**SELENIUM**

* **Automation Testing**

Writing the Test Script with the help of a automation tool by using a Programming language is known as Automation Testing.

* **Automation Tool**

There are two types of tool

Non Functional Tool

Functional Tool

JMETER

LOAD RUNNER

LOAD COMPLETE

Paid Tools

EX-UFT,Test Complete,

* **Selenium Community**

Selenium is a community where all selenium tools are available. It is not a Tool.

Selenium IDE

Selenium

GRID Selenium

Remote control

Selenium WebDriver

* **HTML(Hypertext markup Language)**
* HTML is used to design webpages.Webpages are interface between client(User) and Web server.
* All internet and browser based application required html based web pages to interact the application or server.
* HTML is a collection of html tages,all HTML tag are enclosed with angular brackets (<>).
* Each html tag should have open tag and end tag.
* HTML may or may not contain backend attributes.
* All HTML tags are reserved or inbuild keywords.

**EX.**

<html></html>

**<**title**></**title**>**

**<**body></body>

<input></input>

<a></a>

<select></select>

**Sample Web page:-**

**<html>**

**<title> Selnium Tutorial</title>**

**<body>**

**<input type=’text’ id=’user’ name=’username’/>**

**<input type=’text’ id=’password’ name=’pwd’/>**

**<input type=’button’ value=’Login Now’/>**

**</body>**

**</html>**

**Some HTML tags**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Webelement** | **HTML tag** | | **Mandatory Attribute** | **Value** | **Additional Attribute** | **Value** |
| **Edit box** | **<input>** | **type** | | **text** | **Id or name** | **Custom value** |
| **Button** | **<input>** | **type** | | **Button/submit** | **Id or name** | **Custom value** |
| **checkbox** | **<input>** | **type** | | **Checkbox** | **Id or name** | **Custom value** |
| **Radio button** | **<input>** | **type** | | **Radio** | **Id or name** | **Custom value** |
| **Links** | **<a>** | **href** | | **URL** | **Id or name** | **Custom value** |
| **Select box dropdown** | **<Select>**  **<option>** | **N/A** | | **N/A** | **Id or name** | **Custom value** |
| **Image** | **Img** | **Src** | | **Location of image** | **Id or name** | **Custom value** |
| **Multiple dropdown** | **<Select>**  **<option>** | **Multiple** | | **true** | **Id or name** | **Custom value** |
| **Webtable** | **<table>**  **<tbody>**  **<tr>**  **<td>**  **</td>**  **</tr>**  **</tbody>**  **</table>** | **N/A** | | **N/A** | **Id or name** | **Custom vale** |

**For checkbox**

<input type=’checkbox’ id=’chk’/>

**For Radio button**

**<**input type=’radio’ name=’rd’ id=’radi1’>MALE</input>

**<**input type=’radio’ name=’rd’ id=’radi2’>FEMALE</input>

**For LINK**

<a href=’http://gmail.com’>Gmail</a>

**For Image**

<img src=’image2.jpg’ width=’100’ height=’80’/>

**For Dropdown**

<select name=’sel’>

<option>INDIA</option>

<option>AUS</option>

<option>PAK</option>

<option>Australia</option>

</select>

<a href=’https://gmail.com’ id=’gm’>Gmail</a>

Html tag, mandatory attribute, value, additional attribute, value, visible text, closed tag

* If you want to design multiple select drop down

<select name=’sel2’ multiple=’true’>

<option>INDIA</option>

<option>PAK</option>

<option>USA</option>

<option>UK</option>

* Creating table

<table name=’tab1’ border =’b’>

<tbody>

<tr>

<td> <input type=’checkbox’/></td>

<td><a href=’https://msg1.com’>msg1</a></td>

</tr>

<tr>

<td> <input type=’checkbox’/></td>

<td><a href=’https://msg2.com’>msg2</a></td>

</tr>

<tr>

<td> <input type=’checkbox’/></td>

<td><a href=’https://msg3.com’>msg3</a></td>

</tr>

</tbody>

</table>

Notes

**NOTES**

<h1>----For header

<p>------For Paragraph

<spam>-------for color

<div> or <br>------For division

<b>----- for border of any words

* **Object Identification**
* All selenium tools will identify the object based on locators.
* Locators are written by looking at the html source code of webelement.

All locators are used to identified the weblement.

There are 8 locators which are used to identify elements.

* Id
* Name
* Xpath
* CSS selector
* Linktext
* Partial linktext
* Tagname
* Classname

Id and name directly we can use by seeing html source code.

* **Xpath**

Xpath is a way to navigate to entire html document and identify object based on webelement backend attribute.

**Symbols used while writing Xpath**

* //---Go to entire html document
* /---Go to child
* []---Go to parent
* ------Provide backend attribute for the webelement.
* @----attribute Symbol
* \*-----match any html Tag

**There are two types of X-path**

**----Absolute Xpath**

**-----Relative Xpath**

**------Absolute Xpath-**

* Whenever Xpath is written from root node to child note followed by forward slash is called as absolute X-path.
* In real time we will not use because if the location of element will get change,it will fail to identify the element.
* **Chropath**-----It is the plug-in of chrome browser which is used to inspect the element as well as verify the Xpath.
* **Try for Xpath**-----It is the plug-in for firefox browser which is used to verify xpath in firefox browser.

**--------Relative Xpath-**

* Whenever Xpath is written directly to the webelement using webelement backend attribute is called relative X-path. Relative x-path never fails to identify the object if even though the object location change.

**Syntax: //htmtag[@attribute=’value’]**

* **Case1**

**Whenever the object or webelement not able to find using one attribute ,in such case provide multiple attribute using ‘and keyword’**

Ex.//inut[@type=’text’ and @name=’usr’]

Or //input[@type=’checkbox’ and @name=’usr’]

* **Case2**

**How to identify the object when similar webelement is present in UI.**

//div[@class=’fs’]/input[@type=’button’]

Or

//div[@class=’Rb’]/input[@type=’button’]

* **Case 3**

**How to identify when object is present inside the webtable**.

//tr[2]/td[1]/input

* **Case 4**

**How to identify the webelement when multiple static webtable is present in UI.**

//table[@name=’tab2’]/tbody/tbody/tr[2]/td[1]/input

* **Case 5**

**How to identify the dynamic object when object is present inside dynamic webtable.**

//tr[td[a[@href=’https://msg2.com’]]]/td[1]/input

**Steps to work with dynamic Webtable**

* Find a dependent and independent webelement in webtable
* Write down html tree structure for dependent and independent webelement.
* Find a common parent for dependent and independent element.
* Construct the xpath (by taking help of independent element try to identify the dependent element.)

//tr[td[a[text()=’airtel’]]]/td[6]/input

**Advance Xpath functions**

1)**Text()**

**2)Normalize-space()**

**3)Contains()**

**4)following-sibling**

**5)Descendant**

**1)Text()-**It is used to find the webelement based on visible text

**Syntax-**//html[text()=’Visible text’]

Ex. //h1[text()=’Welcome to Flipkart’]

//a[text()=’Gmail’]

**Note:-**

Text function can’t be used to identify the dynamic text because text() is a complete string matching function.

* Text function always use to identify the object using visible text.
* Text function navigate to entire html source and check for expected value which returns value if complete string matches with UI.
* Text function can’t ignore space before and after the string.
* Text function can’t identify the object using part of string
* Text function can’t identify dynamic object.

**2)Normalize-space**

It is used to ignore the spaces present before and after string and compare string with string.It is the supporting function for attribute and text function only.

**Syntax:**//html[normalize-space(@attribute/text())=’Value/Visible text’]

**Ex.**

**Html source code**

<h2> Product List </h2>

**Xpath**=//h2[normalize-space(text())=’Product List’]

Or

**Html source code**

<input type=’button’ value=’ show’/>

**Xpath**=//input[normalize-space(@value)=’show’)]

**Note:**-Normalize space will not remove space between the string.

**3)Contains()**

**Syntax:-**//htmltag[contains(@attribute/text,’value/visible text’)]

* It is used identify the dynamic object
* Contains() navigate to entire html document and check for expected value, it returns true if expected value partially matches with UI.
* Contains() can automatically ignore the space before and after the string.
* Contains() can identify the object using part of string or complete string.
* Contains() play major role while working Ajax application.
* Ajax is a java framework which is used to develop dynamic object.

Ex.

//input[contains(@value,’RS’)]

Or

//input[contains(text(),’show’)]

**4)Following-sibling and preceding sibling**

* It is used to identify the next immediate siblings from the current html tag.
* Whenever webelement is changing its attribute in such case we will take help of sibling to identify dynamic object.
* It is used to identify next immediate html tag from current html tag.
* Preceding sibling is used to identify previous immediate html tag from current html tag.

**Syntax----**

//htmltag/following-sibling::htmltag

//htmltag/preceding-sibling::htmltag

Ex.

//img[@src=’img1.jpg’]/following-sibling::input[1]

//img[@src=’img1.jpg’]/following-sibling::input[contains(@value,’RS’)]

//span[text()=’Rs 500-Rs900’]/preceding-sibling::input

* **Selenium IDE**
* It is a record and play back tool.
* It is the plug of chrome browser.

**How to download IDE plug-in**

--Open browser

---Navigate to google

---Search for seleniumide for chrome

---click on add-ons links

----click on add to chrome

----click on add extension

----Restart the browser.

-----In upper right side logo of selenium ide will display**.**

**Steps of Selenium Ide**

* **Command-** It is the predefined keyword which is used to write script.
* **Target-** It is used to identify the webelement.
* **Value-** It is used to pass input data to webelement.

**Some command used to write script**

Open, sendkeys, clickAndWait etc…

**Batch execution**

Collection of multiple testcases is called batch execution in a single click is called batch Execution.

**Steps to create Test suite**

* Click on the dropdown in left upper side.
* Select Test suite.
* Add multiple test cases
* Click on Run as Test suite to run test suite.

**Advantage of Selenium IDE**

* It is a UI based and record and playback tool.
* Easy to write /automate test script and easy to execute.
* Batch execution is easier.
* Launches new browser for every testcase.

**Disadvantage of Selenium IDE**

* Works only with html language.
* For loop and if conditional statements are not allowed.
* Can’t create reusable method.
* **Selenium RC**
* It is a collection of java scripts.
* It is not a UI based tool and have record and playback feature.
* It supports multiple browser, operating system and multiple language.

**Disadvantage**

* It will not work with secure network because selenium RC java script method can’t perform action on secure application.
* **Selenium Webdriver**
* It is a collection of core java APIs.
* It is not a UI based and doesn’t have record and playback tool.
* It Support multiple browser and multiple operating system.
* It also support secure network.
* WebDriver is a interface.

**Selenium IDE Selenium RC Selenium WebDriver**

|  |  |  |
| --- | --- | --- |
| It supports Record and playback | It doesn’t supports Record and playback | It doesn’t supports Record and playback |
| Doesn’t required to start server before executing the test script. | Required to start server before executing the test script. | Doesn’t required to start server before executing the test script. |
| It is a GUI Plug-in | It is standalone java program which allow you to run Html test suites. | It actual core API which has binding in a range of languages. |
| Core engine is Javascript based | Core engine is Javascript based | Interacts natively with browser application |
| Very simple to use as it is record & playback. | It is easy and small API | As compared to RC, it is bit complex and large API. |
| It is not object oriented | API’s are less Object oriented | API’s are entirely Object oriented |
| It doesn’t supports of moving mouse cursors. | It doesn’t supports of moving mouse cursors. | It supports of moving mouse cursors. |
| It does not supports listeners | It does not supports listeners | It supports the implementation of listeners |
| It does not support to test iphone/Android applications | It does not support to test iphone/Android applications | It support to test iphone/Android applications |

* **WebDriver Interface**
* It is a collection of jar files.
* Jars are excutable class file which is collection of core java classes.
* Basically it is interface where all browser class implements webdriver interface, so webdriver supports multiple browser.

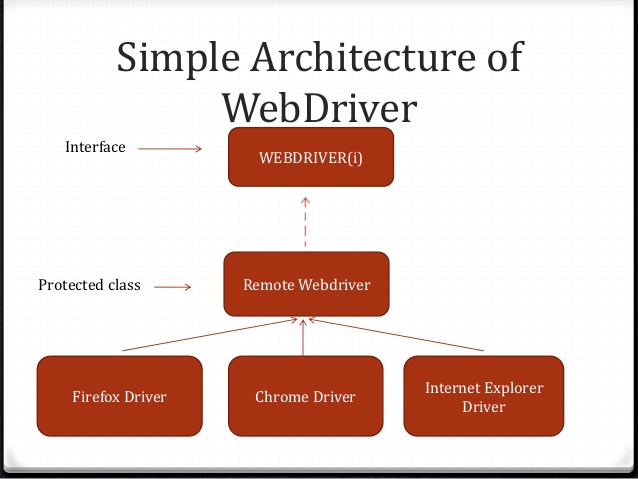
**Steps to download Selenium webdriver.**

* Navigate to google.
* Search for selenium download
* Click on first link.
* Search for selenium client and webdriver language bindings.
* Click on download for specific language.
* Once download unzip the file.

**Steps to import jars in your java project**

* Right click on project and search for build path.
* Navigate to configure build path.
* Click on it.
* Search for classpath and select it.
* Click on add external jars.
* Select the jars from unzipped file(inside lib and outside lib folder).
* Click on OK
* A referenced library should be added in project folder.
* Whenever we have to launch a new browser for every execution. We will create an object of the browser class as all browser class is the subclass of Webdriver interface.
* Whenever webdriver launches the browser all the brower add-ons automatically disabled.
* Webdriver always launches empty browser whenever we create object to browser classes because all browser class default constructor has code to launch empty browser.

**Webdriver driver=new FirefoxDriver();**



* Selenium Webdriver consist of jar.jar is a executable class file.
* Basically Webdriver is a interface ,all browser class implements webdriver interface, so webdriver supports multiple browser.
* It is a latest version of selenium RC and selenium IDE which removes all the disadvantage available in RC and IDE.
* Webdriver always launches a new browser for every execution. Whenever webdriver launches the browser all the browsr add -ons automatically disabled. Get() always take(http) url because webdriver uses http protocol for internal browser communication.
* Webdriver always launches empty browser whenever we create object to browser classes because all browser class default constructor has code to launch empty browser.

**WebDriver d=new FirefoxDriver();**

**FirefoxDriver d=new FirefoxDriver();**

**WebDriver basic method**

**get()**

**findelement()**

**findelements()**

**navigate.to()**

**back()**

**forward()**

**refresh()**

**gettitle()**

**getcurrenturl()**

**getpagesource()**

**getwindowhandle()**

**getwindowhandles()**

**switch to(){**

**frame()**

**window()**

**quit()**

**close()**

**manage()**

**window().maximise()**

**deletecookie()**

* Basically webdriver is a interface which contain browser specific action or method those methods are implemented by all browser classes.
* **Get()**---It is used navigate to any web based application without browser history.
* **Navigate.to()**---It is used navigate to web based application by holding browser history.
* **Findelement()**---It is used to find any one webelement in UI,which always return webelement reference if object is available.
* **Findelements()**---It is used to find multiple webelements in UI, which always return collection of webelement.
* **Getpagesource()**---It is used to get the current page html source code from the UI.
* **manage().window().maximise()**---Whenever the webdriver launches a browser by default browser will be in minimize mode, in order to maximize the browser explicity we go for **manage().window().maximize().**
* **deleteAllcookies()**---It is used to delete all cookies from the browser.
* **Getwindowhandle()**---It is used to handle multiple browser or tabs.
* **Quit()**—It is used to close multiple browser.
* **Close()**---It is used to close current active browser.

**Webelement Basic method**

* Any object available on the browser is called webelement.
* Using findelement() we can identify whether object available on the browser, once object is identified we take help of webelement basic method to perform specific action on the object. It is also an interface.

**WebElement**

**WebElement**

**Sendkeys()**

**Click()**

**Clear()**

**gettext()**

**isDisplayed()**

**isEnabled()**

**isSelected()**

**getTagName()**

**getLocation()**

**getcssvalue()**

**getAttribute()**

* **Sendkey()---**void sendKeys(java.lang.CharSequence... keysToSend)

Use this method to simulate typing into an element, which may set its value.

**Parameters:**

keysToSend - character sequence to send to the element

**Throws:**

java.lang.IllegalArgumentException - if keysToSend is null

* **Click()-**void click()

Click this element. If this causes a new page to load, you should discard all references to this element and any further operations performed on this element will throw a StaleElementReferenceException. Note that if click() is done by sending a native event (which is the default on most browsers/platforms) then the method will \_not\_ wait for the next page to load and the caller should verify that themselves. There are some preconditions for an element to be clicked. The element must be visible and it must have a height and width greater then 0.

**Throws:**

[StaleElementReferenceException](https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/StaleElementReferenceException.html) - If the element no longer exists as initially defined

* **Clear()-**void clear()

If this element is a text entry element, this will clear the value. Has no effect on other elements. Text entry elements are INPUT and TEXTAREA elements. Note that the events fired by this event may not be as you'd expect. In particular, we don't fire any keyboard or mouse events. If you want to ensure keyboard events are fired, consider using something like [sendKeys(CharSequence...)](https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/WebElement.html#sendKeys-java.lang.CharSequence...-) with the backspace key. To ensure you get a change event, consider following with a call to [sendKeys(CharSequence...)](https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/WebElement.html#sendKeys-java.lang.CharSequence...-) with the tab key.

* **getText()-**java.lang.String getText()

Get the visible (i.e. not hidden by CSS) text of this element, including sub-elements.

**Returns:**

The visible text of this element.

* **isDisplayed()-**--boolean isDisplayed()

Is this element displayed or not? This method avoids the problem of having to parse an element's "style" attribute.

**Returns:**

Whether or not the element is displayed

* **IsEnabled()-**--boolean isEnabled()

Is the element currently enabled or not? This will generally return true for everything but disabled input elements.

**Returns:**

True if the element is enabled, false otherwise.

* **Isselected()-**boolean isSelected()

Determine whether or not this element is selected or not. This operation only applies to input elements such as checkboxes, options in a select and radio buttons. For more information on which elements this method supports, refer to the [specification](https://w3c.github.io/webdriver/webdriver-spec.html#is-element-selected).

**Returns:**

True if the element is currently selected or checked, false otherwise.

* **getTagname()-**java.lang.String getTagName()

Get the tag name of this element. **Not** the value of the name attribute: will return "input" for the element <input name="foo" />.

**Returns:**

The tag name of this element.

* **getLocation()-**[Point](https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/Point.html) getLocation()

Where on the page is the top left-hand corner of the rendered element?

**Returns:**

A point, containing the location of the top left-hand corner of the element

* **getCssvalue()-**java.lang.String getCssValue(java.lang.String propertyName)

Get the value of a given CSS property. Color values should be returned as rgba strings, so, for example if the "background-color" property is set as "green" in the HTML source, the returned value will be "rgba(0, 255, 0, 1)". Note that shorthand CSS properties (e.g. background, font, border, border-top, margin, margin-top, padding, padding-top, list-style, outline, pause, cue) are not returned, in accordance with the [DOM CSS2 specification](http://www.w3.org/TR/DOM-Level-2-Style/css.html#CSS-CSSStyleDeclaration) - you should directly access the longhand properties (e.g. background-color) to access the desired values.

**Parameters:**

propertyName - the css property name of the element

**Returns:**

The current, computed value of the property.

* **getAttribute()-**java.lang.String getAttribute(java.lang.String name)

Get the value of the given attribute of the element. Will return the current value, even if this has been modified after the page has been loaded.

**WAIT STATEMENT**

**1.Normal Wait Command**

**Purpose**: This is rarely used, as it always force the browser to wait for a specific time. **Thread. Sleep** is never a good idea and that’s why Selenium provides wait for primitives. If you use them you can specify much higher timeout value which makes tests more reliable without slowing them down as the condition can be evaluated as often as it’s required.

Thread.sleep(second);

## 2.ImplicitlyWait Command

**Purpose**: Selenium WebDriver has borrowed the idea of **implicit waits** from **Watir**. This means that we can 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. We should note that implicit waits will be in place for the entire time the browser is open. This means that any search for elements on the page could take the time the implicit wait is set for.

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

## 3.ExpectedConditions Command

**Purpose**: Models a condition that might reasonably be expected to eventually evaluate to something that is neither null nor false.

WebDriverWait wait = **new** WebDriverWait(driver, 10);

WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id(>someid>)));

4.PageLoadTimeout Command

**Purpose**: 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.



|  |  |
| --- | --- |
| 1 | driver.manage().timeouts().pageLoadTimeout(100, SECONDS); |

5.FluentWait Command

**Purpose**: Each **FluentWait** instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. Furthermore, the user may configure the wait to ignore specific types of exceptions whilst waiting, such as **NoSuchElementExceptions** when searching for an element on the page.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | // Waiting 30 seconds for an element to be present on the page, checking      // for its presence once every 5 seconds.      Wait wait = **new** FluentWait(driver)        .withTimeout(30, SECONDS)        .pollingEvery(5, SECONDS)        .ignoring(NoSuchElementException.**class**);      WebElement foo = wait.until(**new** **Function**() {    **public** WebElement apply(WebDriver driver) {    **return** driver.findElement(By.id("foo"));        }       }); |

**Actions class**

Actions class is an ability provided by Selenium for handling keyboard and mouse events. In [Selenium WebDriver](https://www.browserstack.com/guide/selenium-webdriver-tutorial), handling these events includes operations such as drag and drop, clicking on multiple elements with the control key, among others. These operations are performed using the advanced user interactions API. It mainly consists of *Actions*that are needed while performing these operations.  
  
Action class is defined and invoked using the following syntax:

Actions action = new Actions(driver);

action.moveToElement(element).click().perform();

Action class is useful mainly for mouse and keyboard actions. In order to perform such actions, Selenium provides various methods.

**Mouse Actions in Selenium:**

1. **doubleClick()**: Performs double click on the element
2. **clickAndHold()**: Performs long click on the mouse without releasing it
3. **dragAndDrop()**: Drags the element from one point and drops to another
4. **moveToElement()**: Shifts the mouse pointer to the center of the element
5. **contextClick()**: Performs right-click on the mouse

**Keyboard Actions in Selenium:**

1. **sendKeys()**: Sends a series of keys to the element
2. **keyUp()**: Performs key release
3. **keyDown()**: Performs keypress without release

**Select class**

The Select class is a Webdriver class which basically provides the implementation of the HTML SELECT tag. A Select tag provides the helper methods with select and deselect options. This class can be found under Selenium’s Support.UI.Select package. Select is actually an ordinary class, so its object is also created by the keyword New and also specifies the location of the web element.

Syntax:

* *Select oselect = new Select();*

It will throw up an error asking to add arguments to the command. So specify the web element location using the element locators.

It clearly states that Select is asking for an element type object for its constructor.

After this, once you get the object of the SELECT Class, you can access all the methods that reside in the SELECT class by typing oSelect + dot which will provide all the methods under the Select class. Choose any method according to your test case.

**Different Select commands**

Following are the most commonly used methods to deal with the drop-down list.

**1. Select By VisibleText: select By VisibleText(String arg0): void**

It is very easy to choose or select an option given under any drop downs and multiple selection boxes with this method. It takes a parameter of String which is one of the values of Select element and it returns nothing.

Syntax:

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

**Example:**

* *Select oSelect =new Select(driver.findElement(By.id(“search-box”)));*
* *oSelect.selectByVisibleText(“Blog”);*

**2. selectByIndex: selectByIndex(int arg0) : void**

This method is almost similar to ‘selectByVisibleText’, but the only difference here is that the user has to provide the index number for the option rather than the option text. It takes the integer parameter which is the index value of Select element and it returns nothing.

Synatx:

* *oSelect.selectByIndex(int);*

**Example:**

* *Select oSelect = new Select(driver.findElement(By.id(“Search-box”)));*
* *oSelect.selectByIndex(2)*
* *selectByValue: selectByValue(String arg0) : void*

This method again is similar to what I have discussed earlier, the only difference in this method is that it asks for the value of the option rather than the option text or an index. It takes a String parameter which is one of the values of Select element and it does not return anything.

Syntax:

* *oSelect.selectByValue(“text”);*

**Example:**

* *Select oSelect = new Select(driver.findElement(By.id(“Search-box”)));*
* *oSelect.selectByValue(“Selenium Certification training”);*
* *getOptions: getOptions( ) : List<WebElement>*

This method helps to get all the options belonging to the Select tag. It takes no parameter and returns List<WebElements>.

Syntax:

* *oSelect.getOptions();*

**Example:**

* *Select oSelect = new Select(driver.findElement(By.id(“Search-box”)));*
* *List <WebElement> elementCount = oSelect.getOptions();*
* *System.out.println(elementCount.size());*

So, let’s move ahead to the next topic and learn about the Multiple Select methods

**How does Multiple SELECT command work?**

The multiple select attribute is a boolean expression. When this is present, it specifies that multiple options can be selected at once. These options vary for different operating systems and browsers namely:

* For Windows: Hold on the control (ctrl) button to select multiple options.
* For Mac: Hold down the command button to select multiple options.

It is user-friendly to use check-boxes instead of using different ways of performing operations because you have to inform the user that multiple selections are available. There is a method which actually helps to specify that you can use multiple select options.

isMultiple(): boolean – This method tells whether the SELECT element supports multiple selection options at the same time or not. This method accepts nothing but returns a boolean value(true/false).

Syntax:

* *oSelect.isMultiple();*

**Example:**

* *Select oSelect = new Select(driver.findElement(By.id(Element\_ID)));*
* *oSelect.selectByIndex(index)*
* *oSelect.selectByIndex(index)*
* *// Or can be used as*
* *oSelect.selectByVisibleText(text)*
* *oSelect.selectByVisibleText(text)*
* *// Or can be used as*
* *oSelect.selectByValue(value)*
* *oSelect.selectByValue(value)*

**DeSelect methods**

When you select a particular element on the webpage, there are a few methods which will help in deselecting that element. But the only challenge in these methods is they do not work for DropDown and only work for Multi-Select elements.

In case you want to deselect any pre-selected option, that can be done with either

* deselectAll()
* deselectByIndex
* deselectByValue
* Deselect By Visible text

**Let us understand the methods in detail.**

* deselectAll(): It clears all selected entries. This is only valid when the drop-down element supports multiple selections.

Example:

* *oSelect.deselectAll();*
* deselectByIndex(): It deselects the option at the given index.

Example:

* *oSelect.deselectByIndex(2);*
* Deselect By Value(): This method helps in deselecting the option whose “value” attribute matches the specific parameter.

Example:

* *oSelect.deselectByValue(“13”);*
* Deselect By Visible text():This method helps in deselecting the option that displays the text matching the parameter.

Select class in Selenium WebDriver: How to select an option from the drop-down menu?

I will help you guys understand how this Select method works with a real-time example.

In this case, I will consider working on a famous e-commerce website facebook.com.

* First, add the Java libraries onto your system.
* An IDE where you can write the piece of code. I will consider working on the Eclipse IDE as it is user-friendly.
* Add Selenium libraries onto the project.
* Get the URL of the web page.
* Perform desired actions on the drop-down list.

I have explained this using 2 different programs. The first program will help you to select a value from the drop-down list and the second program helps to perform different operations on the drop-down list.

* First, set the browser driver.
* Get the URL of Facebook.
* Create a WebElement object and find the element by using the element locators.
* Select the object of the WebElement using the Select methods.
* Quit the driver execution.

**Refer to this code:**

* *import org.junit.Test;*
* *import org.openqa.selenium.By;*
* *import org.openqa.selenium.JavascriptExecutor;*
* *import org.openqa.selenium.WebDriver;*
* *import org.openqa.selenium.WebElement;*
* *import org.openqa.selenium.chrome.ChromeDriver;*
* *import org.openqa.selenium.support.ui.Select;*
* *public class SelectClass {*
* *@Test*
* *public static void main(String[] args) throws InterruptedException {*
* *System.setProperty(“webdriver.chrome.driver”, “C:UsersVaishnaviDesktopchromedriver\_win32 (2)chromedriver.exe”);*
* *WebDriver driver = new ChromeDriver();*
* *driver.get(“http://www.facebook.com”);*
* *driver.manage().window().maximize();*
* *//js.executeScript(“window.scrollBy(0,300)”);*
* *WebElement month\_dropdown = driver.findElement(By.id(“day”));*
* *Select oSelect = new Select(month\_dropdown);*
* *oSelect.selectByIndex(3);*
* *Thread.sleep(3000);*
* *WebElement year\_yy = driver.findElement(By.id(“year”));*
* *Select year\_y = new Select(year\_yy);*
* *year\_y.selectByValue(“2000”);*
* *Thread.sleep(3000);*
* *WebElement month\_m = driver.findElement(By.id(“month”));*
* *Select month\_d1 = new Select(month\_m);*
* *month\_d1.selectByVisibleText(“Jul”);*
* *driver.quit();*
* *}*
* *}*

The second program deals with performing actions on the drop-down list. In this case, let us print the number of months and also the names.

* Create a list of WebElements and Select the options.
* Get the size of the month drop-down.
* Print the size of the month list.
* Create another object of the WebElement ele and get the name of the month.
* Print the number using a for loop.
* Quit the driver execution.

*import java.util.List;*

* *import org.junit.Test;*
* *import org.openqa.selenium.By;*
* *import org.openqa.selenium.JavascriptExecutor;*
* *import org.openqa.selenium.WebDriver;*
* *import org.openqa.selenium.WebElement;*
* *import org.openqa.selenium.chrome.ChromeDriver;*
* *import org.openqa.selenium.support.ui.Select;*
* *public class SelectClass2 {*
* *@Test*
* *public static void main(String[] args) throws InterruptedException {*
* *System.setProperty(“webdriver.chrome.driver”, “C:UsersVaishnaviDesktopchromedriver\_win32 (2)chromedriver.exe”);*
* *WebDriver driver = new ChromeDriver();*
* *JavascriptExecutor js = (JavascriptExecutor)driver;*
* *driver.get(“http://www.facebook.com”);*
* *driver.manage().window().maximize();*
* *//js.executeScript(“window.scrollBy(0,300)”);*
* *WebElement month\_dropdown = driver.findElement(By.id(“month”));*
* *Select oSelect = new Select(month\_dropdown);*
* *List&amp;amp;lt;WebElement&amp;amp;gt; month\_list = oSelect.getOptions();*
* *int total\_month = month\_list.size();*
* *System.out.println(“Total count is “+total\_month);*
* *for(WebElement ele:month\_list)*
* *{*
* *String month\_name = ele.getText();*
* *System.out.println(“Months are”+month\_name);*
* *}*
* *driver.quit();*
* *}*
* *}*

**Window Handelling**

*It is a unique identifier that holds the address of all the windows.* Think of it as a pointer to a window, which returns the string value. It is assumed that each browser will have a unique window handle. This window handle function helps to retrieve the handles of all windows.

**Syntax**

1. **get.windowhandle()**: This method helps to get the window handle of the current window
2. **get.windowhandles()**: This method helps to get the handles of all the windows opened
3. **set**: This method helps to set the window handles in the form of a string. *set<string> set= driver.get.windowhandles()*
4. **switch to:** This method helps to switch between the windows
5. **action**: This method helps to perform certain actions on the windows

These are some of the methods that will be used to handle multiple windows in Selenium.

**Multiple Windows**

public class WindowHandle\_Demo {

public static void main(String[] args) throws Exception {

System.setProperty("webdriver.chrome.driver","Path to the driver");

WebDriver driver = new ChromeDriver();

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

// Load the website

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

// It will return the parent window name as a String

String parent=driver.getWindowHandle();

Set<String>s=driver.getWindowHandles();

// Now iterate using Iterator

Iterator<String> I1= s.iterator();

while(I1.hasNext())

{

String child\_window=I1.next();

if(!parent.equals(child\_window))

{

driver.switchTo().window(child\_window);

System.out.println(driver.switchTo().window(child\_window).getTitle());

driver.close();

}

}

//switch to the parent window

driver.switchTo().window(parent);

}

}

**Scenario**: Navigate to the URL. This is the parent window. From the parent window, let’s see how to handle the child windows and then again navigate back to the parent windows.

Steps to execute:

1. Get the handle of the parent window using the command: String parentWindowHandle = driver.getWindowHandle();
2. Print the window handle of the parent window.
3. Find the element on the web page using an ID which is an element locator.
4. Open multiple child windows.
5. Iterate through child windows.
6. Get the handles of all the windows that are currently open using the command: Set<String> allWindowHandles = driver.getWindowHandles(); which returns the set of handles.
7. Use the **SwitchTo** command to switch to the desired window and also pass the URL of the web page.
8. Perform operation on an element in child window.
9. Close the child using driver.close()
10. Move the driver control on parent window using **SwitchTo** command on parent and perform operation on element in parent window.

**Frame Handling**

**iFrame in Selenium Webdriver** is a web page or an inline frame which is embedded in another web page or an HTML document embedded inside another HTML document. The iframe is often used to add content from other sources like an advertisement into a web page. The iframe is defined with the <**iframe**> tag.

We can identify the frames in Selenium using methods given below:

* Right click on the element, If you find the option like ‘This Frame’ then it is an iframe.(Please refer the above diagram)
* Right click on the page and click ‘View Page Source’ and Search with the ‘iframe’, if you can find any tag name with the ‘iframe’ then it is meaning to say the page consisting an iframe.

We can even identify total number of iframes by using below snippet.

Int size = driver.findElements(By.tagName("iframe")).size();

**How to switch over the elements in iframes using Web Driver commands:**

Basically, we can switch over the elements and handle frames in Selenium using 3 ways.

* **By Index**
* **By Name or Id**
* **By Web Element**

**Switch to the frame by index:**

Index is one of the attributes for frame handling in Selenium through which we can switch to it.

Index of the iframe starts with ‘0’.

Suppose if there are 100 frames in page, we can switch to frame in Selenium by using index.

* driver.switchTo().frame(0);
* driver.switchTo().frame(1);

**Switch to the frame by Name or ID:**

Name and ID are attributes for handling frames in Selenium through which we can switch to the iframe.

* driver.switchTo().frame(“iframe1”);
* driver.switchTo().frame(“id of the element”);

**Switch to the frame by Web Element:**

We can even switch to the iframe using web element .

* driver.switchTo().frame(WebElement);

**How to switch back to the Main Frame**

We have to come out of the iframe.

To move back to the parent frame, you can either use switchTo().parentFrame() or if you want to get back to the main (or most parent) frame, you can use switchTo().defaultContent();

driver.switchTo().parentFrame();

driver.switchTo().defaultContent();

**public** **class** Framehandelling {

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

System.*setProperty*("webdriver.chrome.driver", "H:\\chromedriver.exe");

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

driver.manage().timeouts().implicitlyWait(20,TimeUnit.***SECONDS***);

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

driver.get("https://www.selenium.dev/selenium/docs/api/dotnet/index.html");

Thread.*sleep*(3000);

driver.switchTo().frame("TopicContent");

driver.findElement(By.*linkText*("Cookie")).click();

driver.switchTo().defaultContent();

driver.findElement(By.*partialLinkText*("IAllowsFileDetection")).click();

}

}

**Alert POPUP  
Alert Popup** are always implemented using java scripts, so it is also known as java scripts popup.

Webdriver can’t perform action on alert box automatically, in order to work with alert we should pass driver control to alert box before performing action on alert.

Driver.switchto().alert

Using action API method we can perform action on alert box.

Once the operation performed with alert box driver control automatically comes back to main window.

**public** **class** Alertpopup {

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

System.*setProperty*("webdriver.chrome.driver", "H:\\chromedriver.exe");

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

driver.manage().timeouts().implicitlyWait(20,TimeUnit.***SECONDS***);

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

driver.get("https://www.hyrtutorials.com/p/alertsdemo.html");

driver.findElement(By.*id*("alertBox")).click();

Thread.*sleep*(5000);

//Pass control to alert and capture the text present in Alert.

Alert alt= driver.switchTo().alert();

String alerttext=alt.getText();

System.***out***.println(alerttext);

alt.accept();//click on OK button on alert

//alt.dismiss();//click on Cancel button on alert

driver.findElement(By.*id*("promptBox")).click();

Alert alt1=driver.switchTo().alert();

Thread.*sleep*(4000);

alt1.sendKeys("deepeshrajprof@gmail.com");

alt1.dismiss();

}

}

**Calender POPUP**

**It** is always implemented using div html tag but it will be invisible initially, in order to work with div popup perform action to enable the popup & directly write a xpath and perform action on the popup

**Steps to work with Calender Popup**

* 1. Perform action to enable calender icon
  2. Using while loop repeatedly check for expected date , whenever expected date is enable in UI try block through no such element exception.
  3. In order to handle exception & continue the execution we will go for catch block.
  4. While block will terminate the execution whenever expected date available in UI or condition exceeds the limit.

**UnExpected PopUP**

Driver.get(“https://Flipkart.com”);

Try{

Driver.findelement(By.xpath(“closebutton”)).click();

}catch(Exception e)

{

System.out.println(POPUP is not displaying“”);

}

}

* **Working With Multiple Weblements**
* Findelements() method is used to identify webelements in UI which always return list of Webelements.
* Findelements() returns empty list if no links or webelements available in UI.
* In order to capture or perform some specific action in dynamic webtable better practice to go for findelements().

**TestNG Framework Tool**

TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use, such as:

* Annotations.
* Run your tests in arbitrarily big thread pools with various policies available (all methods in their own thread, one thread per test class, etc...).
* Test that your code is multithread safe.
* Flexible test configuration.
* Support for data-driven testing (with @DataProvider).
* Support for parameters.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **1.**[@BeforeSuite](https://www.tutorialspoint.com/testng/testng_annotations_beforesuite.htm)  The annotated method will be run only once before all tests in this suite have run. |  |  |  |  |
| 2 |  |  |  |  | [@AfterSuite](https://www.tutorialspoint.com/testng/testng_annotations_aftersuite.htm)  The annotated method will be run only once after all tests in this suite have run. |
| 3 |  |  |  |  | [@BeforeClass](https://www.tutorialspoint.com/testng/testng_annotations_beforeclass.htm)  The annotated method will be run only once before the first test method in the current class is invoked. |
| 4 |  |  |  |  | [@AfterClass](https://www.tutorialspoint.com/testng/testng_annotations_afterclass.htm)  The annotated method will be run only once after all the test methods in the current class have run. |
| 5 |  |  |  |  | [@BeforeTest](https://www.tutorialspoint.com/testng/testng_annotations_beforetest.htm)  The annotated method will be run before any test method belonging to the classes inside the <test> tag is run. |
| 6 |  |  |  |  | [@AfterTest](https://www.tutorialspoint.com/testng/testng_annotations_aftertest.htm)  The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run. |
| 7 |  |  |  |  | [@BeforeGroups](https://www.tutorialspoint.com/testng/testng_annotations_beforegroups.htm)  The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked. |
| 8 |  |  |  |  | [@AfterGroups](https://www.tutorialspoint.com/testng/testng_annotations_aftergroups.htm)  The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked. |
| 9 |  |  |  |  | [@BeforeMethod](https://www.tutorialspoint.com/testng/testng_annotations_beforemethod.htm)  The annotated method will be run before each test method. |
| 10 |  |  |  |  | [@AfterMethod](https://www.tutorialspoint.com/testng/testng_annotations_aftermethod.htm)  The annotated method will be run after each test method. |
| 11 |  |  |  |  | [@DataProvider](https://www.tutorialspoint.com/testng/testng_annotations_dataprovider.htm)  Marks a method as supplying data for a test method. The annotated method must return an Object[ ][ ], where each Object[ ] can be assigned the parameter list of the test method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation. |
| 12 |  |  |  |  | [@Factory](https://www.tutorialspoint.com/testng/testng_annotations_factory.htm)  Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[ ]. |
| 13 |  |  |  |  | [@Listeners](https://www.tutorialspoint.com/testng/testng_annotations_listeners.htm)  Defines listeners on a test class. |
| 14 |  |  |  |  | [@Parameters](https://www.tutorialspoint.com/testng/testng_annotations_parameters.htm)  Describes how to pass parameters to a @Test method. |
| 15 |  |  |  |  | [@Test](https://www.tutorialspoint.com/testng/testng_annotations_test.htm)  Marks a class or a method as a part of the test. |

**@BeforeClass**: The @BeforeClass annotated method runs before the execution of test methods in a current class.

**@AfterClass:** The @AfterClass annotated method will be executed after all the test methods of a current class have been invoked.

The **@BeforeMethod** is specific to a class not to an XML file. The @BeforeMethod annotated method will be invoked before the execution of each test method where the test method is nothing but a test case. Suppose there are four test methods in a class then the @BeforeMethod annotated method is executed before the execution of each test method. If there are four test methods, then four times @BeforeMethod annotated method will be invoked.The **@AfterMethod** annotation is specific to a class not to an XML file. The @AfterMethod annotated method will be invoked after the execution of each test method. Suppose there are four test methods means that @AfterMethod annotated method will be executed four times.

**@BeforeGroups:** The @BeforeGroups annotated method will run only once before all the test methods belonging to a specified group have been executed

@BeforeGroups annotation which executes the @BeforeGroups annotated method before the execution of the test methods belonging to the group specified in the parameter of the @BeforeGroups annotation.

**@AfterGroups:** The @AfterGroups annotated method will run only once after the execution of all the test methods of a specified group.

**@BeforeSuite:** The **@BeforeSuite** annotated method is executed before the execution of all the test cases defined in the folder.

Generally, @BeforeSuite is used when we have different URLs to run your test cases. Environment variables are set in a @BeforeSuite annotated method so that before executing all the test cases, you need to load all the environment variables for your framework, and then it starts executing your test cases.

The @BeforeSuite annotated method is given as the first priority, so it is executed before all the other test methods.

**@AfterSuite:** The @AfterSuite annotated method is executed after the execution of all the test methods in the Suite. The Suite is basically a testng.xml file so we can say that @AfterSuite annotated method is executed after the execution of an XML file.

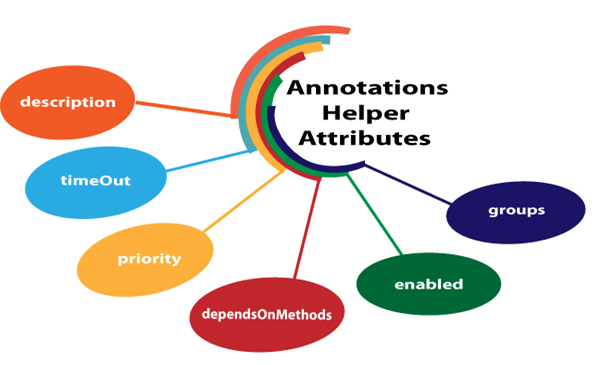
The @BeforeSuite annotation is used to set up or start the selenium drivers while the @AfterSuite annotation is used to stop the selenium web drivers.

**Some of the common attributes are described below:**

46.7M

869

Difference between JDK, JRE, and JVM



* **description**
* **timeOut**
* **priority**
* **dependsOnMethods**
* **enabled**
* **groups**

description

It is a string which is attached to the @Test annotation that describes the information about the test.

**Let's understand through an example.**

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Class1
4. {
6. @Test(description="This is testcase1")
7. **public** **void** testcase1()
8. {
9. System.out.println("HR");
10. }
11. @Test(description="This is testcase2")
12. **public** **void** testcase2()
13. {
14. System.out.println("Software Developer");
15. }
16. @Test(description="This is testcase3")
17. **public** **void** testcase3()
18. {
19. System.out.println("QA Analyst");
20. }
21. }

In the above code, we have added the description attribute in every test. The "**description**" attribute provides information about the test.

dependsOnMethods

When the second test method wants to be dependent on the first test method, then this could be possible by the use of "**dependOnMethods**" attribute. If the first test method fails, then the dependent method on the first test method, i.e., the second test method will not run.

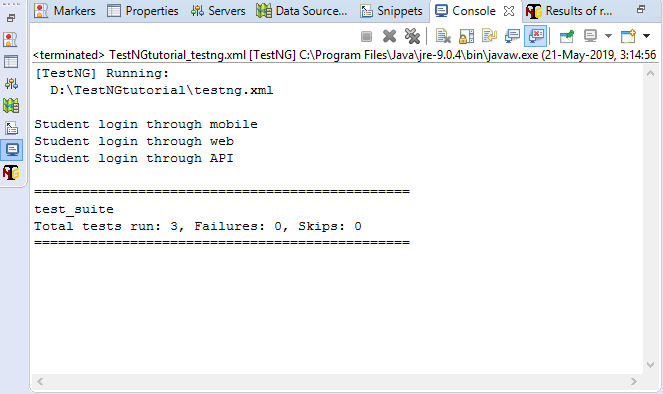
**Let's understand through an example.**

**First case:** When a single value is passed in a parameter.

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Class1
4. {
5. @Test
6. **public** **void** WebStudentLogin()
7. {
8. System.out.println("Student login through web");
9. }
10. @Test
11. **public** **void** MobileStudentLogin()
12. {
13. System.out.println("Student login through mobile");
14. }
15. @Test(dependsOnMethods= {"WebStudentLogin"})
16. **public** **void** APIStudentLogin()
17. {
18. System.out.println("Student login through API");
19. }
20. }

We know that the TestNG executes the test methods in alphabetical order so, in the above program, APIStudentLogin() will execute first. However, we want WebStudentLogin() method to be executed before the execution of the APIStudentLogin() method, so this would only be possible through the "dependsOnMethods" attribute. In the above program, we have specified "dependsOnMethods" attribute in an APIStudentLogin() test method and its value is "WebStudentLogin" which means that WebStudentLogin() method will be executed before the APIStudentLogin() method.

**Output**



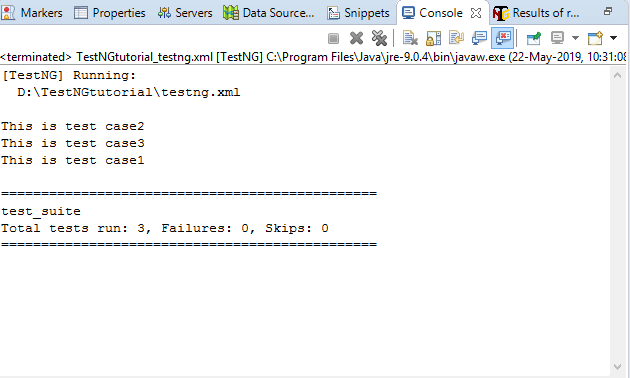
In the above output, MobileStudentLogin() runs before the WebStudentLogin() method as TestNG runs the test methods in an alphabetical order.

**Second case:** When multiple values are passed in a parameter.

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Depends\_On\_Groups
4. {
5. @Test(dependsOnMethods= {"testcase3","testcase2"})
6. **public** **void** testcase1()
7. {
8. System.out.println("This is test case1");
9. }
10. @Test
11. **public** **void** testcase2()
12. {
13. System.out.println("This is test case2");
14. }
15. @Test
16. **public** **void** testcase3()
17. {
18. System.out.println("This is test case3");
19. }
21. }

In the above code, testcase1() is dependent on two methods, i.e., testcase2() and testcase3(), which means that these two methods will be executed before the testcase1().

**Output**



priority

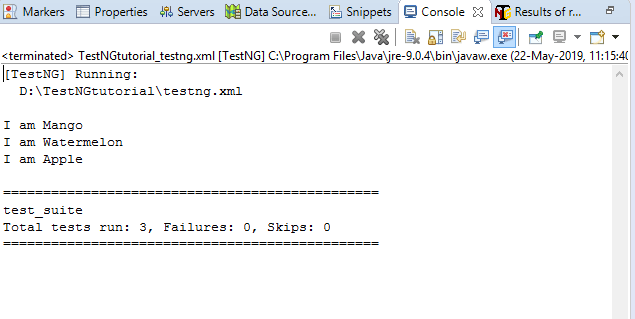
When no 'priority' attribute is specified then the TestNG will run the test cases in alphabetical order. Priority determines the sequence of the execution of the test cases. The priority can hold the integer values between -5000 and 5000. When the priority is set, the lowest priority test case will run first and the highest priority test case will be executed last. Suppose we have three test cases and their priority values are -5000, 0, 15, then the order of the execution will be 0,15,5000. If priority is not specified, then the default priority will be 0.

**Let's understand through an example.**

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Fruits
4. {
5. @Test
6. **public** **void** mango()
7. {
8. System.out.println("I am Mango");
9. }
10. @Test(priority=2)
11. **public** **void** apple()
12. {
13. System.out.println("I am Apple");
14. }
15. @Test(priority=1)
16. **public** **void** watermelon()
17. {
18. System.out.println("I am Watermelon");
19. }
20. }

In the above code, the default priority of mango() test method is 0, so it will be executed first. The watermelon() test method will run after mango() method as the priority of watermelon() test method is 2. The apple() test method has the highest priority, so it will be executed last.

**Output**



enabled

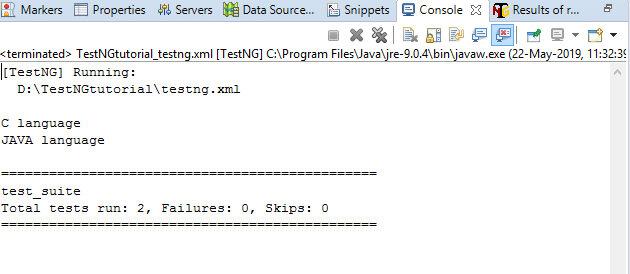
The 'enabled' attribute contains the boolean value. By default, its value is true. If you want to skip some test method, then you need to explicitly specify '**false**' value.

**Let's understand through an example.**

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Programming\_languages
4. {
5. @Test
6. **public** **void** c\_language()
7. {
8. System.out.println("C language");
9. }
10. @Test(enabled=**false**)
11. **public** **void** jira()
12. {
13. System.out.println("JIRA is a testing tool");
14. }
15. @Test
16. **public** **void** java()
17. {
18. System.out.println("JAVA language");
19. }
20. }

In the above code, the value of the enabled attribute in jira() test method is false, so this method will not be invoked.

**Output**



groups

The 'groups' attribute is used to group the different test cases that belong to the same functionality.

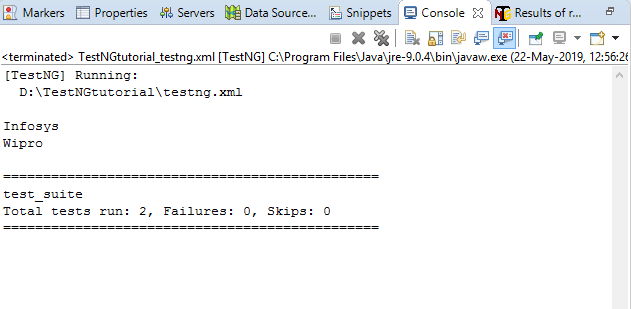
**Let's understand through an example.**

1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Software\_Company
4. {
5. @Test(groups= {"software company"})
6. **public** **void** infosys()
7. {
8. System.out.println("Infosys");
9. }
10. @Test
11. **public** **void** technip()
12. {
13. System.out.println("Technip India Ltd");
14. }
15. @Test(groups= {"software company"})
16. **public** **void** wipro()
17. {
18. System.out.println("Wipro");
19. }
20. }

**testng.xml**

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
3. <suite name="test\_suite">
4. <test name="Software Company">
5. <groups>
6. <run>
7. <include name="software company"/>
8. </run>
9. </groups>
10. <classes>
11. <**class** name="com.javatpoint.Software\_Company"/>
12. </classes>
13. </test> <!-- Test -->
14. </suite> <!-- Suite -->

**Output**



timeOut

If one of the test cases is taking a long time due to which other test cases are failing. To overcome such situation, you need to mark the test case as fail to avoid the failure of other test cases. The timeOut is a time period provided to the test case to completely execute its test case.

**Let's understand through an example.**

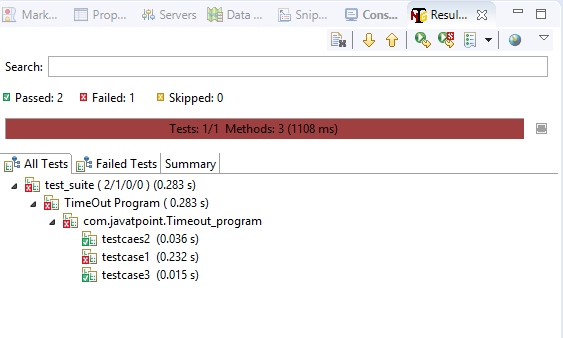
1. **package** com.javatpoint;
2. **import** org.testng.annotations.Test;
3. **public** **class** Timeout\_program
4. {
5. @Test(timeOut=200)
6. **public** **void** testcase1() **throws** InterruptedException
7. {
8. Thread.sleep(500);
9. System.out.println("This is testcase1");
10. }
11. @Test
12. **public** **void** testcaes2()
13. {
14. System.out.println("This is testcase2");
15. }
16. @Test
17. **public** **void** testcase3()
18. {
19. System.out.println("This is testcase3");
20. }
21. }

In the above code, inside the testcase1() method, we have Thread.sleep(500) which means that the testcase1() method will be executed after 500 milliseconds, but we have provided timeOUT attribute with the value 200 means that the testcase1() will be failed after 200 milliseconds.

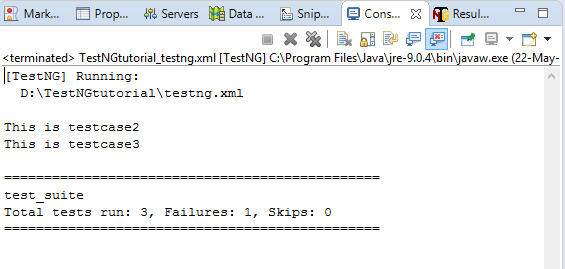
**testng.xml**

1. **<?xml version="1.0" encoding="UTF-8"?>**
2. **<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">**
3. **<suite name="test\_suite">**
4. **<test name="TimeOut Program">**
5. **<classes>**
6. **<class name="com.javatpoint.Timeout\_program"/>**
7. **</classes>**
8. **</test> <!-- Test -->**
9. **</suite> <!-- Suite -->**

**Output**

****

**The above screen shows that one test case is failed and other test cases are passed.**

****

**Batch Execution**

**Executing Multiple testcases in a single click is known as Batch Execution.**

In TestNG, you can define multiple test cases in a single class whereas, in Java, you can define only one test in a single class in the main() method. In Java, if you want to create one more test, then you need to create another java file and define the test in the main() method.

Instead of creating test cases in different classes, we recommend you to use TestNG framework that allows you to create multiple test cases in a single class.

You can create multiple test cases with the help of **@Test** annotation.

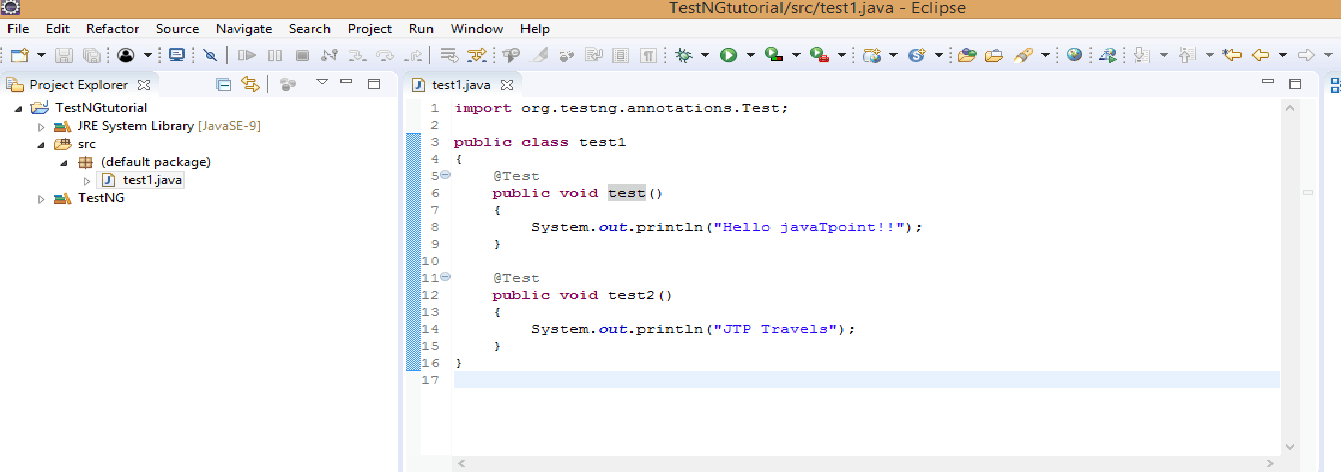
**Let's understand through an example.**

Nested Structure in CKeep Watching

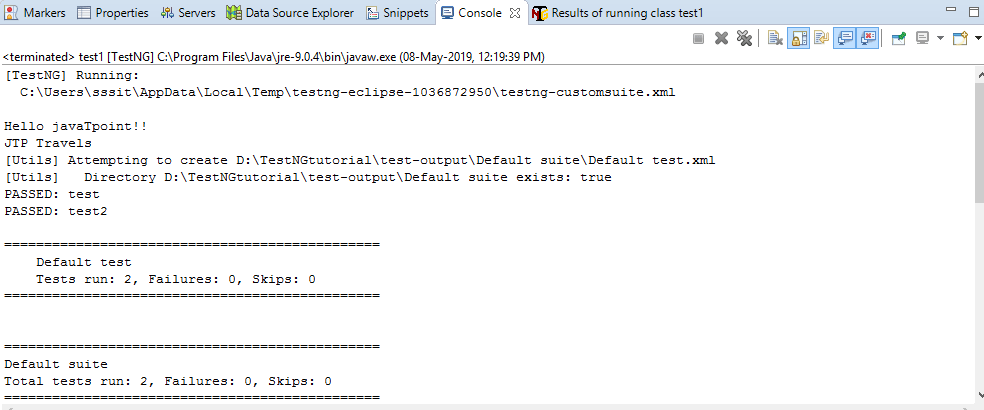
1. **public** **class** test
2. {
3. @Test
4. **public** **void** test1()                      // First test case.
5. {
6. System.out.println("test1");
7. }
8. @Test
9. **public**  **void** test2()                     // Second test case.
10. {
11. System.out.println("test2");
12. }}

The above code consists of a class test. The class test consists of two test cases, i.e., test1() and test2(). You can differentiate the test cases by considering the sequence of test cases. In the above code, the test case **test2()** is written in the second @Test annotation, so this test case will be considered as the second case.

**Source code**



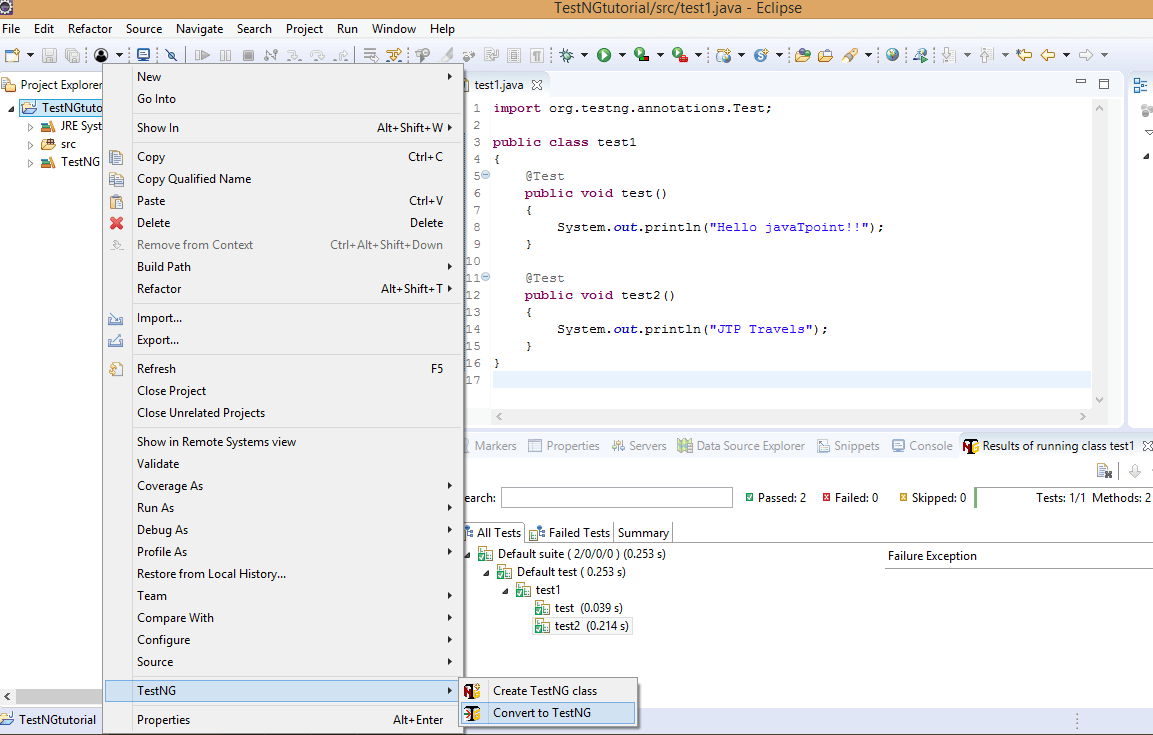
**Output**



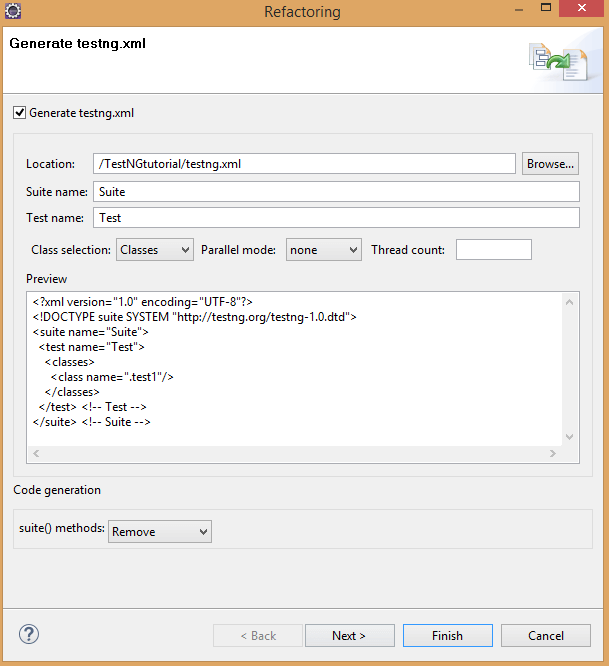
#### **Note: You can trigger all the test cases from a single file known as xml file. Xml file is the heart of TestNG framework.**

## How to create a xml file

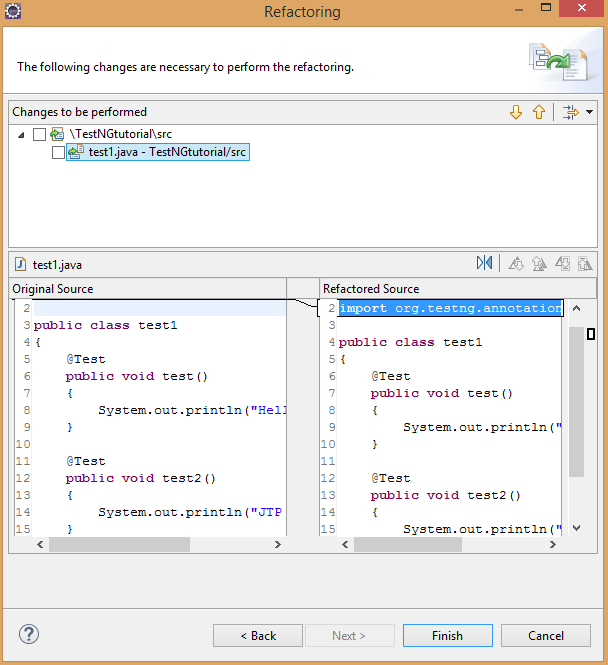
* Right click on the project. Move your cursor down, and you will see TestNG and then click on the **Convert to TestNG**.



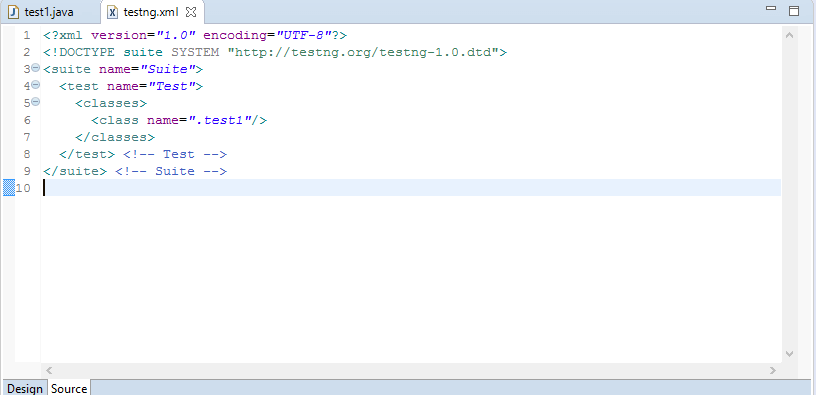
* The below screen shows the preview of the xml file. Click on the **Next** button.



* Click on the **Finish** button.



* The testing.xml file is shown below:



In the above source code of xml file, suite is at the higher hierarchy in TestNG. Inside the , you have to define the test name folder. This test name folder is the name of the folder. For example, In a loan company, there are three different types of modules such as personal loan, home loan and car loan, and each module contain its own test cases. All these test cases are defined in the test name folder.

**Parallel Execution in TestNG:**

* If we want to run methods / classes in different threads, then we need to set ‘parallel’ attributes on the tags ‘[methods](https://stqatools.com/java-methods/)‘ / ‘[classes](https://stqatools.com/class-in-java-and-object-in-java/)‘.
* By using parallel execution, we can reduce the ‘execution time’ because the tests run in different threads simultaneously.
* With parallel execution we can run the same test on different device models, to get more coverage, run different tests on the same device model, cut performance time and get strong results for specific models.

**Advantages of parallel execution in Selenium using TestNG:**

* It saves the execution effort.
* We can cover many tests.
* We also can do cross-browser testing, which will make the app more stable.

**Features of parallel execution in Selenium using TestNG:**

* TestNG handles internally threading concepts which will allow us to run tests in many threads.
* Each thread will be handed over to the individual test , if you have less thread than the test, then only one thread will be available for free.
* You have to separate parallel execution or machine with good resources which can control many browsers at one time.

### **Using parallel attribute Tests / Classes / Methods parallel:**

1. **Tests in parallel then use:**  parallel=”tests” All the test cases inside <test> tag of testng.xml file will run parallel.
2. **Classes in parallel then use:**  parallel=”classes” All the test cases inside a java class will run parallel.
3. **Methods in parallel then use:**  parallel=”methods” All the methods with @Test annotation will execute parallel.

### **Example of Parallel Execution in TestNG:**

* First methods opens **Chrome driver** and navigate to [https://www.stqatools.com](https://www.stqatools.com/) and closes the browser.
* Second methods opens **Firefox driver** and navigate to [https://www.stqatools.com](https://www.stqatools.com/) and close the browser.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | import org.openqa.selenium.WebDriver;  import org.openqa.selenium.chrome.ChromeDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  import org.testng.annotations.Test;    public class Parallel\_TestNG {    @Test  public void getChorme() {    System.setProperty("webdriver.chrome.driver", "D:\\chromedriver.exe");  System.out.println("Chrome Thread : " + Thread.currentThread().getId());  WebDriver driver = new ChromeDriver();  driver.get("https://www.stqatools.com");  driver.close();       }    @Test  public void getFirefox() {    System.setProperty("webdriver.gecko.driver", "D:\\geckodriver.exe");  System.out.println("Firefox Thread : " + Thread.currentThread().getId());  WebDriver driver = new FirefoxDriver();  driver.get("https://www.stqatools.com");  driver.close();       }  } |

### **without mentioning parallel attribute in TestNG.xml:**

**Note:** First in action Chrome browser then Firefox browser one by one.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | &lt;?xml version="1.0" encoding="UTF-8"?&gt;  &lt;!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd"&gt;    &lt;suite name="stqatools"&gt;           &lt;test name="testngTest"&gt;                &lt;classes&gt;                       &lt;class name="package\_name.class\_name" /&gt;                &lt;/classes&gt;         &lt;/test&gt;  &lt;/suite&gt; |

**Output:**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Chrome on Thread : 1    Firefox on Thread : 1    PASSED: getChorme    PASSED: getFirefox |

### **Using parallel attribute To run both the browsers in parallel in TestNG.xml:**

**Note:**  Using  parallel=”methods” both the browsers / methods in action at a time.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | &lt;?xml version="1.0" encoding="UTF-8"?&gt;  &lt;!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd"&gt;    &lt;suite name="stqatools" parallel="methods" thread-count="2"&gt;              &lt;test name="testngTest"&gt;                     &lt;classes&gt;                          &lt;class name="package\_name.class\_name" /&gt;                     &lt;/classes&gt;              &lt;/test&gt;  &lt;/suite&gt; |

**Output:**



|  |  |
| --- | --- |
| 1  2  3 | Firefox on Thread : 12    Chrome on Thread : 11<strong> </strong>  **Grouping Execution** |

TestNG also allows us to group test inside groups, which we discussed in the later section of this tutorial. Grouping saves us from defining many classes in our test source code and then running these classes separately, resulting in avoiding the wastage of our time. This situation also helps us in preventing the recompilation of test cases again and again, according to our needs.

***It is important to note that Groups are declared in the testng.xml file in the TestNG and can be found inside <test> tag or <suite> tag***.

***Also, remember that the groups defined in <test> tag apply to only that particular test tag, but the groups defined in <suite> tag apply to all the <test> tags in the XML file***.

For example, in the first case below, we have defined the group inside the "***suite***" tag.

<suite name="test\_suite">

<groups>

<run>

<include name="UI Test"/>

</run>

</groups>

<test name="Check Login Page">

<classes>

<class name="com.demoqa.Check\_Login\_Page"/>

</classes>

</test>

<test name="Response Status">

<classes>

<class name="com.demoqa.Response\_Status"/>

</classes>

</test>

</suite>

Since the "***groups***" is inside the suite tag, it will run all the tests in the XML file. For the second case, see the following tweak in the above code:

<suite name="test\_suite">

<test name="Check Login Page">

<groups>

<run>

<include name="UI Test"/>

</run>

</groups>

<classes>

<class name="com.demoqa.Check\_Login\_Page"/>

</classes>

</test>

<test name="Response Status">

<classes>

<class name="com.demoqa.Response\_Status"/>

</classes>

</test>

</suite>

Now the groups will work only inside the test tag with the name Check Login Page.

### ***How To Create Groups?***

Before getting more complicated in groups, let's create a simple group test that we discussed in the above section. In the below code, we will check:

* Whether we are getting the title of the webpage correctly or not.
* Find an element "***Sortable***" on the website and click the element.

We will be using our demo website demoqa.com for this task. Refer to our tutorial of [***Selenium Webdriver***](https://www.toolsqa.com/selenium-webdriver/selenium-tutorial/) to know in-depth about the code.

Write the following code inside your file named TestNG.java (You can choose any TestNG test case file you want).

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.annotations.Test;

import org.testng.Assert;

public class TestNG {

WebDriver driver;

// Saving the expected title of the Webpage

String title = "ToolsQA - Demo Website For Automation";

@Test

public void starting\_point(){

System.out.println("This is the starting point of the test");

//Initialize Chrome Driver

//driver.manage().timeouts().implicitlyWait(1,TimeUnit.SECONDS);

driver = new ChromeDriver();

driver.get("https://demoqa.com/");

}

@Test(groups = { "demo" })

public void checkTitle() {

String testTitle = "Free QA Automation Tools For Everyone";

String originalTitle = driver.getTitle();

Assert.assertEquals(originalTitle, testTitle);

}

@Test(groups = { "demo" })

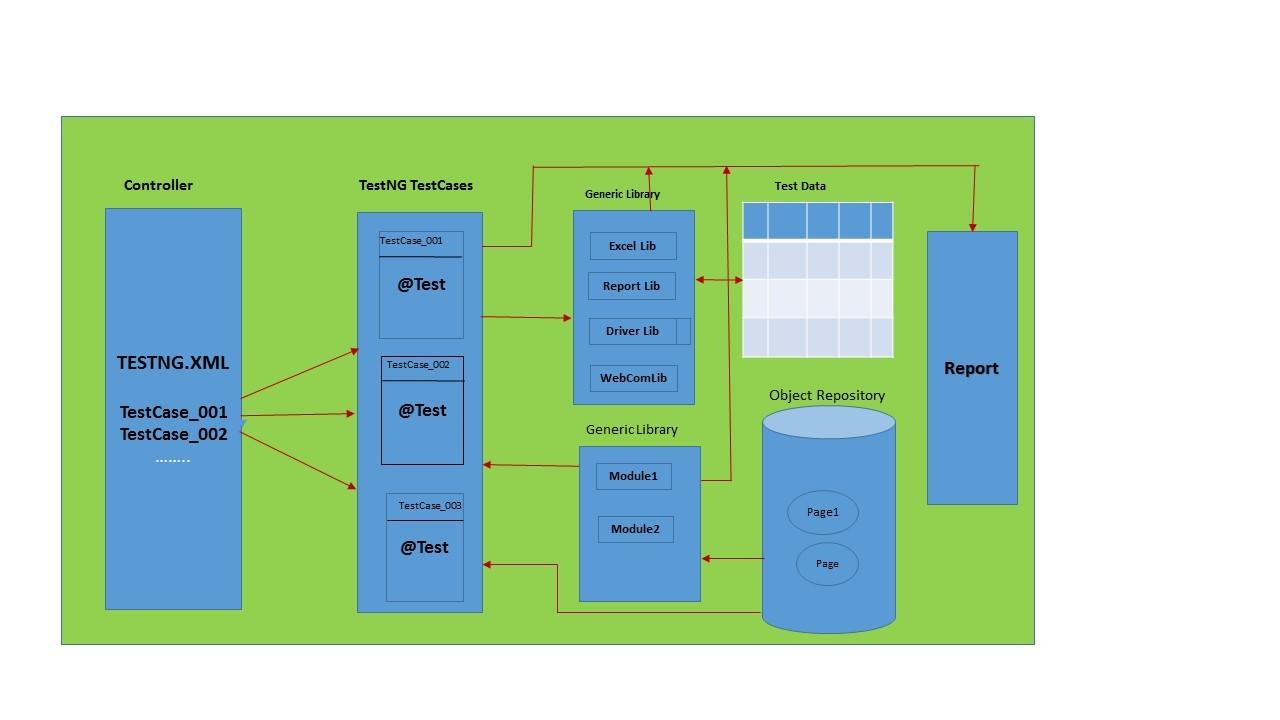
public void click\_element() {

driver.findElement(By.xpath("//\*[@id=\"sidebar\"]/aside[1]/ul/li[1]/a")).click();

System.out.println("Home Page heading is displayed");

}

**Framework**

****

**Framework** is a well organized structure of packages one driver scripts will take care of entire batch execution without any manual interruption.

Or

Framework is a custom tool which is developed by framework developer which contains lots of reusable methods that makes automation T.E life easier.

**Framework components**

1. Driver(TestNg.xml)

2. TestNG Test Scripts

3. Generic Library

4. Page Object Repository

5. Test Data

6. Reports

7. Screenshots

8. Resources