Test Automation Selenium WebDriver using TestNG

Purnima Bindal, Assistant Professor, Department Of Computer Science, PGDAV College, University Of Delhi, India
Sonika Gupta, Software Engineering Analyst, Accenture Sevices Private Limited, India.

Abstract
Test automation tools (softwares) are used to control the execution of tests (Application Under Test) and then comparing actual outcomes to the predicted outcomes. Examples of Test Automation tools are HP Quick Test Professional, Selenium, Test Complete, SOAP UI etc. Selenium is a suite of tools for cross-platform automated testing of web applications. It runs on many browsers and operating systems and can be controlled by many programming languages and testing frameworks. Selenium WebDriver is a functional automation tool to automate the applications. It makes direct calls to the browser using each native support for automation. Selenium Web Driver is also known as Selenium 2.0.

Keywords: Seleniun WebDriver, Test automation, TestNG

Introduction
Selenium is an open source automated testing suite for web applications across different browsers and platforms. Selenium is similar to HP Quick Test Professional but it majorly focuses on automating web-based applications.
It is not just a single tool but a suite of software's, each catering to different testing needs of an organization.
Selenium has four components as listed below -
a) Selenium Integrated Development Environment (IDE)
b) Selenium Remote Control (RC)
c) WebDriver
d) Selenium Grid

Selenium IDE
Selenium IDE is a firefox extension that can automate the browser through a record and playback feature. Selenium IDE was created by Shinya Kasatani from Japan in 2006. It is the simplest framework in the Selenium suite. It is a Firefox plugin and can be installed easily like other plugins[1].

Advantages of Selenium IDE
a) Easy to Use and Install.
b) No Programming Language is required though knowledge of HTML and DOM is needed.
c) Test Cases can be exported to formats/languages such as Python, Ruby, C#, Java etc. which are usable in Selenium RC and WebDriver.
d) It has built-in help and test results reporting module.

Disadvantages of Selenium IDE
a) Available only for Firefox browser.
b) Designed only to create prototypes of tests.
c) No support for iteration and conditional operations.
d) Test Execution is slow as compared to Selenium RC and WebDriver.
e) Parameterization is not supported by Selenium IDE.
f) Mouse movements and keyboard operations are also not supported by Selenium IDE.
g) It does not support encryption and decryption.

Selenium RC
Selenium RC also known as Selenium 1, is the first automated web testing tool that allowed users to use programming languages in creating complex tests. Programming languages supported by RC are Java, C#, PHP, Python, Perl, Ruby.

Advantages of Selenium RC
a) Selenium RC supports Cross-browser and cross-platform.
b) It can perform looping and conditional operations.
c) It can support data-driven testing.
d) It has matured and complete API.
e) It can readily support new browsers.
f) Faster execution than IDE.

Disadvantages of Selenium RC
a) Installation is complicated than IDE.
b) Must have programming knowledge.
c) Needs Selenium RC Server to be running in the background.
d) API contains many redundant and confusing commands.
e) Browser interaction is less interactive.
f) Inconsistent results and uses javascript.
g) Slower execution time than WebDriver.

WebDriver
WebDriver allows the test scripts to communicate directly to the browser and thereby controlling it from the OS level. WebDriver, unlike Selenium RC, does not rely on JavaScript for automation. It is better than Selenium IDE and Selenium RC in many aspects. It implements modern and stable approach in automating the browser's actions. Programming languages supported by WebDriver

are Java, C#, PHP, Python, Perl, Ruby as in Selenium RC.

Advantages of WebDriver

a) It is a web automation framework tool that allows you to execute your tests against different browsers and not just Firefox.
b) It allows us to use programming languages in designing the test scripts.
c) Installation of WebDriver is simpler than Selenium RC.
d) Browser interaction is more realistic.
e) No need for separate component such as the RC Server.
f) Faster execution time than Selenium IDE and Selenium RC.

Disadvantages of WebDriver

a) Installation of WebDriver is more complicated than Selenium IDE.
b) Must have programming knowledge.
c) Cannot readily support new browsers.
d) It has no built-in mechanism for logging runtime messages and generating test results[1].

1.4 **Selenium Grid** is a tool used together with Selenium RC to run parallel tests across different machines and different browsers all at the same time.

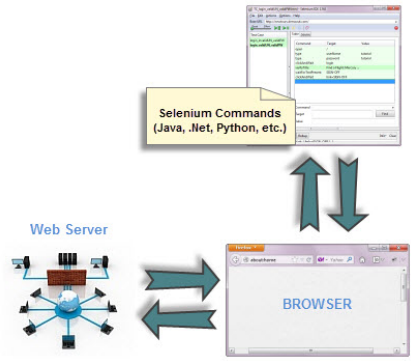
Features of Selenium Grid

a) It enables simultaneous running of tests in multiple browsers and environments.
b) It saves time enormously.
c) It utilizes the hub-and-nodes concept. The hub acts as a central source of Selenium commands to each node connected to it.

Note: Selenium RC and WebDriver was merged to form Selenium 2 and was released in 2011. Selenium 2 has object- oriented API's from WebDriver which can directly interact with browsers. It supports wide range browsers and multiple language bindings[1].

WebDriver's Architecture

WebDriver's Architecture is simpler than Selenium RC's. It controls the browser from the OS level. We need to know one programming language which contains Selenium commands and a browser.



WebDriver is faster than Selenium RC since it speaks directly to the browser and uses the browser's own engine to control it. It interacts with page elements in a more realistic way. For example, if you have a disabled text box on a page, WebDriver cannot enter any value in it just as a real person cannot[2].

1. Locators in WebDriver

In WebDriver automation everything is related to web elements as it is a web application automation tool. Web Elements are DOM Objects present on the Web Page. To perform operations on a Web Element we need to locate the Elements exactly.

Syntax:
WebElementeleent=driver.findElement (By.<Locator>);

As in the above statement we have to specify some locator to identify web element. "By" is the class , in the class we have different static methods to identify elements as below –

a) id
b) name
c) className
d) tagName
e) cssSelector
f) xpath
g) linkText
h) partialLinkText[3]

2.1 **By.id** - locates elements by the value of their "id" attribute.

Ex: *findElement(By.id("someId"))*

2.2 **By.name** - locates elements by the value of the "name" attribute.

Ex: *findElement(By.name("someName"))*

2.3 **By.className** - finds elements based on the value of the "class" attribute.

Ex: *findElement(By.className("someClassName"))*

2.4 **By.tagName** - locates elements by their tag name.

Ex: *findElement(By.tagName("div"))*

2.5 By.cssSelector - finds elements based on the driver's underlying CSS Selector engine.
Ex: *findElement(By.cssSelector("input#email"))*

2.6 By.xpath - locates elements via xpath.
Ex: *findElement(By.xpath("//html/body/div/table/tbody/tr/td[2]"))*

2.7 By.linkText - finds a link element by the exact text it displays.
Ex: *findElement(By.linkText("REGISTRATION"))*

2.8 By.partialLinkText - locates elements that contain the given link text.
Ex: *findElement(By.partialLinkText("REG"))*

2. Steps to Install Selenium WebDriver

a) Download and Install Java Software Development Kit from the link "*http://www.oracle.com/*" from "*Downloads*" tab. The JDK version comes bundled with Java Runtime Environment (JRE) so it is not required to download and install the JRE separately.

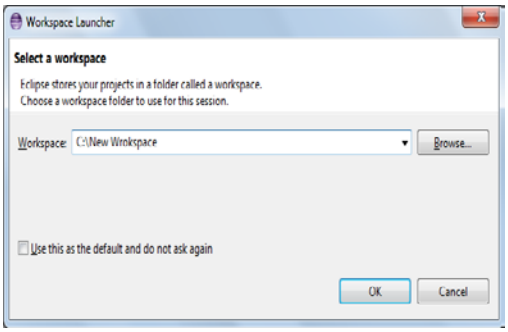
b) Download and Install Eclipse IDE for Java Developers from the link "*http://www.eclipse.org/*" from "*Downloads*" tab. Be sure to choose correctly between Windows 32 Bit and 64 Bit versions. Download the Zip file and extract it(C Drive). There should be an Eclipse folder which contains all the applications files. No Installation is required to use Eclipse.

c) Download Selenium Java Client Driver from the link "*http://docs.seleniumhq.org/*" from "*Downloads*" tab -> "*Selenium Client & WebDriver Language Bindings*" section. The download comes as a ZIP file named "*selenium-java-2.41.0.zip*". Extract the contents of the ZIP file(C Drive). This directory contains all the JAR files that we would later import on Eclipse.

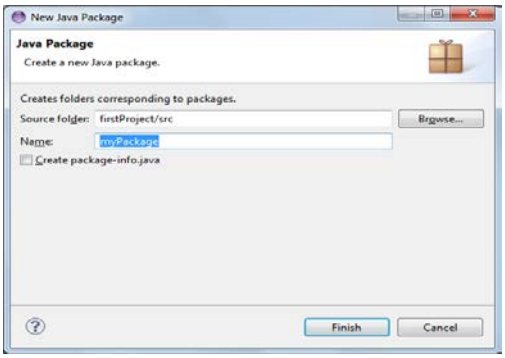
d) Download Selenium Server Standalone Jar file from the link "*http://docs.seleniumhq.org/*" from "*Downloads*" tab -> "*Selenium Server*" section. The download comes as a ZIP file named "*selenium-serverstandalone-2.41.0.jar*". Copy the file in C Drive[3].

3. Configure Eclipse IDE with WebDriver

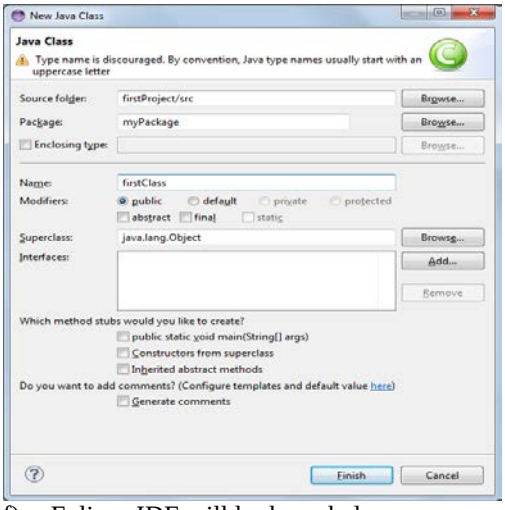
a) Launch "eclipse.exe" from inside the Eclipse folder.
b) Select the Workspace as below –



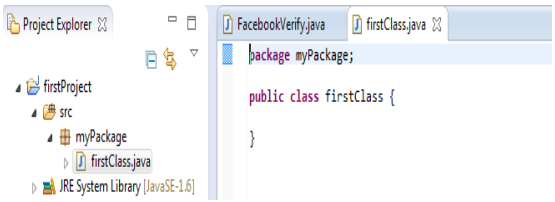
c) Create a new Project. Click on "*File*" > "*New*" > "*Java Project*". Name the Project as "*firstProject*".
d) Right Click on the new project added and create a new package as "*myPackage*".



e) Create a new Java class under "*mypackage*" by right-clicking on it and selecting New > Class as "*firstClass*".

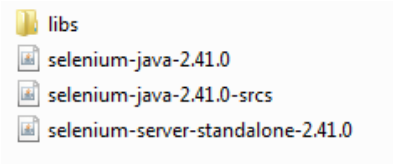


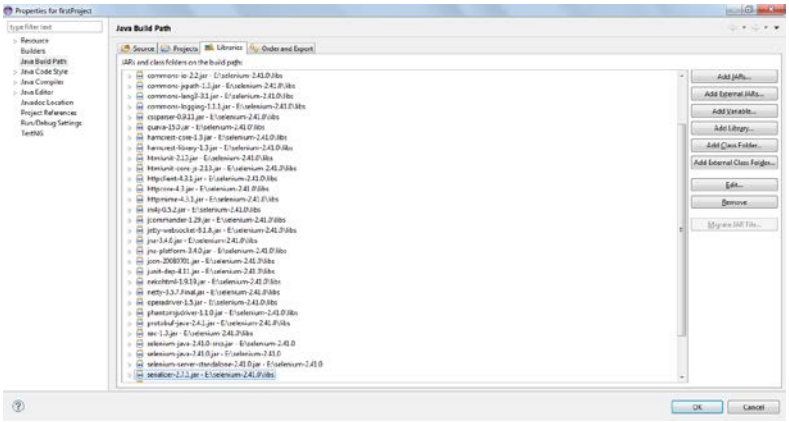
f) Eclipse IDE will looks as below -



g) Right Click on the Project "*firstProject*" > Click on "*Build Path*" > "*Configure Build Path…*"

h) Click on "*Libraries*" tab -> Click on "*Add External JARs…*" button -> Select all the below Jars. Also add all the files from "*libs*" folder as shown in the below image and click on "*Open*".





i) Click on "OK" and all Selenium libraries should get imported into the Project[3].

First WebDriver Code

Below is an example of test case that will check if Facebook homepage is correct.

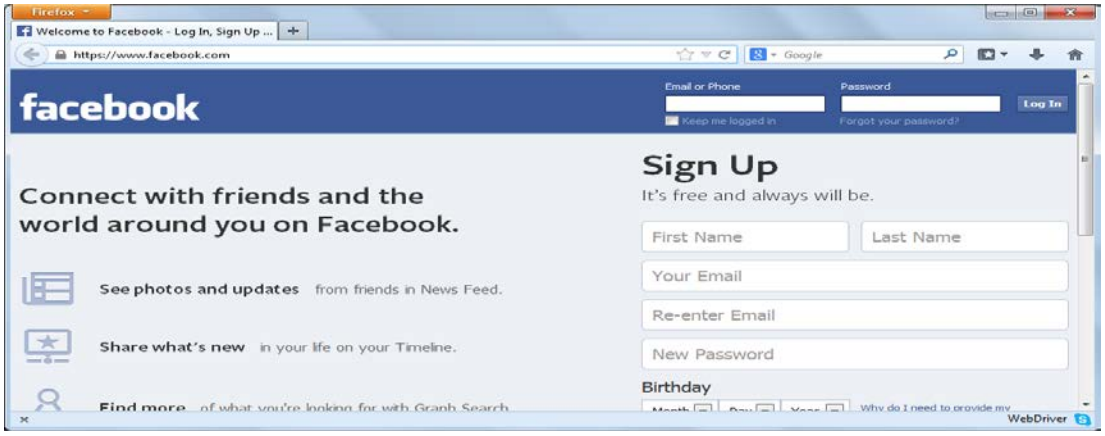
```
package myPackage;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class firstClass
{
        public static void main(String []args)
        {
         String url = "http://www.facebook.com";
                WebDriver driver = new FirefoxDriver();
                driver.get(url);
                String expectedTitle = "Welcome to Facebook - Log In, Sign Up or Learn
                                More";
                String actualTitle = driver.getTitle();
                if (actualTitle.contentEquals(expectedTitle))
                {
                        System.out.println("Test Passed!");
                }
                else
                {
```

```
                    System.out.println("Test Failed");
                }
                driver.quit();
            }
        }
```

Running the Test Case

To run the test case using TestNG, Right Click on Test Case -> click on "*Run As*" -> "*Java Application*".

Output:



Console Window:

Explaining the Code in the Test Case

a) Importing Packages
To start with the code we need to first import following two basic packages –

i. **org.openqa.selenium.*-** this package contains the WebDriver class needed to instantiate a new browser loaded with a specific driver.
ii. **org.openqa.selenium.firefox.FirefoxDriver** – this package contains the FirefoxDriver class needed to instantiate a Firefox-specific driver onto the browser instantiated by the WebDriver class.

b) Initiating a Driver Object
The Driver object is instantiated as below –
WebDriver driver = new FirefoxDriver();
"FirefoxDriver()" with no parameters means that default Firefox profile will be launched by the Java Program.

c) Launching a Browser Session
WebDriver's "**get()**" method is used to launch a new browser session and directs it to the URL which is specified in the parameter.

Driver.get(url);

d) Get the Actual Page Title
WebDriver's "**getTitle()**" method is used obtain the page's title.

actualTitle = driver.getTitle();

e) Compare the Expected and the Actual Result
The below if condition compares the expected and the actual results

if (actualTitle.contentEquals(expectedTitle) {System.*out*.println("Test Passed!");}
Else {System.*out*.println("Test Failed");}

f) Terminating Browser Session

WebDriver "**close()**" method is used to close the browser session.

driver.close();

4. TestNG Framework

TestNG is a testing framework which is designed to simplify a broad range of testing needs, from unit testing to integration testing. "NG" refers to Next Generation. It overcomes the limitations of JUnit. Some of the functionalities in TestNG which makes it more efficient are as follows –

a) It support for Annotations.
b) Test Cases can be grouped more easily.
c) Parallel testing is possible.
d) It supports for data-driven testing.
e) Flexible test configuration.
f) Abilty to re-execute failed test cases[3].

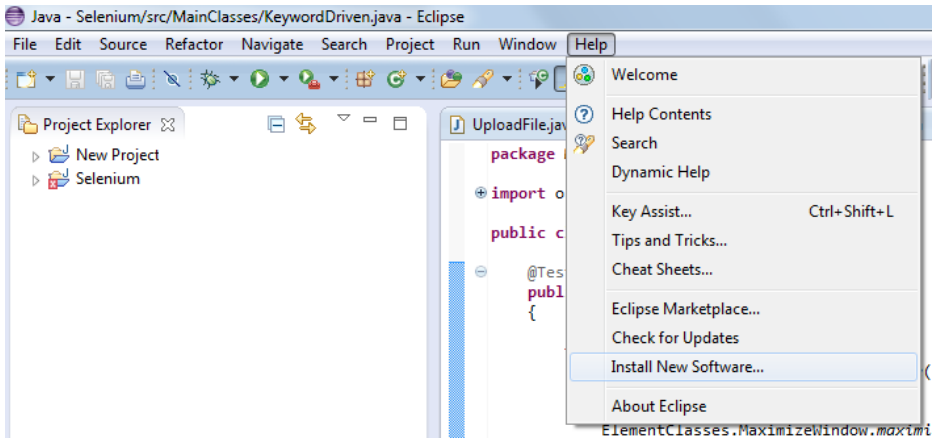
Need for TestNG in Selenium

TestNG can generate reports in the readable format based on the Selenium test results. WebDriver has no native

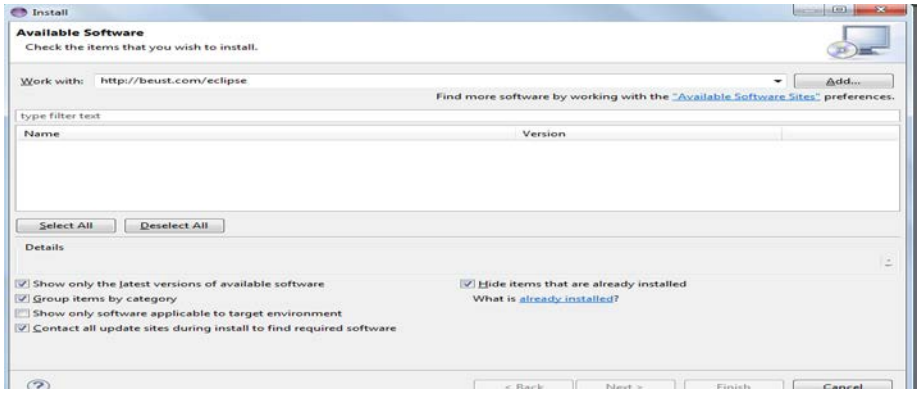
mechanism for generating reports. Also, the uncaught exceptions are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as failed steps in the report[3].

Installing TestNG in Eclipse IDE

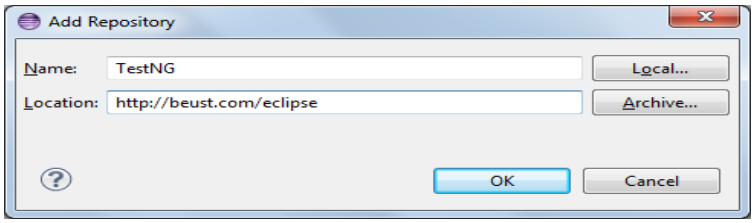
a) Launch Eclipse IDE
b) Click on "Help" -> "Install New Software…"



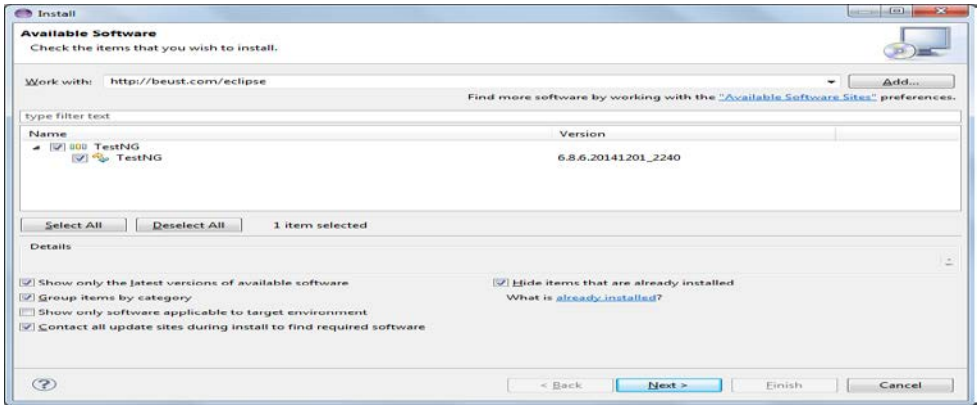
c) Type "http://beust.com/eclipse" in the "Work with:" text box and click add.



d) Write TestNG in name text box and click OK.

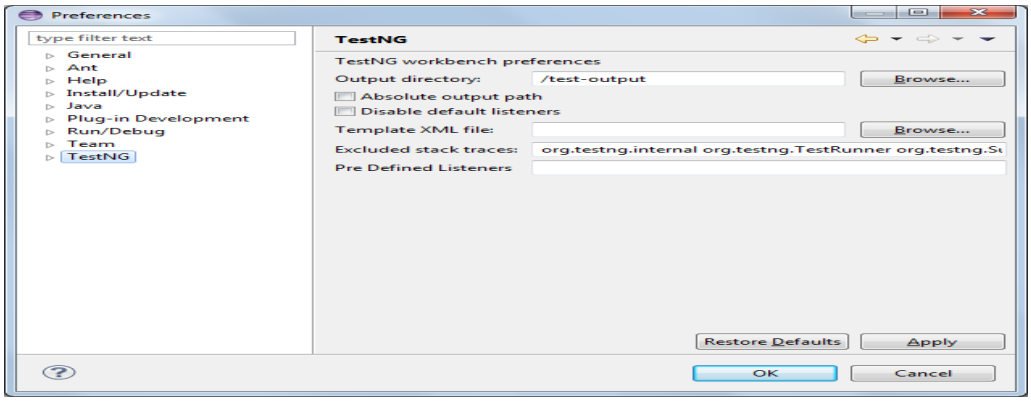


e) TestNG should get displayed under the name section, select the TestNG check box, click next and finish installation.



f) Restart eclipse to finalize installation. After restarting eclipse TestNG will get configured automatically in eclipse IDE.

g) Verify if TestNG was indeed successfully installed. Click Window -> Preferences and see if TestNG is included on the Preferences list[3]



Annotations in TestNG

Annotations in TestNG are lines of code that can control how the method below them will be executed. Annotations are preceded by "@" symbol.

@BeforeSuite: The annotated method will be run before all tests in this suite have run.

@AfterSuite: The annotated method will be run after all tests in this suite have run.

@BeforeTest: The annotated method will be run before any test method belonging to the classes inside the tag is run.

@AfterTest: The annotated method will be run after all the test methods belonging to the classes inside the tag have run.

@BeforeGroups: The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

@AfterGroups: The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

@BeforeClass: The annotated method will be run before the first test method in the current class is invoked.

@AfterClass: The annotated method will be run after all the test methods in the current class have been run.

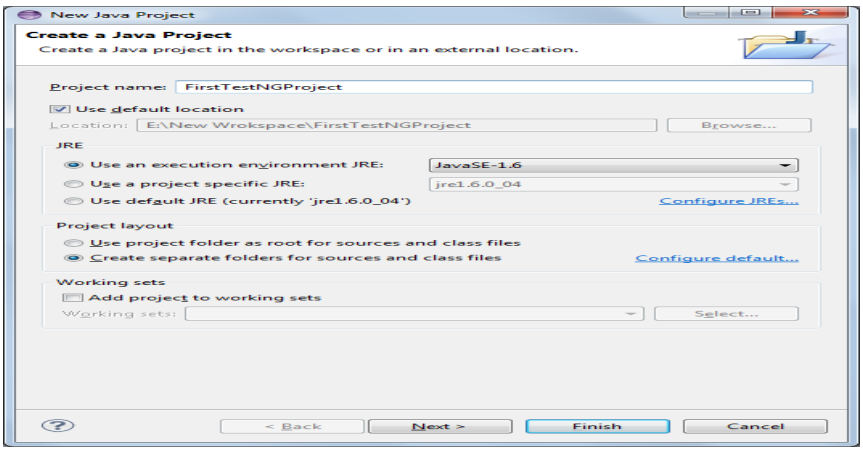
@BeforeMethod: The annotated method will be run before each test method.

@AfterMethod: The annotated method will be run after each test method.

@Test: The annotated method is a part of a test case[1].

Setup TestNG Project

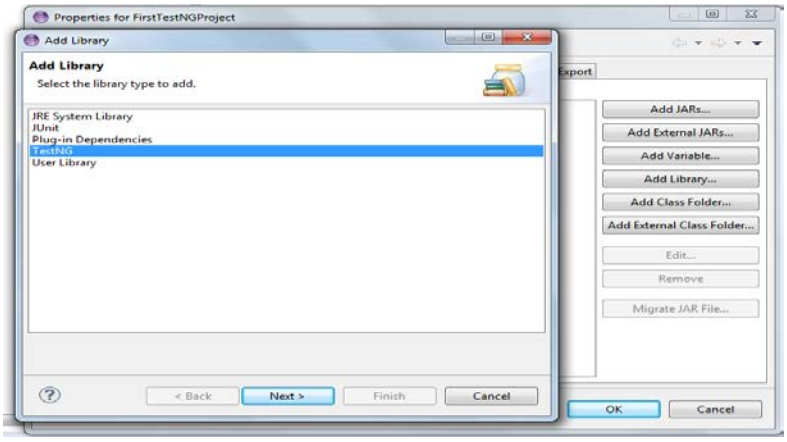
a) Create a new Java Project with name "FirstTestNGProject".

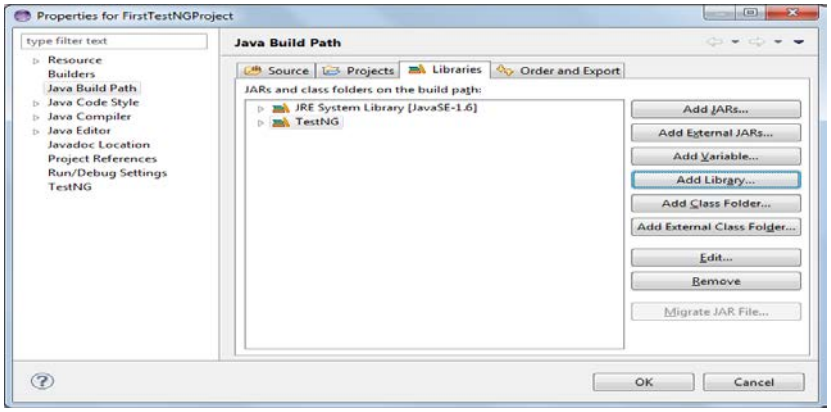


b) Import TestNG Libraries to the Project.

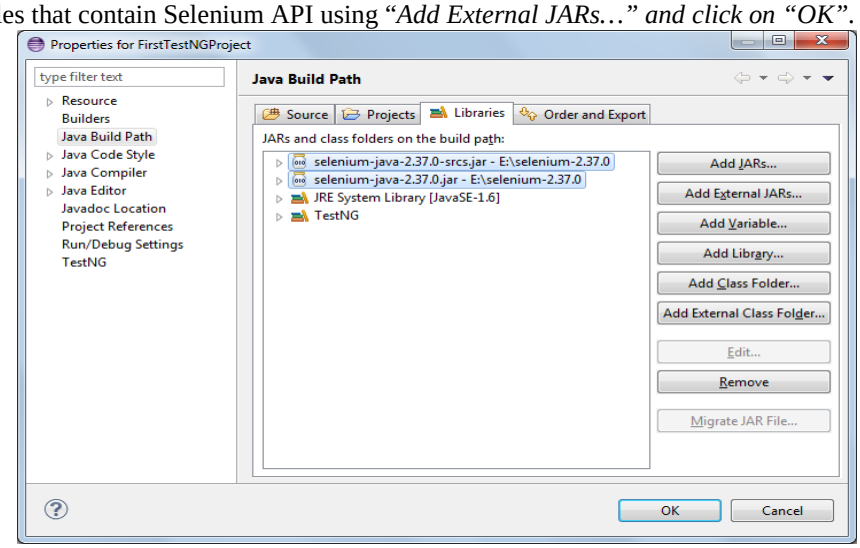
a. Right Click on the Project "*FirstTestNGProject*" -> Click on "*Build Path*" -> "*Configure Build Path…*".

b. Click on "*Libraries*" tab -> Click on "*Add Library…*" button -> Select "*TestNG*" -> Click on "*Next*" -> Click on "Finish".

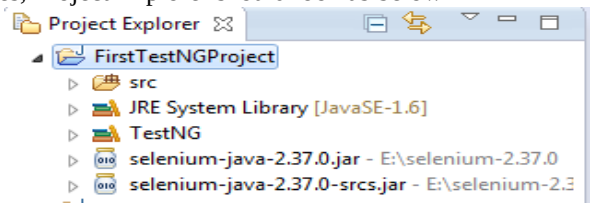




c) Add JAR files that contain Selenium API using *"Add External JARs…" and click on "OK"*.

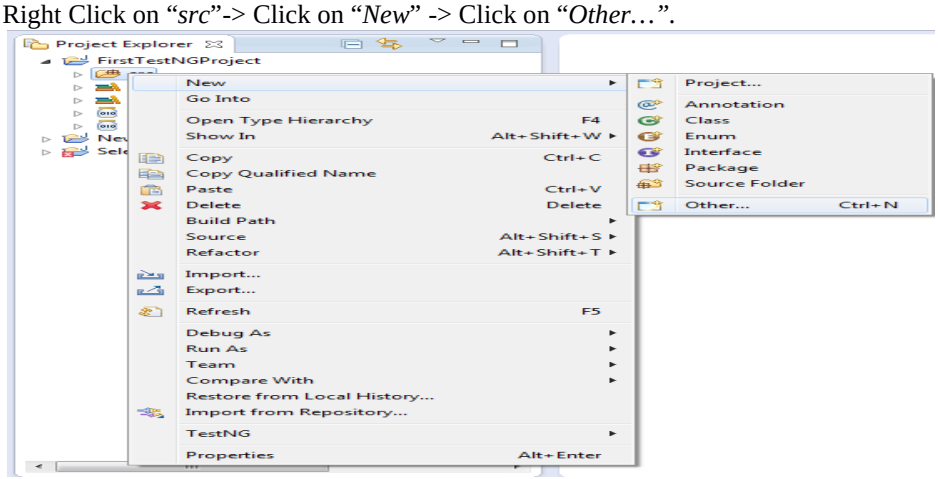


d) After adding the JAR files, Project Explorer should look as below -

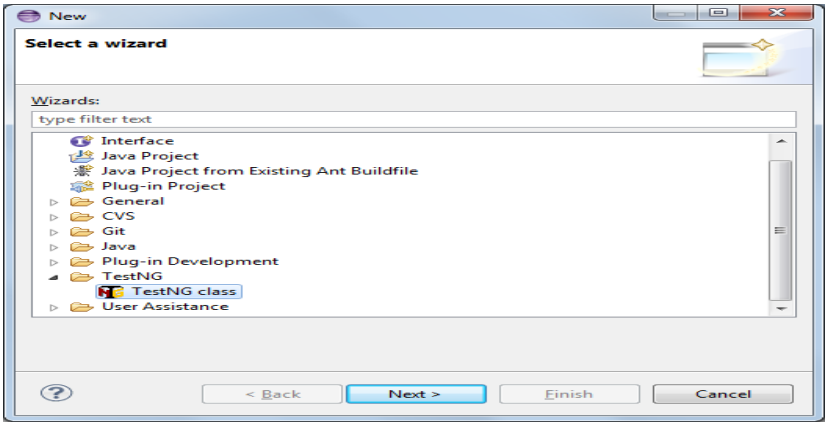


e) Create a new TestNG Class.

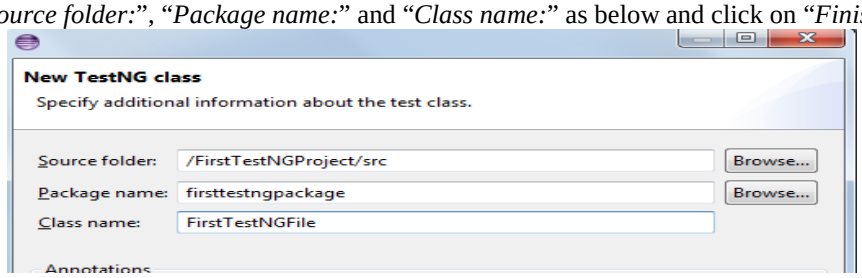
a. Right Click on "*src*"-> Click on "*New*" -> Click on "*Other…*".



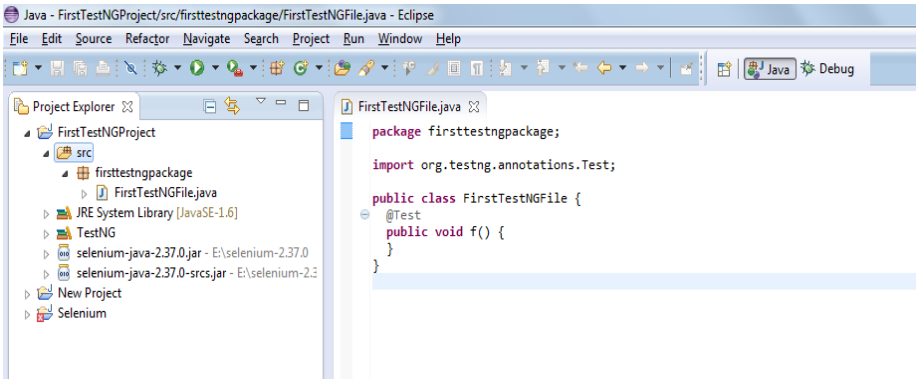
f) Click on "*TestNG class*" -> Click on "*Next*".



g) Enter the "*Source folder:*", "*Package name:*" and "*Class name:*" as below and click on "*Finish*".



h) Eclipse should automatically create a template as below[3] –



7.5 Test Case using TestNG

Below is an example of test case that will check if Facebook homepage is correct.

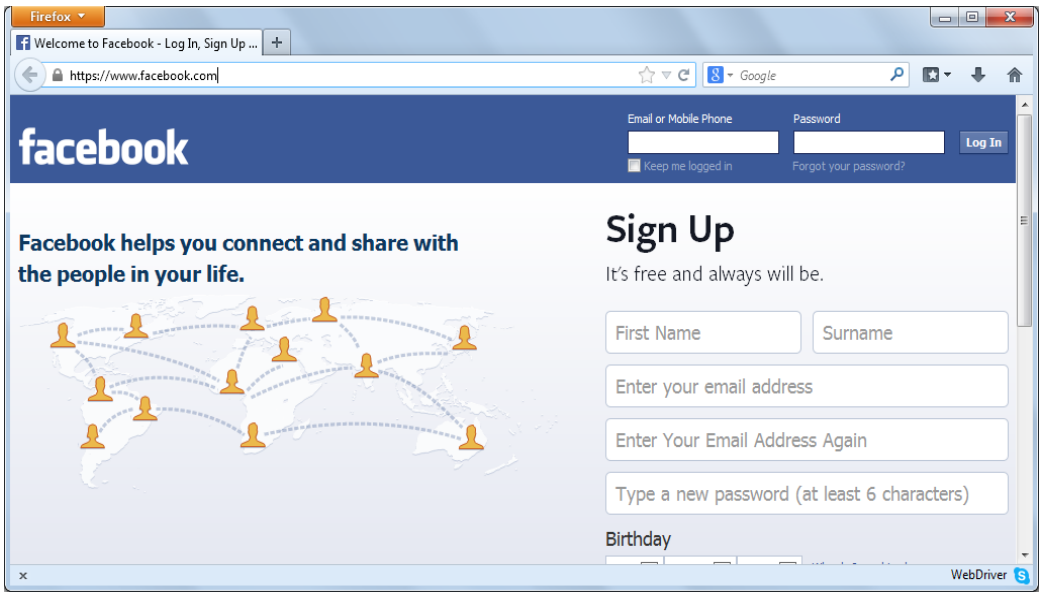
```
package testNGPackage;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.*;

public class FacebookVerify {

  public String url = "http://www.facebook.com";
  public WebDriver driver = new FirefoxDriver();
```

```
@BeforeTest
public void launchBrowser()
{
        driver.get(url);
}

@Test
public void verifyHomePageTitle()
{
        String expectedTitle = "Welcome to Facebook - Log In, Sign Up or Learn More";
        String actualTitle = driver.getTitle();
        System.out.println(actualTitle);
   Assert.assertEquals(actualTitle,expectedTitle);
}

@AfterTest
public void terminateBrowser()
{
 driver.quit();
}
}
```

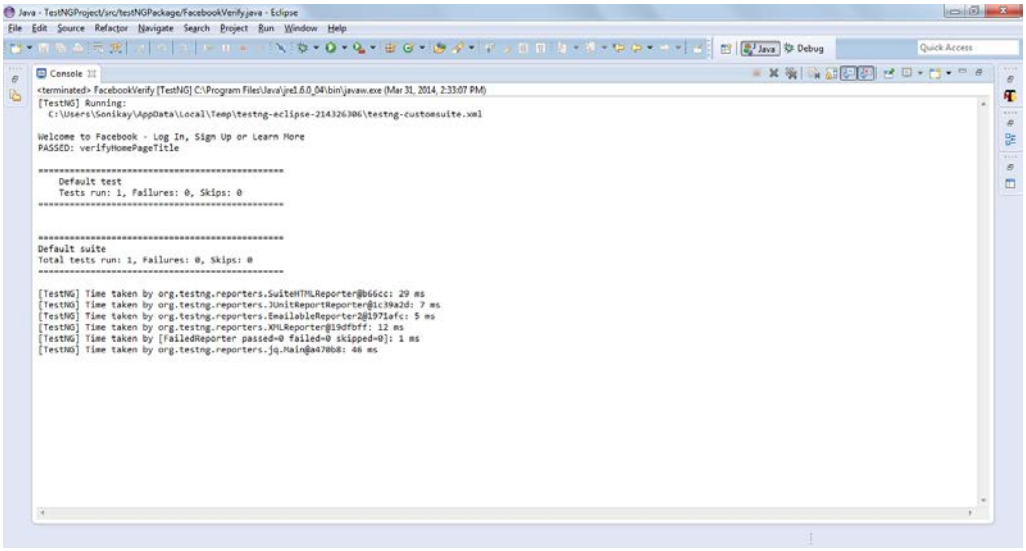
Running Test Case using TestNG

To run the test case using TestNG, Right Click on Test Case -> click on "*Run As*" -> "*TestNG Test*".

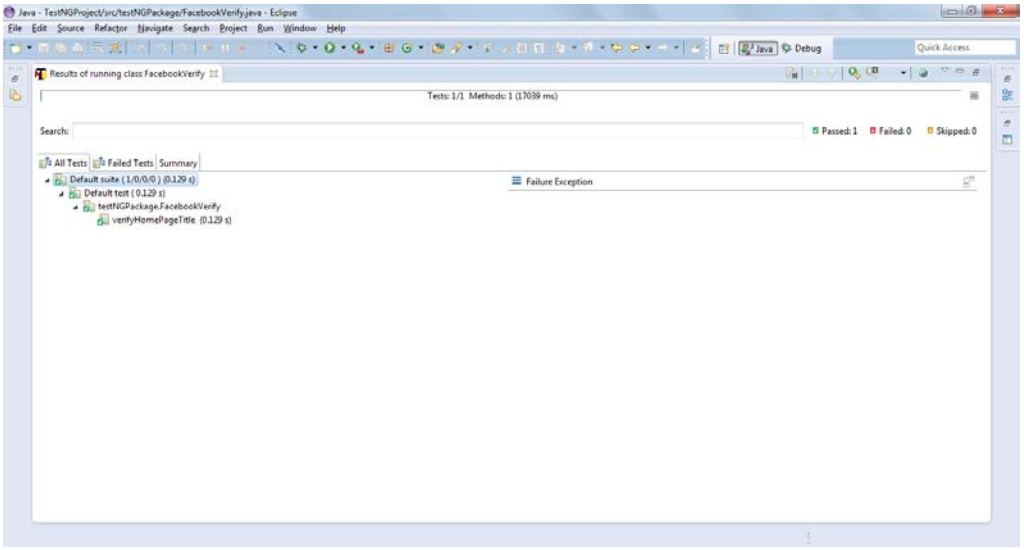
Output:



Console Window:



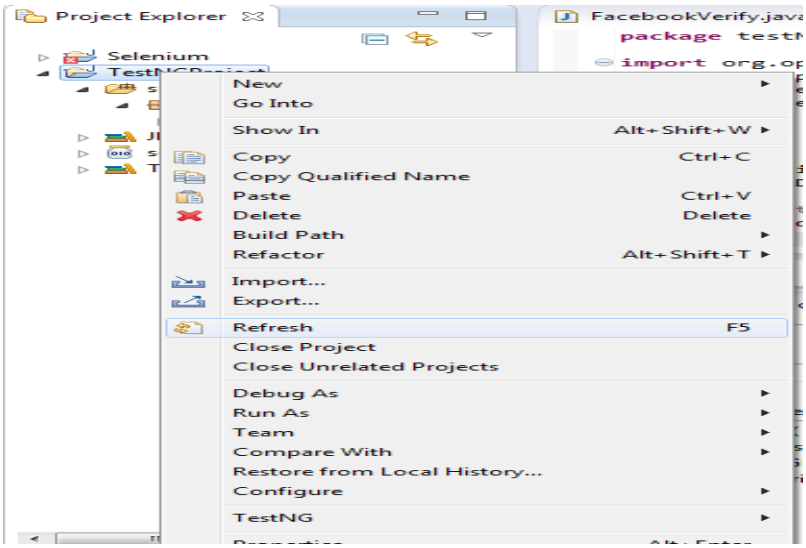
TestNG Result Window –



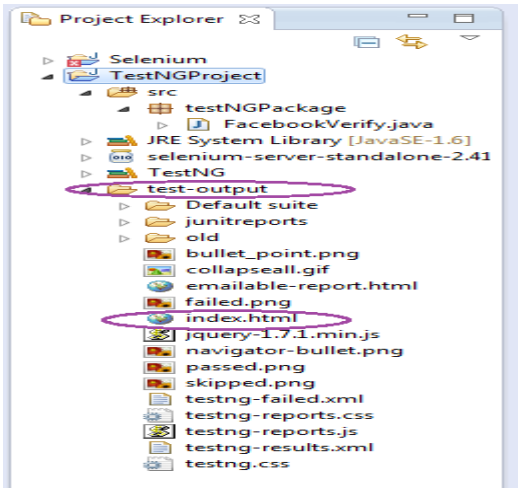
Generating HTML Reports

TestNG has the ability to generate reports in HTML format.

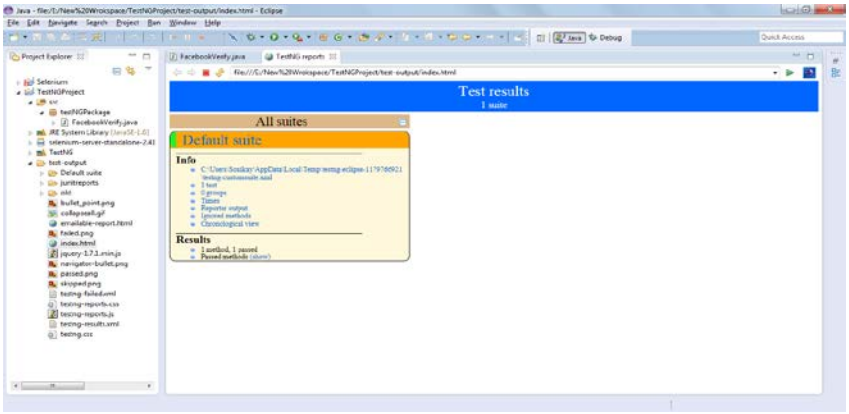
a) After running the test case, right click on the project name in the project explorer window and click on Refresh.



b) "*test-output*" folder sould get created under Project folder. Expand "*test-output*" folder and look for "*index.html*" file. This HTML file is a report of the results of the test run.



c) Double Click on "index.html" file to open it within Eclipse's built in web browser.



Parameters
Annotations in TestNG can be parameterized according to the methods to be executed in a different order using parameter as "priority". Parameters are keywords that modify the annotation's function**.**
Parameters require to assign a value to them by placing a "=" next to them, and then followed by the value. Parameters are enclosed in a pair of parentheses which are placed right after the annotation like the code snippet shown below.

Example: @Test(priority = 0)

TestNG execute the @Test annotation with the lowest priority value up to the largest[3].

5. Object Identification using XPath

Xpath Expression is a way of navigating to HTML Document and can be used to verify or identify the Web Element.

"//" is used to go to the entire HTML Document.
"/" is used to go to Child.
"**[ ]**" is used to go to Parent.

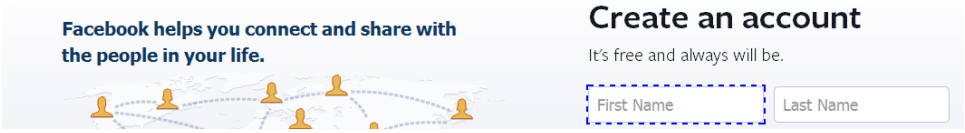
Syntax for writing Xpath –

Below is the syntax to write Xpath -

//tagname[@attribute = 'value']

Application Under Test – Facebook

Example: To write the XPath for the text field "First Name".



HTML Code:

```
<div class="placeholder" aria-hidden="true">First Name</div>
<input id="u_0_1" class="inputtext _58mg _5dba" type="text" aria-label="First
Name" placeholder="" aria-required="1" value="First Name" name="firstname" data-type="text"/>
```

XPath Expression:

//input[@id='u_0_1']
OR **//input[@id = 'u_0_1' and @name = 'firstname']**

Types of XPath –

- **Absolute XPath –** If the location of the path begins from root node to child node followed by "/" it is known as Absolute XPath.

  Example: //div/input[@id = 'u_0_1']

- **Relative XPath –** If the location of the path beigns from the node for which we are writing the XPath by using "//" is known as Relative XPath.

Example: //input[@id='u_0_1']

Note – Writing XPath in Relative Path is considered as a good practice because if any other Web Element is inserted on between the Web Page then Absolute XPath will always fail.

8.3 Text function in XPath –

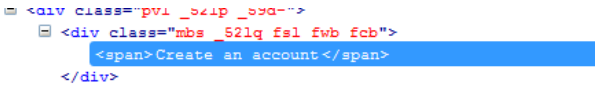
Text function is used to verify if the text is present on the UI. It is not used to verify the backend attributes. It returns true if string is present on the UI and if the string is not present, it returns false. The function cannot veriy part of the string. It is a complete pattern matching function. It navigates to the entire HTML document and search the complete string for the given HTML tag. It cannot matches a text if any space before & after the string is present.

Syntax – **text() = 'Expected String'**

Application Under Test – Facebook

Example – To search for the text "Create an account" on the Facebook page.

HTML Code:



XPath Expression:

//span[text()='Create an account']

Normalize Space function in XPath –

It is used to ignore the space before and after the string.

Syntax – **normalize-space(function/attribute)**

Example - **//span [normalize-space(text())='Create an account']**

Contains function in XPath –

It is not a full pattern matching function. It is used to verify part of the String. It ignores the space before and after string automatically. The function is mainly used when we work with AJAX applications where object attribute changes dynamically.

Syntax – **contains(function/attribute, 'Expected Value')**

Example - **//span [contains(text()), 'Create an']**

Locating GUI Elements

In WebDriver, UI elements are located by using the method "**findElement(By.*locator*())**".

```
public class FacebookVerify {

    public String url = "http://www.facebook.com";
    public WebDriver driver = new FirefoxDriver();

    @BeforeTest
    public   void launchBrowser() {
             driver.get(url);
    }
```

```
        @Test
        public void verifyTagName() {
                String tagName;
                tagName = driver.findElement(By.xpath("//input[@id='u_0_5']")).getTagName();
                System.out.println(tagName);
        }

        @AfterTest
        public void terminateBrowser(){
                driver.quit();
        }
}
```

Other Methods of "By" Class[3]

Variation	Description	Sample
By.className	finds elements based on the value of the "class" attribute	findElement(By.className("someClassName"))
By.cssSelector	finds elements based on the driver's underlying CSS Selector engine	findElement(By.cssSelector("input#email"))
By.id	locates elements by the value of their "id" attribute	findElement(By.id("someId"))
By.linkText	finds a link element by the exact text it displays	findElement(By.linkText("REGISTRATION"))
By.name	locates elements by the value of the "name" attribute	findElement(By.name("someName"))
By.partialLinkText	locates elements that contain the given link text	findElement(By.partialLinkText("REG"))
By.tagName	locates elements by their tag name	findElement(By.tagName("div"))
By.xpath	locates elements via XPath	findElement(By.xpath("//input[@id='u_0_5']")

6. Accesing Forms in WebDriver

A form contains web elements like Input Box, Radio Button, Checkbox, Links, Drop-Down Box etc..

a) Input Box – To enter values in th text fields we use "**sendkeys()"** method.

Example- **driver.findElement(By.*xpath*("//input[@id='u_0_5']")).sendKeys("sonika.gupta@accenture.com")**

To delete values in th text fields we use"clear()**"** method.

Example- **driver.findElement(By.*xpath*("//input[@id='u_0_5']")).clear()**

Example – Select "Jan" from Month Drop-Down

b) RadioButton/CheckBox/Links – To click on radiobutton/checkbox/links we use "**click()"** method. It waits for a new page to load if applicable.

Example-
driver.findElement(By.*xpath*("//input[@id='u_0_d']")).click()

c) Drop-Down Box – To access the values in Drop-Down box, we need to perfom below actions –

a) Import the package org.openqa.selenium.support.ui.Select.
b) Instantiate the drop-down box as a "Select" object in WebDriver

Code - Select drpMonth = **new** Select(driver.findElement(By.*id*("month")));
drpMonth.selectByVisibleText("Jan");

Methods for Select Class[3] –

Methods	Description	Example
selectByVisibleText() deselectByVisibleText()	Selects/deselects the option that displays the text matching the parameter. **Parameter:** The exactly displayed text of a particular option	drpCountry.selectByVisibleText("INDIA");
selectByValue() deselectByValue()	Selects/deselects the option whose "value" attribute matches the specified parameter. **Parameter:** Value of the "value" attribute	drpCountry.selectByValue("1");
selectByIndex() deselectByValue()	Selects/deselects the option at the given index. **Parameter:** The index of the option to be selected.	drpCountry.selectByIndex(0);
isMultiple()	Returns TRUE if the drop-down element allows multiple selections at a time; FALSE if otherwise. Needs parameters needed.	if(drpCountry.isMultiple()){}
deselectAll()	Clears all selected entries. This is only valid when the drop-down element supports multiple selections. No parameters needed.	drpCountry.deselectAll();

- **close()** – This method closes the browser window that WebDriver currently controlls.
- **quit()** – This method closes all the browsers that WebDriver has opened.

- **Get Commands -** Get commands fetch various information about the page/element. Below are some of the "get" commands –

Example - **driver.get(“http://www.google.com”);**

Methods	Sample Usages
get()	It automatically opens a new browser window and fetches the page that you specify inside its parentheses. It is the counterpart of Selenium IDE's "open" command. The parameter must be a String object.
getTitle()	Needs no parameters Fetches the title of the current page Leading and trailing white spaces are trimmed Returns a null string if the page has no title
getPageSource()	Needs no parameters Returns the source code of the page as a String value
getCurrentUrl()	Needs no parameters Fetches the string representing the current URL that the browser is looking at
getText()	Fetches the inner text of the element that you specify

- **Navigate Commands –** Navigate commands allows to refresh, go-into and switch back and forth between different web pages.

Example - **driver.navigate().to(“http://www.facebook.com”);**
driver.navigate().refresh();
driver.navigate.back();
driver.navigate.forward();

Methods	Sample Usages
navigate().to()	It automatically opens a new browser window and fetches the page that you specify inside its parentheses. It does exactly the same thing as the get() method.
navigate().refresh()	Needs no parameters. It refreshes the current page.
navigate().back()	Needs no parameters Takes you back by one page on the browser's history.
navigate().forward()	Needs no parameters Takes you forward by one page on the browser's history.

7. Synchronization in WebDriver

There are two types of waits –

- **Implicit Wait** – is used to set the default waiting time throughout the program. This wait is simpler to code than Explicit Waits. It is declared in the instantiation part of the code. One additional package is required. Before using we need to import the below package in the code –

```
import java.util.concurrent.TimeUnit;
```

Include the below wait statement where ever required in the program –

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);[3]
```

- **Explicit Wait** – is used to set the waiting time for a particular instance only. Explicit Wait is done using the WebDriverWait and ExpectedCondition classes.

We need to import the below packages in the code –

```
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
```

Declare a WebDriverWait variable as "myWaitVar" –

```
WebDriverWait myWaitVar = new WebDriverWait(driver,10);
```

"myWaitVar" is used with ExpectedConditions where explicit wait is required.

```
myWaitVar.until(ExpectedCondition.visibilityofElementLocated(By.id("username")));
driver.findElement(By.id("username")).sendkeys("abc");
```

The ExpectedConditions class offers a wider set of conditions that can be used in conjunction with WebDriverWait's until() method –[3]

Methods	Description	Example
alertIsPresent()	waits until an alert box is displayed.	myWaitVar.until(ExpectedCondition.alertIsPresent());
elementToBeClickable()	waits until an element is visible and enabled.	myWaitVar.until(ExpectedCondition.elementToBeClickable(By.id("username")));
frameToBeAvailableAndSwitchToIt()	waits until the given frame is available, and then switches to it.	myWaitVar.until(ExpectedCondition.frameToBeAvailableAndSwitchToIt("viewIFRAME"));

8. Windows Handling in WebDriver

We have three kinds of windows which we can handle using WebDriver –

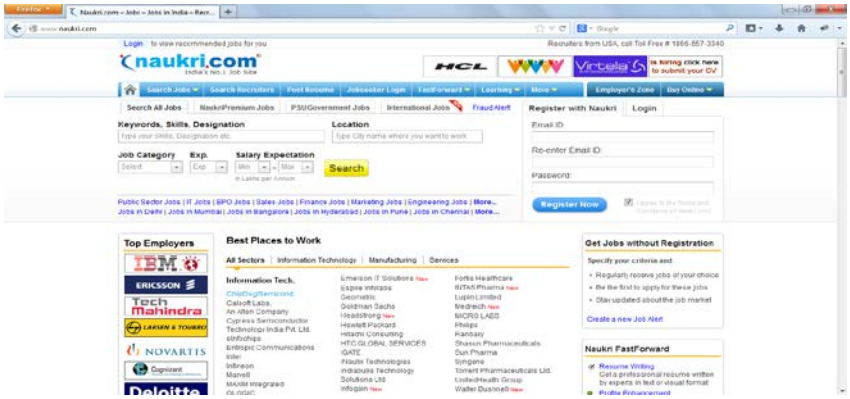
1) HTML Windows

2) Dialogs/Pop-Ups/Alerts
3) Hidden Windows

Some Web Applications has multiple windows. WebDriver supports moving between named windows using the "*switchto()*" method.

Syntax **driver.switchto().windows("windowHandleId");**
driver.switchto().alert();

Example: HTML Windows
Open "*www.naukri.com*" in any browser say – "*Mozilla Firefox*". We can see many child windows getting open along with the "www.*naukri.com*"

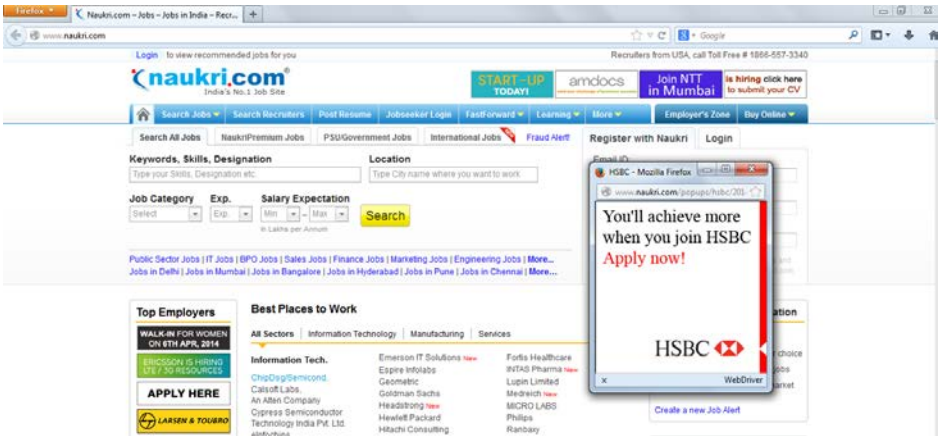


Code:

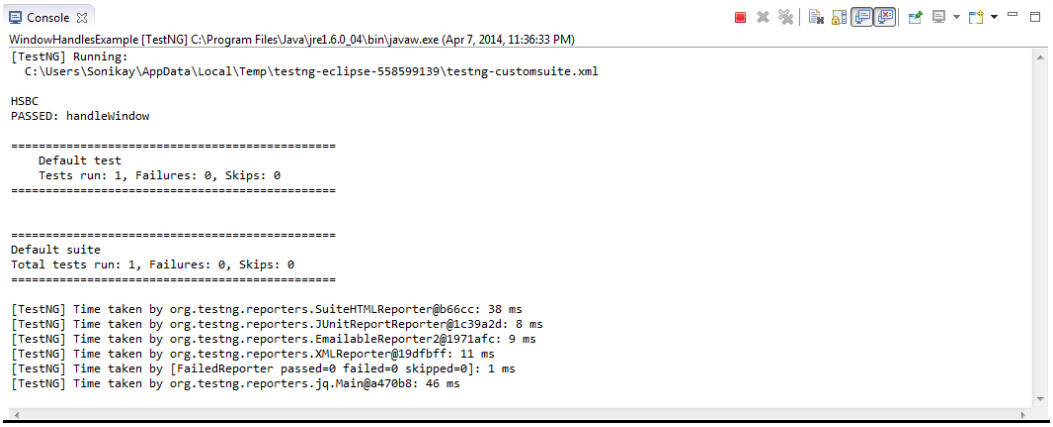
```
package testNGPackage;

import java.util.Iterator;
import java.util.Set;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;

public class WindowHandlesExample {

@Test
public void handleWindow()
{
        WebDriver d1 = new FirefoxDriver();
        d1.get("http://www.naukri.com");
        d1.manage().window().maximize();
```

```
                String parentWindow = d1.getWindowHandle();
                Set<String> set1 = d1.getWindowHandles();

                String childWindow = "";

                Iterator<String> it1 = set1.iterator();
                while(it1.hasNext())
                {
                        childWindow = it1.next();
                        if(!parentWindow.equals(childWindow))
                        {
                                System.out.println(d1.switchTo().window(childWindow).getTitle());
                                d1.switchTo().window(childWindow);
                                d1.close();
                        }
                }
                d1.switchTo().window(parentWindow);
        }
    }
```

Output:



Console:



Example: Alerts

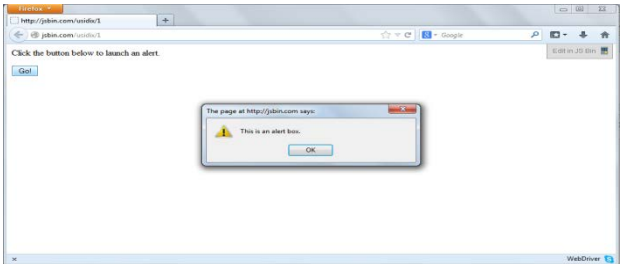
Alerts can be handled using three methods –

- **accept()** – this method is used to accept and close the alert box.
- **dismiss()** – this method is used to reject and close the alert box.
- **gettext()**– this method is used to retrieve the message of the alert box.

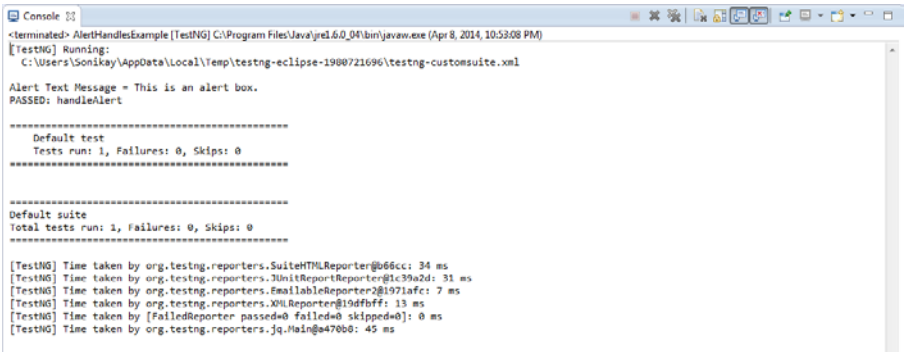
Code:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.annotations.Test;

public class AlertHandlesExample {

@Test
public void handleAlert() {
        WebDriver driver = new FirefoxDriver();
        driver.get("http://jsbin.com/usidix/1");
        driver.findElement(By.xpath("//input[@value='Go!']")).click();
        System.out.println("Alert Text Message = " + driver.switchTo().alert().getText());
        driver.switchTo().alert().accept();
        }
}
```

Output:



Console:



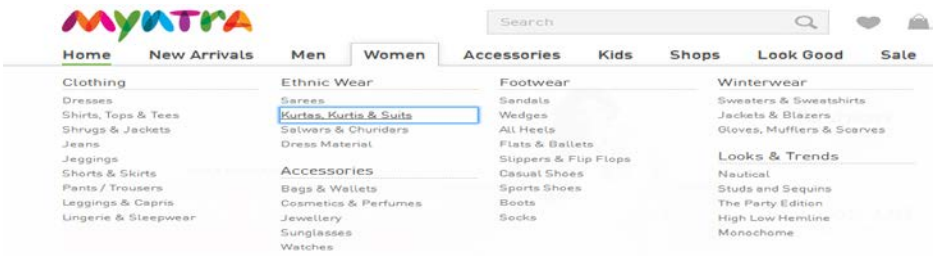
Advanced Programming using WebDriver

WebDriver supports Keyboard operations and Mouse Event operations. Handling special keyboard and mouse events are done using the Advanced User Interactions API. It contains the Actions classe that is needed when executing these events. The following are the most commonly used keyboard and mouse events provided by the Actions class. Below are the commonly used keyboard and mouse events provided by Actions class[3].

Methods	Description
clickAndHold()	Clicks (without releasing) at the current mouse location.
contextClick()	Performs a context-click at the current mouse location.
doubleClick()	Performs a double-click at the current mouse location.
dragAndDrop(source, target)	Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.
dragAndDropBy(source, x-offset, y-offset)	Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.
keyDown(modifier_key)	Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed.
keyUp(modifier _key)	Performs a key release.
moveByOffset(x-offset, y-offset)	Moves the mouse from its current position (or 0,0) by the given offset.
moveToElement(toElement)	Moves the mouse to the middle of the element.
release()	Releases the depressed left mouse button at the current mouse location
sendKeys(onElement, charsequence)	Sends a series of keystrokes onto the element.

Example:

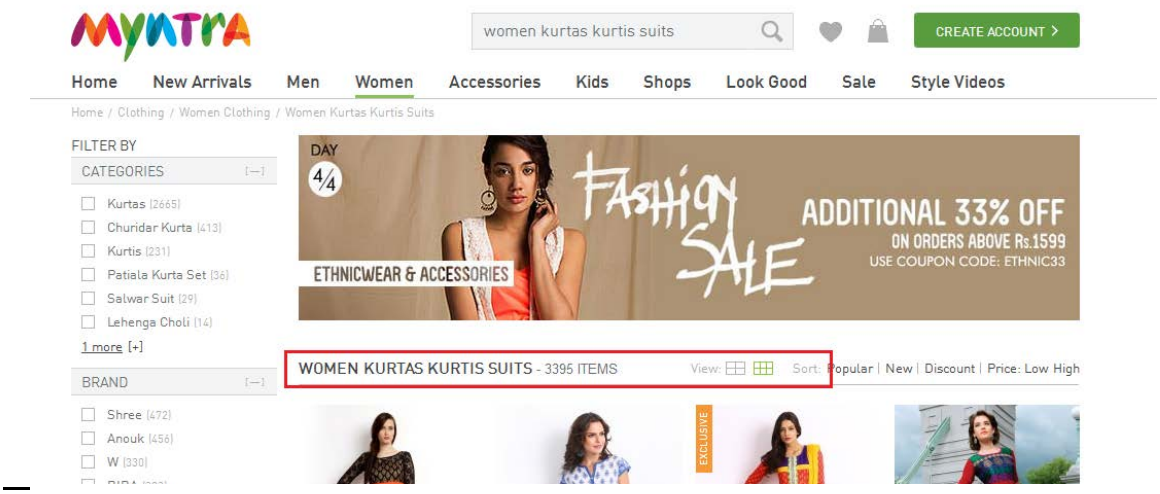
Open *www.myntra.com* and try to click on the sub link as highlighted in the below image –



Code:

```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class Act {

        public static void main(String args[]) {

                WebDriver driver = new FirefoxDriver();
                driver.manage().window().maximize();
                driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
                driver.get("http://www.myntra.com/");
                WebElement ele = driver.findElement(By.xpath("//a[4][text()='Women']"));
                Actions act = new Actions(driver);
                act.moveToElement(ele).perform();
                driver.findElement(By.xpath("//a[contains(text(),'Kurtas, Kurtis &
        Suits')]")).click();
        }
}
```

Output:



Explaining the Code :

Importing Packages

We need to import the below package –

org.openqa.selenium.interactions.Actions;

Instantiating a new Actions Object

Actions act = new Actions(driver);

To perform a mouse-over operation we can use "*moveToElement()*" method

Conclusion

WebDriver is a tool for testing web applications across different browsers using different programming languages.

We can make powerful tests as WebDriver allows to use a programming language in designing the tests. It is faster than Selenium RC because of its simpler architecture. It directly talks to the browser while Selenium RC needs the help of the RC Server in order to do so. Web Driver's API is more concise than Selenium RC's. The only drawbacks of WebDriver are:

a) It cannot readily support new browsers, but Selenium RC can.
b) It does not have a built-in command for automatic generation of test results.

TestNG is a testing framework that is capable of making selenium tests easier to understand and of generating reports that are easy to understand.

The main advantages of TestNG over JUnit are the following:

a) Annotations are easier to use and understand.
b) Test cases can be grouped more easily.
c) TestNG allows us to create parallel tests.

The Console window in Eclipse generates a text-based result while the TestNG window is more useful because it gives a graphical output of the test result. TestNG is capable of generating HTML-based reports. Annotations can use parameters just like the usual Java methods[3].

References

[1] http://www.seleniumhq.org
[2] http://qeworks.com/selenium-webdriver-architecture/
[3] http://www.guru99.com/selenium-tutorial.html
[4] http://www.qtpselenium.com/
[5] http://www.learn-automation.com/