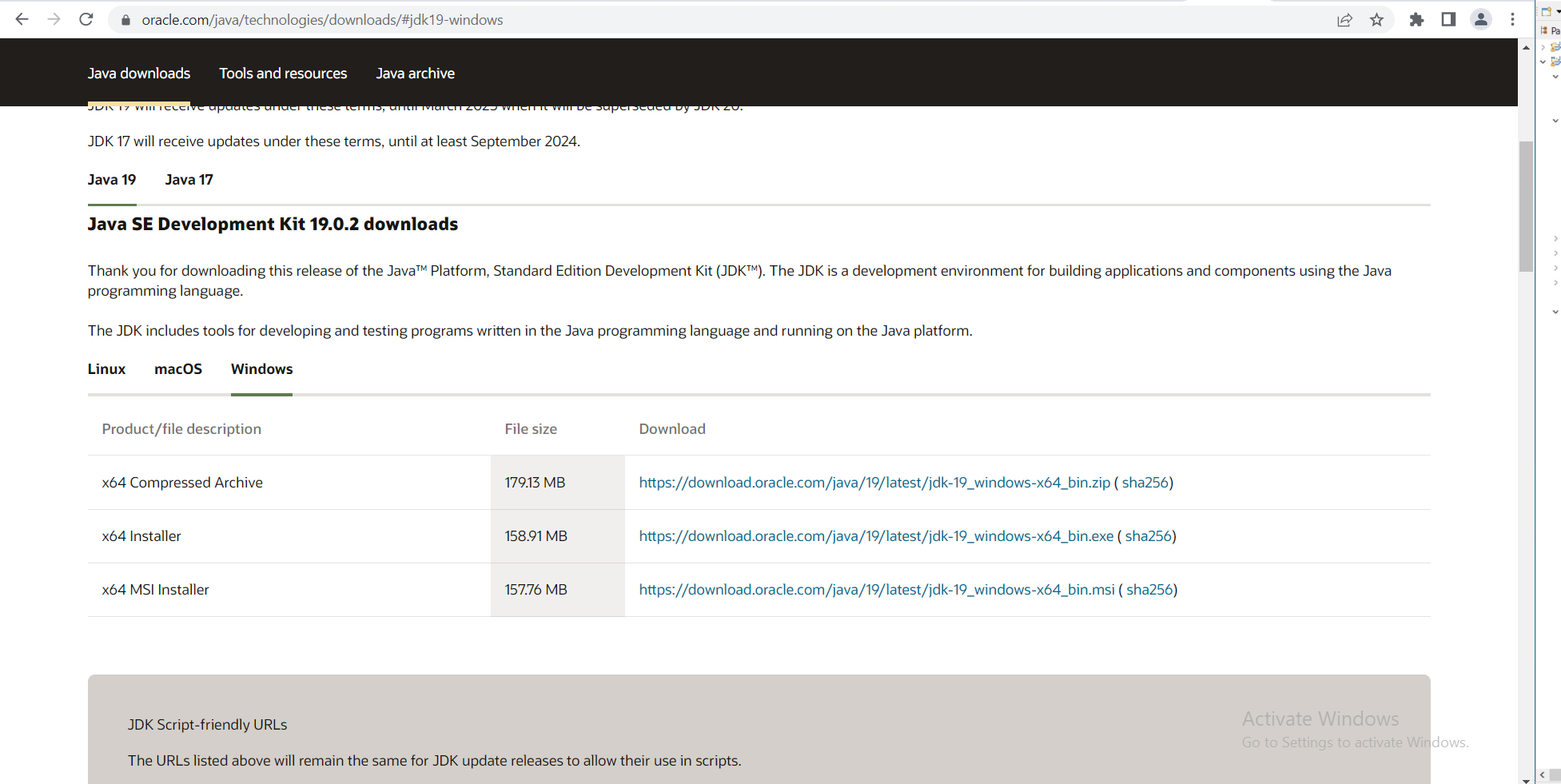
**Configuring Selenium with Eclipse IDE**

**Pre-Requisites:**

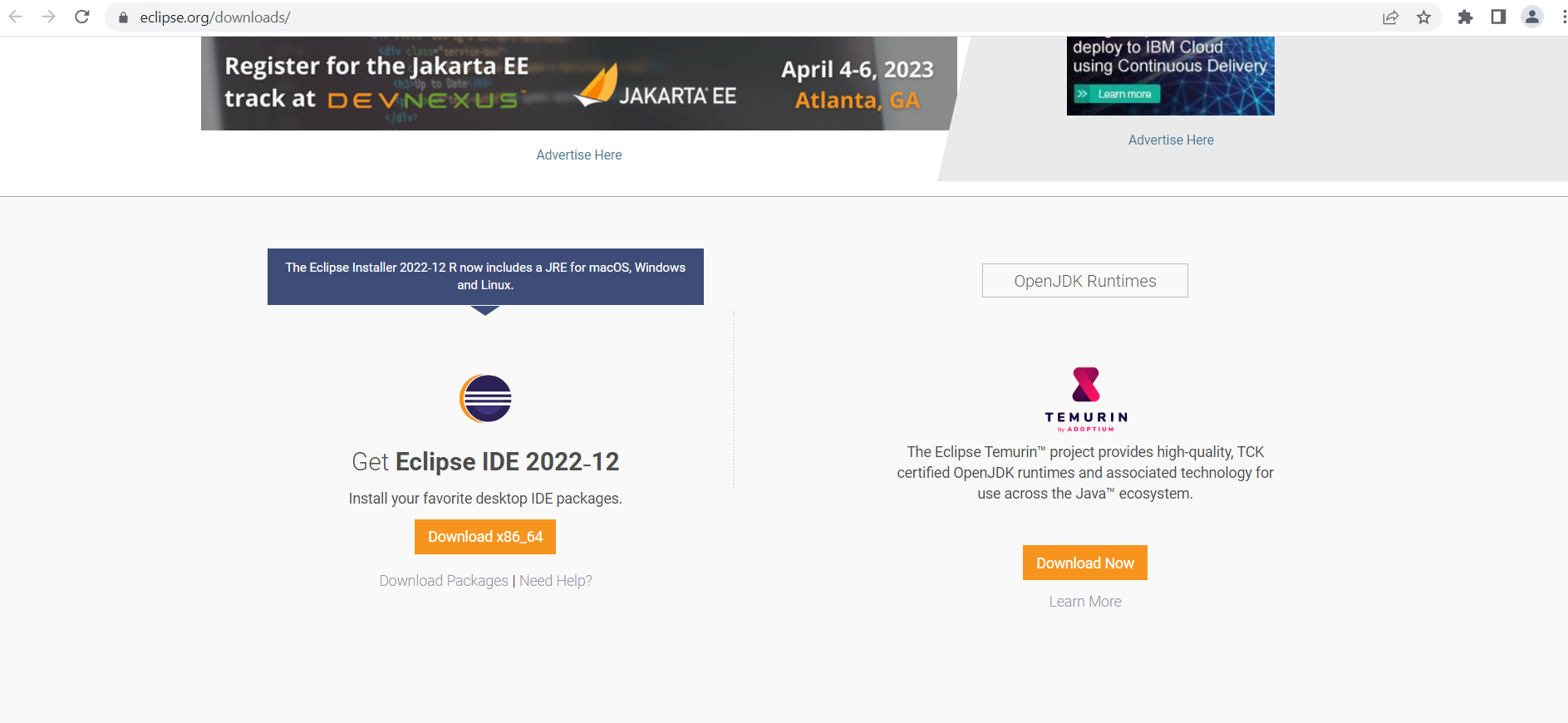
* At first, download Java SE Development Kit 19.0.2 according to Window. Access the following website to start proceedings;

<https://www.oracle.com/java/technologies/downloads/#jdk19-windows>



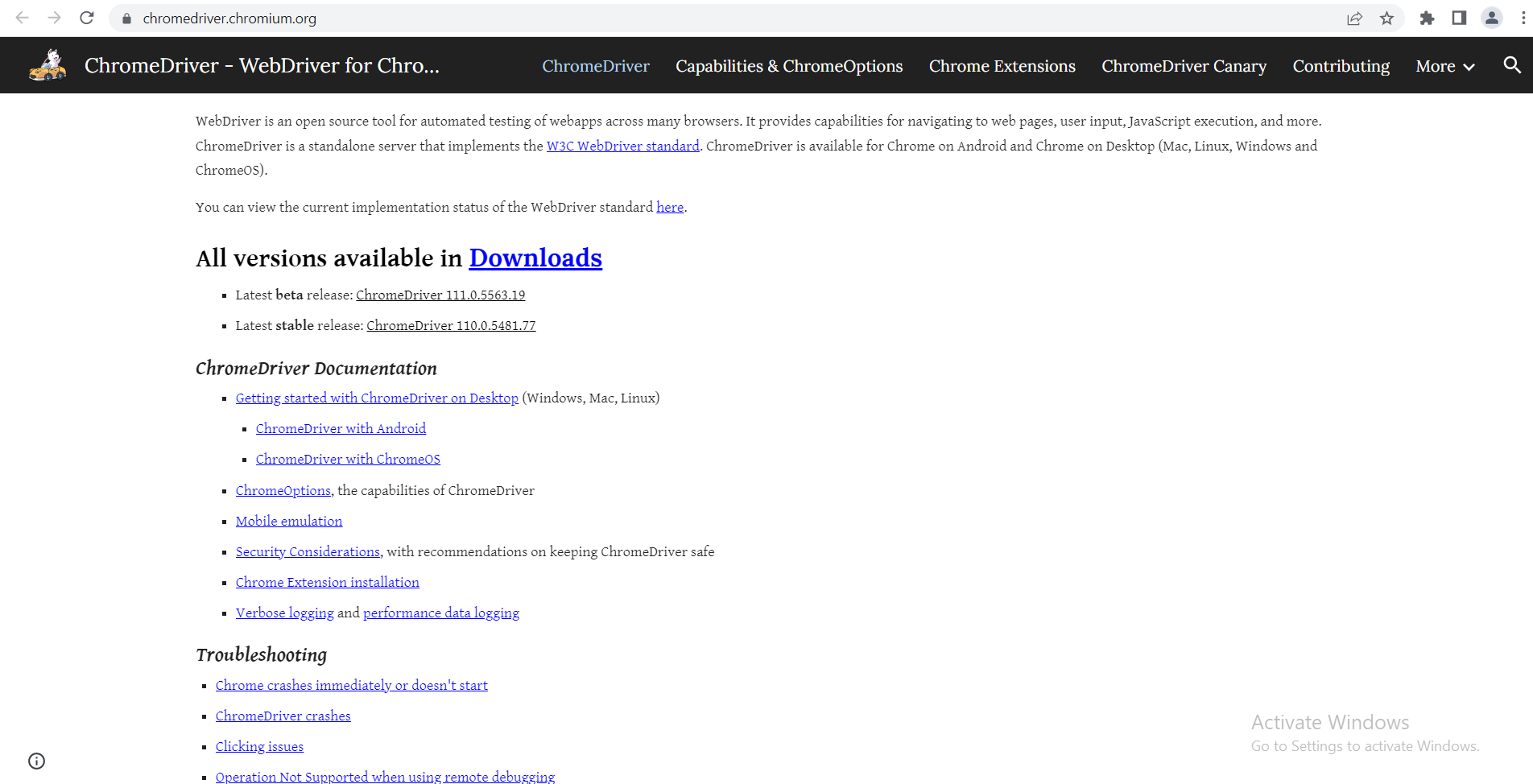
* Install jdk.exe file by double-clicking on the file name in the download location.
* Download Eclipse IDE from the following website;

<https://www.eclipse.org/downloads/>



* Install it in the desired location of your machine.
* Download the relevant Browser Driver – Chrome Driver (for Chrome). Access the following website to get this done.

<https://chromedriver.chromium.org/>



* Download the Latest stable release: ChromeDriver 110.0.5481.77.
* Unzip the downloaded file named chromedriver\_win32. Copy chromedriver.exe file and paste it in the desired location.

**How to Configure Selenium with Eclipse IDE:**

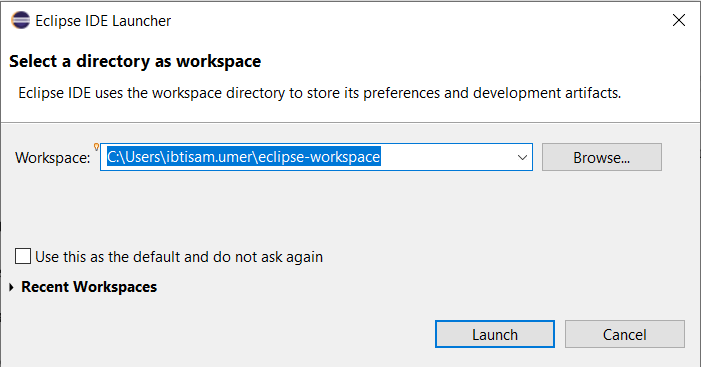
Here are the steps to configure Selenium with Eclipse:

**Step 1: Launch Eclipse**

To launch Eclipse double click on the eclipse.exe file in the download location.

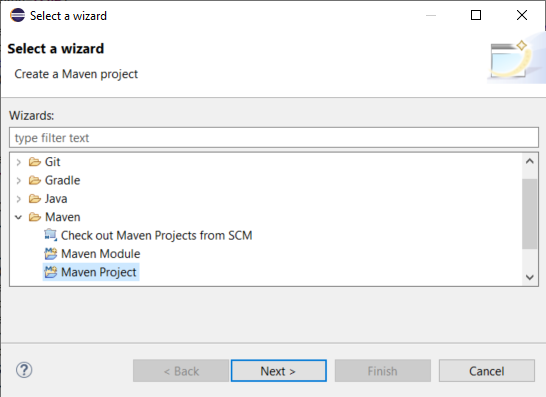
**Step 2: Create Workspace in Eclipse**

This workspace named “**C:\Users\ibtisam.umer\eclipse-workspace**” (this varies from machine to machine) is like any other folder, which will store all the test scripts. Launch this workspace.

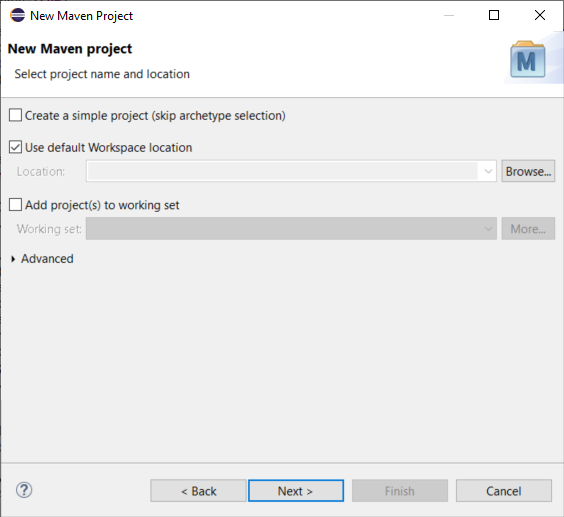


**Step 3: Create New Maven Project**

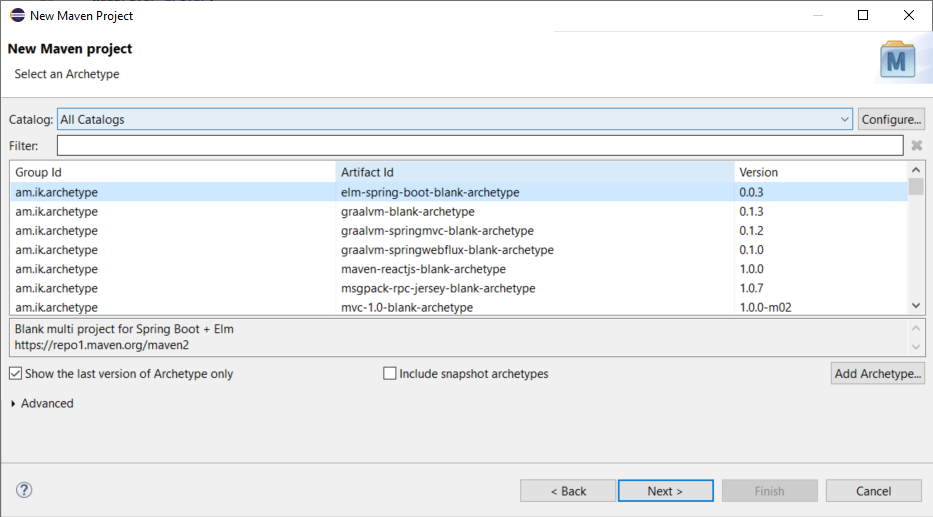
Create a new Maven Project by clicking on **File > New > Other > Maven > Maven Project > Next.**



Click Next again.



Select an Archetype and click Next.



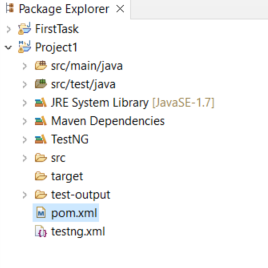
Add Group Id, Artifact Id and click Finish.



A new Maven Project is created.

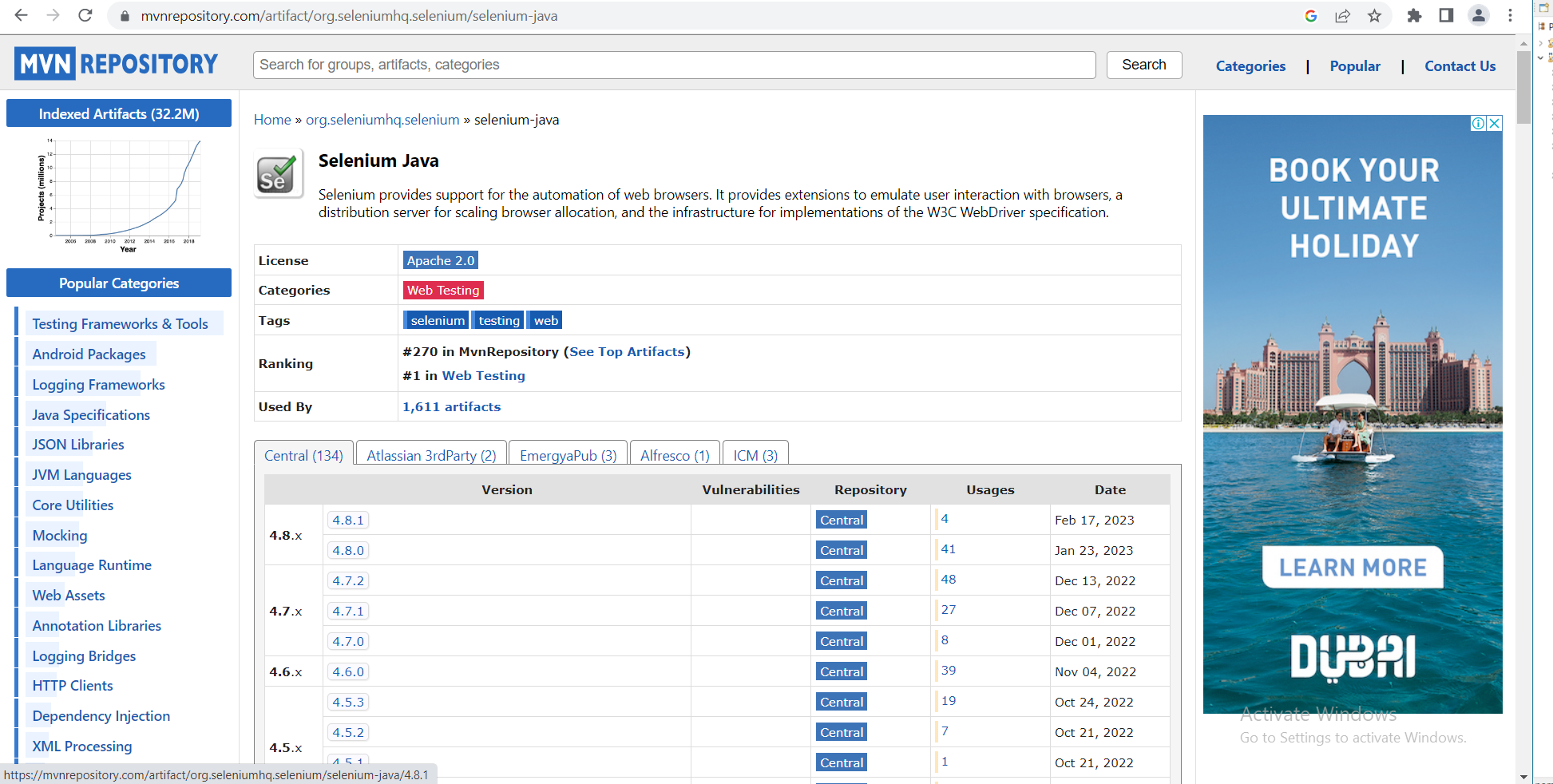
**Step 4: Add Selenium dependencies in pom.xml of Maven File**

**In package explorer, there is a file named pom.xml.**

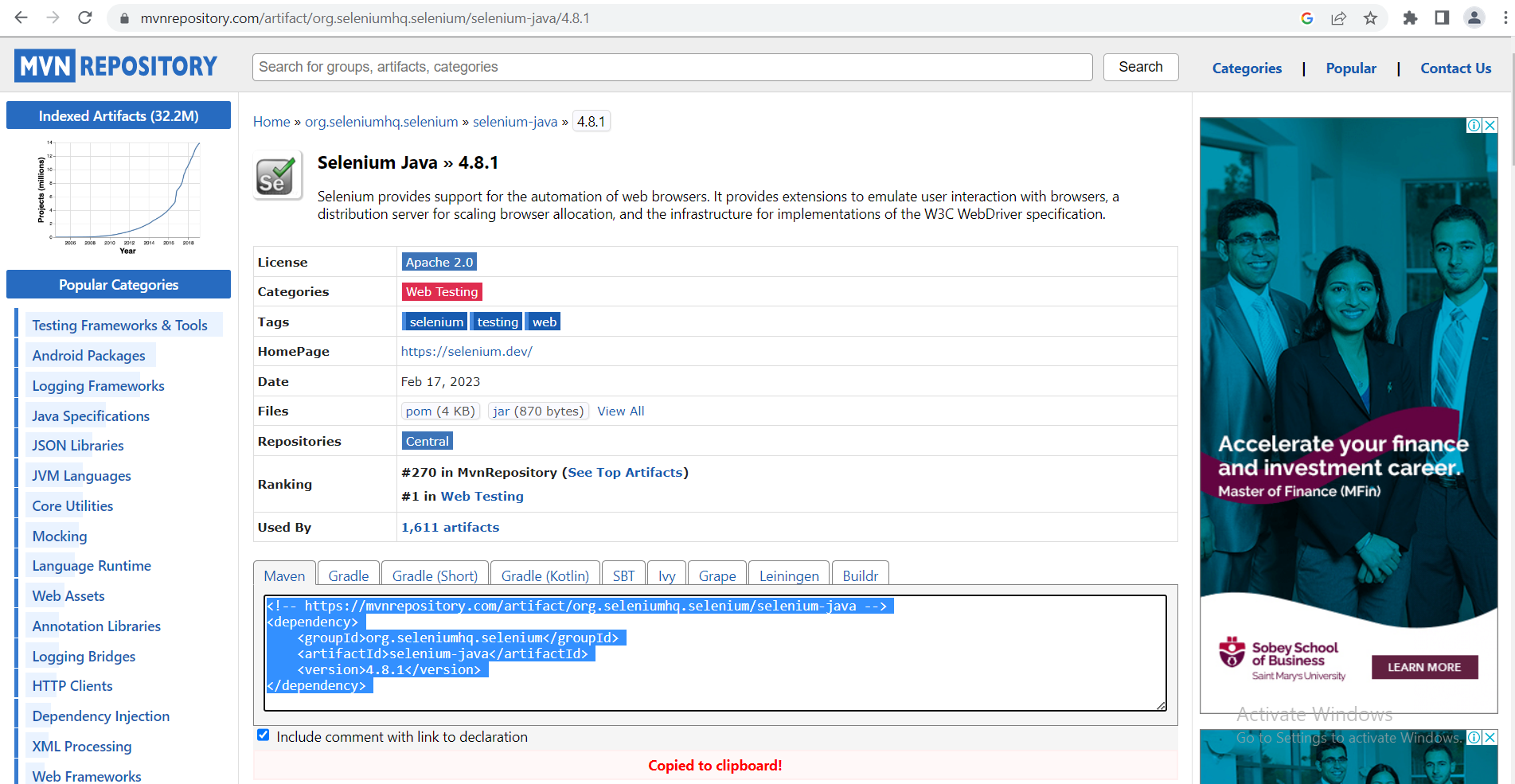


**To add selenium webdriver’s dependencies in pom.xml, access the following website.**

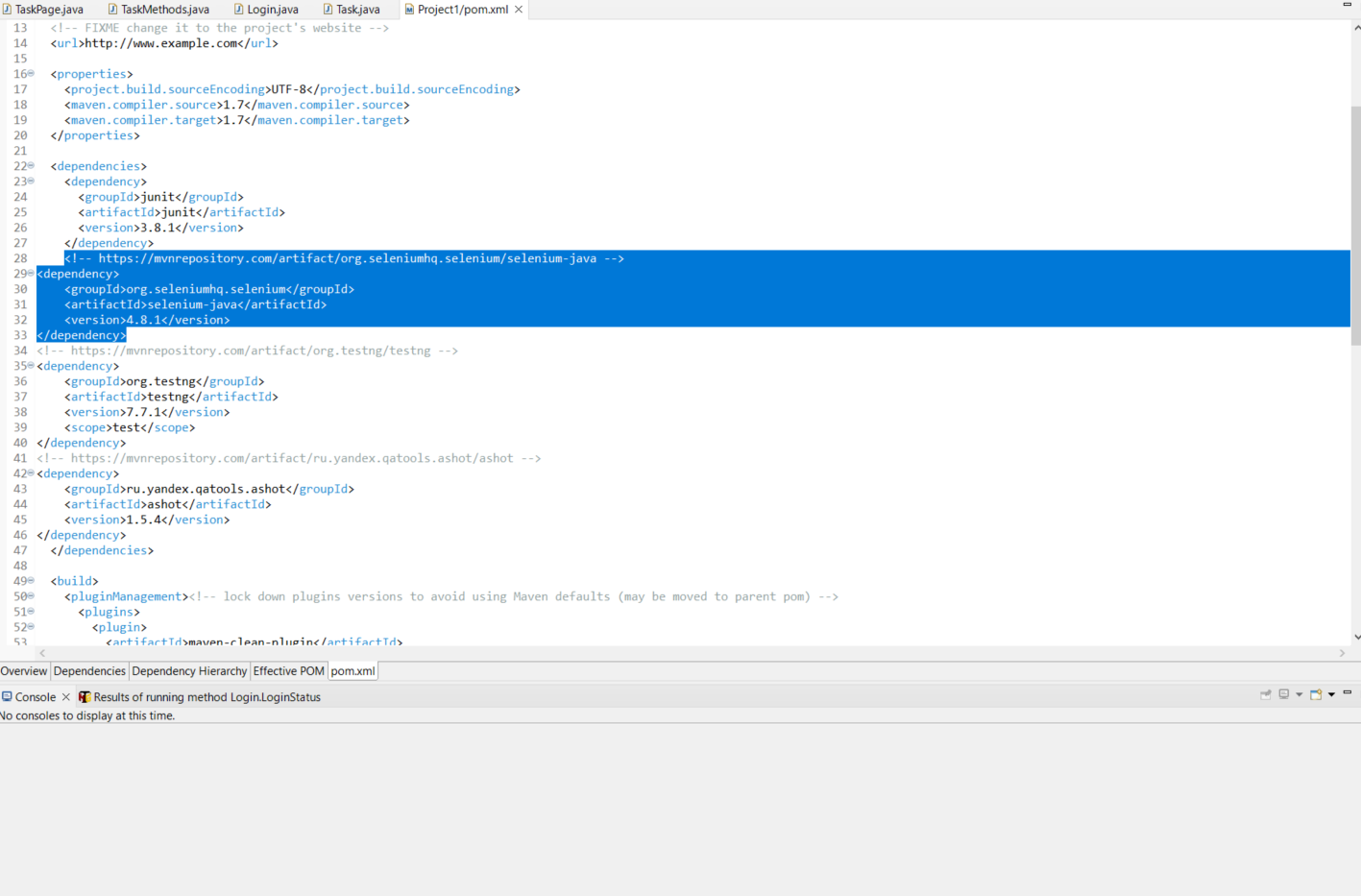
<https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java>



**Click the latest version 4.8.1 and copy the dependency text from there.**



**Now open pom.xml file from the package explorer and paste the copied dependencies there.**



Save the file to make sure that selenium is configured with Eclipse IDE.

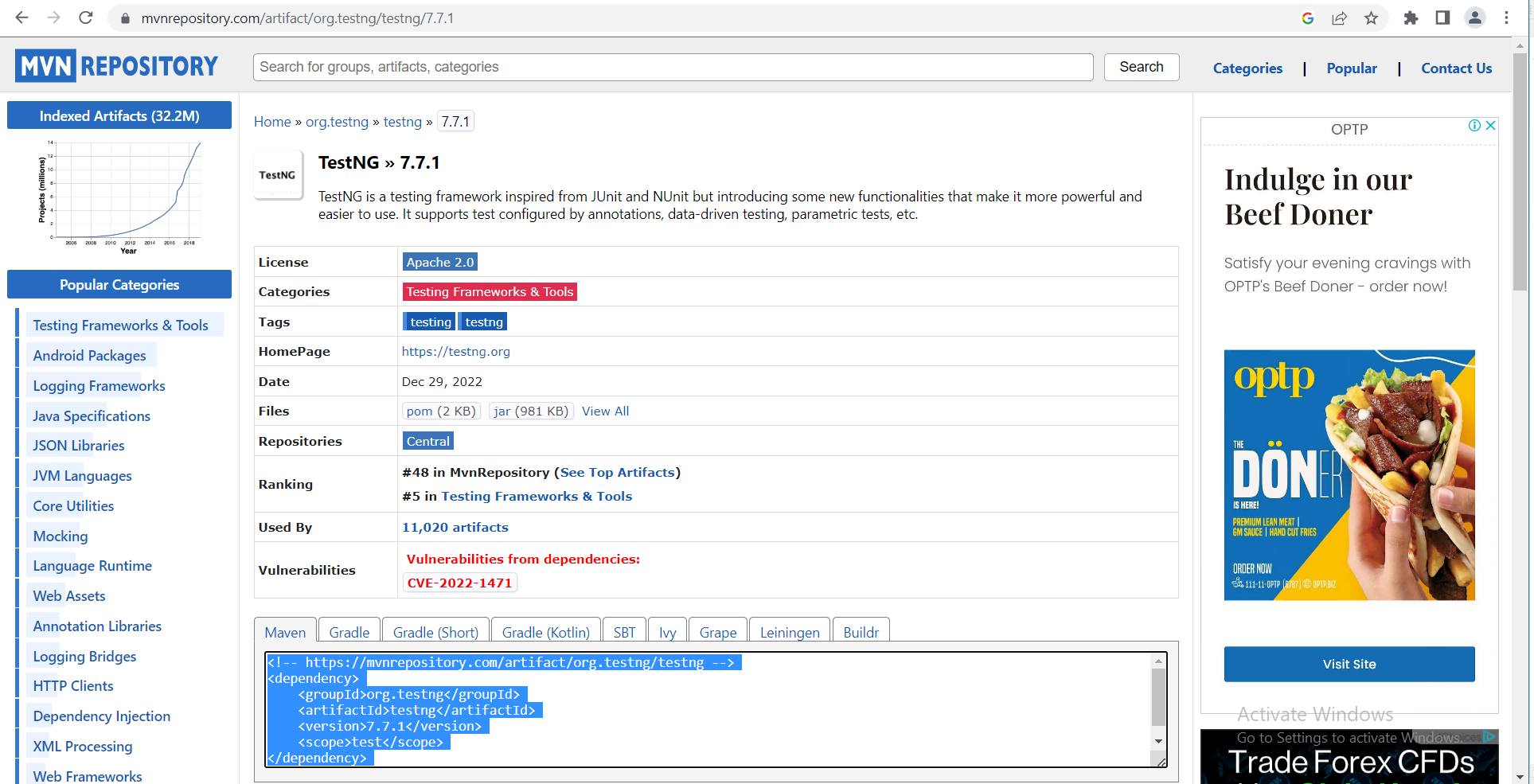
**How to Add TestNG in Eclipse IDE with Selenium:**

**To add TestNG dependencies in pom.xml, access the following website.**

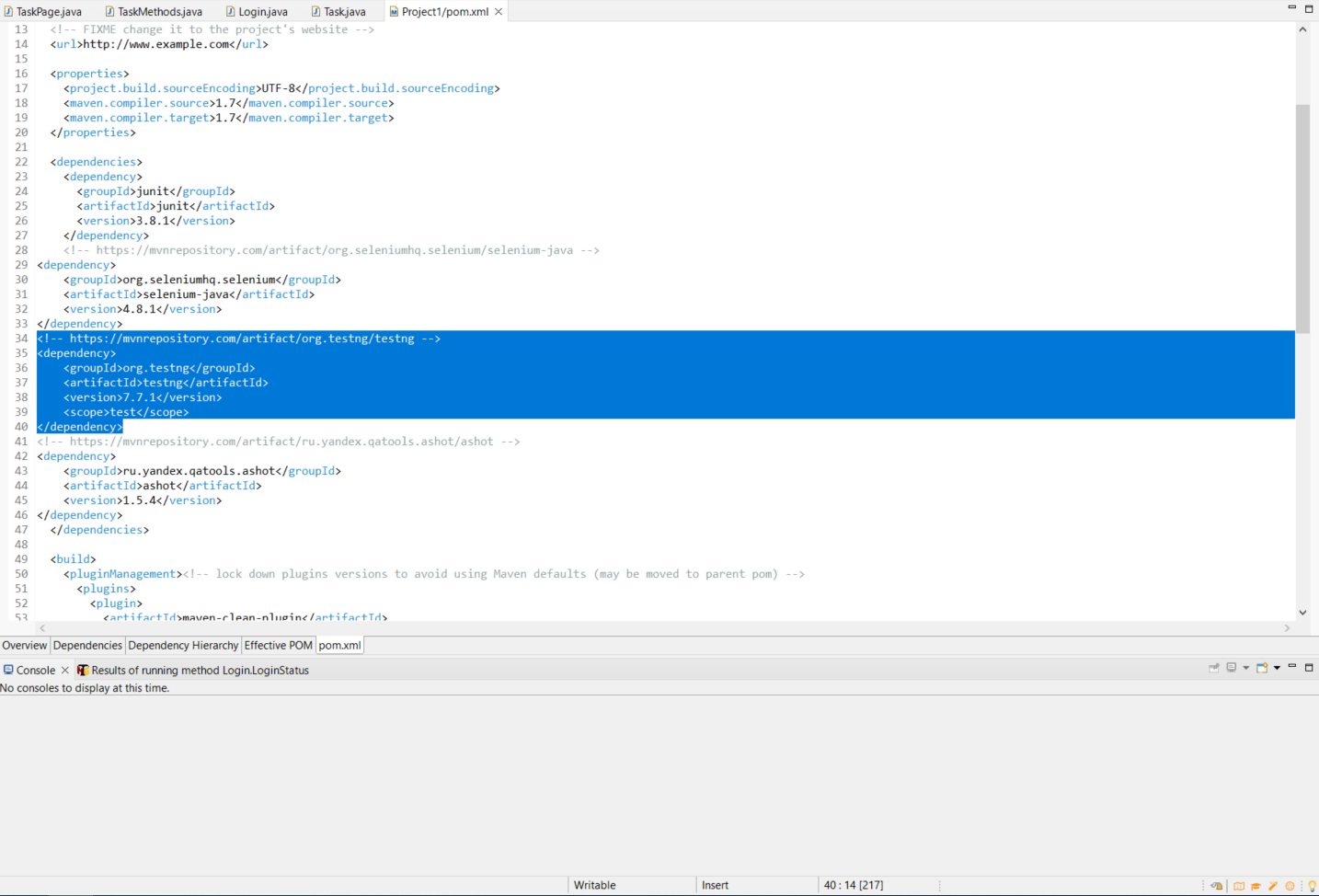
<https://mvnrepository.com/artifact/org.testng/testng>



**Click the latest version 7.7.1 and copy the dependency text from there.**



**Now open pom.xml file from the package explorer and paste the copied dependencies there.**

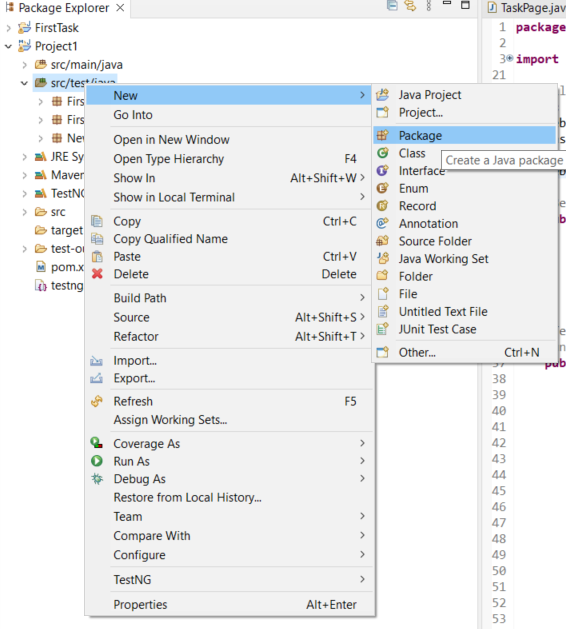


Save the file to make sure that TestNG is added in Eclipse IDE.

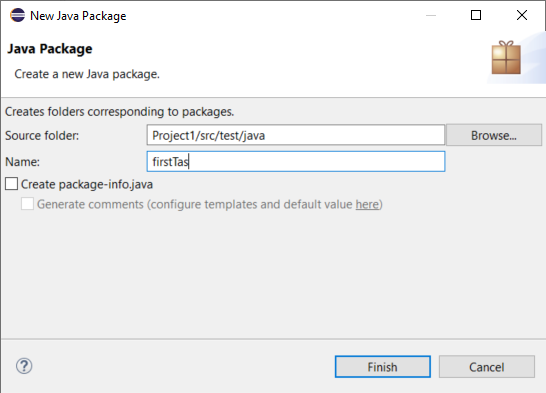
**How to create packages and classes in Maven Project:**

**Java Package Creation**

Create a package by expanding the maven project. Right Click **src/test/java > New > Package**.



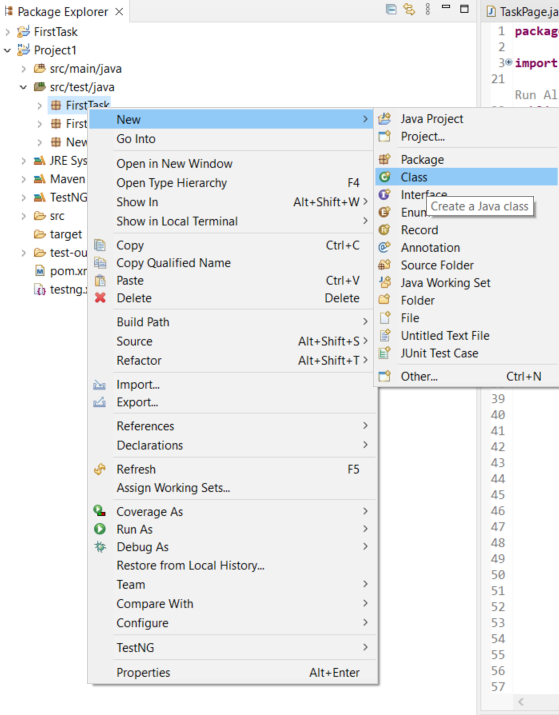
**Enter the name and click Finish.**



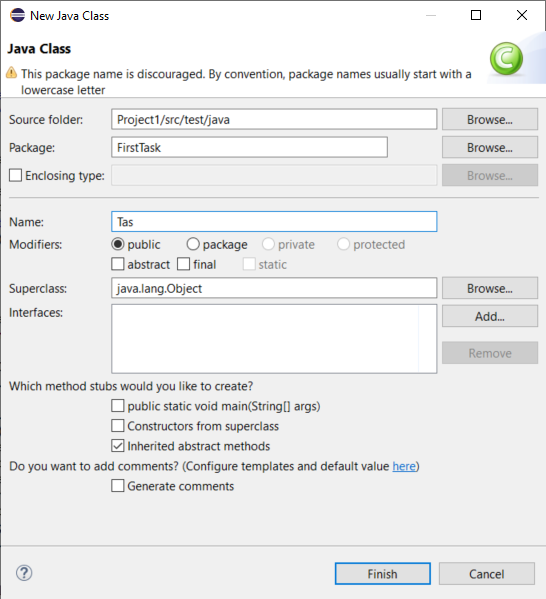
A new java package is created.

**Java Class Creation**

Right click the created **package** **> New > Class**.



Enter the name and click Finish.



A new java class is created.

**How to use selenium in Maven Project:**

Firstly, create a maven project, configure selenium by adding its dependencies in its pom.xml. Create a new package and add a class in it. Let’s access the following website using selenium.

https://phptravels.com/demo

Below is the actual WebDriver code for the logic presented above:

**package** FirstTask;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**public** **class** Test {

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

// **TODO** Auto-generated method stub

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

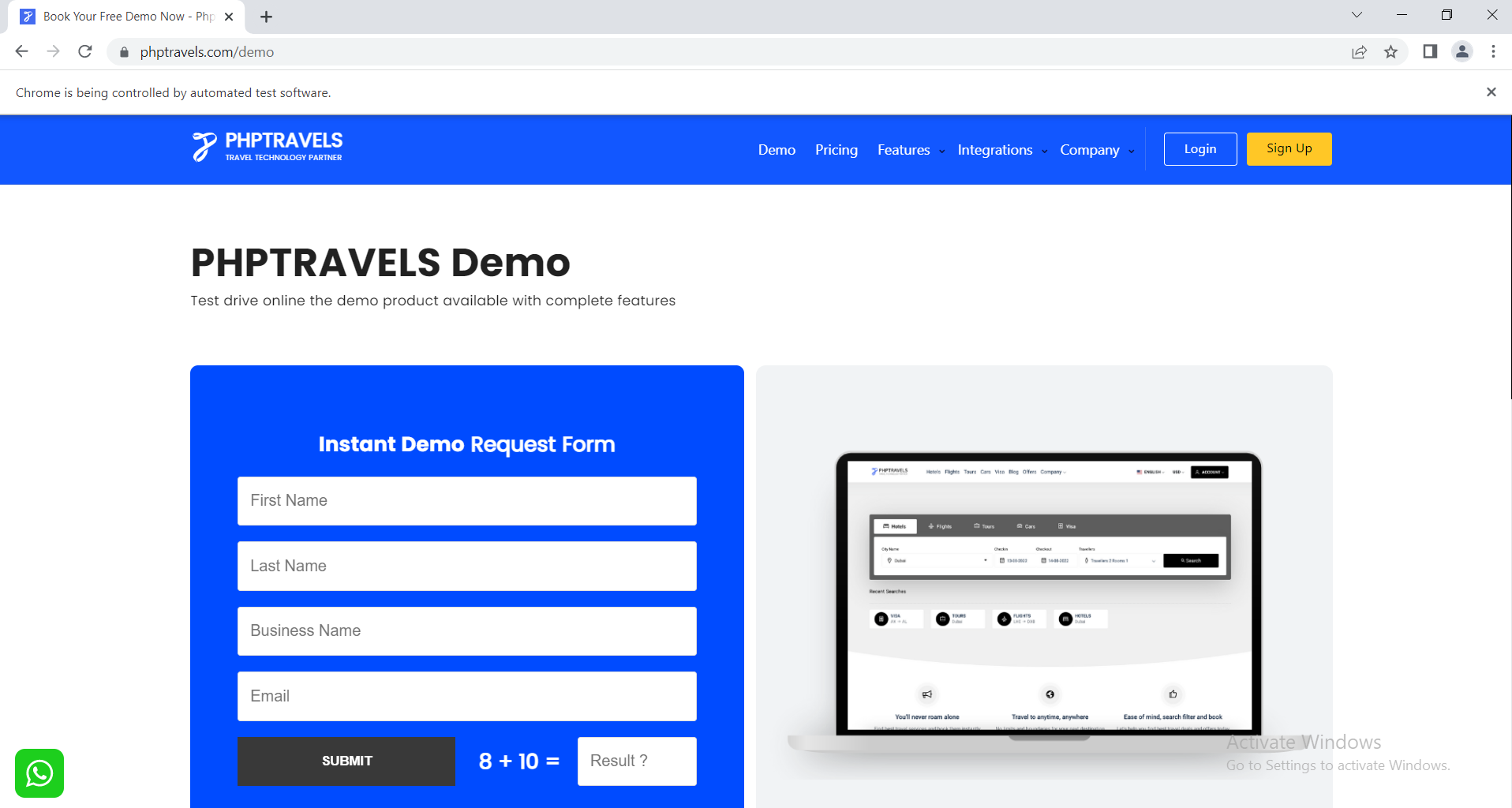
System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\ibtisam.umer\\Desktop\\Selenium\\chromedriver.exe");

driver.get("https://phptravels.com/demo");

driver.manage().window().maximize();

}

}

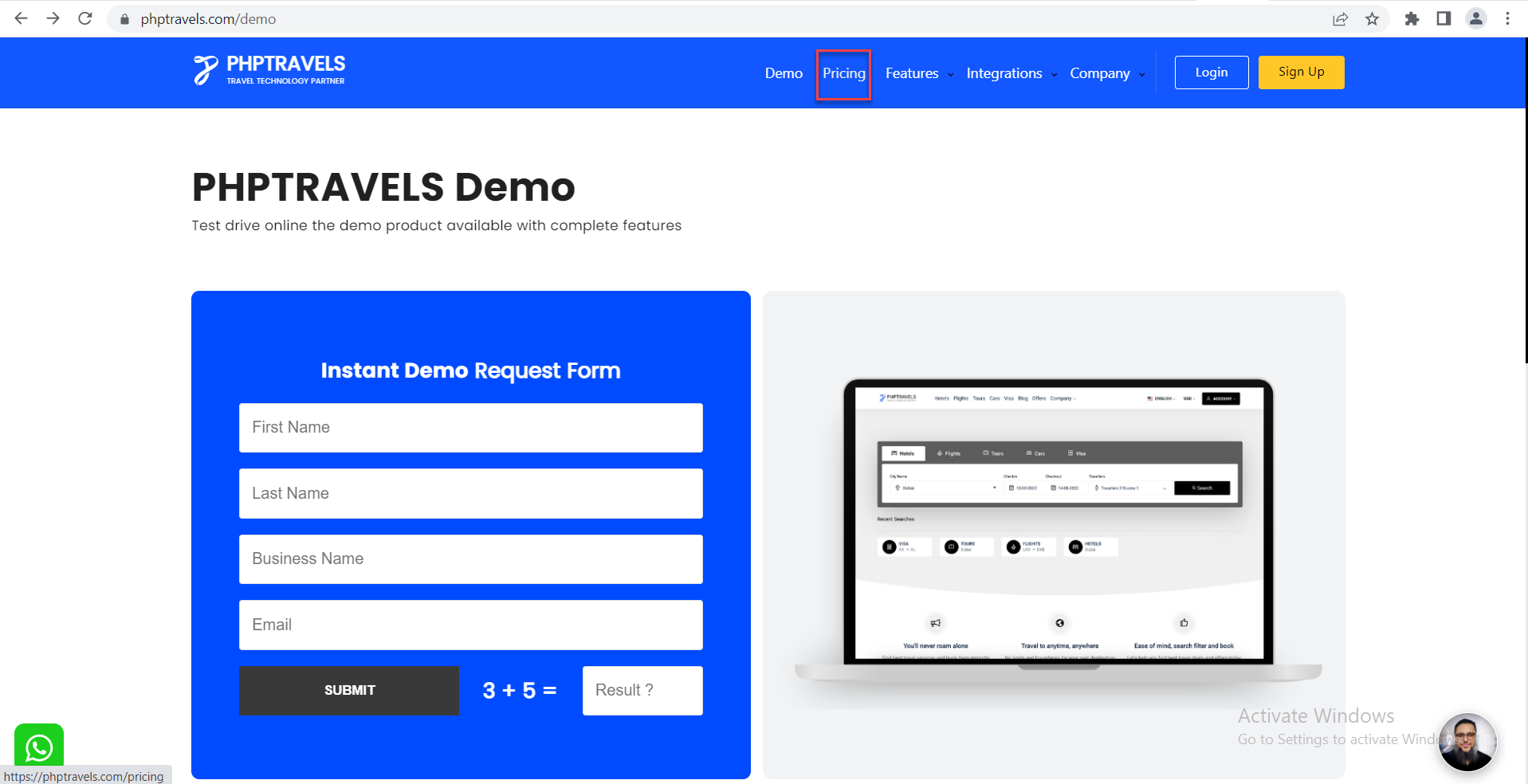


**How to use TestNG with Selenium in Eclipse Maven Project:**

**Importance of TestNG**

* TestNG makes automated tests more structured, readable, maintainable, and user-friendly.
* TestNG provides powerful features and reporting.
* TestNG is preferred by testers for its ability to write powerful test cases with the help of annotations, grouping, and parametrizing.
* TestNG covers all classifications of test automation like Unit testing, Functional testing, End-to-End, and integration testing.

At first add TestNG dependencies in pom.xml and start writing test-cases using @Test annotation of TestNG. For instance, we are writing a test case that the Pricing button at the menu panel of the website is working or not as shown in the figure;



Below is the WebDriver code for the logic presented above:

**package** FirstTask;

**import** java.io.IOException;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.testng.Assert;

**import** org.testng.annotations.BeforeMethod;

**import** org.testng.annotations.Test;

**public** **class** Test1 {

WebDriver driver;

WebElement pricingButton;

WebElement tag;

@BeforeMethod

**public** **void** Setup() {

driver = **new** ChromeDriver();

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\ibtisam.umer\\Desktop\\Selenium\\chromedriver.exe");

driver.get("https://phptravels.com/demo");

driver.manage().window().maximize();

}

@Test

**public** **void** ButtonTest() **throws** InterruptedException, IOException {

pricingButton = driver.findElement(By.*xpath*("//a[@class='jfHeader-menuListLink jfHeader-dynamicLink js-tracking js-myforms']"));

tag = driver.findElement(By.*xpath*("//h1[@class='mb-0']"));

Thread.*sleep*(2000);

pricingButton.click();

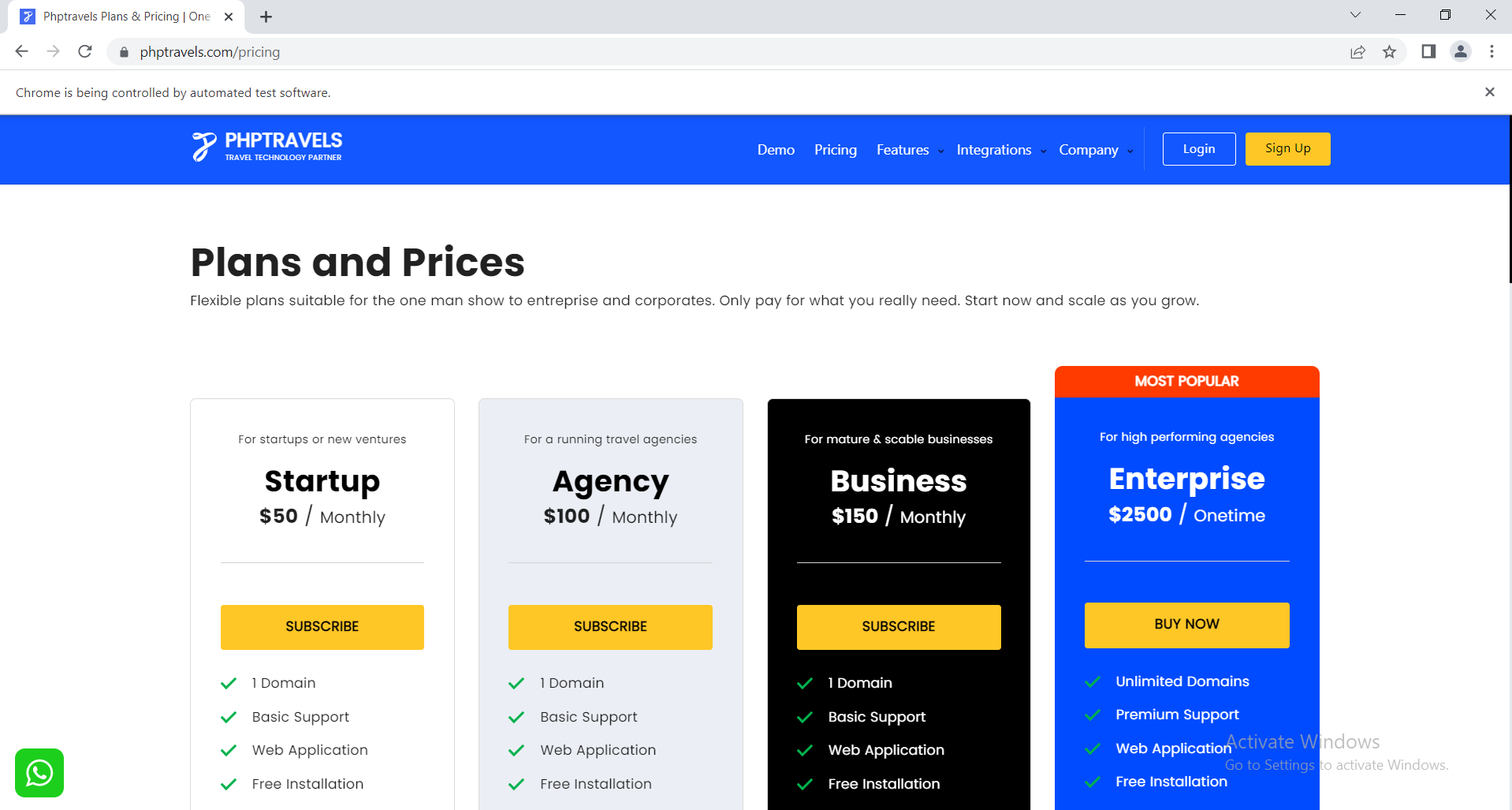
Thread.*sleep*(2000);

Assert.*assertEquals*("Phptravels Plans & Pricing | One Time Payment", driver.getTitle());

}

}

As Pricing button is clicked, following page appears.

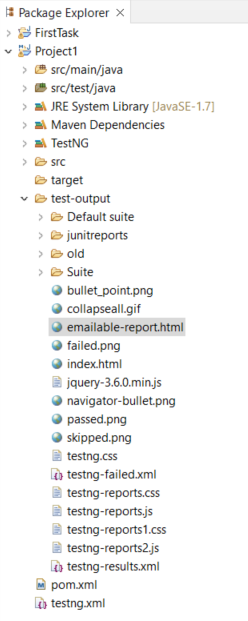


If the title of the newly opened page is same as we are expecting then our test case is declared as passed by TestNG. In this way, we can write as many testcases as possible by using @Test annotation of TestNG and can get their statuses as passed, failed or skipped.

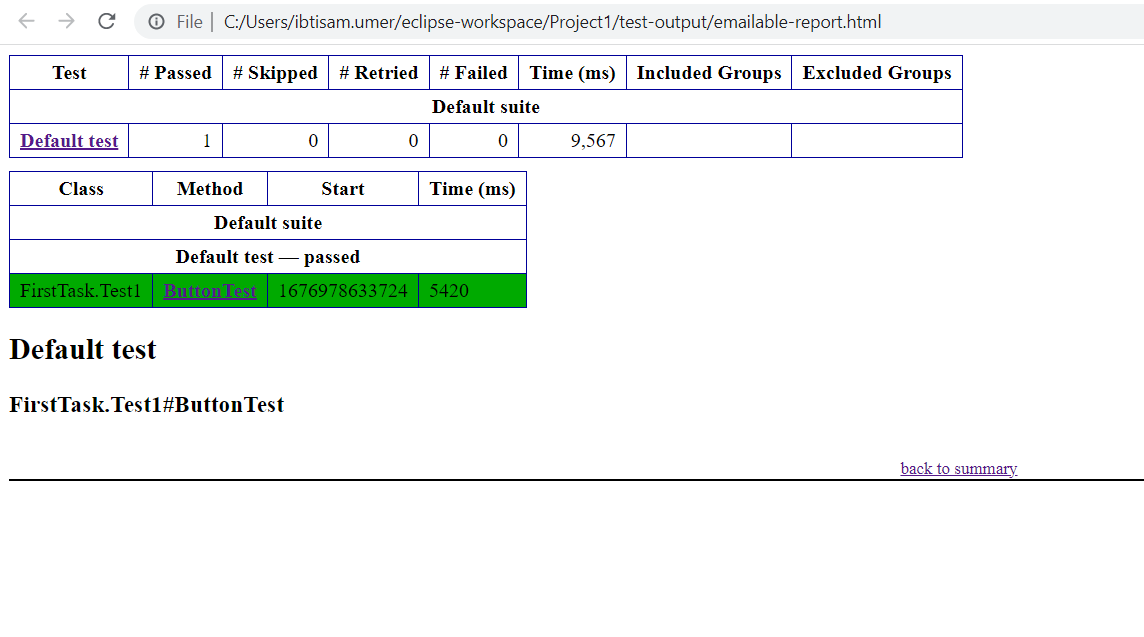
**How to get report using TestNG:**

In order to know about the statuses of the executed test-cases, you can access TestNG default report.

In Package Explorer, **expand test-output > emailable-report.html**.



Click it to get the detailed report of execution status. Let’s have a look at the report of the testcase we just executed.



**How to take screenshot using Selenium:**

To capture a screenshot of webpage when the test case fails, you can do it easily with the help of TestNG. TestNG provide a separate ITestResult interface which keeps the track of executed test details. We just need to capture the state of test and call the screenshot method. @AfterMethod annotation can be used to get the execution status of the test case and if it falls in the ‘Failed’ category, the screenshot of the webpage is captured.

By doing this you could easily identify where exactly the script got failed by seeing the screenshot.

Below mentioned script shows how to capture a screenshot of failed test case using Selenium WebDriver.

**package** FirstTask;

**import** java.io.File;

**import** java.io.IOException;

**import** org.apache.commons.io.FileUtils;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.OutputType;

**import** org.openqa.selenium.TakesScreenshot;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** org.testng.Assert;

**import** org.testng.ITestResult;

**import** org.testng.annotations.AfterMethod;

**import** org.testng.annotations.BeforeMethod;

**import** org.testng.annotations.Test;

**public** **class** Test1 {

WebDriver driver;

WebElement pricingButton;

WebElement tag;

@BeforeMethod

**public** **void** Setup() {

driver = **new** ChromeDriver();

System.*setProperty*("webdriver.chrome.driver", "C:\\Users\\ibtisam.umer\\Desktop\\Selenium\\chromedriver.exe");

driver.navigate().to("https://phptravels.com/demo");

driver.manage().window().maximize();

}

@Test

**public** **void** ButtonTest() **throws** InterruptedException, IOException {

pricingButton = driver.findElement(By.*xpath*("//a[@class='jfHeader-menuListLink jfHeader-dynamicLink js-tracking js-myforms']"));

tag = driver.findElement(By.*xpath*("//h1[@class='mb-0']"));

Thread.*sleep*(2000);

pricingButton.click();

Thread.*sleep*(2000);

Assert.*assertEquals*("Phptravels Plans & Pricing | One Time Payment//", driver.getTitle(),"Pricing Button is not working");

}

@AfterMethod

**public** **void** Closure(ITestResult result) {

**if**(ITestResult.***FAILURE***==result.getStatus()) {

**try**

{

File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.***FILE***);

FileUtils.*copyFile*(scrFile, **new** File("C:\\Users\\ibtisam.umer\\Desktop\\"+result.getName()+".png"));

}

**catch** (IOException e1)

{ // **TODO** Auto-generated catch block

e1.printStackTrace();

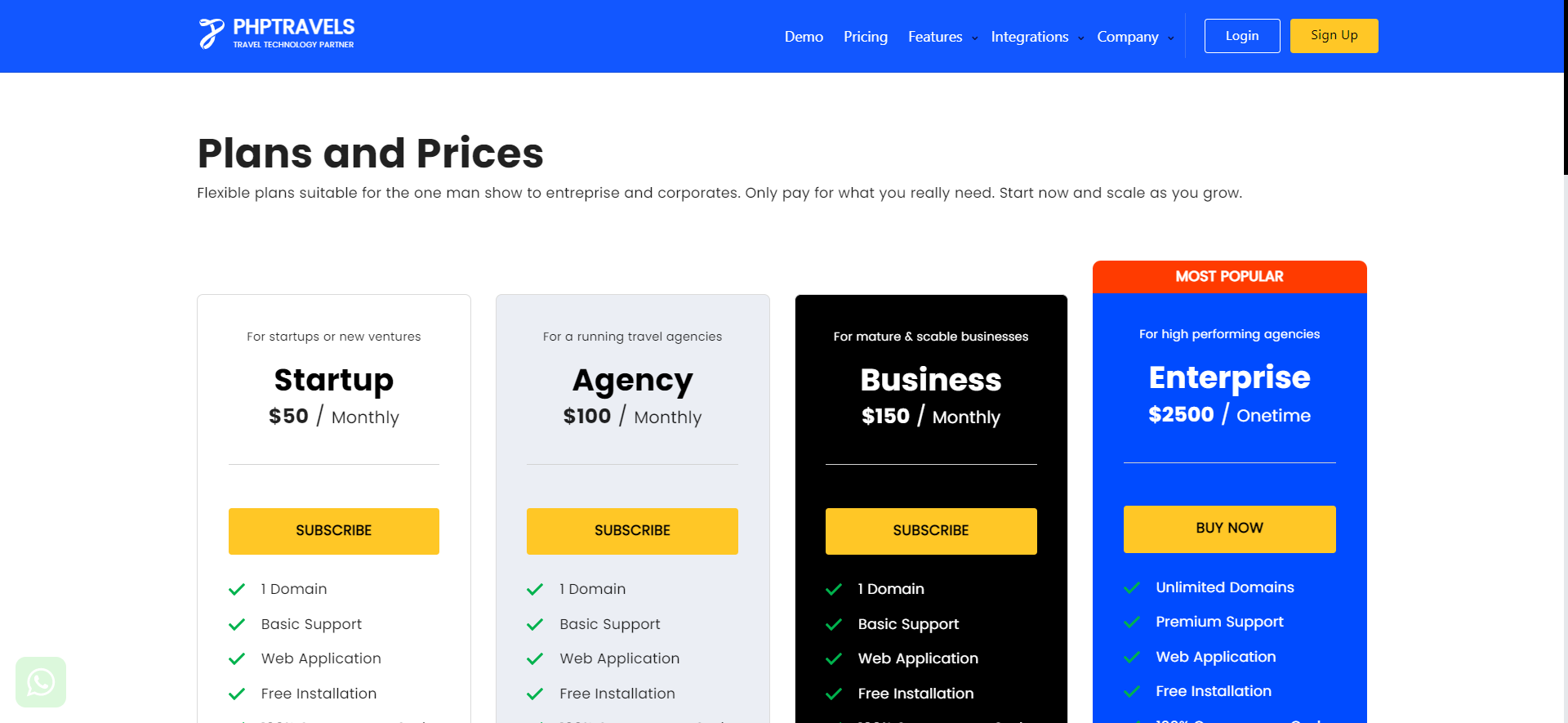
}

}

}

}

Captured Screenshot is;



The test-case failed as the expected title of the webpage was not same as the actual title. The screenshot was captured and saved in the location we described in the script.