AUTOMATION TESTING

GUIDELINE

JUN 2016

# Contents

[Contents 2](#_Toc455674531)

[1. BASIC TrAINING 4](#_Toc455674532)

[**1.1.** **Installation** 4](#_Toc455674533)

[1.1.1 Install Eclipse 4](#_Toc455674534)

[1.1.2 Install TestNG 4](#_Toc455674535)

[1.1.3 Install Maven for eclipse 6](#_Toc455674536)

[1.1.3 Install Selenium WebDriver Java Client 8](#_Toc455674537)

[1.1.4 Basic git and guide to clone code, create branch and push code to git: 10](#_Toc455674538)

[**1.2.** **Inspectors Tools & Locators** 14](#_Toc455674539)

[**1.3.** **WebDriver Commands** 14](#_Toc455674540)

[1.3.1 Browser Commands 14](#_Toc455674541)

[1.3.2 Navigation commands 15](#_Toc455674542)

[1.3.3 WebElement commands 15](#_Toc455674543)

[1.3.4 FindElement commands: 19](#_Toc455674544)

[1.3.5 CheckBox & Radio Button Operations 22](#_Toc455674545)

[1.3.6 DropDown & Multiple Select Operations 23](#_Toc455674546)

[1.3.7 Wait Commands: 26](#_Toc455674547)

[**1.4** **Alerts & Windows** 28](#_Toc455674548)

[1.4.1 Switch Commands 28](#_Toc455674549)

[1.4.2 Handling of Alerts 29](#_Toc455674550)

[1.4.3 Handling Iframes 31](#_Toc455674551)

[**1.4.** **Action & Robot Class** 36](#_Toc455674552)

[1.5.1 Drag And Drop in Selenium WebDriver 36](#_Toc455674553)

[1.5.2 Mouse Hover action in Selenium WebDriver 37](#_Toc455674554)

[**1.6** **Tips and Tricks** 39](#_Toc455674555)

[1.6.1 HTTP Proxy Authentication with Selenium WebDriver: 39](#_Toc455674556)

[2 ADVANCED TrAINING 40](#_Toc455674557)

[**2.1**  **TestNG Framework** 40](#_Toc455674558)

[2.1.1 TestNG Introduction 40](#_Toc455674559)

[2.1.2 Basic Annotation, Groups and Dependent 40](#_Toc455674560)

[2.1.3 TestNG prioritizing and Sequencing 46](#_Toc455674561)

[2.1.4 Skipping a Test Case 47](#_Toc455674562)

[2.1.5 TestNG Parameter, Data Provider and Factory 47](#_Toc455674563)

[2.1.6 Cross Browser and Parallel Testing 51](#_Toc455674564)

[2.1.7 Testng.xml 53](#_Toc455674565)

[**2.2 Data Driven Approach** 53](#_Toc455674566)

[2.2.1 Download and Add POI library 53](#_Toc455674567)

[2.2.2 Data Driven Framework (Apache POI - Excel) 56](#_Toc455674568)

[**2.3 Log4j Logging** 62](#_Toc455674569)

[2.3.1 Download Log4j and add log4j.jar 63](#_Toc455674570)

[2.3.2 Test Case with Log4j 64](#_Toc455674571)

[2.3.3 Log4j LogManager 66](#_Toc455674572)

[2.3.4 Log 4j Appenders 69](#_Toc455674573)

[2.3.5 Log4j Loggers 72](#_Toc455674574)

[**2.4 Page Object Model (POM) and Page Factory** 74](#_Toc455674575)

[2.4.1 Page Object Model 74](#_Toc455674576)

[2.4.2 Page Factory 78](#_Toc455674577)

# BASIC TrAINING

## **Installation**

### 1.1.1 Install Eclipse

*Step 1: Download eclipse*

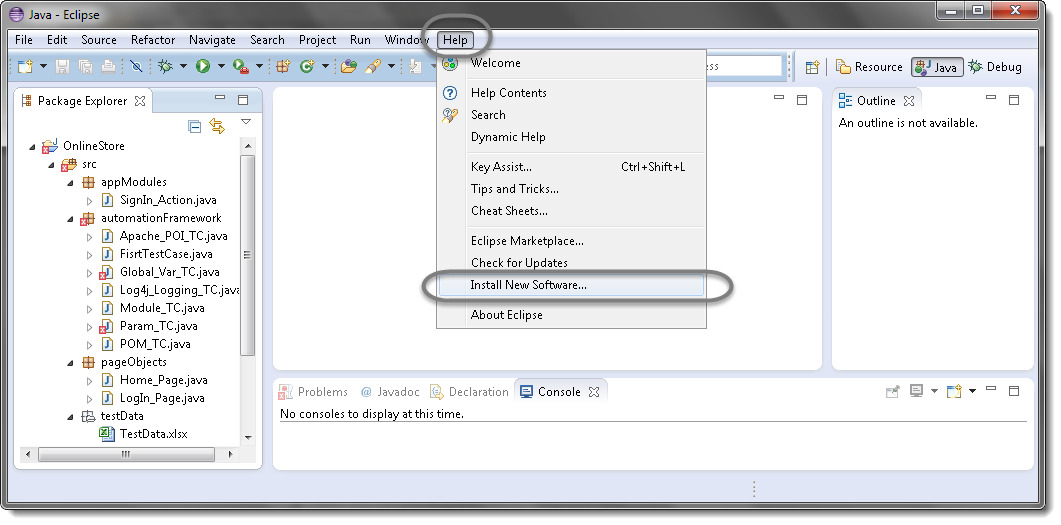
* Go to page: <https://eclipse.org/downloads/> and download whatever eclipse version you want (I am using luna and kepler versions)

*Step 2: Install eclipse*

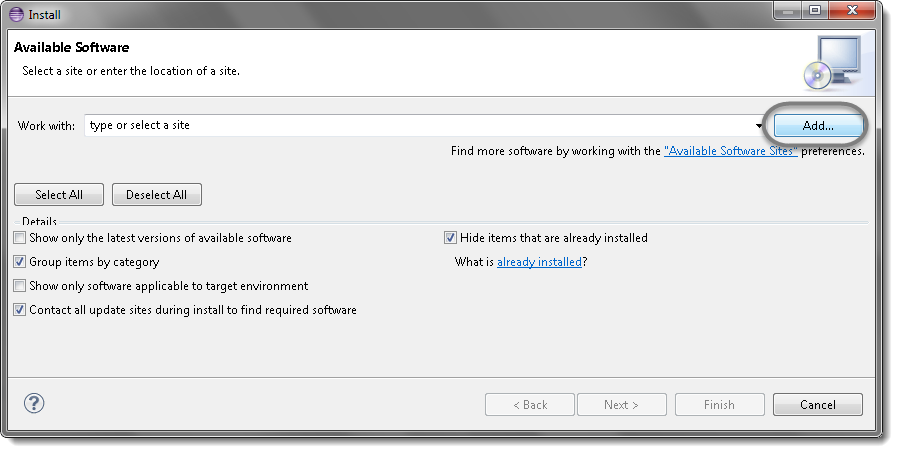
*Step 3: Open Eclipse*

### 1.1.2 Install TestNG

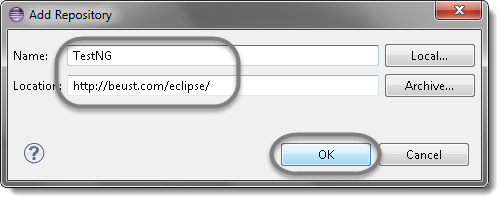
*Step 1: Launch the Eclipse IDE and from Help menu, click “****Install New Software****”.*



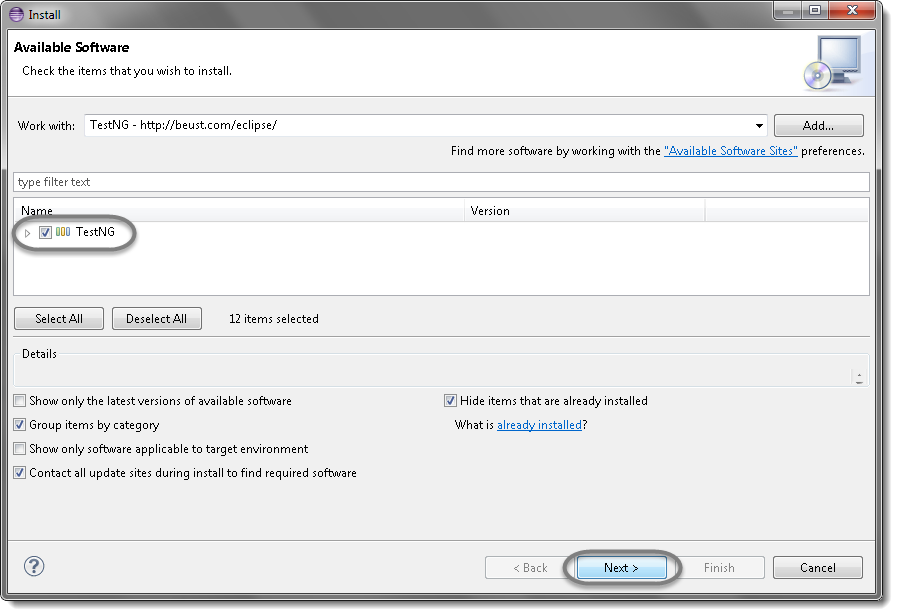
*Step2: You will see a dialog window, click “Add” button.*



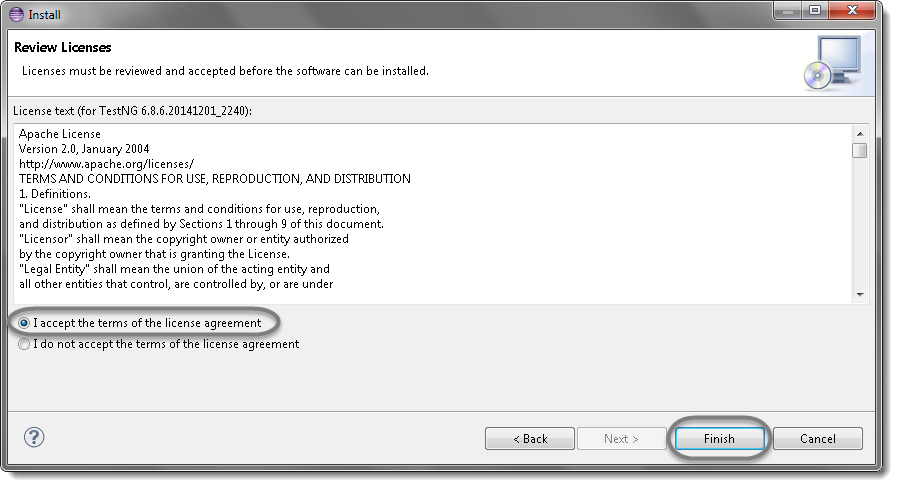
*Step3: Type name as you wish, take “****TestNG****” and type “****http://beust.com/eclipse/****” as location. Click OK.*

**

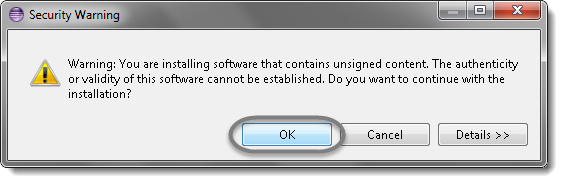
*Step4: Select TestNG and press “****Next****” button.*



*Step5. Click “****I accept the terms of the license agreement****” then click****Finish****.*

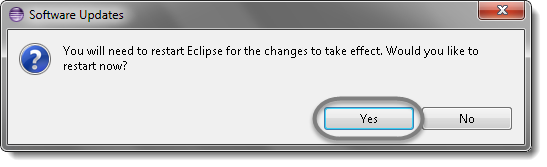


(Note: You may or may not encounter a Security warning, if in case you do just click **OK**.



*Step6. Click****Next****again on the succeeding dialog box until it prompts you to Restart the Eclipse.*

*Step7. You are all done now, just Click****Yes to restart eclipse***

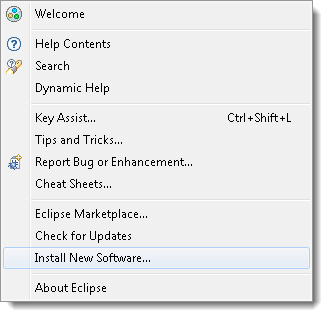


### 1.1.3 Install Maven for eclipse

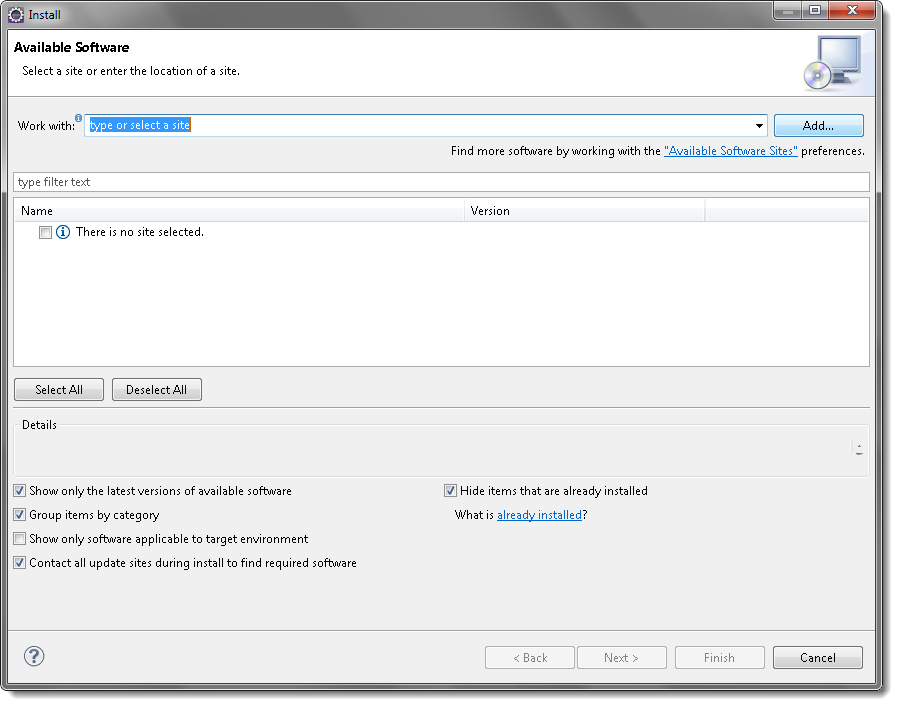
Note: with the newest version of eclipse, you do not need to install Maven. It is integrated in eclipse

Step1.Open eclipse (choose workspace A)

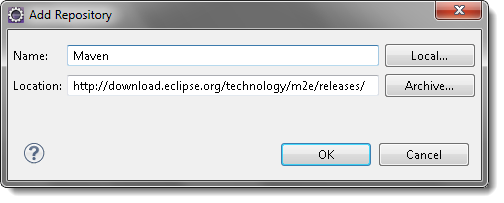
Step2. Click on the ***Help***from the top menu in ***Eclipse***and select ‘***Install New Software’***



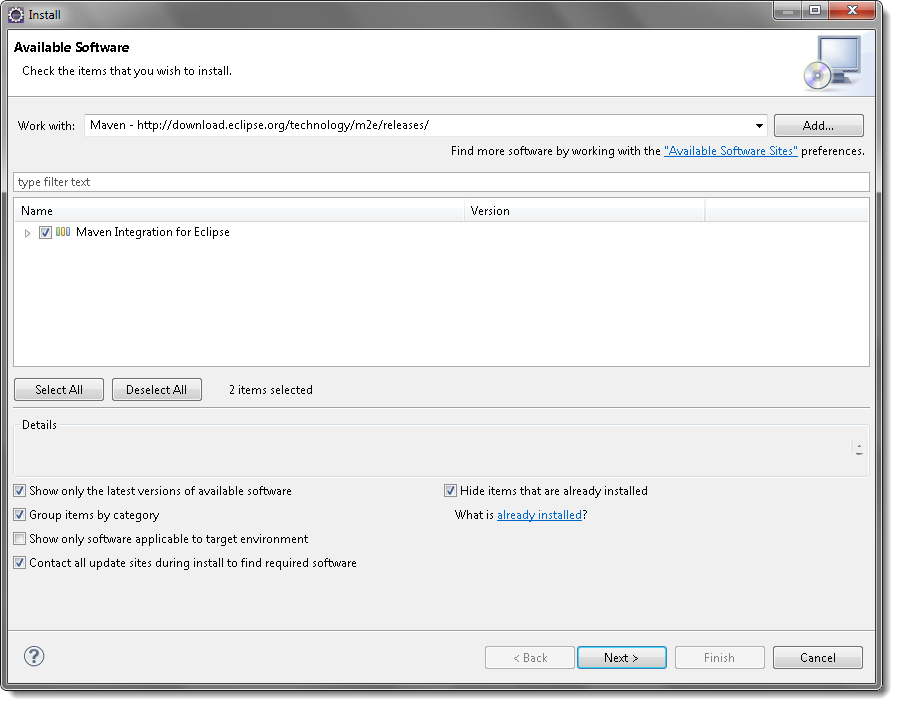
Step3. Click on the ***Add*** button on the newly opened window.



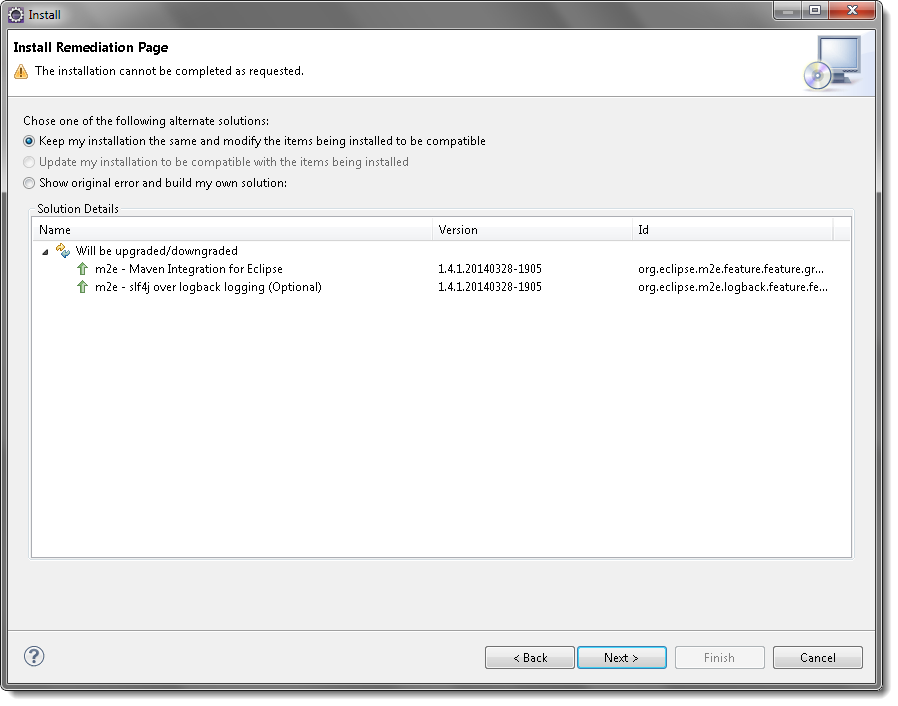
Step4. In the Name box, type ‘**Maven**‘ and in the Location box, type ‘<http://download.eclipse.org/technology/m2e/releases>" or <http://download.eclipse.org/technology/m2e/milestones/1.0>



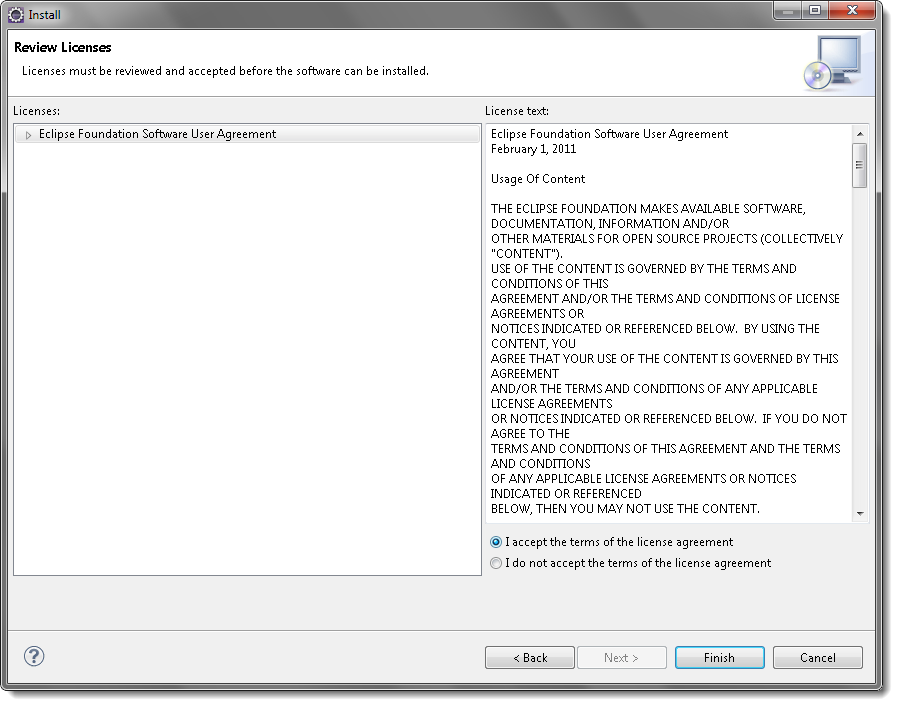
Step5. A check-box will appear in the pop window, **Check** the check-box and click on **Next** button.



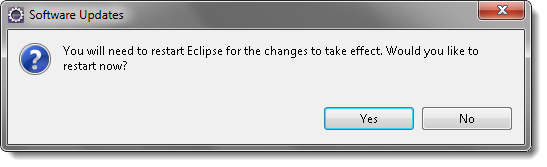
Step6. Keep the default settings and click on **Next** button.



Step7. Accept the ‘**Terms and Conditions**’ and move forward by clicking on **Finish** button.

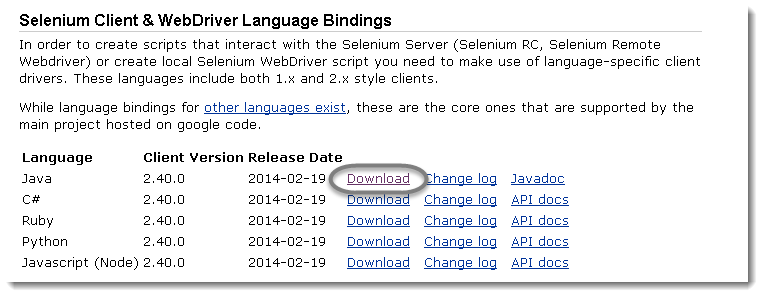


Step8. Once the installation is finished, it will ask you to restart the Eclipse. Please click on **Yes**, so that changes can be reflected properly.

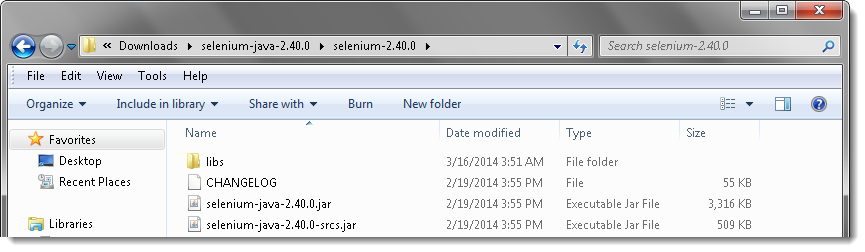


### 1.1.3 Install Selenium WebDriver Java Client

*Step1. Go to:* [*http://docs.seleniumhq.org/download/*](http://docs.seleniumhq.org/download/)*. On that page click on ‘****Download****’ link of java client driver as shown in the below image.*

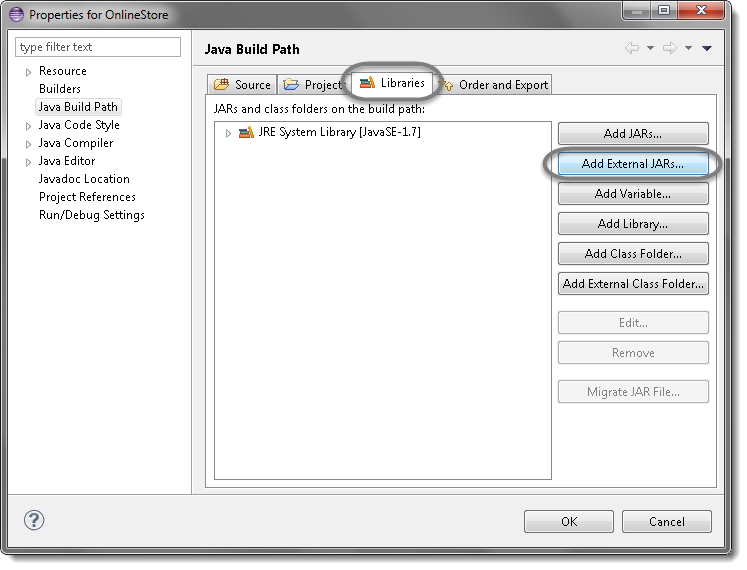
****

*Step2. Extract this file, after extract successfully*

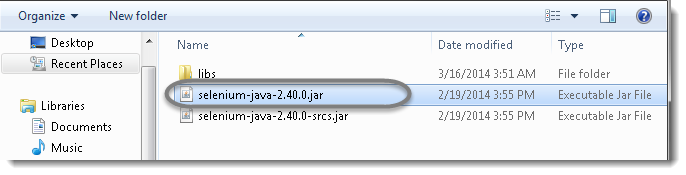
****

*Step3.* *Launch the Eclipse IDE & Create a Workspace*

*Step4.* Right click*on Project 🡪 Select Properties*🡪Java build path*. Then navigate to Libraries tab and*click*Add External JARs.*



***Step5.*** *Add Selenium Java jar, you may add the source file too.*

****

### 1.1.4 Basic git and guide to clone code, create branch and push code to git:

- Download and install gitbash: Go to: <https://git-scm.com/downloads> to download and install gitbash

- Create repository, clone source code from bitbucket, create a new branch to code and push code to bitbucket:

***+ Step 1: Open gitbash application***

***+ Step 2: Go to the folder that you want to create a repository*** by command:

cd [path-to-folder-A]

Ex: cd /E/niteco-training-automation

***+ Step 3: Initiate git*** by command:

git init

git remote add origin <http://bitbucket.niteco.se:7990/scm/qa/qa-automation-training.git>

***+ Step 4: Clone source code from gitlab to repository*** by command:

git clone <http://bitbucket.niteco.se:7990/scm/qa/qa-automation-training.git>

Note: At this step, you maybe have to enter your username and password in bitbucket for authorization

**+ Step 5: Get the latest code and switch to master branch:**

* Go to the folder that you want to create a repository by command:

cd [path-to-folder-A]

* Get latest code by command:

git fetch

Note: At this step, you maybe have to enter your username and password in bitbucket for authorization

* Checkout master branch by command:

git checkout master

* Get the latest code of master branch from bitbucket by command:

git rebase origin/master

***+ Step 6: Create new branch from maser branch to code***

* Go to the folder that you want to create a repository by command:

cd [path-to-folder-A]

* Create new branch for your automation training course **(named rule is: ‘[yourname]AutomationTraining** ’**. Please follow this rule when creating your own branch**. Ex:

git branch tuluuAutomationTraining

* Switch to tuluuAutomationTraining branch to continue coding by command:

git checkout tuluuAutomationTraining

***+ Step 7: Coding in* tuluuAutomationTraining *branch***

* ***Create your own project and code. Rules:***

***+ Project name: [your name]-automation-training***

***+ Each exercise will be a separate package (Each package can be include many classes due to your coding)***

***+ Step 8: After coding finishes, we need to push code from your branch to bitbucket:***

* Go to the folder that you want to create a repository by command:

cd [path-to-folder-A]

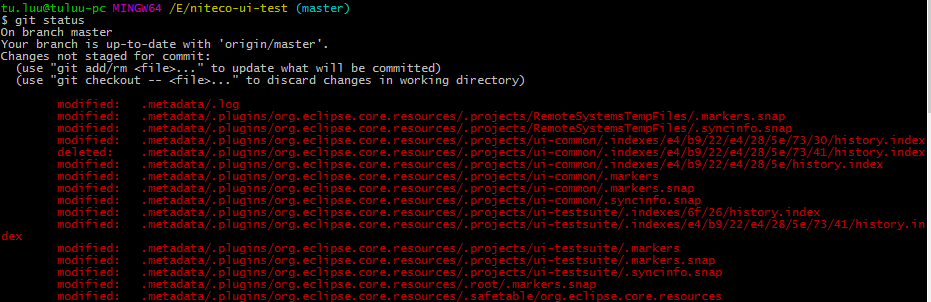
* Check status of branch by command:

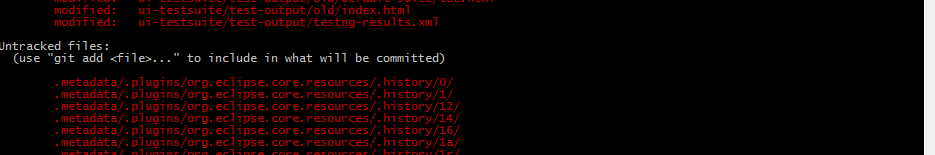
git status

* Add changes by commands:

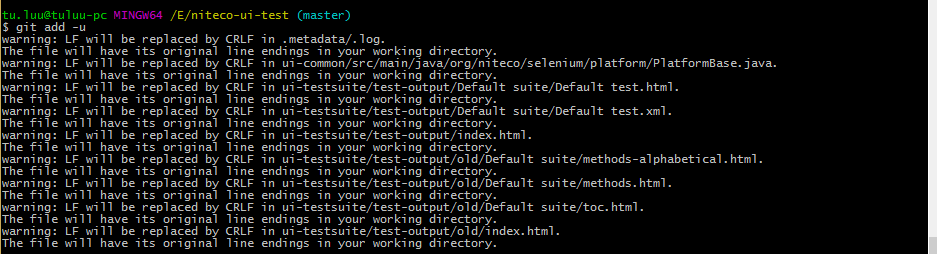
git add –u

(**Note that**: after command “git status”, we will get 2 parts of changed files: *“Changes not staged for commit”* and *“Untracked files:”*

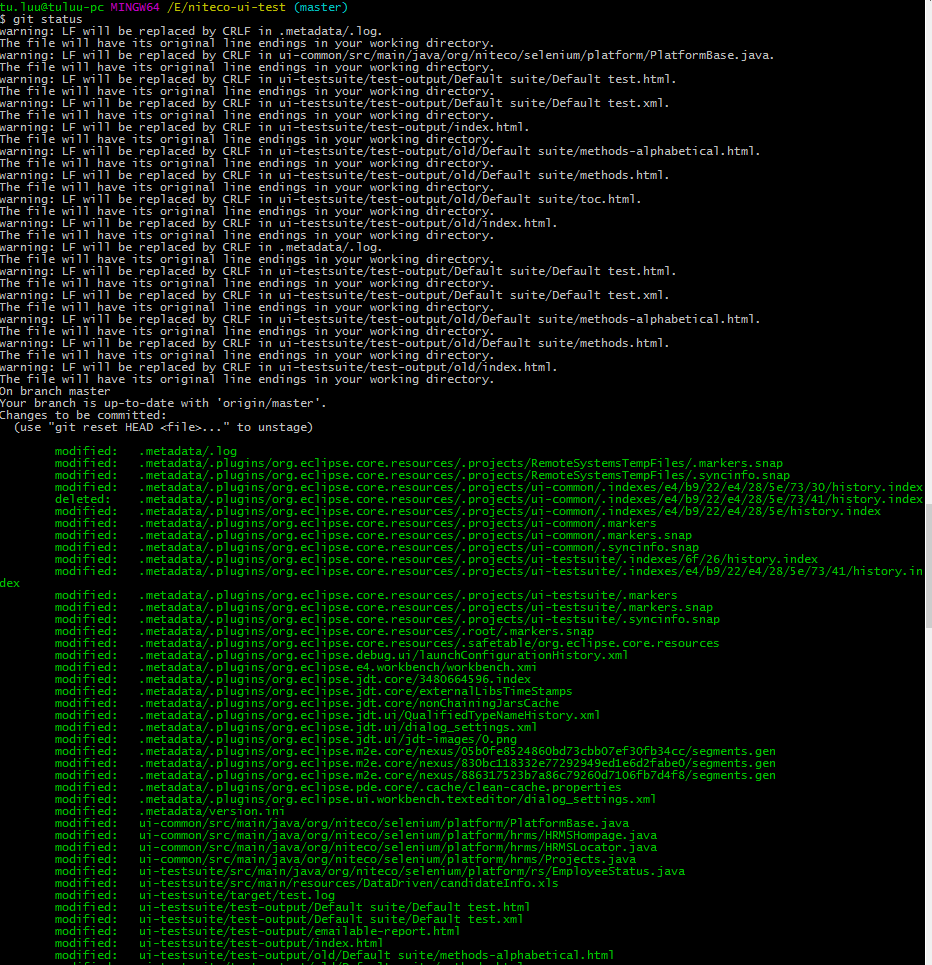


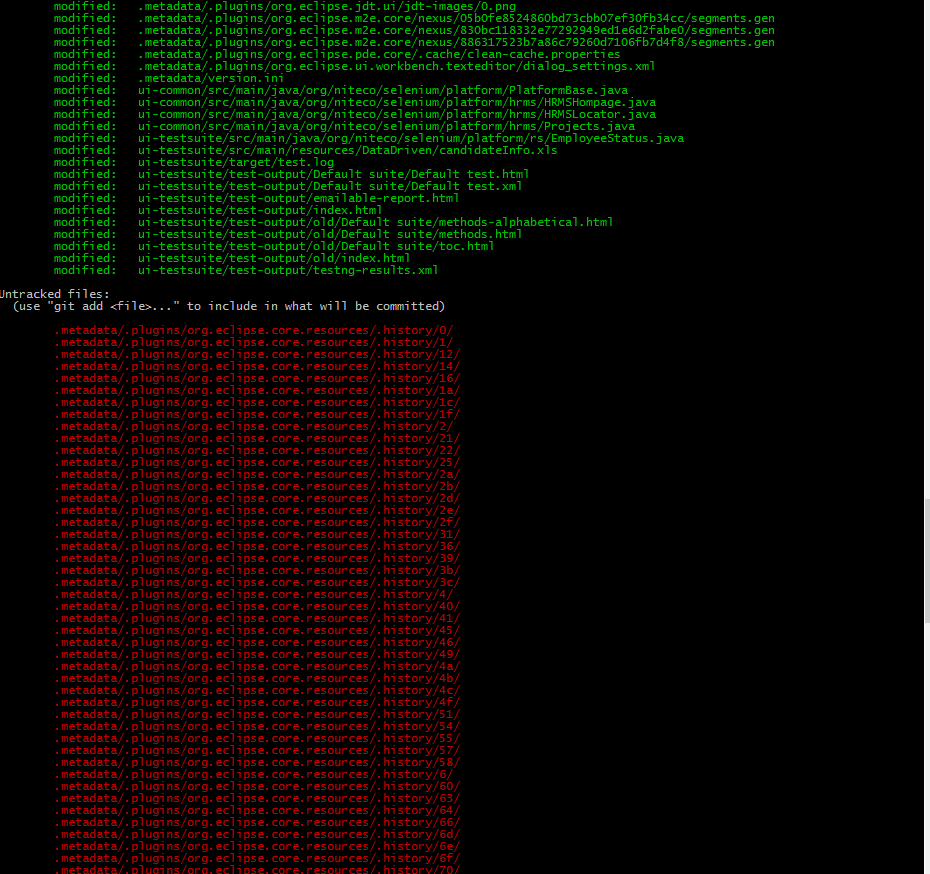


*Command “git add -u” only add ALL the changed files in “Changes not staged for commit”.*



*Check which changed files added successfully (the green files are added, the red files are not added) by command: git status*





*We have to check in “Untracked files:” part and add each changed file that we want to add by command: git add [file]. Then check to make sure that file was added successfully by command: git status*



* Commit changes:

git commit –m “description of this commit”



* Push commit to your branch in gitlab:

git push origin tuAutomationTraining:tuAutomationTraining

**Note: Some other git commands:**

+ Command to check which branch you are in: git branch

+ Command to delete a branch name ‘branchA’: git branch branchA -D

+ Command to see the commits: git log –oneline

+ Command to revise your code to origin: git stash

## **Inspectors Tools & Locators**

**Install FireBug: This is an add-ons in firefox, so you can install it as every other firefox add-ons**

**Install FirePath: This is an add-ons in firefox, so you can install it as every other firefox add-ons**

## **WebDriver Commands**

### 1.3.1 Browser Commands

* **Get Command**

*get(String arg0) : void* – This method ***Load***a new web page in the current browser window. Accepts String as a parameter and returns nothing.

*Command*– ***driver.get(appUrl);***

* **Get Title Command**

***getTitle() : String*** – This method fetches the ***Title*** of the current page. Accepts nothing as a parameter and returns a String value.

***Command – driver.getTitle();***

As the return type is String value, the output must be stored in String object/variable.

* **Get Current URL Command**

***getCurrentUrl() : String*** – This method fetches the string representing the ***Current URL*** which is opened in the browser. Accepts nothing as a parameter and returns a String value.

***Command – driver.getCurrentUrrl();***

As the return type is String value, the output must be stored in String object/variable.

* **Get Page Source Command**

***getPageSource() : String*** – This method returns the ***Source Code***of the page. Accepts nothing as a parameter and returns a String value.

***Command – driver.getPageSource();***

As the return type is String value, the output must be stored in String object/variable.

* **Close Command**

***close() : void*** – This method **Close** only the current window the WebDriver is currently controlling. Accepts nothing as a parameter and returns nothing.

***Command – driver.close();***

Quit the browser if it’s the last window currently open.

* **Quit Command**

***quit() : void*** – This method **Closes** all windows opened by the WebDriver. Accepts nothing as a parameter and returns nothing.

***Command – driver.quit();***

Close every associated window.

**Exercise:**

* *Launch a new Firefox browser.*
* *Open* [*http://newtours.demoaut.com/*](http://newtours.demoaut.com/)
* *Get Page Title name and Title length*
* *Print Page Title and Title length on the Eclipse Console.*
* *Get Page URL and verify if the it is a correct page opened*
* *Get Page Source (HTML Source code) and Page Source length*
* *Print Page Length on Eclipse Console.*
* *Close the Browser*

### 1.3.2 Navigation commands

* **Navigate To Command**

***to(String arg0) : void*** – This method ***Loads***a new web page in the current browser window. It accepts a String parameter and returns nothing.

Command – ***driver.navigate().to(appUrl);***

It does exactly the same thing as the ***driver.get(appUrl)*** method. Where **appUrl**is the website address to load. It is best to use a fully qualified URL.

* **Forward Command**

***forward() : void*** – This method does the same operation as clicking on the ***Forward Button*** of any browser. It neither accepts nor returns anything.

Command – ***driver.navigate().forward();***

Takes you forward by one page on the browser’s history.

* **Back Command**

***back() : void*** – This method does the same operation as clicking on the ***Back Button*** of any browser. It neither accepts nor returns anything.

Command – ***driver.navigate().back();***

Takes youback by one page on the browser’s history.

* **Refresh Command**

***refresh() : void*** – This method ***Refresh*** the current page. It neither accepts nor returns anything.

Command – ***driver.navigate().refresh();***

Perform the same function as pressing F5 in the browser.

**Exercise:**

* *Launch new Browser*
* *Open* [*http://newtours.demoaut.com/*](http://newtours.demoaut.com/) *website*
* *Click on Registration link using “*driver.findElement(By.*xpath* //\*[contains(text(),'SIGN-ON')]”)).click();*“*
* *Come back to Home page (Use ‘Back’ command)*
* *Again go back to Registration page (This time use ‘Forward’ command)*
* *Again come back to Home page (This time use ‘To’ command)*
* *Refresh the Browser (Use ‘Refresh’ command)*
* *Close the Browser*

### 1.3.3 WebElement commands

WebElement represents an HTML element. HTML documents are made up by HTML elements. HTML elements are written with a start tag, with end tag, with the content in between: <tagname> content </tagname>

* **Clear Command**

*clear( ) : void* – If this element is a text entry element, this will clear the value. This method accepts nothing as a parameter and returns nothing.

*Command* – ***element.clear();***

*Example:*

|  |
| --- |
| WebElement element = driver.findElement(By.id("UserName"));  element.clear();  //Or can be written as  driver.findElement(By.id("UserName")).clear(); |

* **Sendkeys Command**

*sendKeys(CharSequence… keysToSend ) : void* – This simulate typing into an element, which may set its value. This method accepts CharSequence as a parameter and returns nothing.

*Command* – ***element.sendKeys(“text”);***

*Example:*

|  |
| --- |
| **WebElement element = driver.findElement(By.id("UserName"));**  **element.sendKeys("ToolsQA");**  **//Or can be written as**  **driver.findElement(By.id("UserName")).sendKeys("ToolsQA");** |

* **Click Command**

*click( ) : void* – This simulates the clicking of any element. Accepts nothing as a parameter and returns nothing.

*Command* – ***element.click();***

Clicking is perhaps the most common way of interacting with web elements like text elements, links, radio boxes and many more**.**

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.linkText("ToolsQA"));  element.click();  //Or can be written as  driver.findElement(By.linkText("ToolsQA")).click(); |

***Note****: Most of the time we click on the links and it causes a new page to load, this method will attempt to wait until the page has loaded properly before handing over the execution to next statement. But If click() causes a new page to be loaded via an event or is done by sending a native event for example through javascript, then the method will not wait for it to be loaded.*

*There are some preconditions for an element to be clicked. The element must be****Visible****and it must have a****Height and Width****greater than 0.*

* **IsDisplayed Command**

*isDisplayed( ) : boolean* – This method determines if an element is currently being displayed or not. This accepts nothing as a parameter but returns boolean value(true/false).

*Command* – ***element.isDisplayed();***

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.id("UserName"));  boolean status = element.isDisplayed();  //Or can be written as  boolean staus = driver.findElement(By.id("UserName")).isDisplayed(); |

***Note***: This will return***true****if the element is present on the page and throw a* ***NoSuchElementFound****exception if the element is not present on the page. This refers the property of the element, sometimes the element is present on the page but the property of the element is set to****hidden****, in that case this will return****false****, as the element is present in the DOM but not visible to us.*

* **IsEnabled Command**

*isEnabled( ) : boolean* – This determines if the element currently is ***Enabled or not***? This accepts nothing as a parameter but returns boolean value(true/false).

*Command*– ***element.isEnabled();***

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.id("UserName"));  boolean status = element.isEnabled();  //Or can be written as  boolean staus = driver.findElement(By.id("UserName")).isEnabled();  //Or can be used as  WebElement element = driver.findElement(By.id("userName"));  boolean status = element.isEnabled();  // Check that if the Text field is enabled, if yes enter value  if(status){  element.sendKeys("ToolsQA");  } |

* **IsSelected Command**

*isSelected( ) : boolean* – Determine whether or not this element is selected or not. This accepts nothing as a parameter but returns boolean value(true/false).

*Command* – ***element.isSelected();***

This operation only applies to input elements such as ***Checkboxes***, ***Select Options*** and ***Radio Buttons***. This returns ***True*** if the element is currently*selected or checked*, ***false*** otherwise.

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.id("Sex-Male"));  boolean status = element.isSelected();  //Or can be written as  boolean staus = driver.findElement(By.id("Sex-Male")).isSelected(); |

* **Submit Command**

*submit( ) : void*– This method works well/better than the *click()* if the current element is a form, or an element within a form. This accepts nothing as a parameter and returns nothing.

*Command* – ***element.submit();***

If this causes the current page to change, then this method will wait until the new page is loaded.

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.id("SubmitButton"));  element.submit();  //Or can be written as  driver.findElement(By.id("SubmitButton")).submit(); |

* **getText Command**

*getText( ) : String*– This method will fetch the visible (i.e. not hidden by CSS) innerText of the element. This accepts nothing as a parameter but returns a String value.

*Command*– ***element.submit();***

This returns an innerText of the element, including sub-elements, without any leading or trailing whitespace.

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.xpath("anyLink"));  String linkText = element.getText(); |

* **getTagName Command**

*getTagName( ) : String*– This method gets the tag name of this element. This accepts nothing as a parameter and returns a String value.

*Command*– ***element.getTagName();***

This does not return the value of the name attribute but return the tag for e.g. “***input***”*for the element****<input name="foo"/>***.

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.id("SubmitButton"));  String tagName = element.getTagName();  //Or can be written as  String tagName = driver.findElement(By.id("SubmitButton")).getTagName(); |

* **getCSSValue Command**

*getCssvalue( ) : String*– This method Fetch CSS property value of the give element. This accepts nothing as a parameter and returns a String value.

*Command*– ***element.getCssValue();***

* **getAttribute Command**

*getAttribute(String Name) : String*– This method gets the value of the given attribute of the element. This accepts the String as a parameter and returns a String value.

*Command*– ***element.getAttribute();***

Attributes are Ids, Name, Class extra and using this method you can get the value of the attributes of any given element.

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.id("SubmitButton"));  String attValue = element.getAttribute("id"); //This will return "SubmitButton" |

* **getSize Command**

*getSize( ) : Dimension*– This method fetch the width and height of the rendered element. This accepts nothing as a parameter but returns the Dimension object.

*Command* – ***element.getSize();***

This returns the size of the element on the page.

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.id("SubmitButton"));  Dimension dimensions = element.getSize();  System.out.println(“Height :” + dimensions.height + ”Width : "+ dimensions.width); |

* **getLocation Command**

*getLocation( ) : Point*– This method locate the location of the element on the page. This accepts nothing as a parameter but returns the Point object.

*Command*– ***element.getLocation();***

This returns the ***Point object***, from which we can get X and Y coordinates of specific element.

Example:

|  |
| --- |
| WebElement element = driver.findElement(By.id("SubmitButton"));  Point point = element.getLocation();  System.out.println("X cordinate : " + point.x + "Y cordinate: " + point.y); |

### ****1.3.4 FindElement commands:****

* **By id**

*id(String id) : By* – This is the most efficient and preferred way to locate an element, as most of the times *IDs* are unique. It takes a parameter of String which is a *Value of ID attribute* and it returns a ***BY object*** to ***findElement()*** method.

*Command*– ***driver.findElement(By.id(“Element ID”));***

With this strategy, If no element has a matching id attribute, a ***NoSuchElementException*** will be raised.

Example:



WebElement element = driver.findElement(By.id("submit"));

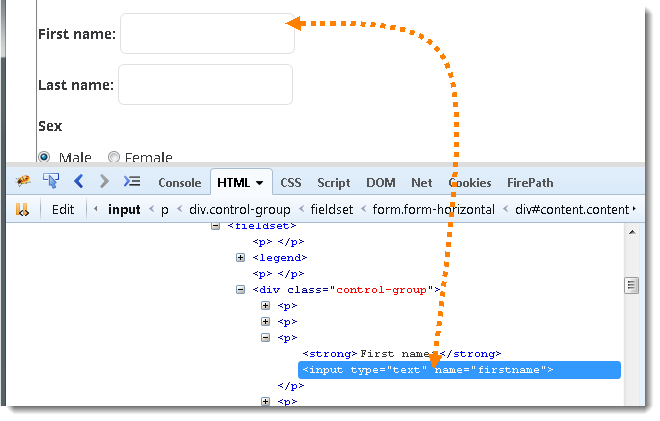
* **By Name**

*name(String name) : By* –when UI developer make it having non-unique names on a page or auto-generating the names. It takes a parameter of String which is a *Value of NAME attribute* and it returns a ***BY object*** to ***findElement()*** method.

*Command*– ***driver.findElement(By.name(“Element NAME”));***

With this strategy, the first element with the name attribute value matching the location will be returned. If no element has a matching name attribute, a ***NoSuchElementException***will be raised.

Example:



**WebElement element = driver.findElement(By.name("firstname"));**

* **By ClassName**

*className(String className) : By* – This finds elements based on the value of the *CLASS* attribute.

*Command* – ***driver.findElement(By.className(“Element CLASSNAME”));***

If an element has many classes then this will match against each of them.

****

Example:

WebElement parentElement = driver.findElement(By.className("btn btn-info"));

* **By TagName**

*tagName(String name) : By* – With this you can find elements by their *TAGNAMES*.

*Command*– ***driver.findElement(By.tagName(“Element TAGNAME”));***



Example:

WebElement element = driver.findElement(By.tagName("button"));

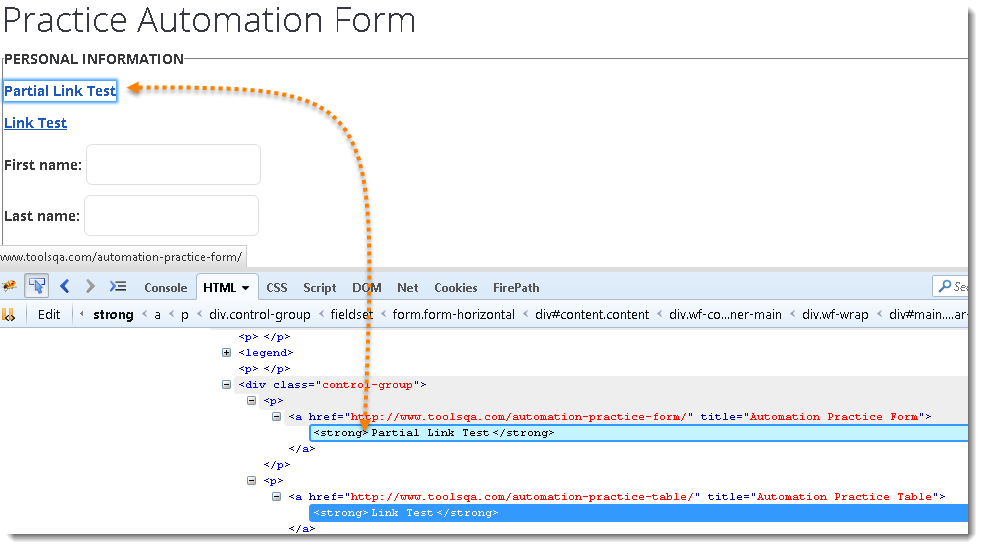
* **By LinkText & PartialLinkText**

*LinkText(String linkText) : By* – With this you can find elements of *“****a****” tags(****Link****)* with the link names. ***object*** to ***findElement()*** method.

*partialLinkText(String linkText) : By* – With this you can find elements of *“****a****” tags(****Link****)* with the partial link names.

*Command* – *driver.findElement(By.linkText(“Element LINKTEXT”));*

*Command* – *driver.findElement(By.partialLinkText(“Element LINKTEXT”));*

****

Example:

|  |
| --- |
| **WebElement element = driver.findElement(By.linkText("Partial Link Test"));**  **//Or can be identified as**  **WebElement element = driver.findElement(By.partialLinkText("Partial");** |

* **By CSS Selector**

CSS Selectors have many formats, but we will only focus on the most common ones.

* + - * Tag and ID
      * Tag and class
      * Tag and attribute
      * Tag, class, and attribute
      * Inner text

***References:***

[*http://www.guru99.com/locators-in-selenium-ide.html*](http://www.guru99.com/locators-in-selenium-ide.html)

* **By XPath**

*xpath(String xpathexpression) : By* – It is most popular and majorly used locating element technique or the easiest way to locate element in WebDriver.

*Command*– ***driver.findElement(By.xpath(“Element XPATHEXPRESSION”));***

**References:**

*http://www.guru99.com/xpath-selenium.html*

**Exercise:**

* *Launch new Browser*
* *Open* [*http://newtours.demoaut.com/*](http://newtours.demoaut.com/) *website*
* *Click on SING-ON link*
* *Type User Name & Password*
* Click on submit button
* *Close the Browser*

### ****1.3.5 CheckBox & Radio Button Operations****

* **IsSelected**: Used for Radio Button/Check Box and you just need to select the deselected Radio Button/Check Box.With this statement, you can get to know that the element is selected or not.

***Example:***

|  |
| --- |
| // Store all the elements of same category in the list of WebLements  List oRadioButton = driver.findElements(By.name(""));  // Create a boolean variable which will hold the value (True/False)  boolean bValue = false;  // This statement will return True, in case of first Radio button is selected  bValue = oRadioButton.get(0).isSelected();  // This will check that if the bValue is True means if the first radio button is selected  if(bValue = true){  // This will select Second radio button, if the first radio button is selected by default  oRadioButton.get(1).click();  }else{  // If the first radio button is not selected by default, the first will be selected  oRadioButton.get(0).click();  } |

***Note: Name***is always same for the same group of Radio Buttons/Check Boxes but their***Values***are different.

* ***With Value:*** you can even select Radio Buttons/Check Boxes with their Values.

***Example***

|  |
| --- |
| **// Find the checkbox or radio button element by Name**  **List oCheckBox = driver.findElements(By.name("tool"));**  **// This will tell you the number of checkboxes are present**  **int iSize = oCheckBox.size();**  **// Start the loop from first checkbox to last checkboxe**  **for(int i=0; i < iSize ; i++ ){**  **// Store the checkbox name to the string variable, using 'Value' attribute**  **String sValue = oCheckBox.get(i).getAttribute("value");**  **// Select the checkbox it the value of the checkbox is same what you are looking for**  **if (sValue.equalsIgnoreCase("")){**  **oCheckBox.get(i).click();**  **// This will take the execution out of for loop**  **break;**  **}**  **}** |

**Exercise:**

* Launch new Browser
* Open “[***http://toolsqa.com/automation-practice-form/***](http://toolsqa.com/automation-practice-form/)“
* Challenge One – Select the deselected Radio button (female) for category Sex (Use IsSelected method)
* Challenge Two – Select the Third radio button for category ‘Years of Exp’ (Use Id attribute to select Radio button)
* Challenge Three – Check the Check Box ‘Automation Tester’ for category ‘Profession'( Use Value attribute to match the selection)
* Challenge Four – Check the Check Box ‘Selenium IDE’ for category ‘Automation Tool’ Use cssSelector)

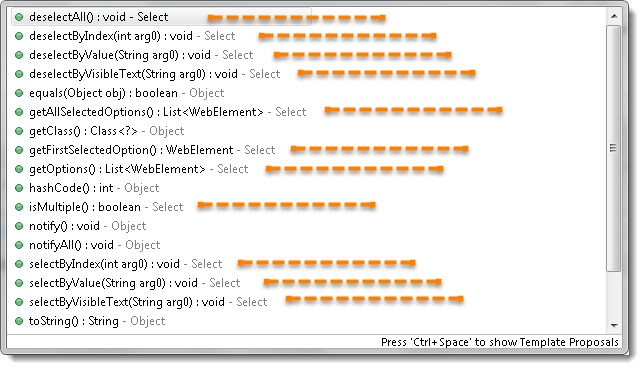
### ****1.3.6 DropDown & Multiple Select Operations****

Require to import **‘*import org.openqa.selenium.support.ui.Select*’** package and to use it we need to create a new ***Select Object*** of class ***Select*.**

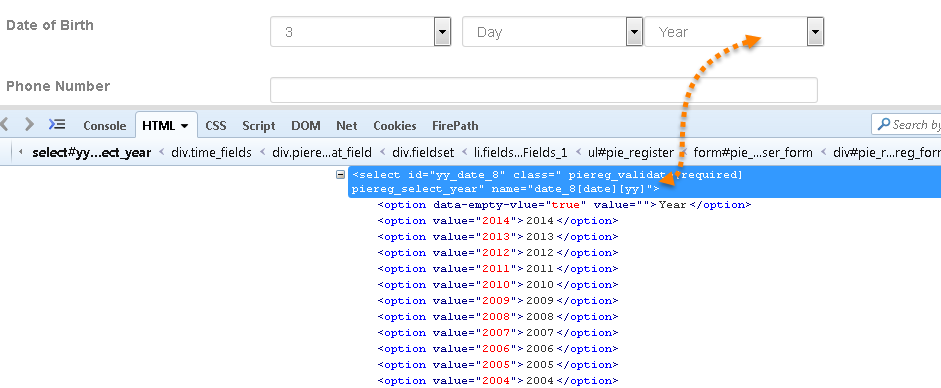
*Example:*

|  |
| --- |
| WebElement element = driver.findElement(By.id("Country"));  Select oSelect = new Select(element);  //Or it can be also written as  Select oSelect = new Select(driver.findElement(By.id("Country"))); |

***Note:***Select class only works for elements with <select> tags.



***Different Select Commands:***



* **selectByVisibleText**

*selectByVisibleText(String arg0) : void*– It takes a parameter of String which is one of the *Value of Select element* and it returns nothing.

*Command*– *oSelect.selectByVisibleText(“text”);*

*Example*

|  |
| --- |
| ***//***To select the value 2010.  Select oSelect = new Select(driver.findElement(By.id("yy\_date\_8")));  oSelect.selectByVisibleText("2010"); |

* **selectByValue**

*selectByValue(String arg0) : void*

*Command*– ***oSelect.selectByValue(“text”);***

*Example:*

|  |
| --- |
| //To select the value 2014.  Select oSelect = new Select(driver.findElement(By.id("yy\_date\_8")));  oSelect.selectByValue("2014"); |

***Note:***The value of an option and the text of the option may not be always same and there can be a possibility that the value is not assigned to Select webelement. If the value is given in the Select tag then only you can use the ***selectByValue***method.

* **getOptions**

*getOptions( ) : List<WebElement>*–This gets the all options belonging to the Select tag. It takes no parameter and returns *List<WebElements>*.

*Command* – ***oSelect.getOptions();***

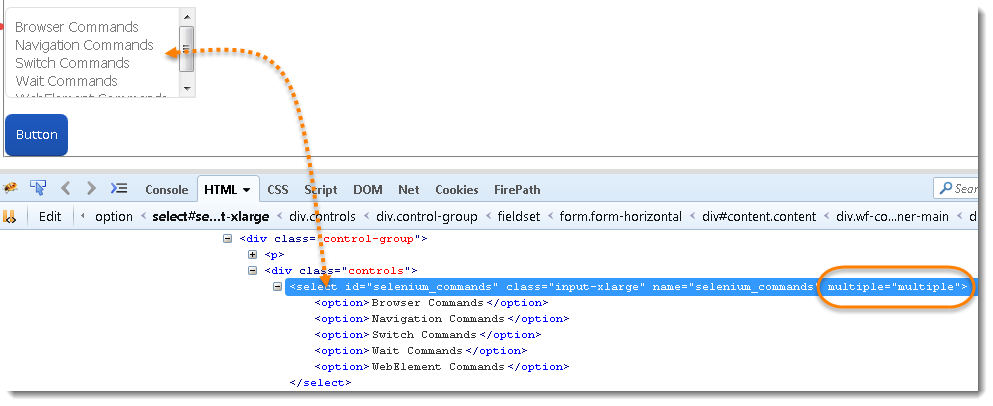
*Example:*

|  |
| --- |
| ***//***To get the ***Count*** of the total elements inside *SELECT*.  Select oSelect = new Select(driver.findElement(By.id("yy\_date\_8")));  List <WebElement> elementCount = oSelect.getOptions();  System.out.println(elementCount.size()); |

*Example:*To get the ***Count*** of the total elements inside SELECTand to ***Print*** the text value of every element present in theSELECT.

|  |
| --- |
| Select oSelect = new Select(driver.findElement(By.id("yy\_date\_8")));  List <WebElement> elementCount = oSelect.getOptions();  int iSize = elementCount.size();  for(int i =0; i>iSize ; i++){  String sValue = elementCount.get(i).getText();  System.out.println(sValue);  } |

* **Deselect Methods**



* ***deselectAll( ) : void***– Clear all selected entries. This is only valid when the SELECT supports multiple selections.

Command – ***oSelect.deselectAll;***

* *deselectByIndex(int arg0) : void*–Deselect the option at the given index.

*Command*– ***oSelect.deselectByIndex;***

* *deselectByValue(String arg0) : void*–Deselect all options that have a value matching the argument.

*Command*– ***oSelect.deselectByValue;***

* *deselectByVisibleText(String arg0) : void*– Deselect all options that display text matching the argument.

*Command*– ***oSelect.deselectByVisibleText***

* **isMultiple**

*isMultiple( ) : boolean*– This tells whether the SELECT element support multiple selecting options at the same time or not.

*Command* – ***oSelect.isMultiple();***

**Exercise 1 (Drop Down Box/List):**

* *Launch new Browser*
* *Open “http://*[*toolsqa*](http://toolsqa.com/selenium-webdriver/dropdown-multiple-select-operations/)*.com/automation-practice-form/”*
* *Select ‘Continents’ Drop down ( Use Id to identify the element )*
* *Select option ‘Europe’ (Use selectByIndex)*
* *Select option ‘Africa’ now (Use selectByVisibleText)*
* *Print all the options for the selected drop down and select one option of your choice*
* *Close the browser*

**Exercise 2 (Multiple Selection Box/List):**

* *Launch new Browser*
* *Open “http://*[*toolsqa*](http://toolsqa.com/selenium-webdriver/dropdown-multiple-select-operations/)*.com/automation-practice-form/”*
* *Select ‘Selenium Commands’ Multiple selection box (Use Name locator to identify the element )*
* *Select option ‘Browser Commands’ and then deselect it (Use selectByIndex and deselectByIndex)*
* *Select option ‘Navigation Commands’ and then deselect it (Use selectByVisibleText and deselectByVisibleText)*
* *Print and select all the options for the selected Multiple selection list.*
* *Deselect all options*
* *Close the browser*

### ****1.3.7 Wait Commands:****

* **ImplicitlyWait Command**: tell Selenium that we would like it to wait for a certain amount of time before throwing an **exception** that it cannot find the element on the page. Any search for elements on the page could take the time the implicit wait is set for.

The default setting is 0. While execution, if WebDriver cannot find element to act upon immediately, it will wait for specified amount of time. During this time, no attempt is made to find an element. After completion of specified time, WebDriver will try to find an element once. Exception is displayed if element is not found after that.

*Example:*

|  |
| --- |
| WebDriver driver = new FirefoxDriver();  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  driver.get("http://url\_that\_delays\_loading");  WebElement myDynamicElement = driver.findElement(By.id("myDynamicElement")); |

* **FluentWait Command**:uses two parameters. Timeout value and polling frequency. Let’s say we have timeout value as 30 seconds and polling frequency as 2 seconds. WebDriver will check for element for every 2 seconds until timeout value (30 seconds). After timeout value is exceeded without any result, exception is thrown.

fluent wait is helpful in AJAX applications as well as in scenarios when element load time fluctuates often. Strategic use of fluent wait significantly improves automation efforts.

*Example:*

|  |
| --- |
| Wait wait = new FluentWait(driver).withTimeout(30, SECONDS).pollingEvery(2, SECONDS).ignoring(NoSuchElementException.class);  WebElement testElement = wait.until(new Function() {  public WebElement apply(WebDriver driver) {  return driver.findElement(By.id("testId"));  }  }); |

* **ExpectedConditions Command (Explicit wait):** when some element takes more time to load. Setting implicit wait for such cases doesn’t make sense as browser will wait unnecessarily for the same time for every element, increasing the automation time. Explicit wait helps here by bypassing implicit wait altogether for some specific elements.

*Example:*

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id(>someid>))); |

* **Timeout**
* PageLoadTimeOut: sets the amount of time to wait for a page load to complete before throwing an error. If the timeout is negative, page loads can be indefinite

*Example:*

driver.manage().timeouts().pageLoadTimeout(100, SECONDS);

* SetScriptTimeout Command: Sets the amount of time to wait for an asynchronous script to finish execution before throwing an error. If the timeout is negative, then the script will be allowed to run indefinitely.

*Example:*

driver.manage().timeouts().setScriptTimeout(100,SECONDS);

* **Sleep Command:** using sleeper in your Selenium tests is not a good practice and should be avoided. Reason for this is, it forces browser to wait for specified amount of time. It unnecessarily delays other automation steps. You can implement sleeper in your test with below code.

*Example:* thread.sleep(1000);

***References*:** *http://www.testingdiaries.com/selenium-webdriver-wait/*

**Exercise:**

  - *Launch new Browser*

*- Open URL “http://toolsqa.com/automation-practice-switch-windows/”*

*- Click on the Button “Timing Alert”*

*- Accept the Alert (Alert will take 3 second to get displayed, Use WebDriverWait to wait for it)*

## **Alerts & Windows**

### 1.4.1 Switch Commands

Some web applications have many frames or multiple windows. Selenium WebDriver assigns an alphanumeric id to each window as soon as the WebDriver object is instantiated. This unique alphanumeric id is called**window handle**. Selenium uses this unique **id** to switch control among several windows. In simple terms, each unique window has a unique ID, so that Selenium can differentiate when it is switching controls from one window to the other.

***Basic Commands:***

* **GetWindowHandle:** To get the **window handle** of the current window.

*Example:* String handle= driver.getWindowHandle();//Return a string of alphanumeric window handle

* **GetWindowHandles:** To get the **window handle** of **all** the current windows.

*Example:* Set<String> handle= driver.getWindowHandles();//Return a set of window handle

* **SwitchTo Window**
  + Purpose 1: WebDriver supports **moving** between named windows using the “switchTo” method:

driver.switchTo().window("windowName");

* + Purpose2: Alternatively, you can pass a “**window handle**” to the “switchTo().window()” method. Knowing this, it’s possible to iterate over every open window like so:

for (String handle : driver.getWindowHandles()) {

driver.switchTo().window(handle);}

* + Purpose 3**:** Switching between windows with **Iterators**:

|  |
| --- |
| driver.findElement(By.id(“id of the link which opens new window”)).click();  //wait till two windows are not opened  waitForNumberofWindowsToEqual(2);//this method is for wait  Set handles = driver.getWindowHandles();  firstWinHandle = driver.getWindowHandle();  handles.remove(firstWinHandle);  String winHandle=handles.iterator().next();  if (winHandle!=firstWinHandle){  //To retrieve the handle of second window, extracting the handle which does not match to first window handle  secondWinHandle=winHandle; //Storing handle of second window handle  //Switch control to new window  driver.switchTo().window(secondWinHandle); |

* **SwitchTo Frame:** WebDriver supports moving between named frames using the “**switchTo**” method.

*Example:* driver.switchTo().frame("frameName");

* **SwitchTo Popup:** WebDriver supports moving between named PopUps using the “**switchTo**” method.You cab **accept, dismiss, read** its contents or even type into a prompt. This interface works equally well on alerts, confirms, and prompts.

*Example:* Alert alert = driver.switchTo().alert();

**Exercise 1:**

- *Launch new Browser*

*- Open URL “http://*[*toolsqa*](http://toolsqa.com/selenium-webdriver/switch-commands/)*.com/automation-practice-switch-windows/”*

*- Get Window name (Use GetWindowHandle command)*

*- Click on Button “New Message Window”, it will open a Pop Up Window*

*- Get all the Windows name ( Use GetWindowHandles command)*

*- Close the Pop Up Window (Use Switch Command to shift window)*

**Exercise 2:**

*- Launch new Browser*

*- Open URL “http://*[*toolsqa*](http://toolsqa.com/selenium-webdriver/switch-commands/)*.com/automation-practice-switch-windows/”*

*- Click on Button “Alert Box”, it will open a Pop Up Window generated by JavaScript*

*- Switch to Alert window (Use ‘SwitchTo()Alert() command)*

*- Close the Pop Up Window (Use Accept command)*

### 1.4.2 Handling of Alerts

Type of Alerts:

* Simple alert
* Confirmation alert
* Prompt alert

Alert interface gives us following methods to deal with the alert:

* ***accept()****To accept the alert*
* ***dismiss()****To dismiss the alert*
* ***getText()****To get the text of the alert*
* ***sendKeys()****To write some text to the alert*

**Simple Alert:** Simple alerts just have a***OK*** button on them.

*Example*

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** InterruptedException {  WebDriver driver;  System.*setProperty*("webdriver.chrome.driver", "E:\\Setup\\BrowserDrivers\\chromedriver.exe");  driver = **new** ChromeDriver();  driver.get("http://sislands.com/coin70/week1/dialogbox.htm");  driver.manage().window().maximize();  driver.findElement(By.*xpath*("//input[@value = 'alert']")).click();  Alert javascriptAlert = driver.switchTo().alert();  System.***out***.println(javascriptAlert.getText()); // Get text on alert box  javascriptAlert.accept();  }  } |

**Confirmation Alert:** This alert comes with an option to accept or dismiss the alert. To accept the alert you can use **Alert.accept()** and to dismiss you can use the***Alert.dismiss()***. Here is the code to dismiss a prompt alert.

*Example:*

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** InterruptedException {    WebDriver driver;  System.*setProperty*("webdriver.ie.driver", "E:\\Setup\\BrowserDrivers\\IEDriverServer.exe");  driver = **new** InternetExplorerDriver();  driver.get("http://sislands.com/coin70/week1/dialogbox.htm");  driver.manage().window().maximize();  driver.findElement(By.*xpath*("//input[@value = 'confirm']")).click();  Alert javascriptconfirm = driver.switchTo().alert();  javascriptconfirm.accept();  javascriptconfirm = driver.switchTo().alert();  System.***out***.println(javascriptconfirm.getText()); // Get text on alert box  javascriptconfirm.accept();  driver.findElement(By.*xpath*("//input[@value = 'confirm']")).click();  javascriptconfirm = driver.switchTo().alert();  javascriptconfirm.dismiss();  javascriptconfirm = driver.switchTo().alert();  System.***out***.println(javascriptconfirm.getText()); // Get text on alert box  javascriptconfirm.accept();  } |

**Prompt Alerts:** In prompt alerts you get an option to add text to the alert box. This is specifically used when some input is required from the user. We will use the***sendKeys()***method to type something in the Prompt alert box. Here is the code

*Example:*

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** InterruptedException {    WebDriver driver;  System.*setProperty*("webdriver.ie.driver", "E:\\Setup\\BrowserDrivers\\IEDriverServer.exe");  driver = **new** InternetExplorerDriver();  driver.get("http://sislands.com/coin70/week1/dialogbox.htm");  driver.manage().window().maximize();  driver.findElement(By.*xpath*("//input[@value = 'prompt']")).click();  Alert javascriptprompt = driver.switchTo().alert();  javascriptprompt.sendKeys("This is Selenium Training");  System.***out***.println(javascriptprompt.getText()); // Get text on alert box  javascriptprompt.accept();  javascriptprompt = driver.switchTo().alert();  System.***out***.println(javascriptprompt.getText()); // Get text on alert box  javascriptprompt.accept();  driver.findElement(By.*xpath*("//input[@value = 'prompt']")).click();  javascriptprompt = driver.switchTo().alert();  System.***out***.println(javascriptprompt.getText()); // Get text on alert box  javascriptprompt.dismiss();  javascriptprompt = driver.switchTo().alert();  System.***out***.println(javascriptprompt.getText()); // Get text on alert box  javascriptprompt.accept();  } |

### 1.4.3 Handling Iframes

iFrame is a HTML document embedded inside an HTML document. iFrame is defined by an <iframe></iframe> tag in HTML.

Here is an sample HTML code of a HTML page which contains two iFrames.

|  |
| --- |
| <html>  <body>  <div class="box">  <iframe name="iframe1" id="IF1" height="600" width="400" src="http://toolsqa.com"></iframe>  </div>  <div class="box">  <iframe name="iframe2" id="IF2" height="600" width="400" align="left" src="http://demoqa.com"></iframe>  </div>  </body>  </html> |

To Switch between iFrames we have to use the driver’s ***switchTo().frame*** command. We can use the *switchTo().frame()* in three ways:

* *switchTo.frame(int frameNumber)*:Pass the frame index and driver will switch to that frame.
* *switchTo.frame(string frameNameOrId)*:Pass the frame element Name or ID and driver will switch to that frame.
* *switchTo.frame(WebElement frameElement)*:Pass the frame web element and driver will switch to that frame.

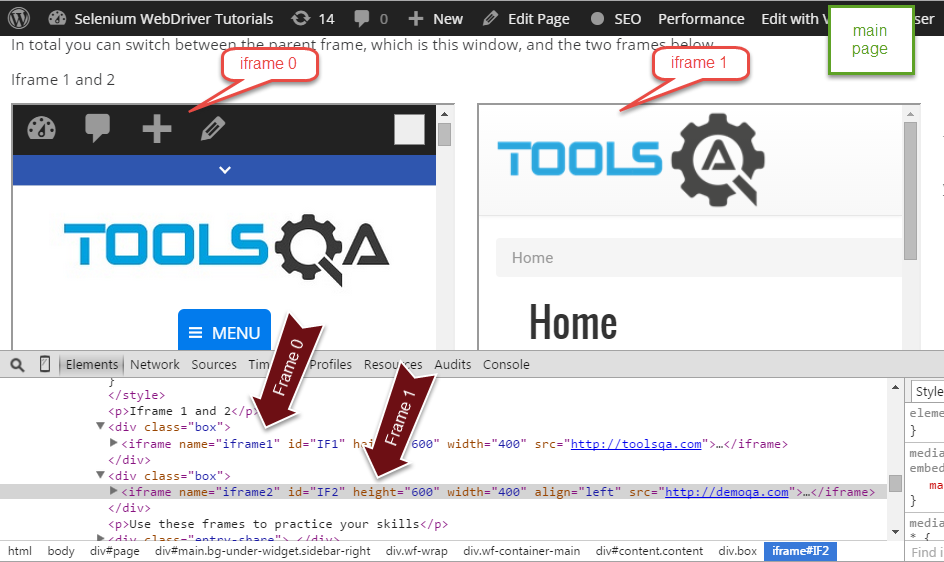
*1.4.3.1 Basic iFrame implementations*

* ***Find total number of iFrames on a webpage***

*Example:*

|  |
| --- |
| WebDriver driver = new FirefoxDriver();  driver.get("http://toolsqa.com/iframe-practice-page/");  //By executing a java script  JavascriptExecutor exe = (JavascriptExecutor) driver;  Integer numberOfFrames = Integer.parseInt(exe.executeScript("return window.length").toString());  System.out.println("Number of iframes on the page are " + numberOfFrames);  //By finding all the web elements using iframe tag  List<WebElement> iframeElements = driver.findElements(By.tagName("iframe"));  System.out.println("The total number of iframes are " + iframeElements.size()); |

* ***Switch to Frames by Index*:** Index of an iFrame is the position at which it occurs in the HTML page. In the above example we have found total number of iFrames. In the sample page we have two IFrames, index of iFrame starts from 0. So there are two iFrames on the page with index 0 and 1. Index 0 will be the iFrame which exists earlier in the HTML page. Refer to the image below:



to switch to0th iframe we can simple write ***driver.switchTo().frame(0)***. Here is the sample code.

*Example:*

|  |
| --- |
| public static void main(String[] args) throws InterruptedException {  WebDriver driver = new FirefoxDriver();  driver.get("http://toolsqa.com/iframe-practice-page/");  //Switch by Index  driver.switchTo().frame(0);  driver.quit();  } |

* ***Switch to Frames by Name:*** We can switch to the iFrame using the name by using the command ***driver.switchTo().frame(“iframe1”)***.

*Example:*

|  |
| --- |
| WebDriver driver = new FirefoxDriver();  driver.get("http://toolsqa.com/iframe-practice-page/");  //Switch by frame name  driver.switchTo().frame("iframe1");  driver.quit(); |

* **Switch to Frames by ID**

*Example:*

|  |
| --- |
| WebDriver driver = new FirefoxDriver();  driver.get("http://toolsqa.com/iframe-practice-page/");  //Switch by frame ID  driver.switchTo().frame("IF1");  driver.quit(); |

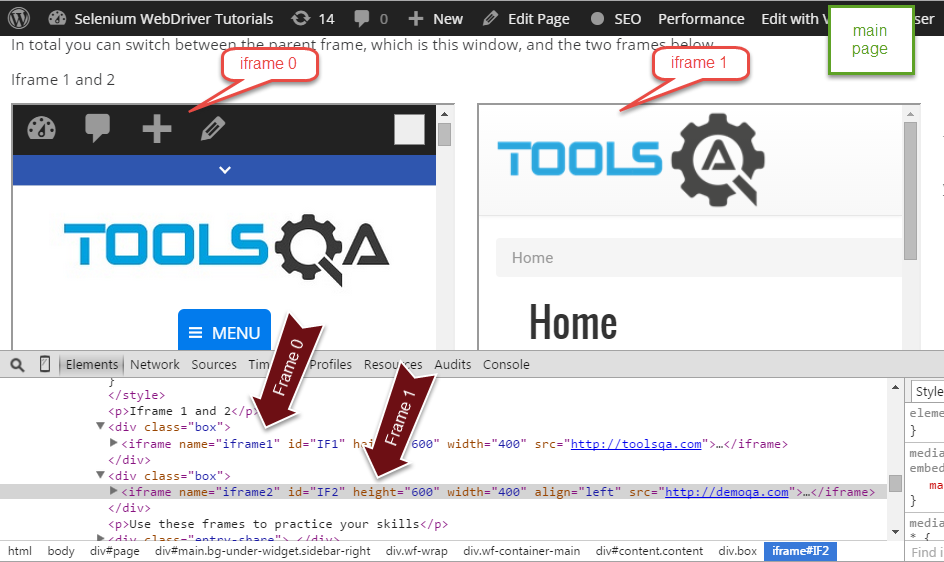
* ***Switch to Frames by WebElement***

|  |
| --- |
| WebDriver driver = new FirefoxDriver();  driver.get("http://toolsqa.com/iframe-practice-page/");  //First find the element using any of locator stratedgy  WebElement iframeElement = driver.findElement(By.id("IF1"));  //now use the switch command  driver.switchTo().frame(iframeElement);  driver.quit(); |

* ***Switching back to Main page from Frame: Using command : switchTo().defaultContent()***

|  |
| --- |
| WebDriver driver = new FirefoxDriver();  driver.get("http://toolsqa.com/iframe-practice-page/");  //First find the element using any of locator stratedgy  WebElement iframeElement = driver.findElement(By.id("IF1"));  //now use the switch command  driver.switchTo().frame(0);  //Do all the required tasks in the frame 0  //Switch back to the main window  driver.switchTo().defaultContent();  driver.quit(); |

*1.4.3.2 Interact with elements inside an Iframe*



*Step 1:* Navigate to the page with multiple iFrames.

|  |
| --- |
| driver.get("http://toolsqa.com/iframe-practice-page/"); |

*Step 2: Switch to first frame and then find First Name and Last name element.*

|  |
| --- |
| /\* Switch to the first frame, remember frame index starts from 0 \*/  driver.switchTo().frame(0);  /\* now find the First name field \*/  WebElement firstName = driver.findElement(By  .xpath("//\*[@id='content']/form/fieldset/div[1]/p[3]/input"));  /\* now find the Last name field \*/  WebElement lastName = driver.findElement(By  .xpath("//\*[@id='content']/form/fieldset/div[1]/p[4]/input")); |

*Step 3:* *Fill some value in the first name and Last name value*

|  |
| --- |
| firstName.sendKeys("Virender");  lastName.sendKeys("Singh"); |

*Step 4:* Switch to second frame (Critical step):Switching to the 2nd frame, frame index 1

***Note: when the*** *page has two frame 0 and 1. Now if you are inside frame 0 you cannot switch to frame 1. Because frame 1 is not a part of frame 0, but its a part of main page. Any attempt to directly switch to frame 1 from frame 0 will give an exception. Simple switch back to the main page using the SwitchTo().defaultContent() command. Once on the main page switch to the frame 1 as shown in the code below.*

|  |
| --- |
| driver.switchTo().frame(1); |

*Step 5:*Find the first image element and then click on it

Once you are in second frame, rest is just similar to regular findElement and click command. As shown in the code below.

|  |
| --- |
| WebElement imageElement = driver.findElement(By  .xpath("//\*[@id='post-9']/div/div[1]/div/p[1]/a/img"));  imageElement.click();  System.out.println("Switching successfull"); |

* + **Complete Code:**

|  |
| --- |
| WebDriver driver = new FirefoxDriver();  // Step 1: Navigate to the page with multiple iframes  driver.get("http://toolsqa.com/iframe-practice-page/");  // Step 2: Switch to first frame and then find First Name and Last name  // element  /\* Switch to the first frame, remember frame index starts from 0 \*/  driver.switchTo().frame(0);  /\* now find the First name field \*/  WebElement firstName = driver.findElement(By  .xpath("//\*[@id='content']/form/fieldset/div[1]/p[3]/input"));  /\* now find the Last name field \*/  WebElement lastName = driver.findElement(By  .xpath("//\*[@id='content']/form/fieldset/div[1]/p[4]/input"));  // Step 3: Fill some value in the first name and last name files  firstName.sendKeys("Virender");  lastName.sendKeys("Singh");  // Step 4: Switching to the 2nd frame, frame index 1  driver.switchTo().defaultContent();  driver.switchTo().frame(1);  //Step 5: Find image element and click on that  WebElement imageElement = driver.findElement(By  .xpath("//\*[@id='post-9']/div/div[1]/div/p[1]/a/img"));  imageElement.click();  System.out.println("Switching successfull"); |

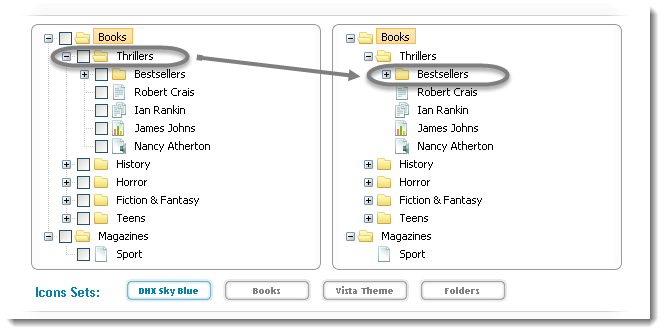
## **Action & Robot Class**

### 1.5.1 Drag And Drop in Selenium WebDriver

The action chain generator implements the **Builder**pattern to create a Composite Action containing a group of other actions. This should ease building actions by configuring an **Actions** chains generator instance and invoking its **build( )** method to get the complex action.

*Example:*

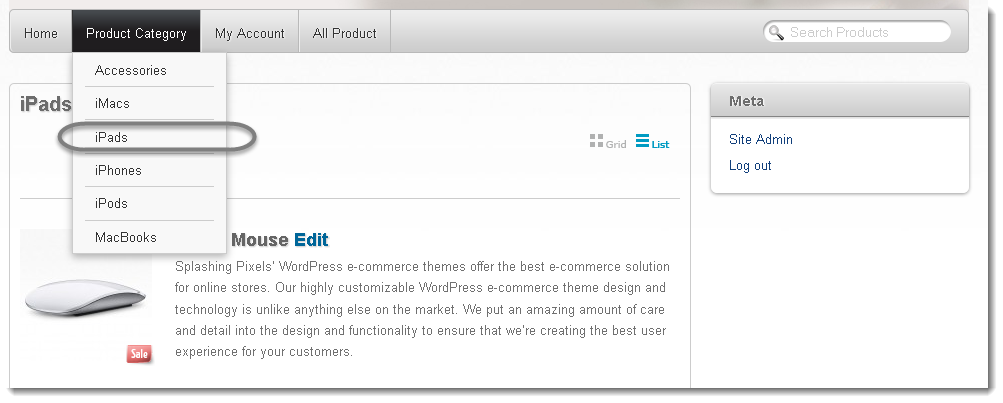
In this example we will drag the **Thriller** folder from the left table on to the **Bestsellers** folder of the right side table.



|  |
| --- |
| package practiceTestCases;  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.Action;  import org.openqa.selenium.interactions.Actions;  public class DragAndDrop {  public static void main(String[] args) throws InterruptedException {  WebDriver driver = new FirefoxDriver();  String URL = "http://www.dhtmlx.com/docs/products/dhtmlxTree/index.shtml";  driver.get(URL);  // It is always advisable to Maximize the window before performing DragNDrop action  driver.manage().window().maximize();  driver.manage().timeouts().implicitlyWait(10000, TimeUnit.MILLISECONDS);  WebElement From = driver.findElement(By.xpath(".//\*[@id='treebox1']/div/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[1]/td[4]/span"));  WebElement To = driver.findElement(By.xpath(".//\*[@id='treebox2']/div/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[2]/td[2]/table/tbody/tr[1]/td[4]/span"));  Actions builder = new Actions(driver);  Action dragAndDrop = builder.clickAndHold(From)  .moveToElement(To)  .release(To)  .build();  dragAndDrop.perform();  }  } |

### Mouse Hover action in Selenium WebDriver

*Example:* Go to<http://store.demoqa.com/> **--**> Click on Product category link on the top menu > Then select any of the items from the drop down menu:



One way of doing this is by using **Action** class:

|  |
| --- |
| package automationFramework;  import java.util.concurrent.TimeUnit;  import org.openqa.selenium.\*;  import org.openqa.selenium.interactions.Actions;  public class mouseHover{  public static WebDriver driver;  public static void main(String[] args) {  driver = new FirefoxDriver();  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  driver.get("http://www.onlinestore..com");  WebElement element = driver.findElement(By.linkText("Product Category"));  Actions action = new Actions(driver);  action.moveToElement(element).build().perform();  driver.findElement(By.linkText("iPads")).click();  }  } |

It can be done differently like this

|  |
| --- |
| WebElement element = driver.findElement(By.linkText("Product Category"));  Actions action = new Actions(driver);  action.moveToElement(element).moveToElement(driver.findElement(By.linkText("iPads"))).click().build().perform(); |

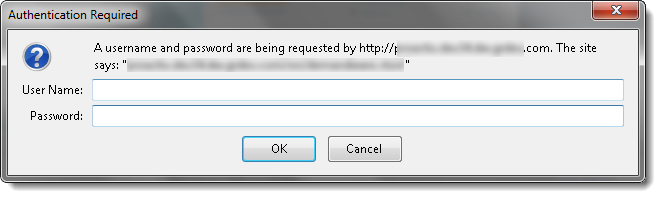
***Note:*** *With some of the browser it happens that once*[*mouse hover*](http://toolsqa.com/selenium-webdriver/mouse-hover-action/)*action is performed, the menu list disappear with in the fractions of seconds before Selenium identify the next submenu item and perform click action on it. In that case it is better to use ‘perform()’ action on the main menu to hold the menu list till the time Selenium identify the sub menu item and click on it.*

|  |
| --- |
| WebElement element = driver.findElement(By.linkText("Product Category"));  Actions action = new Actions(driver);  action.moveToElement(element).perform();  WebElement subElement = driver.findElement(By.linkText("iPads"));  action.moveToElement(subElement);  action.click();  action.perform(); |

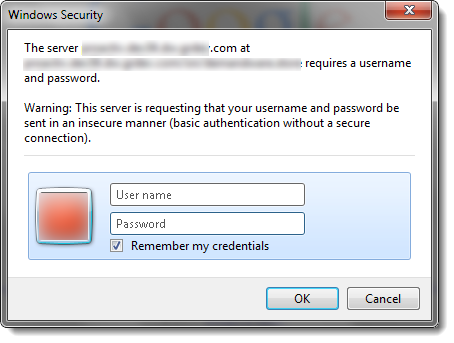
## **Tips and Tricks**

### 1.6.1 HTTP Proxy Authentication with Selenium WebDriver:

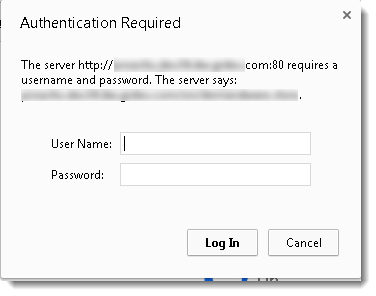
* **Firefox Browser Authentication Window**

****

* **Internet Explorer Browser Authentication Window**

****

* **Chrome Browser Authentication Window**

****

To handle [**HTTP Proxy Authentication**](http://toolsqa.com/selenium-webdriver/http-proxy-authentication/), we can simply send **Username & Password** through **URL** and in most of the cases it works perfectly fine.

**driver.get("http://UserName:Password@Example.com");**

# ADVANCED TrAINING

## **2.1 TestNG Framework**

### 2.1.1 TestNG Introduction

***TestNG*** is a testing framework inspired from ***JUnit*** and ***NUnit*** but introducing some new functionality that make it more powerful and easier to use.

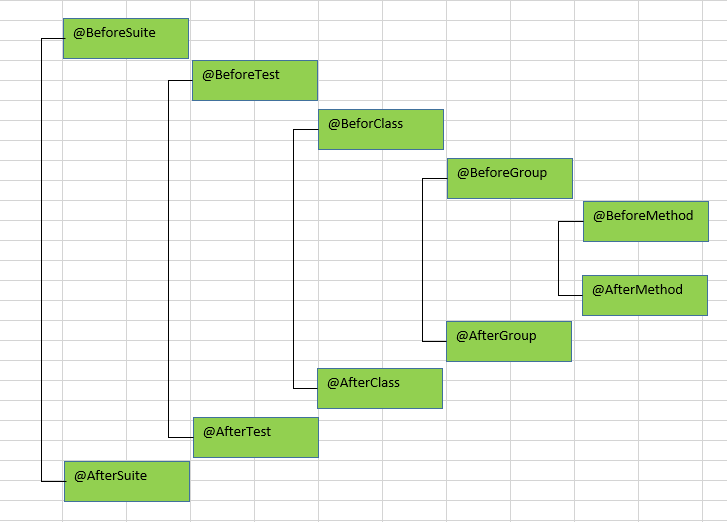
It is an open source automated testing framework; where **NG** of Test**NG** means **N**ext **G**eneration. TestNG is similar to JUnit but it is much more powerful than JUnit but still it’s inspired by JUnit. It is designed to be better than JUnit, especially when testing integrated classes. Pay special thanks to *Cedric Beust who is the creator of TestNG*.

There are number of benefits but from Selenium perspective, major advantages of TestNG are :

* It gives the ability to produce ***HTML Reports*** of execution
* ***Annotations***made testers life easy
* Test cases can be ***Grouped & Prioritized*** more easily
* ***Parallel***testing is possible
* Generates ***Logs***
* Data ***Parameterization***is possible

### 2.1.2 Basic Annotation, Groups and Dependent

*2.1.2.1 Basic Annotation, Groups*



|  |  |
| --- | --- |
| Annotation | Description |
| @BeforeSuite | Run before all tests in this suite have run. |
| @AfterSuite | Run after all tests in this suite have run. |
| @BeforeTest | Run before any test method belonging to the classes inside the test tag is run. |
| @AfterTest: | Run after all the test methods belonging to the classes inside the test tag have run. |
| @BeforeClasss | Run before the first test in the current class is invoked |
| @AfterTest | Run after all the test method in the current class is invoked |
| @BeforeGroups | Run before the first test method that belongs to any of these groups is invoked. |
| @AfterGroup | Run after the method that belongs to any of these groups is invoked |
| @BeforeMethod | Run before each test method |
| @AfterMethod | Run after each method |
| @Test | Is a part of a test case |

*Example:*

Step1. Create class A

|  |
| --- |
| package org.niteco.qa.CloudTesting;  import org.testng.annotations.AfterClass;  import org.testng.annotations.AfterGroups;  import org.testng.annotations.BeforeClass;  import org.testng.annotations.BeforeGroups;  import org.testng.annotations.Test;  import org.testng.annotations.BeforeMethod;  import org.testng.annotations.AfterMethod;  public class A {  @Test(groups={"group1"})  public void test\_A01(){  System.out.println("test A01");  }  @Test(groups={"group1"})  public void test\_A02() {  System.out.println("test A02");  }  @BeforeMethod  public void beforeMethod() {  System.out.println("before method in Class A");  }  @AfterMethod  public void afterMethod() {  System.out.println("after method in Class A");  }  @BeforeGroups("group1")  public void beforeGroup(){  System.out.println("before group1");  }  @AfterGroups("group1")  public void afterGroup(){  System.out.println("after group1");  }  @BeforeClass  public void beforeClass(){  System.out.println("before Class A");  }  @AfterClass  public void afterClass(){  System.out.println("after Class A");  }  } |

Step2. Create class B

|  |
| --- |
| **package** org.niteco.qa.CloudTesting;  **import** org.testng.annotations.AfterClass;  **import** org.testng.annotations.AfterGroups;  **import** org.testng.annotations.AfterMethod;  **import** org.testng.annotations.BeforeClass;  **import** org.testng.annotations.BeforeGroups;  **import** org.testng.annotations.BeforeMethod;  **import** org.testng.annotations.Test;  **public** **class** B {  @Test(groups={"group2"})  **public** **void** test\_A01(){  System.***out***.println("test B01");  }  @Test(groups={"group2"})  **public** **void** test\_B02() {  System.***out***.println("test B02");  }  @BeforeMethod  **public** **void** beforeMethod() {  System.***out***.println("before method in Class B");  }  @AfterMethod  **public** **void** afterMethod() {  System.***out***.println("after method in Class B");  }  @BeforeGroups("group2")  **public** **void** beforeGroup(){  System.***out***.println("before group2");  }  @AfterGroups("group2")  **public** **void** afterGroup(){  System.***out***.println("after group2");  }  @BeforeClass  **public** **void** beforeClass(){  System.***out***.println("before Class B");  }  @AfterClass  **public** **void** afterClass(){  System.***out***.println("after Class B");  }  } |

Step3. Create class TestBase

|  |
| --- |
| **package** org.niteco.qa.CloudTesting;  **import** org.testng.annotations.AfterSuite;  **import** org.testng.annotations.AfterTest;  **import** org.testng.annotations.BeforeSuite;  **import** org.testng.annotations.BeforeTest;  **public** **class** TestBase {  @BeforeTest  **public** **void** beforeTest() {  System.***out***.println("before Test");  }  @AfterTest  **public** **void** afterTest() {  System.***out***.println("after Test");  }  @BeforeSuite  **public** **void** beforeSuite() {  System.***out***.println("before Suite");  }  @AfterSuite  **public** **void** afterSuite() {  System.***out***.println("after sutie");  }  } |

Step4. Create testing.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  <suite name=*"Suite"*>  <test name=*"Test"*>  <classes>  <class name=*"org.niteco.qa.CloudTesting.TestBase"*/>  <class name=*"org.niteco.qa.CloudTesting.A"* />  <class name=*"org.niteco.qa.CloudTesting.B"* />  </classes>  </test> <!-- Test -->  </suite> <!-- Suite --> |

Step5. Right click on Testn.xml then select Run as TestNG. Check the output

|  |
| --- |
| [TestNG] Running:  E:\Automation\Selenium\Project\CloudTesting\testng.xml  before Suite  before Test  before Class A  before group1  before method in Class A  test A01  after method in Class A  before method in Class A  test A02  after method in Class A  after group1  after Class A  before Class B  before group2  before method in Class B  test B01  after method in Class B  before method in Class B  test B02  after method in Class B  after group2  after Class B  after Test  after suite  ===============================================  Suite  Total tests run: 4, Failures: 0, Skips: 0  =============================================== |

* + - 1. *Dependent Test:*

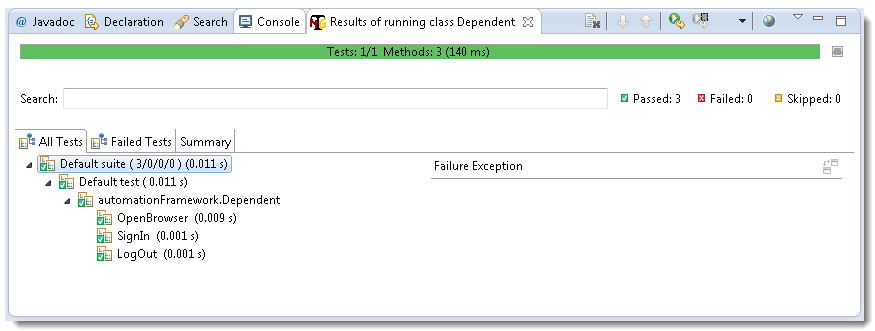
TestNG allows you to specify dependencies either with:

* Using attributes *dependsOnMethods* in @Test annotations OR
* Using attributes *dependsOnGroups* in @Test annotations.

*Example:*

|  |
| --- |
| package automationFramework;  import org.testng.annotations.Test;  public class Dependent {    @Test (dependsOnMethods = { "OpenBrowser" })    public void SignIn() {    System.out.println("This will execute second (SignIn)");    }    @Test    public void OpenBrowser() {    System.out.println("This will execute first (Open Browser)");    }    @Test (dependsOnMethods = { "SignIn" })    public void LogOut() {    System.out.println("This will execute third (Log Out)");    } |

The output look like this:



### 2.1.3 TestNG prioritizing and Sequencing

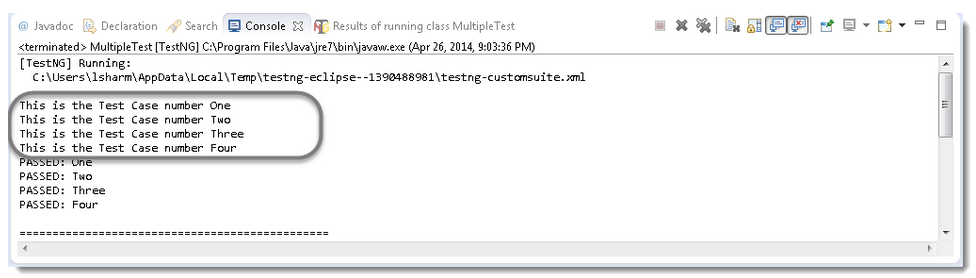
By default, methods annotated by @Test are executed alphabetically (method name).

**Use “priority” parameter.** Simply assign priority to all @Test methods starting from 0(Zero). TestNG will execute the @Test annotation with the lowest priority value up to the largest.

*Example*

|  |
| --- |
| package automationFramework;  import org.openqa.selenium.WebDriver;  import org.testng.annotations.Test;  public class MultipleTest {  public WebDriver driver;  @Test(priority = 0)  public void One() {  System.out.println("This is the Test Case number One");  }  @Test(priority = 1)  public void Two() {  System.out.println("This is the Test Case number Two");  }  @Test(priority = 2)  public void Three() {  System.out.println("This is the Test Case number Three");  }  @Test(priority = 3)  public void Four() {  System.out.println("This is the Test Case number Four");  }  } |

The output is:

****

### 2.1.4 Skipping a Test Case

@Test (enabled = false)

To use two or more parameters in a single annotation, separate them with a comma:

@Test (priority = 3, enabled = false)

### 2.1.5 TestNG Parameter, Data Provider and Factory

*2.1.5.1 TestNG Parameter*

Define parameters in the testing.xml then use them in test method.

*Example:*

Scenario:

1. Open <http://store.demoqa.com/>
2. Input user and password
3. Click Login
4. Click Logout

*Step1. Create the class TestngParameters*

|  |
| --- |
| import java.util.concurrent.TimeUnit;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  import org.testng.annotations.Test;  import org.testng.annotations.Parameters;  public class TestngParameters {  private static WebDriver driver;    @Test    @Parameters({ "sUsername", "sPassword" })    public void test(String sUsername, String sPassword) {    driver = new FirefoxDriver();        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);        driver.get("http://www.store.demoqa.com");        driver.findElement(By.xpath(".//\*[@id='account']/a")).click();        driver.findElement(By.id("log")).sendKeys(sUsername);        driver.findElement(By.id("pwd")).sendKeys(sPassword);        driver.findElement(By.id("login")).click();        driver.findElement(By.xpath(".//\*[@id='account\_logout']/a")).click();        driver.quit();    }  } |

*Step2. Create the testng.xml then create two parameters “sUsername” and “sPassword”*

|  |
| --- |
| <suite name="Suite">  <test name="ToolsQA">  <parameter name="sUsername" value="testuser\_1"/>  <parameter name="sPassword" value="Test@123"/>  <classes>  <class name="automationFramework.TestngParameters" />  </classes>  </test>  </suite> |

Now, run the testng.xml, which will run the parameterTest method. TestNG will try to find a parameter named sUsername & sPassword.

* + - 1. *TestNG DataProviders*
* To pass the complex data such as complex object, objects read from a property file or a data base or excel file,etc…we using Dataproviders. A Dataprovider is a method annotated with @DataProvider. A Dataprovider returns an array of objects.
* A test method that uses DataProvider will be executed a multiple number of times based on the data provided by the DataProvider. The test method will be executed using the same instance of the test class to which the test method belongs.

*Example:*

|  |
| --- |
| public class DataProviderClass  {      @BeforeClass      public void beforeClass() {          System.out.println("Before class executed");      }      @Test(dataProvider = "dataMethod")      public void testMethod(String param) {          System.out.println("The parameter value is: " + param);      }      @DataProvider      public Object[][] dataMethod() {          return new Object[][] { { "one" }, { "two" } };      }  } |

The output is

|  |
| --- |
| Before class executed  The parameter value is: one  The parameter value is: two  PASSED: testMethod("one")  PASSED: testMethod("two") |

*2.1.5.3 Factory*

A factory will execute all the test methods present inside a test class using a separate instance of the respective class.

*Example:*

The below class contains the testMethod and beforeClass methods. The constructor of the test class takes a String argument value. Both beforeClass and testMethod print a message onto console.

|  |
| --- |
| public class SimpleTest {      private String param = "";      public SimpleTest(String param) {          this.param = param;      }      @BeforeClass      public void beforeClass() {          System.out.println("Before SimpleTest class executed.");      }      @Test      public void testMethod() {          System.out.println("testMethod parameter value is: " + param);      }  }  public class SimpleTestFactory  {      @Factory      public Object[] factoryMethod() {          return new Object[] {                                  new SimpleTest("one"),                                  new SimpleTest("two")                              };      }  } |

The output is

|  |
| --- |
| Before SimpleTest class executed.  testMethod parameter value is: two  Before SimpleTest class executed.  testMethod parameter value is: one  PASSED: testMethod  PASSED: testMethod |

**Compare DataProvider and Factory**

|  |  |
| --- | --- |
| DataProvider | Factory |
| DataProvider executes the test method (testMethod) for a single instance of the test class. | Factory implementation executes the test method for each individual instance of the test class |
| Dataprovider is used to provide parameters to a test. If you provide dataprovider to a test, the test will be run taking different sets of value each time. This is useful for a scenario like where you want to login into a site with different sets of username and password each time. | TestNG factory is used to create instances of test classes dynamically. This is useful if you want to run the test class any number of times. For example, if you have a test to login into a site and you want to run this test multiple times,then its easy to use TestNG factory where you create multiple instances of test class and run the tests (may be to test any memory leak issues). |

### 2.1.6 Cross Browser and Parallel Testing

*2.1.6.1 Cross Browser: use TestNG parameter*

*Example:*

*Step1. Create class MultiBrowser*

|  |
| --- |
| **package** org.niteco.qa.CloudTesting;  **import** org.openqa.selenium.By;  **import** org.openqa.selenium.WebDriver;  **import** org.openqa.selenium.chrome.ChromeDriver;  **import** org.openqa.selenium.ie.InternetExplorerDriver;  **import** org.testng.annotations.AfterClass;  **import** org.testng.annotations.BeforeClass;  **import** org.testng.annotations.Parameters;  **import** org.testng.annotations.Test;  **public** **class** MultiBrowser {  **public** WebDriver driver;  @Parameters("browser")  @BeforeClass  // Passing Browser parameter from TestNG xml  **public** **void** beforeTest(String browser) {  // If the browser is Firefox, then do this  **if**(browser.equalsIgnoreCase("chrome")) {  System.*setProperty*("webdriver.chrome.driver", "E:\\Setup\\BrowserDrivers\\chromedriver.exe");  driver = **new** ChromeDriver();  // If browser is IE, then do this  }**else** **if** (browser.equalsIgnoreCase("ie")) {  // Here I am setting up the path for my IEDriver  System.*setProperty*("webdriver.ie.driver", "E:\\Setup\\BrowserDrivers\\IEDriverServer.exe");  driver = **new** InternetExplorerDriver();  }  // Doesn't the browser type, lauch the Website  driver.get("http://www.store.demoqa.com");  }  // Once Before method is completed, Test method will start  @Test **public** **void** login() **throws** InterruptedException {  driver.findElement(By.*xpath*(".//\*[@id='account']/a")).click();  driver.findElement(By.*id*("log")).sendKeys("testuser\_1");  driver.findElement(By.*id*("pwd")).sendKeys("Test@123");  driver.findElement(By.*id*("login")).click();  }  @AfterClass **public** **void** afterTest() {  driver.quit();  }  } |

*Step2. Create testing.xml*

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  <suite name=*"Suite"*>  <test name=*"Test01"*>  <parameter name =*"browser"* value=*"chrome"*/>  <classes>  class name=*"org.niteco.qa.CloudTesting.MultiBrowser"* />  </classes>  </test> <!-- Test -->  <test name=*"Test02"*>  <parameter name =*"browser"* value=*"ie"*/>  <classes>  <class name=*"org.niteco.qa.CloudTesting.MultiBrowser"* />  </classes>  </test> <!-- Test -->  </suite> <!-- Suite --> |

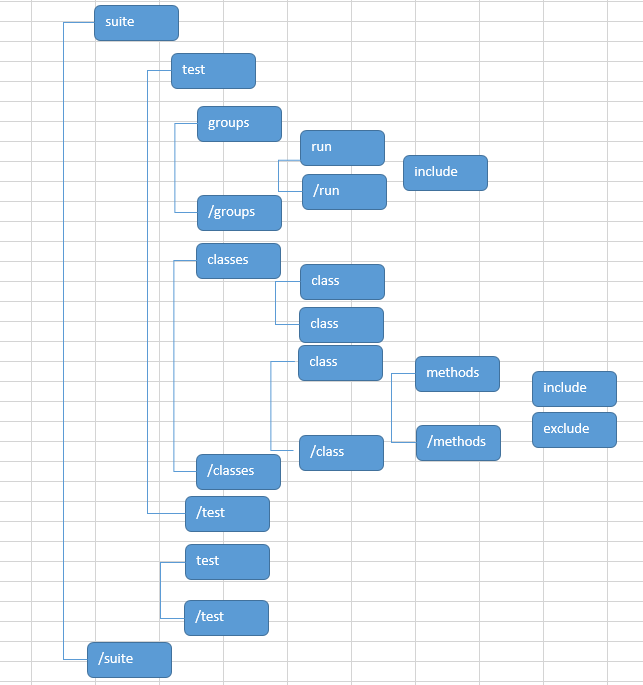
**Note:**TestNg will execute the test one by one

*2.1.6.2 Parallel Testing*

Now just set the “**parallel**” attribute to “**tests**” in the above used xml and give a run again. This time you will notice that your both browsers will open almost simultaneously and your test will run in parallel.

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  <suite name=*"Suite"* parallel=*"tests"*>  <test name=*"Test01"*>  <parameter name =*"browser"* value=*"chrome"*/>  <classes>  <class name=*"org.niteco.qa.CloudTesting.MultiBrowser"* />  </classes>  </test> <!-- Test -->  <test name=*"Test02"*>  <parameter name =*"browser"* value=*"ie"*/>  <classes>  <class name=*"org.niteco.qa.CloudTesting.MultiBrowser"* />  </classes>  </test> <!-- Test -->  </suite> <!-- Suite --> |

### Testng.xml

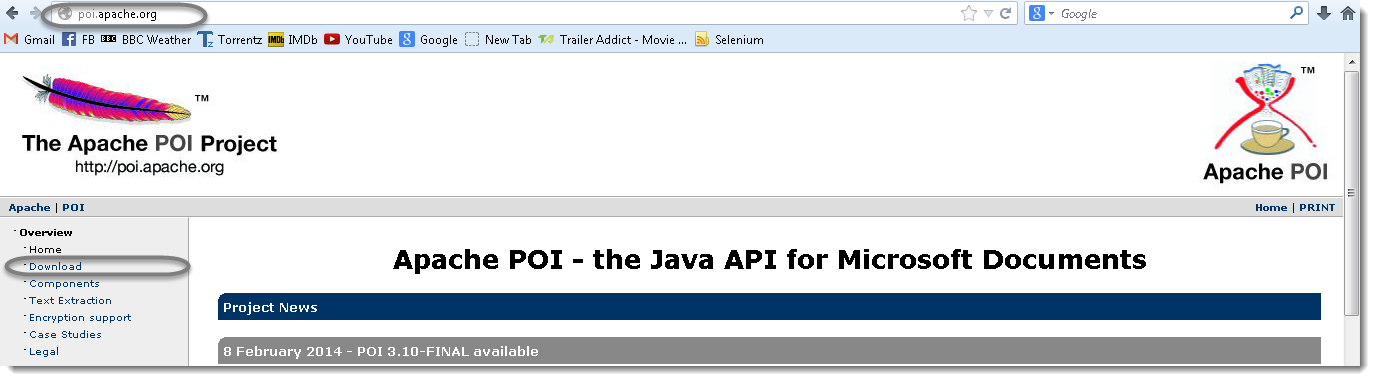


## **2.2 Data Driven Approach**

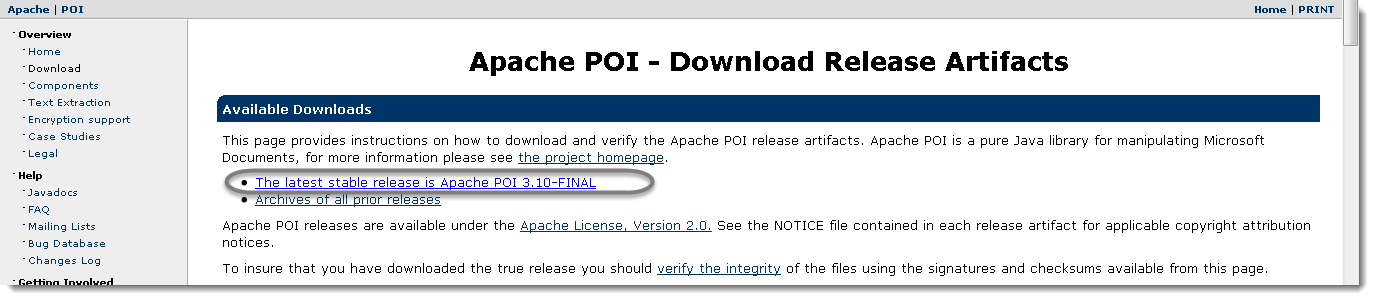
### 2.2.1 Download and Add POI library

*Step1. Download Apache POI*

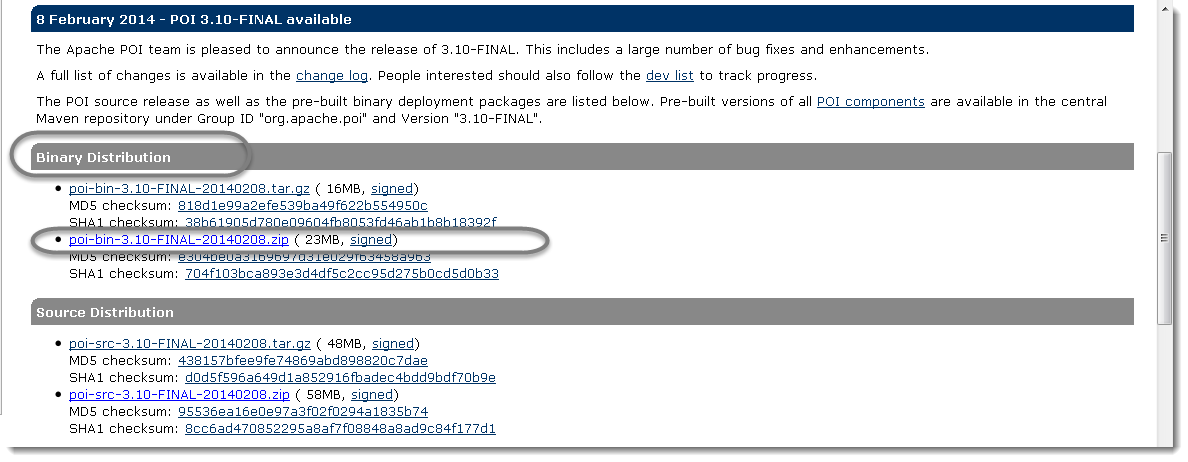
Go to http://poi.apache.org/ services and click on ‘**Download**‘ on the left side menu.



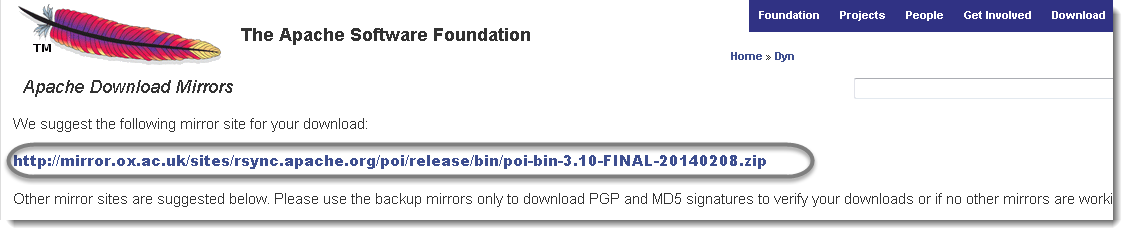
Click on the link for Apchae POI under ‘**Available Downloads**‘.



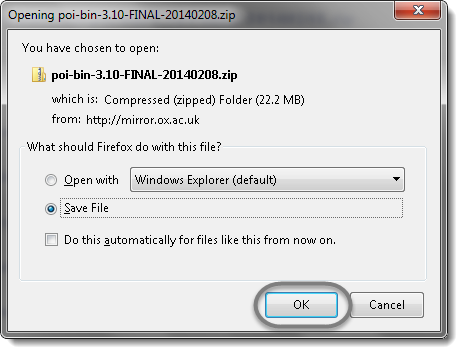
Click on the ZIP file to start the downloading.



Click on the highlighted link at the top of the page.

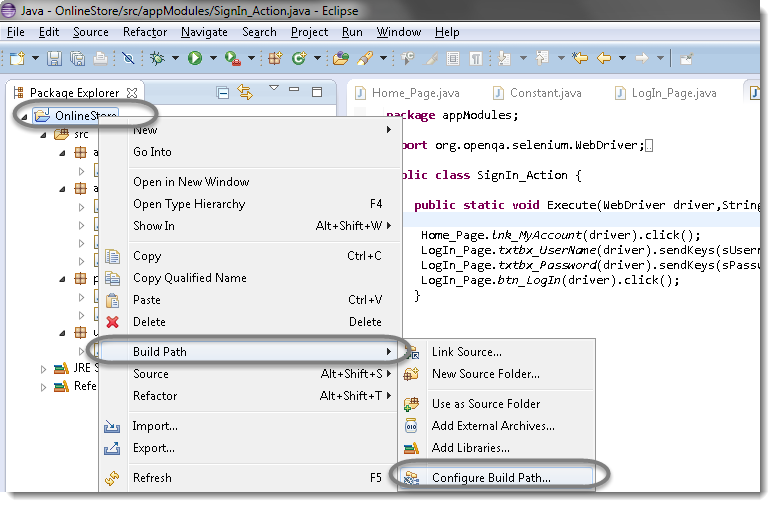


Select the radio button for ‘**Save File’** and click **OK**. Zip file will be saved on the system with in few seconds.

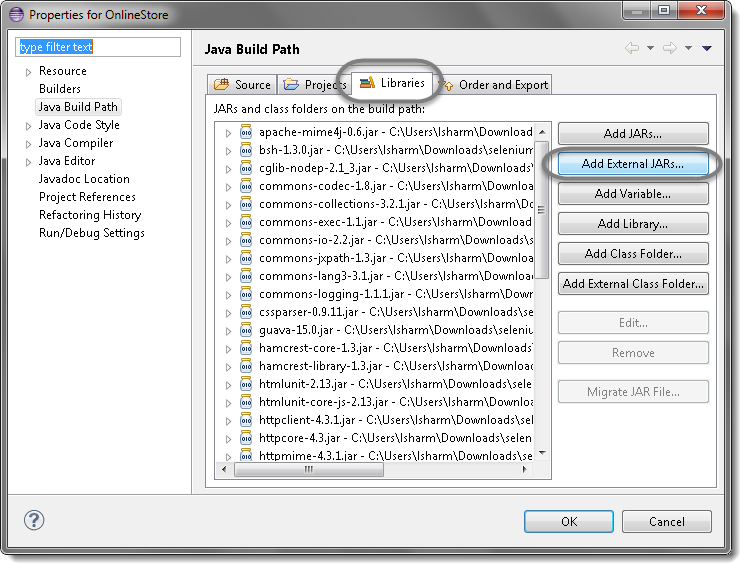


*Step2. Add Apache POI Jars*

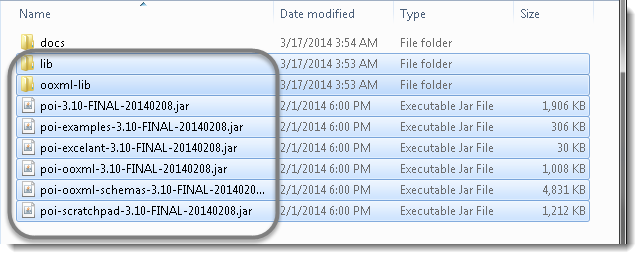
Right click the project name, navigate to **Build Path** and select ‘**Configure Build Path**‘.

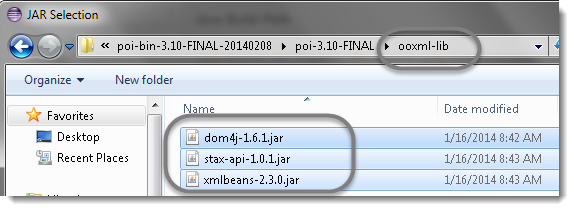


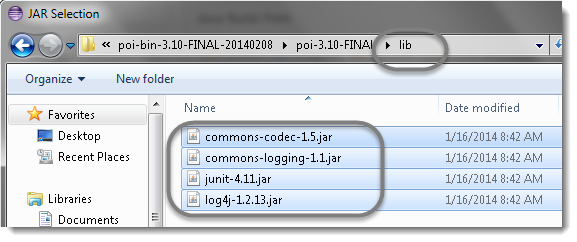
Click on **Add External JARS** and navigate to the folder where you have kept the [Apache POI](http://toolsqa.com/selenium-webdriver/add-apache-poi-jars/) jar files.



Select the Executable Jar files from all the folders.







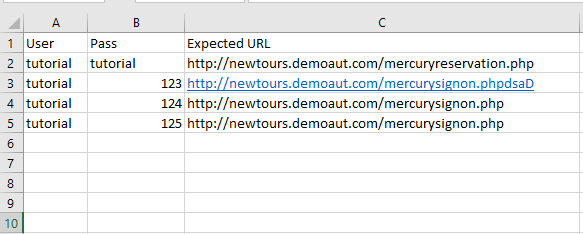
### 2.2.2 Data Driven Framework (Apache POI - Excel)

*This example for reading data from excel then return the two-dimension array for using in data provider*

*Step1. Create the method to read data from excel file then return an array of object*

|  |
| --- |
| package org.niteco.qa.NewToursDemo;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import java.util.Iterator;  import java.util.Properties;  import org.apache.poi.ss.usermodel.Cell;  import org.apache.poi.ss.usermodel.Row;  import org.apache.poi.xssf.usermodel.XSSFCell;  import org.apache.poi.xssf.usermodel.XSSFRow;  import org.apache.poi.xssf.usermodel.XSSFSheet;  import org.apache.poi.xssf.usermodel.XSSFWorkbook;  public class ExcelUtils {  @SuppressWarnings("null")  public static String[][] getExcelData(String filePath, String sheetName)  throws IOException {  FileInputStream inputStream = null;  XSSFWorkbook workbook = null;  XSSFSheet sheet;  XSSFRow rowData;  int x = 0, y = 0;  int totalRow = 0;  int totalCol = 0;  inputStream = new FileInputStream(filePath);  workbook = new XSSFWorkbook(inputStream);  sheet = workbook.getSheet(sheetName);  Iterator<Row> iterator = sheet.iterator();  // Get the size of array  while (iterator.hasNext()) {  Row nextRow = iterator.next();  Iterator<Cell> cellIterator = nextRow.cellIterator();    while (cellIterator.hasNext()) {  Cell cell = cellIterator.next();  totalRow = cell.getRowIndex();  totalCol = cell.getColumnIndex();  System.out.println("Total Row :" + totalRow);  System.out.println("Total Column :" + totalCol);  }  }  String testData[][] = new String[totalRow][totalCol + 1];  // Get data then put to array  x = 0;  for (int i = 1; i <= totalRow; i++, x++) {  y = 0;  for (int j = 0; j <= totalCol; j++, y++) {  rowData = sheet.getRow(i);  Cell cellData = rowData.getCell(j);  switch (cellData.getCellType()) {  case Cell.CELL\_TYPE\_STRING:  testData[x][y] = cellData.getStringCellValue();  System.out.print(testData[x][y]);  break;  case Cell.CELL\_TYPE\_BOOLEAN:  testData[x][y] = Boolean.toString(cellData  .getBooleanCellValue());  System.out.print(testData[x][y]);  break;  case Cell.CELL\_TYPE\_NUMERIC:  if (cellData.getCellStyle().getDataFormatString().contains("%")) {  Double value = cellData.getNumericCellValue() \* 100;  testData[x][y] = String.valueOf(value);  }  else  testData[x][y] = String.valueOf(cellData  .getNumericCellValue());  System.out.print(testData[x][y]);  break;  }  System.out.print(" ");  }  System.out.println();  }  workbook.close();  inputStream.close();  // return null;  return testData;  }  } |

*Step2. Create excel file and put test data in this file*



*Step 3: Create the class TestData to define test data for test method*

|  |
| --- |
| **package** org.niteco.qa.data;  **import** java.io.IOException;  **import** java.util.Properties;  **import** org.niteco.qa.NewToursDemo.CommonOperations;  **import** org.niteco.qa.NewToursDemo.ExcelUtils;  **import** org.testng.annotations.DataProvider;  **public** **class** TestData {  **static** Properties *p*;  @DataProvider(name="TC01")  **public** **static** Object[][] getLoginData() **throws** IOException {  *p* = CommonOperations.*readConfig*();  Object[][] data = ExcelUtils.*getExcelData*(*p*.getProperty("dataFile"),*p*.getProperty("TC01"));  **return** data;  }  } |

*Step4. Create test method to pass the test data from excel*

|  |
| --- |
| **package** org.niteco.qa.test;  **import** java.io.IOException;  **import** org.niteco.qa.NewToursDemo.TestBase;  **import** org.niteco.qa.action.LoginAction;  **import** org.niteco.qa.data.TestData;  **import** org.testng.Assert;  **import** org.testng.annotations.AfterMethod;  **import** org.testng.annotations.BeforeMethod;  **import** org.testng.annotations.Test;  **public** **class** TestLogin **extends** TestBase {  LoginAction objLoginAction;  @BeforeMethod  **public** **void** setupBeforeTest() **throws** IOException{  initTest();  }  @Test(dataProvider="TC01", dataProviderClass=TestData.**class**)  **public** **void** TC01(String user, String pwd, String expectedURL){  objLoginAction = **new** LoginAction(driver);  objLoginAction.login(user, pwd);  Assert.*assertTrue*(driver.getCurrentUrl().contains(expectedURL));  }  @AfterMethod  **public** **void** teardownMethod(){  driver.quit();  }  } |

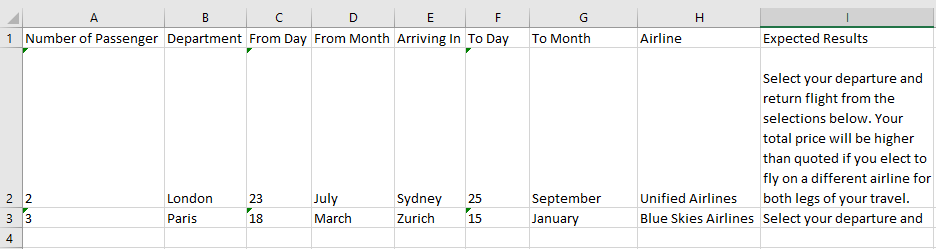
***2.1.6 Combine DataProvider & Factory***

Sometime, you need to pass the parameters on the annotated methods like @BeforeTest, @BeforeClass, @BeforeMethod,etc…You can combine the Dataprovider and Factory.

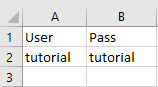
*Example:*

*Step1. Create the test data file then put data in this file*

*1.1 Data for test method (TC02) put to the sheet “TC02”*



*1.2 Data for BeforeMethod (setupBeforeMethod) put to the sheet “Login”*



*Step 2. Create the method to define test data*

|  |
| --- |
| @DataProvider(name="TC02")  **public** **static** Object[][] getFlightFinderData() **throws** IOException {  *p* = CommonOperations.*readConfig*();  Object[][] data = ExcelUtils.*getExcelData*(*p*.getProperty("dataFile"),*p*.getProperty("TC02"));  **return** data;  }  @DataProvider(name="login")  **public** **static** Object[][] getLoginValidData() **throws** IOException {  *p* = CommonOperations.*readConfig*();  Object[][] data = ExcelUtils.*getExcelData*(*p*.getProperty("dataFile"),*p*.getProperty("Login"));  **return** data;  } |

*3. Create test method*

|  |
| --- |
| **package** org.niteco.qa.test;  **import** java.io.IOException;  **import** org.niteco.qa.NewToursDemo.TestBase;  **import** org.niteco.qa.action.FlightFinderAction;  **import** org.niteco.qa.action.LoginAction;  **import** org.niteco.qa.data.TestData;  **import** org.testng.Assert;  **import** org.testng.annotations.AfterMethod;  **import** org.testng.annotations.BeforeMethod;  **import** org.testng.annotations.Factory;  **import** org.testng.annotations.Test;  **public** **class** TestFlightFinder **extends** TestBase {  LoginAction objLoginAction;  FlightFinderAction objFFAction;  String user;  String pwd;  @Factory(dataProvider="login",dataProviderClass=TestData.**class**)  **public** TestFlightFinder(String user, String pwd){  **this**.user = user;  **this**.pwd = pwd;  System.***out***.println("init class");  }  @BeforeMethod  **public** **void** setupBeforeMethod() **throws** IOException{  initTest();  objLoginAction = **new** LoginAction(driver);  objFFAction = **new** FlightFinderAction(driver);  objLoginAction.login(user, pwd);  }  @Test(dataProvider="TC02", dataProviderClass=TestData.**class**)  **public** **void** TC02(String numPassenger, String department, String fromDay, String fromMonth, String arrive, String toDay, String toMonth, String airline, String expectedResults){    objFFAction.findFlight(numPassenger, department, fromDay, fromMonth, arrive, toDay, toMonth, airline);  **try** {  Thread.*sleep*(3000);  } **catch** (InterruptedException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  Assert.*assertTrue*(driver.getPageSource().contains(expectedResults));  }  @AfterMethod  **public** **void** teardownMethod(){  driver.quit();  }  } |

## **2.3 Log4j Logging**

Log4j is a brilliant logging API available both on Java and .net framework. It consists of four main components:

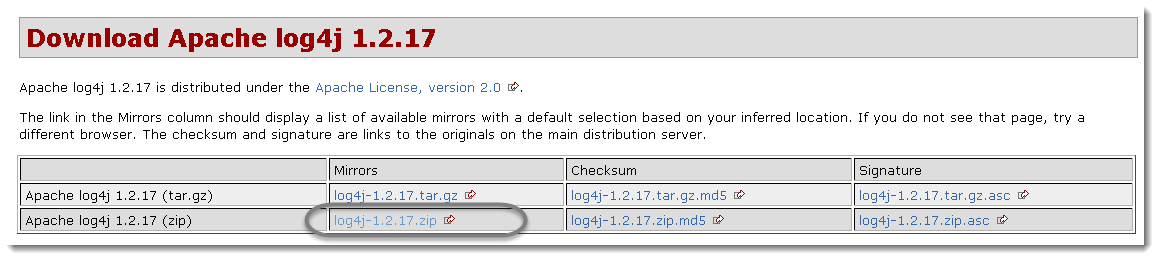
* *LogManager*
* *Loggers*
* *Appenders*
* *Layouts*

### 2.3.1 Download Log4j and add log4j.jar

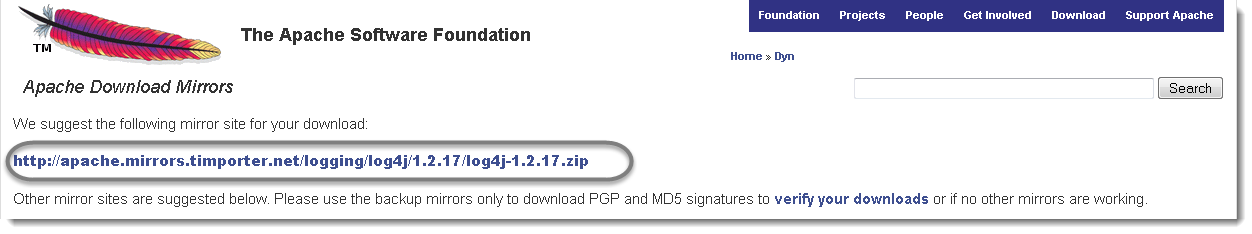
* Download Log4j
  + Go to http://logging.apache.org/ and click **Apache log4j**.
  + Click on “**Download**” on the left side menu.



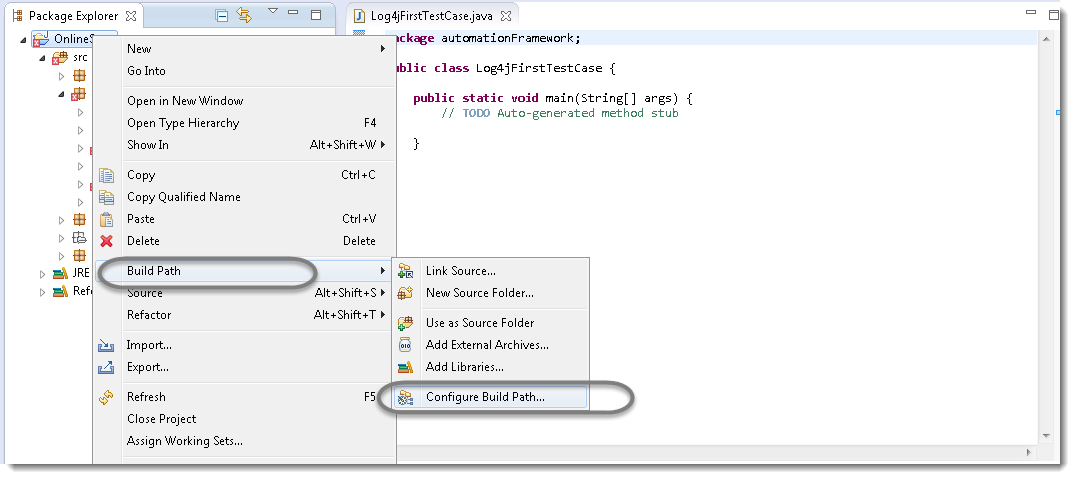
* + Click on the ZIP file under **Mirrors**column.



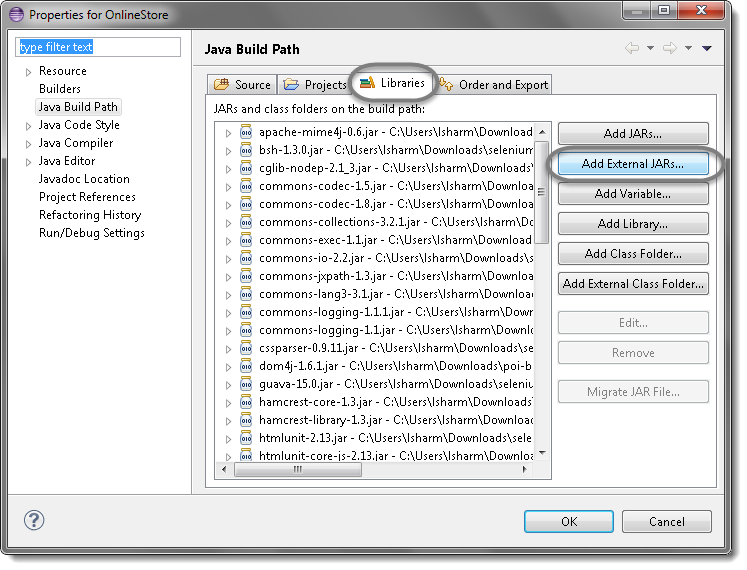
* + Click on the highlighted link at the top of the page.



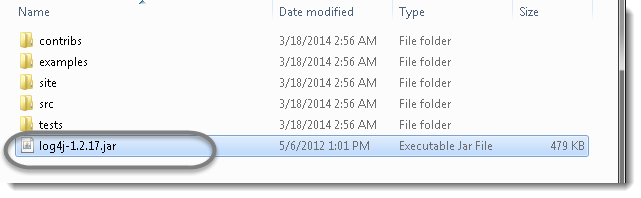
* Add Log4j jar
  + Right click the project name and navigate to **Build Path** and select “**Configure Build Path**“.



* + Click on **Add External JARS** and navigate to the folder where you have kept the Log4j jar files.



* + Select the Executable Jar File and click **Open**



### 2.3.2 Test Case with Log4j

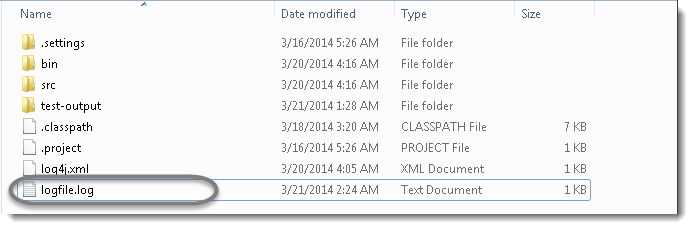
*Step 1: Create a new XML file – log4j.xml and place it under the Project root folder and Paste the following code in the log4j.xml file*

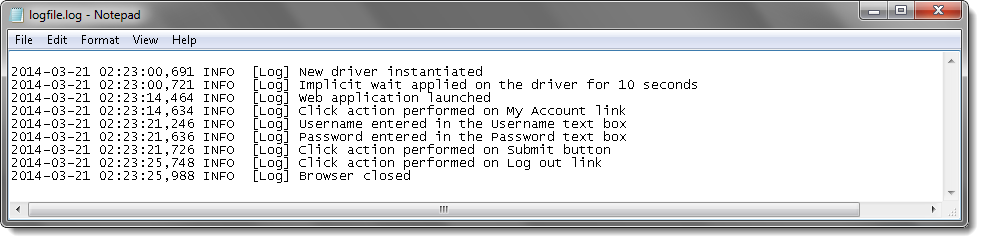
|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">  <log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/" debug="false">  <appender name="fileAppender" class="org.apache.log4j.FileAppender">  <param name="Threshold" value="INFO" />  <param name="File" value="logfile.log"/>  <layout class="org.apache.log4j.PatternLayout">  <param name="ConversionPattern" value="%d %-5p [%c{1}] %m %n" />  </layout>  </appender>  <root>  <level value="INFO"/>  <appender-ref ref="fileAppender"/>  </root>  </log4j:configuration> |

*Step 2: Now include logging code in to your test script*

|  |
| --- |
| package automationFramework;  import java.util.concurrent.TimeUnit;  import org.apache.log4j.Logger;  import org.apache.log4j.xml.DOMConfigurator;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  public class Log4j {  private static WebDriver driver;  private static Logger Log = Logger.getLogger(Log4j.class.getName());  public static void main(String[] args) {  DOMConfigurator.configure("log4j.xml");  // Create a new instance of the Firefox driver  driver = new FirefoxDriver();  Log.info("New driver instantiated");  //Put a Implicit wait, this means that any search for elements on the page could take the time the implicit wait is set for before throwing exception  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  Log.info("Implicit wait applied on the driver for 10 seconds");  //Launch the Online Store Website  driver.get("http://www.onlinestore.toolsqa.com");  Log.info("Web application launched");  // Find the element that's ID attribute is 'account'(My Account)  driver.findElement(By.id("account")).click();  Log.info("Click action performed on My Account link");  // Find the element that's ID attribute is 'log' (Username)  // Enter Username on the element found by above desc.  driver.findElement(By.id("log")).sendKeys("testuser\_1");  Log.info("Username entered in the Username text box");  // Find the element that's ID attribute is 'pwd' (Password)  // Enter Password on the element found by the above desc.  driver.findElement(By.id("pwd")).sendKeys("Test@123");  Log.info("Password entered in the Password text box");  // Now submit the form. WebDriver will find the form for us from the element  driver.findElement(By.id("login")).click();  Log.info("Click action performed on Submit button");  // Print a Log In message to the screen  System.out.println(" Login Successfully, now it is the time to Log Off buddy.");  // Find the element that's ID attribute is 'account\_logout' (Log Out)  driver.findElement(By.id("account\_logout"));  Log.info("Click action performed on Log out link");  // Close the driver  driver.quit();  Log.info("Browser closed");  }  } |

*Step 3: Check the output file “logfile.txt”. The output will look like below*





### 2.3.3 Log4j LogManager

Is the Manager of all logger objects. This is the static class that you refer to for creating Logger objects. *LogManager* also keeps a list of all the loggers being created by the application. If I were to summarize, *LogManager* does following work

*+ Create instances of Logger objects.*  
*+ Store references of all the created logger objects.*  
*+ Allow reuse of same logger object in different parts of the code.*

*Example 1****:*** This shows how we can create different logger instances out of a LogManager.

|  |
| --- |
| package Log4jSample;  import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;  public class SampleEntry {  public static void main(String[] args) {  //This is how we create a logger object  Logger logger1 = LogManager.getLogger("Logger1");  Logger logger2 = LogManager.getLogger("Logger2");  Logger logger3 = LogManager.getLogger("Logger3");  BasicConfigurator.configure();  logger1.info("This is logger 1");  logger2.info("This is logger 2");  logger3.info("This is logger 3");  }  } |

*Example 2*: We can also retrieve all the logger objects inside LogManager at a particular instance by using thegetCurrentLoggers() method

|  |
| --- |
| package Log4jSample;  import java.util.Enumeration;  import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;  public class SampleEntry {  public static void main(String[] args) {  // TODO Auto-generated method stub  Logger logger1 = LogManager.getLogger("Logger1");  Logger logger2 = LogManager.getLogger("Logger2");  Logger logger3 = LogManager.getLogger("Logger3");  BasicConfigurator.configure();  logger1.info("This is logger 1");  logger2.info("This is logger 2");  logger3.info("This is logger 3");  Enumeration loggers = LogManager.getCurrentLoggers();  while(loggers.hasMoreElements()){  logger3.info("Logger name is " + loggers.nextElement().getName());  }  }  } |

*Example 3****:*** One very important property of LogManager it lets us retrieve an existing logger object by name. Also, if we try to create a logger object with the same name as an existing logger object, it will pass on the reference of the existing logger object instead of creating one.

|  |
| --- |
| package Log4jSample;  import java.util.Enumeration;  import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;  public class SampleEntry {  public static void main(String[] args) {  Logger logger1 = LogManager.getLogger("Logger1");  Logger logger2 = LogManager.getLogger("Logger2");  Logger logger3 = LogManager.getLogger("Logger3");  BasicConfigurator.configure();  logger1.info("This is logger 1");  logger2.info("This is logger 2");  logger3.info("This is logger 3");  Logger logger1\_2 = LogManager.getLogger("Logger1");  Logger logger1\_3 = LogManager.getLogger("Logger1");  //You will see that LogManager doesnt create a new instance of logger  //Object with name Logger1, instead passes on the reference to the  //existing Logger1 object. We can confirm this with following lines  logger1\_2.info("The name of this logger is " + logger1\_2.getName());  if(logger1\_3.equals(logger1))  {  logger1\_3.info("Both objects are same");  }  else  {  logger1\_3.info("Logger1 and logger1\_2 are different objects");  }  }  } |

Out put of this code is

|  |
| --- |
| 1 [main] INFO Logger1 - This is logger 1  2 [main] INFO Logger2 - This is logger 2  2 [main] INFO Logger3 - This is logger 3  2 [main] INFO Logger1 - The name of this logger is Logger1  2 [main] INFO Logger1 - Both objects are same |

### 2.3.4 Log 4j Appenders

* ***File Appender****:*To log in a file

*Example:*

|  |
| --- |
| package Log4jSample;  import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.FileAppender;  import org.apache.log4j.Layout;  import org.apache.log4j.Level;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;  import org.apache.log4j.SimpleLayout;  public class SampleEntry {  public static void main(String[] args) {  BasicConfigurator.configure();  Logger OurLogger = LogManager.getLogger("OurLogger");  //create a FileAppender object and  //associate the file name to which you want the logs  //to be directed to.  //You will also have to set a layout also, here  //We have chosen a SimpleLayout  FileAppender fileAppender = new FileAppender();  fileAppender.setFile("logfile.txt");  fileAppender.setLayout(new SimpleLayout());  //Add the appender to our Logger Object.  //from now onwards all the logs will be directed  //to file mentioned by FileAppender  OurLogger.addAppender(fileAppender);  fileAppender.activateOptions();  //lets print some log 10 times  int x = 0;  while(x < 11){  OurLogger.debug("This is just a log that I want to print " + x);  x++;  }  }  } |

 This is how the logs look in the logfile:

|  |
| --- |
| DEBUG - This is just a log that I want to print 0  DEBUG - This is just a log that I want to print 1  DEBUG - This is just a log that I want to print 2  DEBUG - This is just a log that I want to print 3  DEBUG - This is just a log that I want to print 4  DEBUG - This is just a log that I want to print 5  DEBUG - This is just a log that I want to print 6  DEBUG - This is just a log that I want to print 7  DEBUG - This is just a log that I want to print 8  DEBUG - This is just a log that I want to print 9  DEBUG - This is just a log that I want to print 10 |

* ***Console Appenders:*** directs the logs to ***System.err*** and***System.out*** streams. These streams are also read by Console and hence the output is displayed at the console as well.

*Example:*

|  |
| --- |
| import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.ConsoleAppender;  import org.apache.log4j.FileAppender;  import org.apache.log4j.Layout;  import org.apache.log4j.Level;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;  import org.apache.log4j.SimpleLayout;  public class SampleEntry {  public static void main(String[] args) {  // TODO Auto-generated method stub  BasicConfigurator.configure();  Logger OurLogger = LogManager.getLogger("OurLogger");  //create a ConsoleAppender object  //You will also have to set a layout also, here  //We have chosen a SimpleLayout  ConsoleAppender ConsoleAppender = new ConsoleAppender();  ConsoleAppender.setLayout(new SimpleLayout());  //Add the appender to our Logger Object.  //from now onwards all the logs will be directed  //to file mentioned by FileAppender  OurLogger.addAppender(ConsoleAppender);  ConsoleAppender.activateOptions();  //lets print some log 10 times  int x = 0;  while(x &lt; 11){  OurLogger.debug("This is just a log that I want to print " + x);  x++;  }  }  } |

* ***JDBC Appenders:*** *are*used to write logs to a Data base. These appenders accept data base connection credentials to connect to DB.

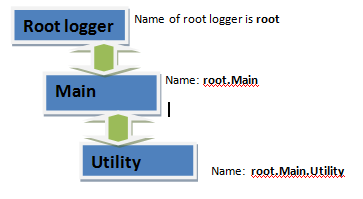
*Example:*

|  |
| --- |
| package Log4jSample;  import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.ConsoleAppender;  import org.apache.log4j.FileAppender;  import org.apache.log4j.Layout;  import org.apache.log4j.Level;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;  import org.apache.log4j.SimpleLayout;  import org.apache.log4j.jdbc.JDBCAppender;  public class SampleEntry {  public static void main(String[] args) {  BasicConfigurator.configure();  Logger OurLogger = LogManager.getLogger("OurLogger");  //create a JDBCAppender class instance  JDBCAppender dataBaseAppender = new JDBCAppender();  //Provide connection details to the  //Database appender  dataBaseAppender.setURL("jdbc:mysql://localhost/test"); //Connection url  dataBaseAppender.setUser("User1"); //Username for the DB connection  dataBaseAppender.setPassword("ThisPassword"); //Password for the DB connection  dataBaseAppender.setDriver("com.mysql.jdbc.Driver"); // Drivers to use to connect to DB  //set the sql insert statement that you want to use  dataBaseAppender.setSql("INSERT INTO LOGS VALUES ('%x', now() ,'%C','%p','%m'");  //activate the new options  dataBaseAppender.activateOptions();  //Attach the appender to the Logger  OurLogger.addAppender(dataBaseAppender);  int x = 0;  while(x < 11){  OurLogger.debug("This is just a log that I want to print " + x);  x++;  }  }  } |

Note: This code explains how to set up a ***JDBCAppender***object and use it for logging. This statement is used to insert logs in the desired database table. We have used the statementINSERT INTO LOGS VALUES (‘%x’, now() ,’%C’,’%p’,’%m’) It says that logs are inserted in the table named LOGS.

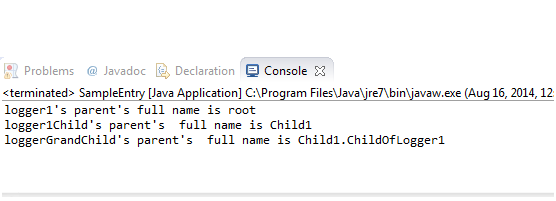
### 2.3.5 Log4j Loggers

* This is the most important class that you will need. This is the object which lets you log information to the required Log location, be it console or a file or even a database.
* *Logger* objects follow hierarchy similar to class hierarchy in any OOP language. Naming convention of *Logger* hierarchy is in the name. Each objects name decide which hierarchy it follows. For example we have a logger named “*Main.Utility*“. So Utility is the child of Main and Main is the father of Utility. Also, all *Loggers* are derived from *root Logger*. The actual hierarchy will be *root.Main.Utility* with root being ancestor of Utility and Father of Main. This can be shown in a diagram as
* These relationships are managed by the LogManager class. Lets illustrate it using an example:



|  |
| --- |
| package Log4jSample;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;  public class SampleEntry {  public static void main(String[] args) {  // TODO Auto-generated method stub  Logger chance = LogManager.getLogger(SampleEntry.class.getName());  Logger logger1 = LogManager.getLogger("Child1");  Logger logger1Child = logger1.getLogger("Child1.ChildOfLogger1");  Logger loggerGrandChild = LogManager.getLogger("Child1.ChildOfLogger1.GrandChild");  System.out.println("logger1's full name is " + logger1.getParent().getName());  System.out.println("logger1Child's full name is " + logger1Child.getParent().getName());  System.out.println("loggerGrandChild's full name is " + loggerGrandChild.getParent().getName());  }  } |

Output:



***Logging levels:*** Logger class have following print methods that help you log information.

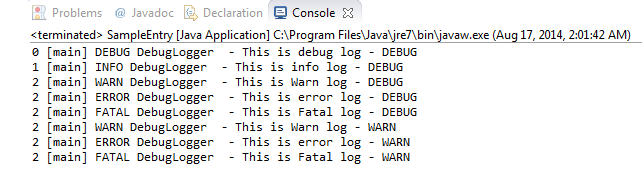
* *Trace*
* *Debug*
* *Info*
* *Warn*
* *Error*
* *Fatal*

You can set log level of a logger using the*Logger.setLevel* method. Once you set the Log level of your logger only loggers with that and higher level will be logged. Log levels have following order  
***TRACE < DEBUG < INFO < WARN < ERROR < FATAL.***

**Example:**

|  |
| --- |
| package Log4jSample;  import org.apache.log4j.BasicConfigurator;  import org.apache.log4j.Level;  import org.apache.log4j.LogManager;  import org.apache.log4j.Logger;  public class SampleEntry {  public static void main(String[] args) {  BasicConfigurator.configure();  Logger Mylogger = LogManager.getLogger("DebugLogger");  //Setting up the log level of both loggers  Mylogger.setLevel(Level.DEBUG);  Mylogger.trace("This is the trace log - DEBUG");  Mylogger.debug("This is debug log - DEBUG");  Mylogger.info("This is info log - DEBUG");  Mylogger.warn("This is Warn log - DEBUG");  Mylogger.error("This is error log - DEBUG");  Mylogger.fatal("This is Fatal log - DEBUG");  Mylogger.setLevel(Level.WARN);  Mylogger.trace("This is the trace log - WARN");  Mylogger.debug("This is debug log - WARN");  Mylogger.info("This is info log - WARN");  Mylogger.warn("This is Warn log - WARN");  Mylogger.error("This is error log - WARN");  Mylogger.fatal("This is Fatal log - WARN");  }  } |

* + Output:

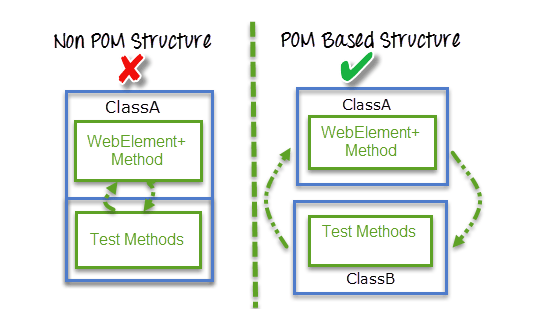


You can see that when Log level was DEBUG all the logs DEBUG to FATAL are displayed. Once the log level is set to WARN all the logs from WARNS to FATAL are displayed.

## **2.4 Page Object Model (POM) and Page Factory**

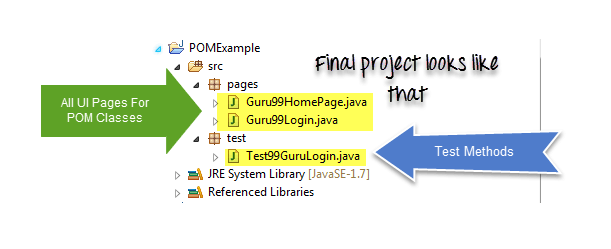
### 2.4.1 Page Object Model

2.4.1.1 Introduction

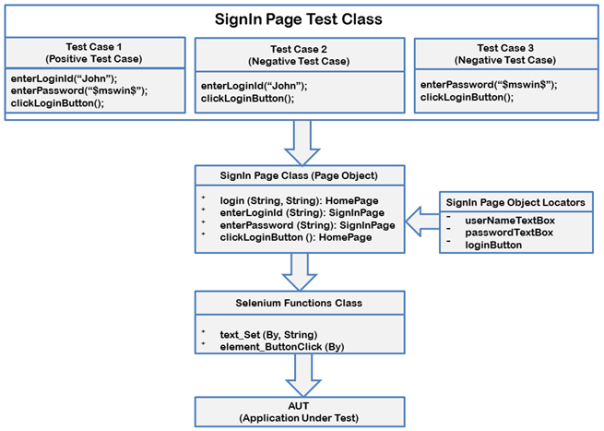


* **Page Object Model** is a design pattern to create **Object Repository** for web UI elements.
* Under this model, for each web page in the application there should be corresponding page class.
* This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.
* Name of these methods should be given as per the task they are performing i.e., if a loader is waiting for payment gateway to be appear, POM method name can be waitForPaymentScreenDisplay().

*Example:* project structure



*Example:* UML diagram of test case flow



Advantages of POM:

* Make our code cleaner and easy to understand
* Use the same object repository for a different purpose with different tools
* Method get more realistic names which can be easily mapped with the operation in UI.

*2.4.1.2 How to implement*

We have a test case:

Step1. Go to Guru 99 Demo site

Step2. In home page check text **"Guru99 Bank"** is present

Step3. Login into application

Step4. Verify that the Home page contains text as "Manger Id : demo"

* + This test case deal with 2 pages: Login page and Home page. So that we need to create 2 POM classes
* Create Guru99 Login Page class

|  |
| --- |
| package pages;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  public class Guru99Login {  WebDriver driver;  By user99GuruName = By.name("uid");  By password99Guru = By.name("password");  By titleText =By.classame("barone");  By login = By.name("btnLogin");  public Guru99Login(WebDriver driver){  this.driver = driver;  }  //Set user name in textbox  public void setUserName(String strUserName){  driver.findElement(user99GuruName).sendKeys(strUserName);;  }  //Set password in password textbox  public void setPassword(String strPassword){  driver.findElement(password99Guru).sendKeys(strPassword);  }  //Click on login button  public void clickLogin(){  driver.findElement(login).click();  }  //Get the title of Login Page  public String getLoginTitle(){  return driver.findElement(titleText).getText();  }  /\*\*  \* This POM method will be exposed in test case to login in the application  \* @param strUserName  \* @param strPasword  \* @return  \*/  public void loginToGuru99(String strUserName,String strPasword){  //Fill user name  this.setUserName(strUserName);  //Fill password  this.setPassword(strPasword);  //Click Login button  this.clickLogin();  }  } |

* Create the class Guru99 Home page

|  |
| --- |
| package pages;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  public class Guru99HomePage {  WebDriver driver;  By homePageUserName = By.xpath("//table//tr[@class='heading3']");  public Guru99HomePage(WebDriver driver){  this.driver = driver;  }  //Get the User name from Home Page  public String getHomePageDashboardUserName(){  return driver.findElement(homePageUserName).getText();  }  } |

* Create the test class

|  |
| --- |
| package test;  import java.util.concurrent.TimeUnit;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  import org.testng.Assert;  import org.testng.annotations.BeforeTest;  import org.testng.annotations.Test;  import pages.Guru99HomePage;  import pages.Guru99Login;  public class Test99GuruLogin {  WebDriver driver;  Guru99Login objLogin;  Guru99HomePage objHomePage;  @BeforeTest  public void setup(){  driver = new FirefoxDriver();  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  driver.get("http://demo.guru99.com/V4/");  }  /\*\*  \* This test case will login in http://demo.guru99.com/V4/  \* Verify login page title as guru99 bank  \* Login to application  \* Verify the home page using Dashboard message  \*/  @Test(priority=0)  public void test\_Home\_Page\_Appear\_Correct(){  //Create Login Page object  objLogin = new Guru99Login(driver);  //Verify login page title  String loginPageTitle = objLogin.getLoginTitle();  Assert.assertTrue(loginPageTitle.toLowerCase().contains("guru99 bank"));  //login to application  objLogin.loginToGuru99("mgr123", "mgr!23");  // go the next page  objHomePage = new Guru99HomePage(driver);  //Verify home page Assert.assertTrue(objHomePage.getHomePageDashboardUserName().toLowerCase().contains("manger id : mgr123"));  } |

### 2.4.2 Page Factory

*2.4.2.1 Introduction*

* PageFactory Class is an extension to the Page Object Model
* It is used to initialize the elements of the Page Object or instantiate the Page Objects itself.
* It is used to initialize element of a page class without having to use “Findelemetn” or “FindElements”
* Annotations “@FindBy” can be used to supply descriptive names of target objects to improve code readability. This annotation can accept the tagName, **partialLinkText, name, linkText, id, css, className, xpath**as attributes.



* + Now we refactor the example above using Page Factory
* GuruLogin page

|  |
| --- |
| package PageFactory;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.WebElement;  import org.openqa.selenium.support.FindBy;  import org.openqa.selenium.support.PageFactory;  public class Guru99Login {  /\*\*  \* All WebElements are identified by @FindBy annotation  \*/  WebDriver driver;  @FindBy(name="uid")  WebElement user99GuruName;  @FindBy(name="password")  WebElement password99Guru;  @FindBy(className="barone")  WebElement titleText;  @FindBy(name="btnLogin")  WebElement login;  public Guru99Login(WebDriver driver){  this.driver = driver;  //This initElements method will create all WebElements  PageFactory.initElements(driver, this);  }  //Set user name in textbox  public void setUserName(String strUserName){  user99GuruName.sendKeys(strUserName);  }  //Set password in password textbox  public void setPassword(String strPassword){  password99Guru.sendKeys(strPassword);  }  //Click on login button  public void clickLogin(){  login.click();  }  //Get the title of Login Page  public String getLoginTitle(){  return titleText.getText();  }  /\*\*  \* This method will be exposed in test case to login in the application  \* @param strUserName  \* @param strPasword  \* @return  \*/  public void loginToGuru99(String strUserName,String strPasword){  //Fill user name  this.setUserName(strUserName);  //Fill password  this.setPassword(strPasword);  //Click Login button  this.clickLogin();  }  } |

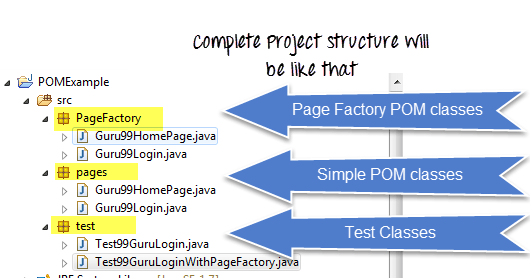
* Guru99 Homepage

|  |
| --- |
| package PageFactory;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.WebElement;  import org.openqa.selenium.support.FindBy;  import org.openqa.selenium.support.PageFactory;  public class Guru99HomePage {  WebDriver driver;  @FindBy(xpath="//table//tr[@class='heading3']")  WebElement homePageUserName;  public Guru99HomePage(WebDriver driver){  this.driver = driver;  //This initElements method will create all WebElements  PageFactory.initElements(driver, this);  }  //Get the User name from Home Page  public String getHomePageDashboardUserName(){  return homePageUserName.getText();  }  } |

* Create the test case

|  |
| --- |
| package test;  import java.util.concurrent.TimeUnit;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  import org.testng.Assert;  import org.testng.annotations.BeforeTest;  import org.testng.annotations.Test;  import PageFactory.Guru99HomePage;  import PageFactory.Guru99Login;  public class Test99GuruLoginWithPageFactory {  WebDriver driver;  Guru99Login objLogin;  Guru99HomePage objHomePage;  @BeforeTest  public void setup(){  driver = new FirefoxDriver();  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  driver.get("http://demo.guru99.com/V4/");  }  /\*\*  \* This test go to http://demo.guru99.com/V4/  \* Verify login page title as guru99 bank  \* Login to application  \* Verify the home page using Dashboard message  \*/  @Test(priority=0)  public void test\_Home\_Page\_Appear\_Correct(){  //Create Login Page object  objLogin = new Guru99Login(driver);  //Verify login page title  String loginPageTitle = objLogin.getLoginTitle();  Assert.assertTrue(loginPageTitle.toLowerCase().contains("guru99 bank"));  //login to application  objLogin.loginToGuru99("mgr123", "mgr!23");  // go the next page  objHomePage = new Guru99HomePage(driver);  //Verify home page  Assert.assertTrue(objHomePage.getHomePageDashboardUserName().toLowerCase().contains("manger id : mgr123"));  }  } |

* + The completed project structure



*Note:*

*Should use Encapsulation – OOPs Principle (Some time you only allow another to get/set your data – attribute of one object)*

***Encapsulation*** *in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as single unit. In encapsulation the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class, therefore it is also known as data hiding.*

*To achieve encapsulation in Java*

* *Declare the variables of a class as private.*
* *Provide public setter and getter methods to modify and view the variables values.*

***References:***

<http://www.guru99.com/page-object-model-pom-page-factory-in-selenium-ultimate-guide.html>

<https://solutionscafe.wordpress.com/2014/01/06/page-object-pattern/>

http://toolsqa.com/selenium-webdriver/c-sharp/encapsulation-oops-principle/