Components in Selenium :

1.Selenium WebDriver - Selenium WebDriver is used to automate web applications using browser's native methods.
2.Selenium IDE - A firefox plugin that works on record and play back principle.
3.Selenium RC - Selenium Remote Control(RC) is officially deprecated by selenium and it used to work on javascript to automate the web applications.
4.Selenium Grid - Allows selenium tests to run in parallel across multiple machines.

Advantages of Selenium
1. Selenium is pure open source, freeware and portable tool.
2. Selenium supports variety of languages that include Java, Perl, Python, C#, Ruby, Groovy, Java Script, and VB Script. etc.
3. Selenium supports many operating systems like Windows, Macintosh, Linux, Unix etc.
4. Selenium supports many browsers like Internet explorer, Chrome, Firefox, Opera, Safari etc.
5. Selenium can be integrated with ANT or Maven kind of framework for source code compilation.
6. Selenium can be integrated with TestNG testing framework for testing our applications and generating reports.
7. Selenium can be integrated with Jenkins or Hudson for continuous integration.
8. Selenium can be integrated with other open source tools for supporting other features.
9. Selenium can be used for Android, IPhone, Blackberry etc. based application testing.
10. Selenium supports very less CPU and RAM consumption for script execution.
11. Selenium comes with different component to provide support to its parent which is Selenium IDE, Selenium Grid and Selenium Remote Control (RC).

Disadvantages:

2. Selenium only supports web based application and does not support windows based application.
3. It is difficult to test Image based application.
4. Selenium need outside support for report generation activity like dependence on TestNG or Jenkins.
5. Selenium does not support built in add-ins support.
6. Selenium user lacks online support for the problems they face.
7. Selenium does not provide any built in IDE for script generation and it need other IDE like Eclipse for writing scripts.
8. Selenium script creation time is bit high.
10. Selenium does not support file upload facility.
11. Selenium partially supports for Dialog boxes.

Selenium Vs QTP :

Open source, free to use, and free of charge where QTP is commercial.
Selenium is Highly extensible where QTP having Limited add-ons
Can run tests across different browsers where QTP can only run tests in Firefox, Internet Explorer and Chrome
Supports various operating systems where QTP can only be used in Windows,
Supports mobile devices where QTP Supports Mobile app test automation (iOS & Android) using HP solution called - HP Mobile Center
Can execute tests while the browser is minimized where in QTP, Needs to have the application under test to be visible on the desktop
Can execute tests in parallel.. In QTP, can only execute in parallel but using Quality Center which is again a paid product.

Implicit Wait Vs Expilict Wait ::

An implicit wait is a type of wait which waits for a specified time while locating an element before throwing NoSuchElementException.
As by default selenium tries to find elements immediately when required without any wait. So, it is good to use implicit wait.
This wait is applied to all the elements of the current driver instance.
Syntax :: driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
An explicit wait is a type of wait which is applied to a particular web element untill the expected condition specified is met.

```
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id("elementId")));
```

How to run Chrome, IE, Safari, Opera and Mozilla Firefox on selenium web driver using java?.

1) Import the classes whatever is being used.
2) set the property of the drivers using the System.getProperty(“driver”,”PathWhereTheExeFile IsStored”);
3) Instantiate the browser. ex: WebDriver driver=new FirefoxDriver();
4) Open the URL. Using the object of the driver and function. driver.get(URL)

1 Different importing statements:

import org.openqa.selenium.WebDriver;	**WebdriverInterface**	
import org.openqa.selenium.safari.SafariDriver;	**Safari**	
import org.openqa.selenium.firefox.FirefoxDriver;	**Firefox**	
import org.openqa.selenium.chrome.ChromeDriver;	**Chrome**	
import org.openqa.selenium.ie.InternetExplorerDriver;	**IE**	

2) **SETPROPERTY and browser launching**

Sl.no	**Browser**	**example**	**Browser instantiation**
1	Firefox	System.getProperty(“webdriver.gecko.driver”,”c://geckodriver.exe”);	WebDriver driver = new FirefoxDriver();
2	Chrome	System.getProperty(“webdriver.chrome.driver”,”c://chromedriver.exe”);	WebDriver driver = new ChromeDriver();
3	IE	System.getProperty(“webdriver.ie.driver”, ”c://IEDriverServer.exe”);	`WebDriver driver = new InternetExplorerDriver();`
4	EDGE		
5	Safari		
6	Opera	`System.setProperty("webdriver.ie.driver", "operadriver.exe");`	`WebDriver  driver = new OperaDriver();`

Different Drivers

HTMLUnit and Firefox are two browsers that WebDriver can directly automate - meaning that no other separate component is needed to install or run while the test is being executed. For other browsers, a separate program is needed. That program is called as the **Driver Server**.

A driver server is different for each browser. For example, Internet Explorer has its own driver server which you cannot use on other browsers. Below is the list of driver servers and the corresponding browsers that use them.

You can download these drivers on seleniumhq.org

Browser	Name of Driver Server	Remarks
HTMLUnit	HtmlUnitDriver	WebDriver can drive HTMLUnit using HtmlUnitDriver as driver server
Firefox	Mozilla GeckoDriver	WebDriver can drive Firefox without the need of a driver server Starting Firefox 35 & above one needs to use gecko driver created by Mozilla for automation
Internet Explorer	Internet Explorer Driver Server	Available in 32 and 64-bit versions. Use the version that corresponds to the architecture of your IE
Chrome	ChromeDriver	Though its name is just "ChromeDriver", it is, in fact, a Driver Server, not just a driver. The current version can support versions higher than Chrome v.21
Opera	OperaDriver	Though its name is just "OperaDriver", it is, in fact, a Driver Server, not just a driver.
PhantomJS	GhostDriver	PhantomJS is another headless browser just like HTMLUnit.
Safari	SafariDriver	Though its name is just "SafariDriver", it is, in fact, a Driver Server, not just a driver.

To Maximize the Window of the browser:
driver.manage().window().maximize();

To Enter the URL in the Browser

`voidget(Stringurl);`	driver.get("URL");

To close the Browser:

`voidclose();`	driver.close();

To Get the title of the page(on top)
driver.getTitle(); → Returns string

To Open new Tab.
driver.findElement(By.tagName("body")).sendKey(Keys.CONTROL+ "t");

<u>Simple Program to open browser,navigating to the page and verifying the page Title</u>

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class PG1 {


    public static void main(String[] args) {
        // declaration and instantiation of objects/variables
       WebDriver driver ;
       System.setProperty("webdriver.firefox.marionette","C:\\geckodriver.e
xe");
       driver = new FirefoxDriver();
        String baseUrl = "http://newtours.demoaut.com";
        String expectedTitle = "Welcome: Mercury Tours";
        String actualTitle = "";

        // launch Fire fox and direct it to the Base URL
        driver.get(baseUrl);
        // get the actual value of the title
        actualTitle = driver.getTitle();

        /*
         * compare the actual title of the page with the expected one and
print
         * the result as "Passed" or "Failed"
         */
        if (actualTitle.contentEquals(expectedTitle)){
            System.out.println("Test Passed!");
        } else {
            System.out.println("Test Failed");
        }

        //close Fire fox
        driver.close();

        // exit the program explicitly
        System.exit(0);
    }

}
```

Get Commands

Get commands fetch various important information about the page/element. Here are some important "get" commands you must be familiar with.

get() *Sample usage:*	• It automatically opens a new browser window and fetches the page that you specify inside its parentheses. • It is the counterpart of Selenium IDE's "open" command. • The parameter must be a **String** object.
getTitle() *Sample usage:*	• Needs no parameters • Fetches the title of the current page • Leading and trailing white spaces are trimmed • Returns a null string if the page has no title
getPageSource()*Sample usage:*	• Needs no parameters • Returns the **source code of the page** as a String value
getCurrentUrl()*Sample usage:*	• Needs no parameters • Fetches the string representing the **current URL** that the browser is looking at
getText() *Sample usage:*	• Fetches the **inner text** of the element that you specify • Example <p att1=”11” att2=”22”>Hello</p> • driver.findElement(By.tagName(“p”)).getText();
getAttribute(String strObj)	• Input is attribute name and returns the value of a attribute • driver.getAttribute(“att1”); returns String as 11

	driver.findElement(By.xpath("//a[text()='TechBeamers']"))
	
	.getCssValue("color");
	
	driver.findElement(By.xpath("//a[text()='TechBeamers']"))
	
	.getCssValue("background-color");

Navigate commands

These commands allow you to refresh,go-into and switch back and forth between different web pages.

navigate().to()	• It automatically **opens a new browser window and fetches the page** that you specify inside its parentheses. • **It does exactly the same thing as the get() method.** • **Navigate().to("www.google.com")**
navigate().refresh()*Sample usage:*	• Needs no parameters. • It **refreshes** the current page. • Navigate().refresh();
navigate().back()*Sample usage:*	• Needs no parameters • Takes you **back by one page** on the browser's history.
navigate().forward()*Sample usage:*	• Needs no parameters • Takes you **forward by one page** on the browser's history.

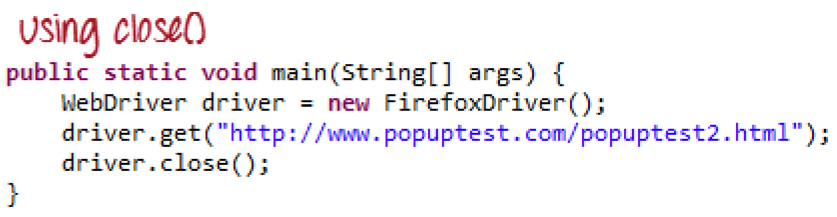
Closing and Quitting Browser Windows

close() *Sample usage:*	• Needs no parameters • **It closes only the browser window that WebDriver is currently controlling**.
quit() *Sample usage:*	• Needs no parameters • **It closes all windows that WebDriver has opened.**

close()
- will only close a single window

quit()
- will close all windows

To clearly illustrate the difference between close() and quit(), try to execute the code below. It uses a webpage that automatically pops up a window upon page load and opens up another after exiting.


using close()

```
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    driver.get("http://www.popuptest.com/popuptest2.html");
    driver.close();
}
```

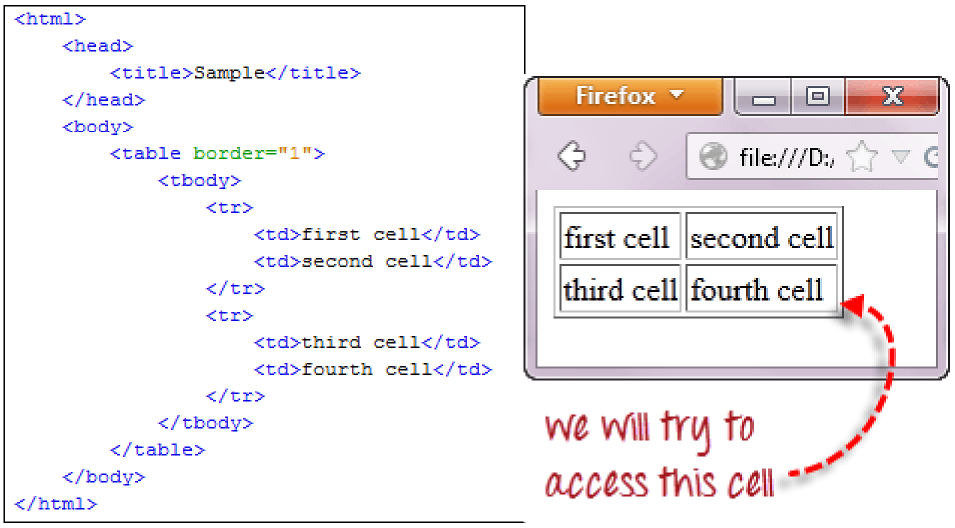
Notice that only the parent browser window was closed and not the two pop-up windows.

By Locators

Variation	Description	Sample
By.**className**	finds elements based on the value of the "class" attribute	findElement(By.className("someClassName"))
By.**cssSelector**	finds elements based on the driver's underlying CSS Selector engine	findElement(By.cssSelector("input#email"))
By.**id**	locates elements by the value of their "id" attribute	findElement(By.id("someId"))
By.**linkText**	finds a link element by the exact text it displays	findElement(By.linkText("REGISTRATION"))
By.**name**	locates elements by the value of the "name" attribute	findElement(By.name("someName"))
By.**partialLinkText**	locates elements that contain the given link text	findElement(By.partialLinkText("REG"))
By.**tagName**	locates elements by their tag name	findElement(By.tagName("div"))
By.**xpath**	locates elements via XPath	findElement(By.xpath("//html/body/div/table/tbody/tr/td[2]/table/tbody/tr[4]/td"))

XPath Syntax

Consider the HTML code below.



```
<html>
    <head>
        <title>Sample</title>
    </head>
    <body>
        <table border="1">
            <tbody>
                <tr>
                    <td>first cell</td>
                    <td>second cell</td>
                </tr>
                <tr>
                    <td>third cell</td>
                    <td>fourth cell</td>
                </tr>
            </tbody>
        </table>
    </body>
</html>
```

We will use XPath to get the inner text of the cell containing the text "fourth cell."

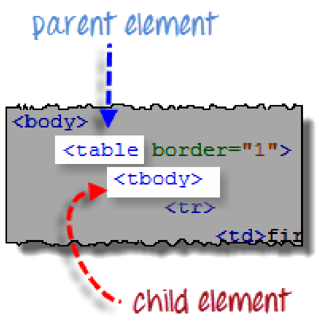
Step 1 - Set the Parent Element (table)

XPath locators in WebDriver always start with a double forward slash "//" and then followed by the parent element. Since we are dealing with tables, the parent element should always be the <table> tag. The first portion of our XPath locator should, therefore, start with "//table".

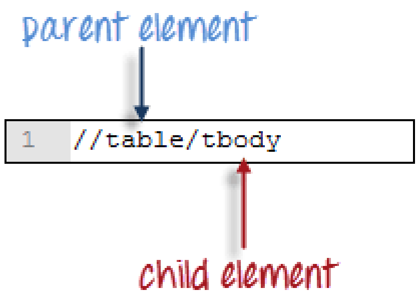
```
//table
```

Step 2 - Add the child elements

The element immediately under <table> is <tbody> so we can say that <tbody> is the "child" of <table>. And also, <table> is the "parent" of <tbody>. All child elements in XPath are placed to the right of their parent element, separated with one forward slash "/" like the code shown below.



```
<body>
  <table border="1">
    <tbody>
      <tr>
        <td>fir
```



```
//table/tbody
```

Step 3 - Add Predicates

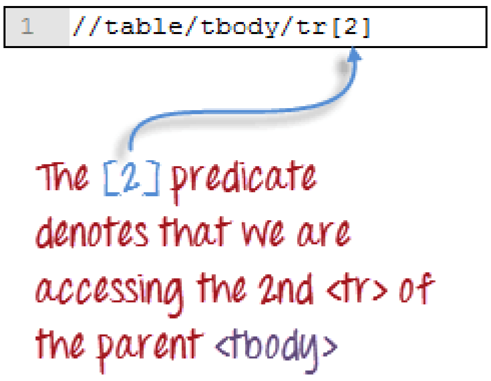
The <tbody> element contains two <tr> tags. We can now say that these two <tr> tags are "children" of <tbody>. Consequently, we can say that <tbody> is the parent of both the <tr> elements.

Another thing we can conclude is that the two <tr> elements are siblings. **Siblings refer to child elements having the same parent**.

To get to the <td> we wish to access (the one with the text "fourth cell"), we must first access the **second** <tr> and not the first. If we simply write "//table/tbody/tr", then we will be accessing the first <tr> tag.

So, how do we access the second <tr> then? The answer to this is to use **Predicates**.

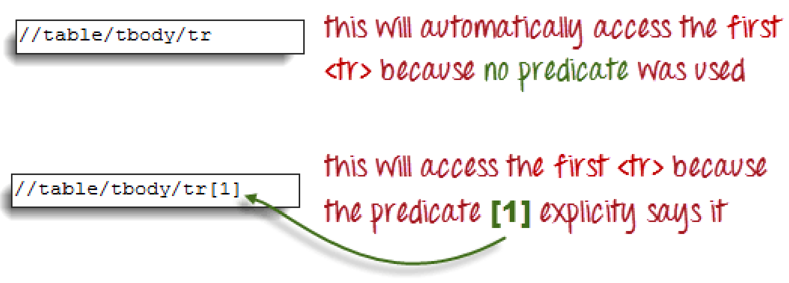
Predicates are numbers or HTML attributes enclosed in a pair of square brackets "[ ]" that distinguish a child element from its siblings. Since the <tr> we need to access is the second one, we shall use "[2]" as the predicate.



```
//table/tbody/tr[2]
```

The [2] predicate denotes that we are accessing the 2nd <tr> of the parent <tbody>

If we won't use any predicate, XPath will access the first sibling. Therefore, we can access the first <tr> using either of these XPath codes.



```
//table/tbody/tr
```

this will automatically access the first <tr> because no predicate was used

```
//table/tbody/tr[1]
```

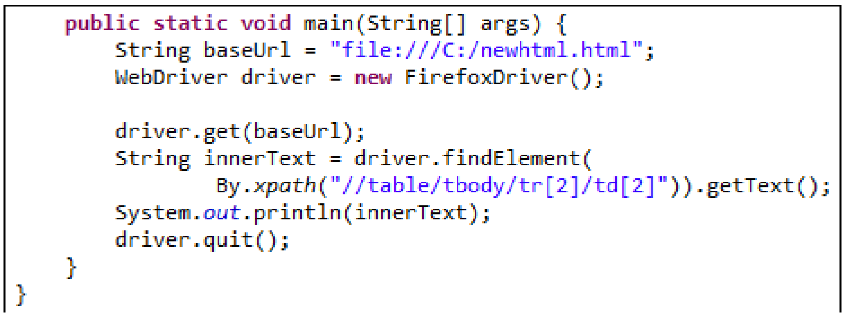
this will access the first <tr> because the predicate **[1]** explicity says it

Step 4 - Add the Succeeding Child Elements Using the Appropriate Predicates

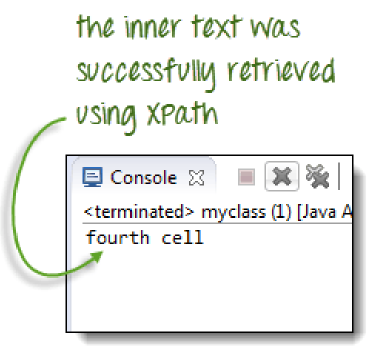
The next element we need to access is the second <td>. Applying the principles we have learned from steps 2 and 3, we will finalize our XPath code to be like the one shown below.

```
//table/tbody/tr[2]/td[2]
```

Now that we have the correct XPath locator, we can already access the cell that we wanted to and obtain its inner text using the code below. It assumes that you have saved the HTML code above as "newhtml.html" within your C Drive.



```
    public static void main(String[] args) {
        String baseUrl = "file:///C:/newhtml.html";
        WebDriver driver = new FirefoxDriver();

        driver.get(baseUrl);
        String innerText = driver.findElement(
                By.xpath("//table/tbody/tr[2]/td[2]")).getText();
        System.out.println(innerText);
        driver.quit();
    }
}
```



the inner text was successfully retrieved using xpath

```
<terminated> myclass (1) [Java A
fourth cell
```

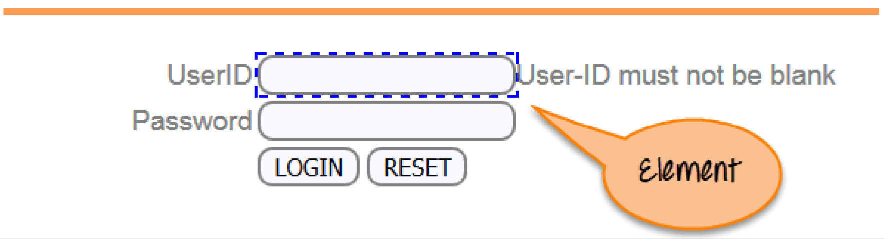
Using XPath Handling complex & Dynamic elements in Selenium

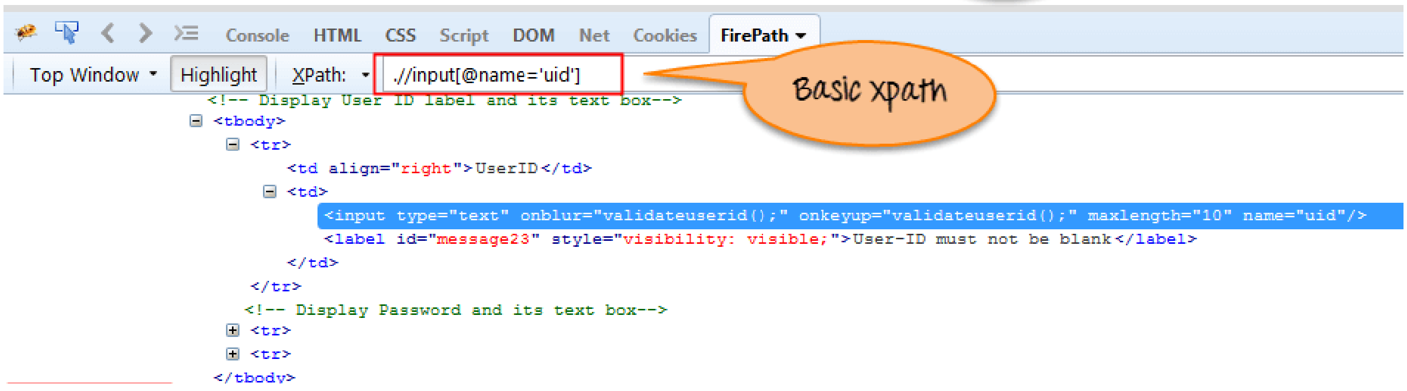
1) Basic XPath:

XPath expression select nodes or list of nodes on the basis of attributes like **ID , Name, Classname**, etc. from the XML document as illustrated below.

```
Xpath=//input[@name='uid']
```

Here is a link to access the page http://demo.guru99.com/v1/





```
<!-- Display User ID label and its text box-->
<tbody>
  <tr>
    <td align="right">UserID</td>
    <td>
      <input type="text" onblur="validateuserid();" onkeyup="validateuserid();" maxlength="10" name="uid"/>
      <label id="message23" style="visibility: visible;">User-ID must not be blank</label>
    </td>
  </tr>
  <!-- Display Password and its text box-->
  <tr>
  <tr>
</tbody>
```

Some more basic xpath expressions:

```
Xpath=//input[@type='text']
Xpath= //label[@id='message23']
Xpath= //input[@value='RESET']
Xpath=//*[@class='barone']
Xpath=//a[@href='http://demo.guru99.com/']
Xpath= //img[@src='//cdn.guru99.com/images/home/java.png']
```

2) Contains() : Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information.

The contain feature has an ability to find the element with partial text as shown in below example.

In this example, we tried to identify the element by just using partial text value of the attribute. In the below XPath expression partial value 'sub' is used in place of submit button. It can be observed that the element is found successfully.

Complete value of 'Type' is 'submit' but using only partial value 'sub'.

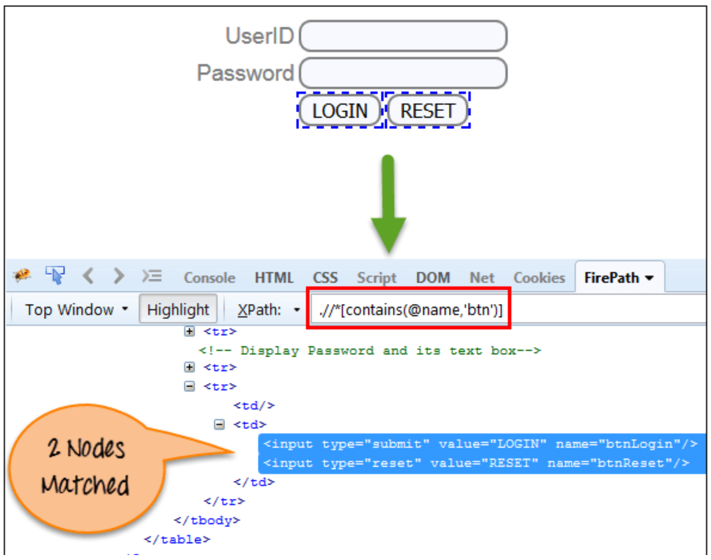
<input type='submit' />

```
//*[contains(@type,'sub')]
```

Complete value of 'name' is 'btnLogin' but using only partial value 'btn'.

```
.//*[contains(@name,'btn')]
```

In the above expression, we have taken the 'name' as an attribute and 'btn' as an partial value as shown in the below screenshot. This will find 2 elements (LOGIN & RESET) as their 'name' attribute begins with 'btn'.

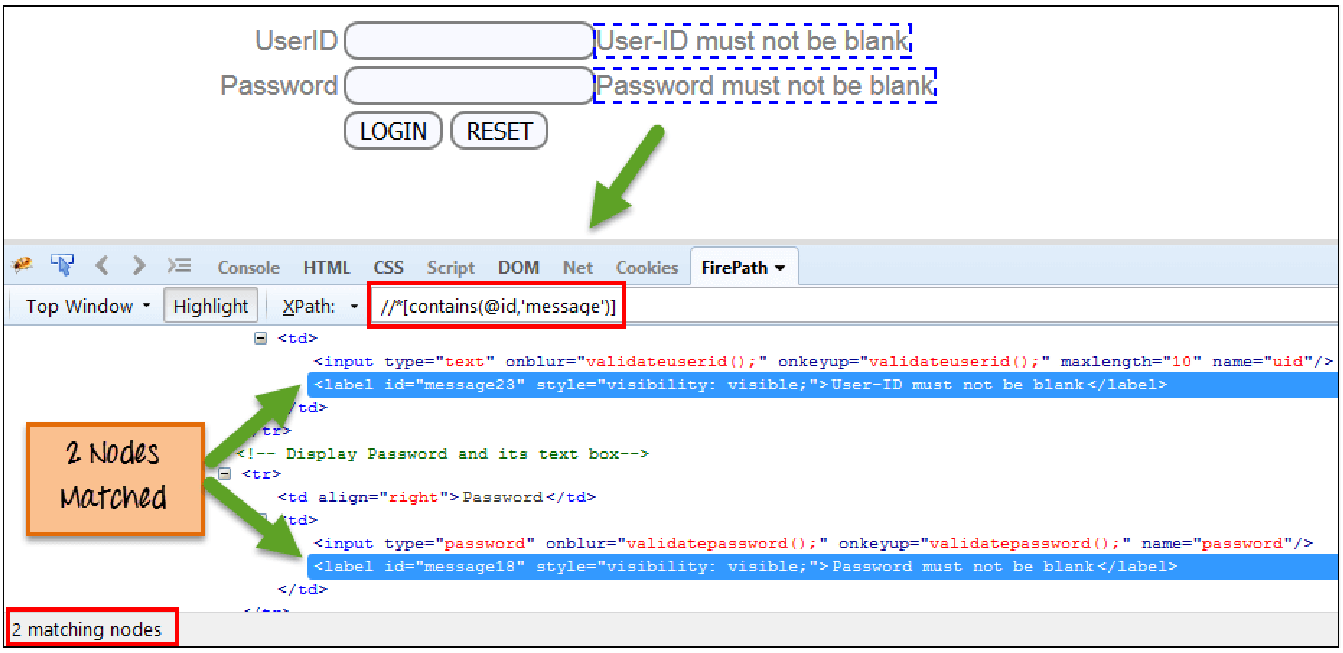


.//*[contains(@name,'btn')]

```
<tr>
  <!-- Display Password and its text box-->
<tr>
<tr>
    <td/>
  <td>
      <input type="submit" value="LOGIN" name="btnLogin"/>
      <input type="reset" value="RESET" name="btnReset"/>
    </td>
  </tr>
 </tbody>
</table>
```

Similarly, in the below expression, we have taken the 'id' as an attribute and 'message' as a partial value. This will find 2 elements ('User-ID must not be blank' & 'Password must not be blank') as its 'name' attribute begins with 'message'.

```
Xpath=//*[contains(@id,'message')]
```


UserID
Password
User-ID must not be blank
Password must not be blank
LOGIN
RESET
Console HTML CSS Script DOM Net Cookies FirePath
Top Window
Highlight
XPath:
2 Nodes Matched

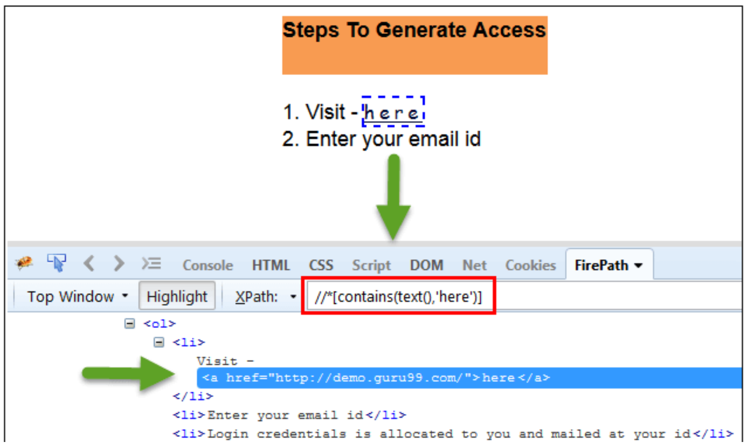
//*[contains(@id,'message')]

```
<td>
   <input type="text" onblur="validateuserid();" onkeyup="validateuserid();" maxlength="10" name="uid"/>
   <label id="message23" style="visibility: visible;">User-ID must not be blank</label>
 </td>
</tr>
<!-- Display Password and its text box-->
<tr>
  <td align="right">Password</td>
  <td>
   <input type="password" onblur="validatepassword();" onkeyup="validatepassword();" name="password"/>
   <label id="message18" style="visibility: visible;">Password must not be blank</label>
  </td>
```

2 matching nodes

In the below expression, we have taken the "text" of the link as an attribute and 'here' as a partial value as shown in the below screenshot. This will find the link ('here') as it displays the text 'here'.

```
Xpath=//*[contains(text(),'here')]
Xpath=//*[contains(@href,'guru99.com')]
```



```
//*[contains(text(),'here')]
```

```
<ol>
  <li>
    Visit -
    <a href="http://demo.guru99.com/">here</a>
  </li>
  <li>Enter your email id</li>
  <li>Login credentials is allocated to you and mailed at your id</li>
```

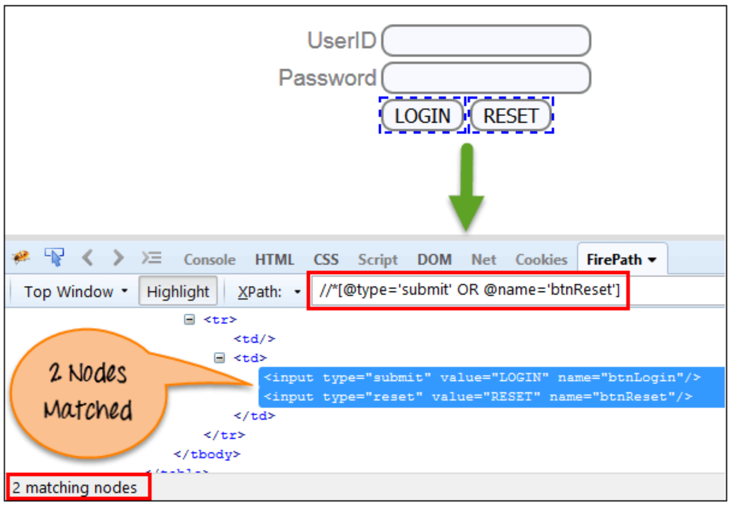
3) Using OR & AND:

In OR expression, two conditions are used, whether 1st condition OR 2nd condition should be true. It is also applicable if any one condition is true or maybe both. Means any one condition should be true to find the element.

In the below XPath expression, it identifies the elements whose single or both conditions are true.

```
//*[@type='submit' OR @name='btnReset']
//*[contains(@type, 'sub') OR contains(@name, btn)]
```

Highlighting both elements as "LOGIN " element having attribute 'type' and "RESET" element having attribute 'name'.



```
//*[@type='submit' OR @name='btnReset']
```

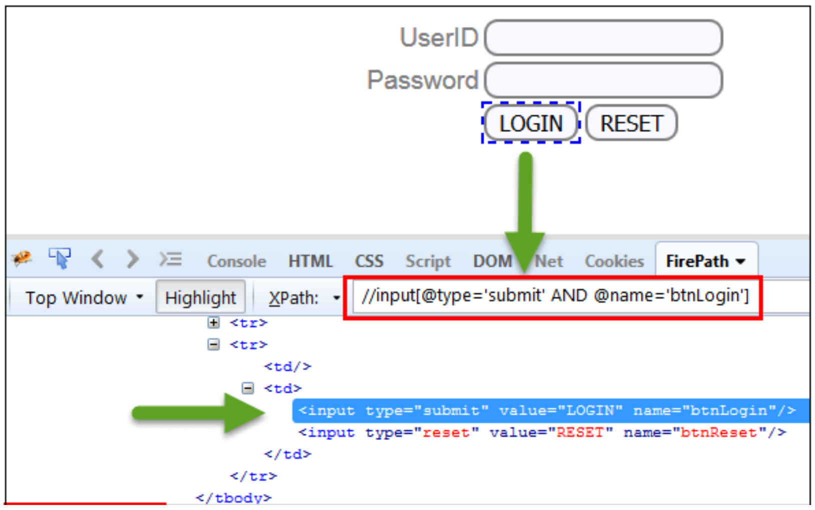
```
<tr>
  <td/>
  <td>
    <input type="submit" value="LOGIN" name="btnLogin"/>
    <input type="reset" value="RESET" name="btnReset"/>
  </td>
</tr>
</tbody>
```

2 matching nodes

In AND expression, two conditions are used, both conditions should be true to find the element. It fails to find element if any one condition is false.

```
//input[@type='submit' AND @name='btnLogin']
```

In below expression, highlighting 'LOGIN' element as it having both attribute 'type' and 'name'.



```
//input[@type='submit' AND @name='btnLogin']
```

```
<tr>
<tr>
  <td/>
  <td>
    <input type="submit" value="LOGIN" name="btnLogin"/>
    <input type="reset" value="RESET" name="btnReset"/>
  </td>
</tr>
</tbody>
```

4) Start-with function: Start-with function finds the element whose attribute value changes on refresh or any operation on the webpage. In this expression, match the starting text of the attribute is used to find the element whose attribute changes dynamically. You can also find the element whose attribute value is static (not changes).

For example -: Suppose the ID of particular element changes dynamically like:

Id=" message12"

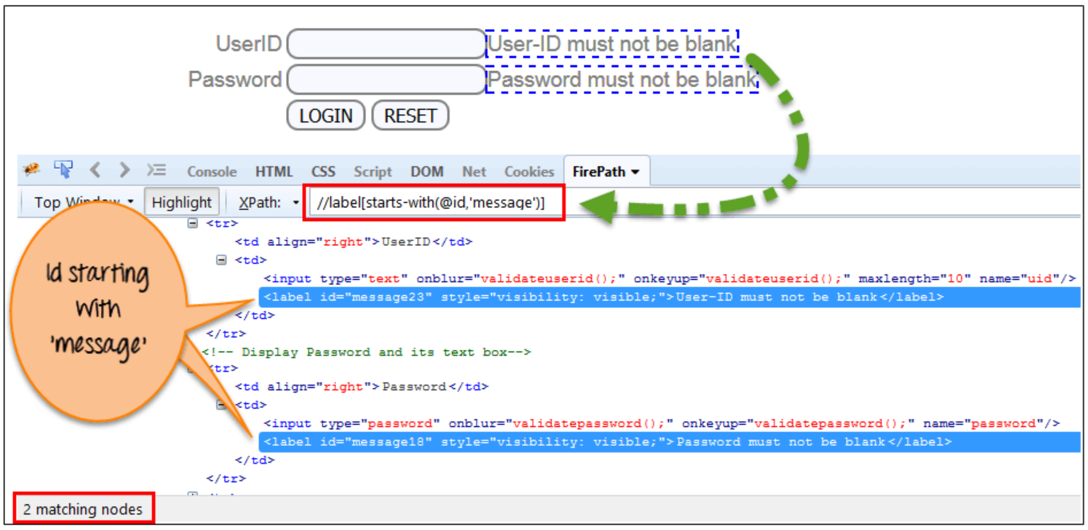
Id=" message345"a

Id=" message8769"

and so on.. but the initial text is same. In this case, we use Start-with expression.

In the below expression, there are two elements with an id starting "message"(i.e., 'User-ID must not be blank' & 'Password must not be blank'). In below example, XPath finds those element whose 'ID' starting with 'message'.

```
Xpath=//label[starts-with(@id,'message')]
```

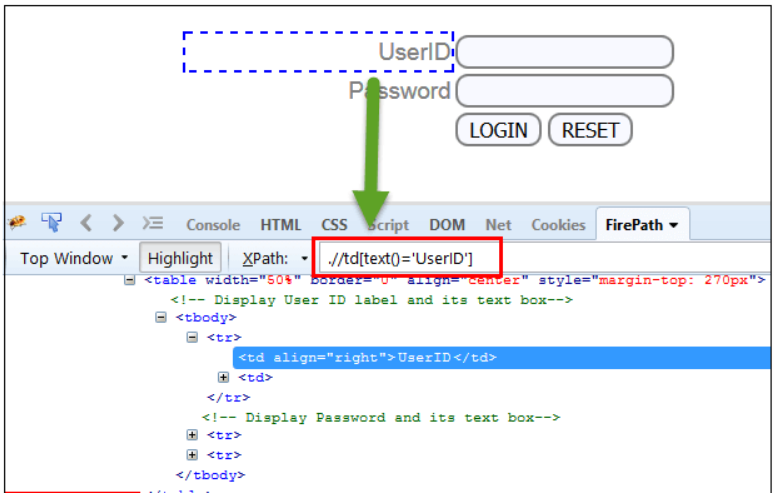

UserID
Password
User-ID must not be blank
Password must not be blank
LOGIN
RESET
Console HTML CSS Script DOM Net Cookies FirePath
Top Window
Highlight
XPath:
Id starting with 'message'
2 matching nodes

```
//label[starts-with(@id,'message')]
```

```
<tr>
  <td align="right">UserID</td>
  <td>
    <input type="text" onblur="validateuserid();" onkeyup="validateuserid();" maxlength="10" name="uid"/>
    <label id="message23" style="visibility: visible;">User-ID must not be blank</label>
  </td>
</tr>
<!-- Display Password and its text box-->
<tr>
  <td align="right">Password</td>
  <td>
    <input type="password" onblur="validatepassword();" onkeyup="validatepassword();" name="password"/>
    <label id="message18" style="visibility: visible;">Password must not be blank</label>
  </td>
</tr>
```

5) Text(): In this expression, with text function, we find the element with exact text match as shown below. In our case, we find the element with text "UserID".

```
Xpath=//td[text()='UserID']
```



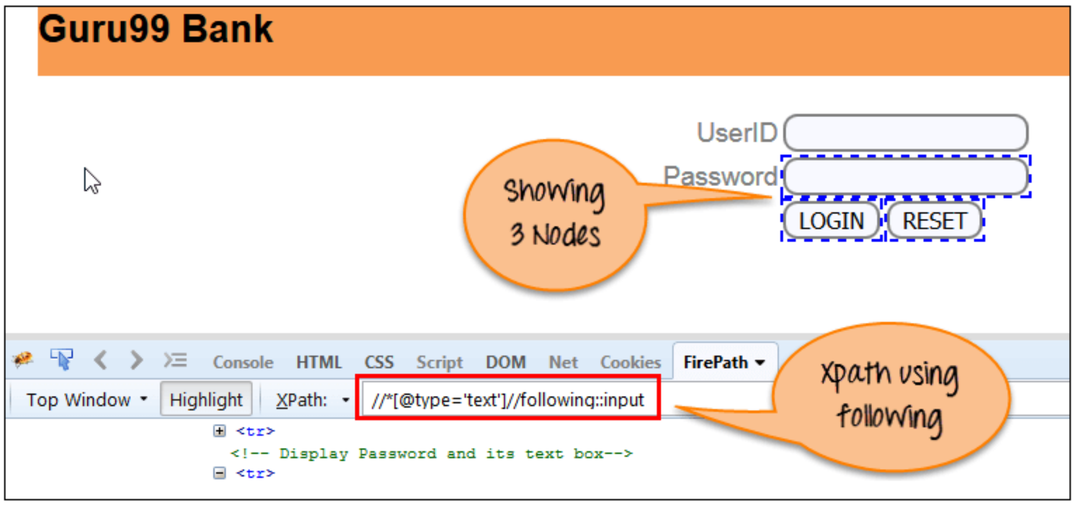
```
.//td[text()='UserID']
<table width="50%" border="0" align="center" style="margin-top: 270px">
  <!-- Display User ID label and its text box-->
  <tbody>
    <tr>
      <td align="right">UserID</td>
      <td>
    </tr>
    <!-- Display Password and its text box-->
    <tr>
    <tr>
  </tbody>
```

6) XPath axes methods: These XPath axes methods are used to find the complex or dynamic elements. Below we will see some of these methods.

For illustrating these XPath axes method, we will use the Guru99 bank demo site.

a) Following: Selects all elements in the document of the current node( ) [UserID input box is the current node] as shown in the below screen.

```
Xpath=//*[@type='text']//following::input
```



```
//*[@type='text']//following::input
<tr>
<!-- Display Password and its text box-->
<tr>
```

There are 3 "input" nodes matching by using "following" axis- password, login and reset button. If you want to focus on any particular element then you can use the below XPath method:

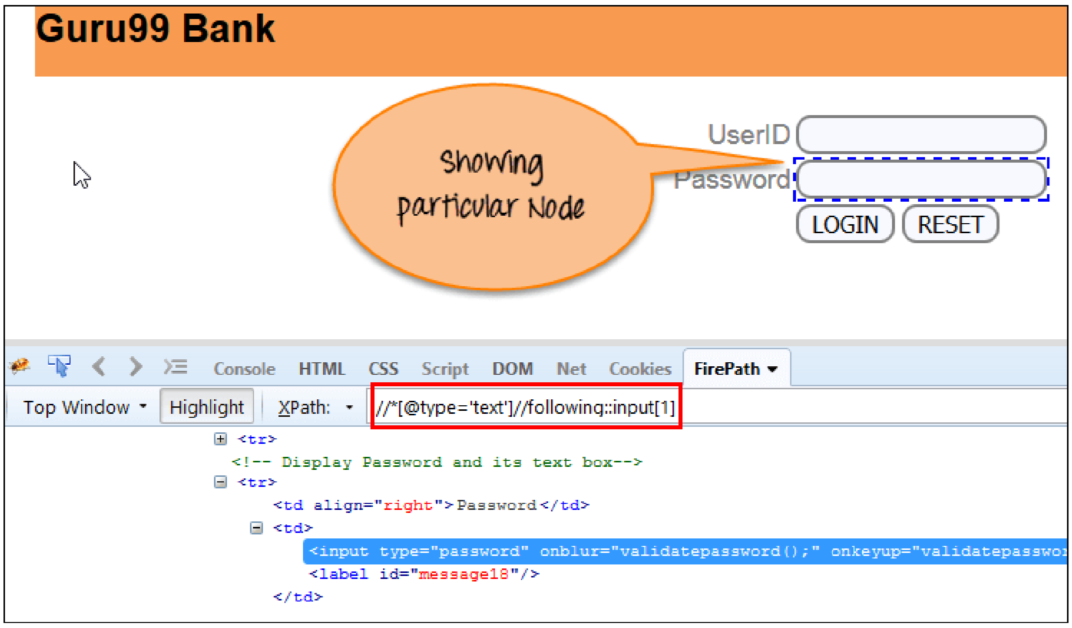
<tbody>
 	
<tr>
 	
<td>1</td>
 	
<td>2</td>
 	
</tr>
 	
<tr>
 	
<td>3</td>
	`//tbody/tr/following::tr`
	Selects the next following sibling of a tr… i.e tr[2] and tr[3]
	
	`//tbody/tr/following::td`
	Selects the td which is next to tr[1] , i.e tr[2]/td[1], tr[2]/td[2], tr[3]/td[1], tr[3]/td[2]

<td>4</td></tr> <tr><td>5</td><td>6</td></tr> </tbody>	

```
Xpath=//*[@type='text']//following::input[1]
```

You can change the XPath according to the requirement by putting [1],[2]…………and so on.

With the input as '1', the below screen shot finds the particular node that is 'Password' input box element.



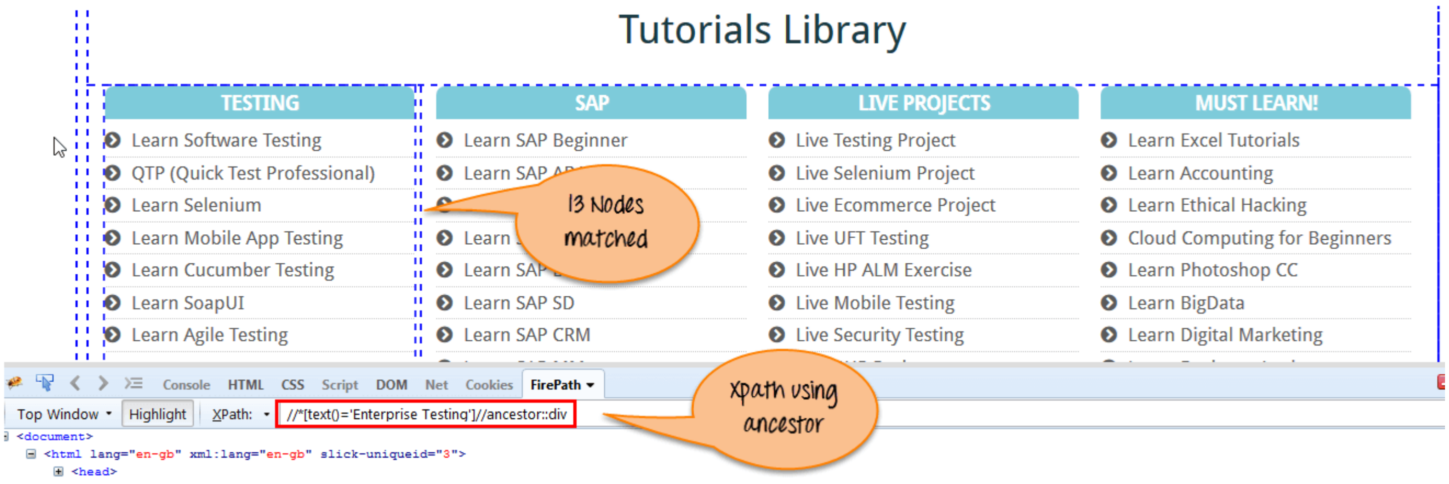
XPath: //*[@type='text']//following::input[1]

```
⊞ <tr>
  <!-- Display Password and its text box-->
⊟ <tr>
      <td align="right">Password</td>
    ⊟ <td>
        <input type="password" onblur="validatepassword();" onkeyup="validatepasswo
        <label id="message18"/>
      </td>
```

b) Ancestor: The ancestor axis selects all ancestors element (grandparent, parent, etc.) of the current node as shown in the below screen.

In the below expression, we are finding ancestors element of the current node("ENTERPRISE TESTING" node).

```
Xpath=//*[text()='Enterprise Testing']//ancestor::div
```



TESTING	SAP	LIVE PROJECTS	MUST LEARN!
Learn Software Testing	Learn SAP Beginner	Live Testing Project	Learn Excel Tutorials
QTP (Quick Test Professional)	Learn SAP AP	Live Selenium Project	Learn Accounting
Learn Selenium		Live Ecommerce Project	Learn Ethical Hacking
Learn Mobile App Testing	Learn	Live UFT Testing	Cloud Computing for Beginners
Learn Cucumber Testing	Learn SAP	Live HP ALM Exercise	Learn Photoshop CC
Learn SoapUI	Learn SAP SD	Live Mobile Testing	Learn BigData
Learn Agile Testing	Learn SAP CRM	Live Security Testing	Learn Digital Marketing

XPath: //*[text()='Enterprise Testing']//ancestor::div

```
<document>
 ⊟ <html lang="en-gb" xml:lang="en-gb" slick-uniqueid="3">
    ⊞ <head>
```

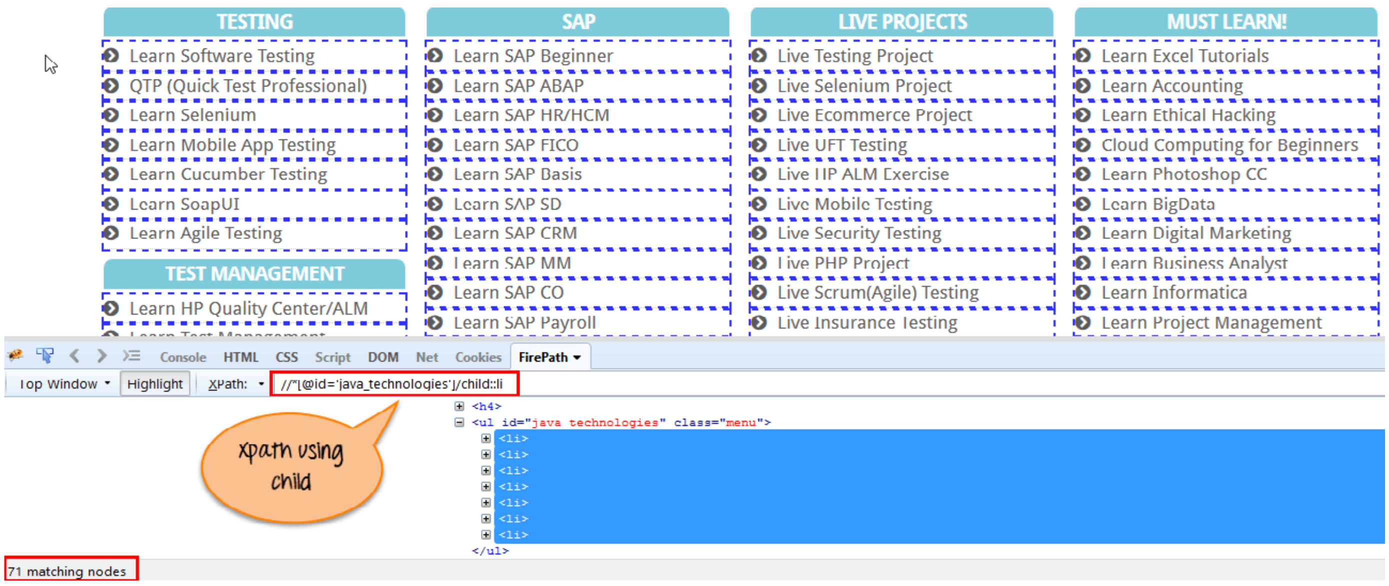
There are 13 "div" nodes matching by using "ancestor" axis. If you want to focus on any particular element then you can use the below XPath, where you change the number 1, 2 as per your requirement:

```
Xpath=//*[text()='Enterprise Testing']//ancestor::div[1]
```

You can change the XPath according to the requirement by putting [1], [2]…………and so on.

c) Child : Selects all children elements of the current node (Java) as shown in the below screen.

```
Xpath=//*[@id='java_technologies']/child::li
```



TESTING	SAP	LIVE PROJECTS	MUST LEARN!
Learn Software Testing	Learn SAP Beginner	Live Testing Project	Learn Excel Tutorials
QTP (Quick Test Professional)	Learn SAP ABAP	Live Selenium Project	Learn Accounting
Learn Selenium	Learn SAP HR/HCM	Live Ecommerce Project	Learn Ethical Hacking
Learn Mobile App Testing	Learn SAP FICO	Live UFT Testing	Cloud Computing for Beginners
Learn Cucumber Testing	Learn SAP Basis	Live HP ALM Exercise	Learn Photoshop CC
Learn SoapUI	Learn SAP SD	Live Mobile Testing	Learn BigData
Learn Agile Testing	Learn SAP CRM	Live Security Testing	Learn Digital Marketing
TEST MANAGEMENT	Learn SAP MM	Live PHP Project	Learn Business Analyst
Learn HP Quality Center/ALM	Learn SAP CO	Live Scrum(Agile) Testing	Learn Informatica
	Learn SAP Payroll	Live Insurance Testing	Learn Project Management

```
//*[@id='java_technologies']/child::li
```

```
<h4>
<ul id="java_technologies" class="menu">
    <li>
    <li>
    <li>
    <li>
    <li>
    <li>
    <li>
</ul>
```

71 matching nodes

There are 71 "li" nodes matching by using "child" axis. If you want to focus on any particular element then you can use the below xpath:

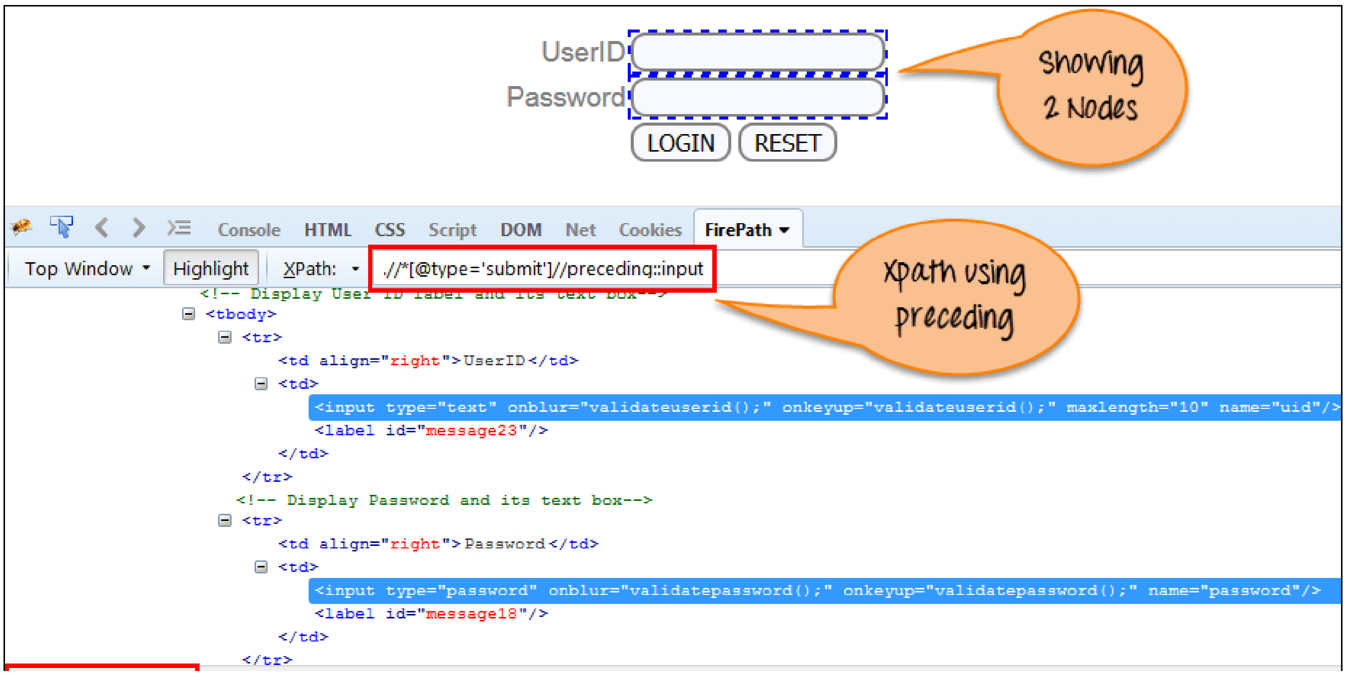
```
Xpath=//*[@id='java_technologies']/child::li[1]
```

You can change the xpath according to the requirement by putting [1],[2]…………and so on.

d) Preceding: Select all nodes that come before the current node as shown in the below screen.

In the below expression, it identifies all the input elements before "LOGIN" button that is **Userid** and **password** input element.

```
Xpath=//*[@type='submit']//preceding::input
```



```
.//*[@type='submit']//preceding::input
```

```
      <!-- Display User ID label and its text box-->
    <tbody>
      <tr>
          <td align="right">UserID</td>
        <td>
            <input type="text" onblur="validateuserid();" onkeyup="validateuserid();" maxlength="10" name="uid"/>
            <label id="message23"/>
          </td>
      </tr>
      <!-- Display Password and its text box-->
      <tr>
          <td align="right">Password</td>
        <td>
            <input type="password" onblur="validatepassword();" onkeyup="validatepassword();" name="password"/>
            <label id="message18"/>
          </td>
      </tr>
```

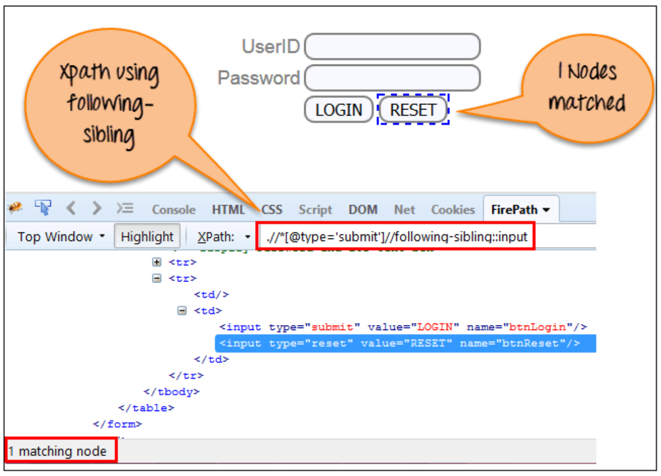
There are 2 "input" nodes matching by using "preceding" axis. If you want to focus on any particular element then you can use the below XPath:

```
Xpath=//*[@type='submit']//preceding::input[1]
```

You can change the xpath according to the requirement by putting [1],[2]…………and so on.

e) Following-sibling: Select the following siblings of the context node. Siblings are at the same level of the current node as shown in the below screen. It will find the element after the current node.

```
 xpath=//*[@type='submit']//following-sibling::input
```


Xpath using following-sibling
UserID
Password
LOGIN
RESET
1 Nodes matched
Console HTML CSS Script DOM Net Cookies FirePath
Top Window
Highlight
XPath:

```
.//*[@type='submit']//following-sibling::input
```

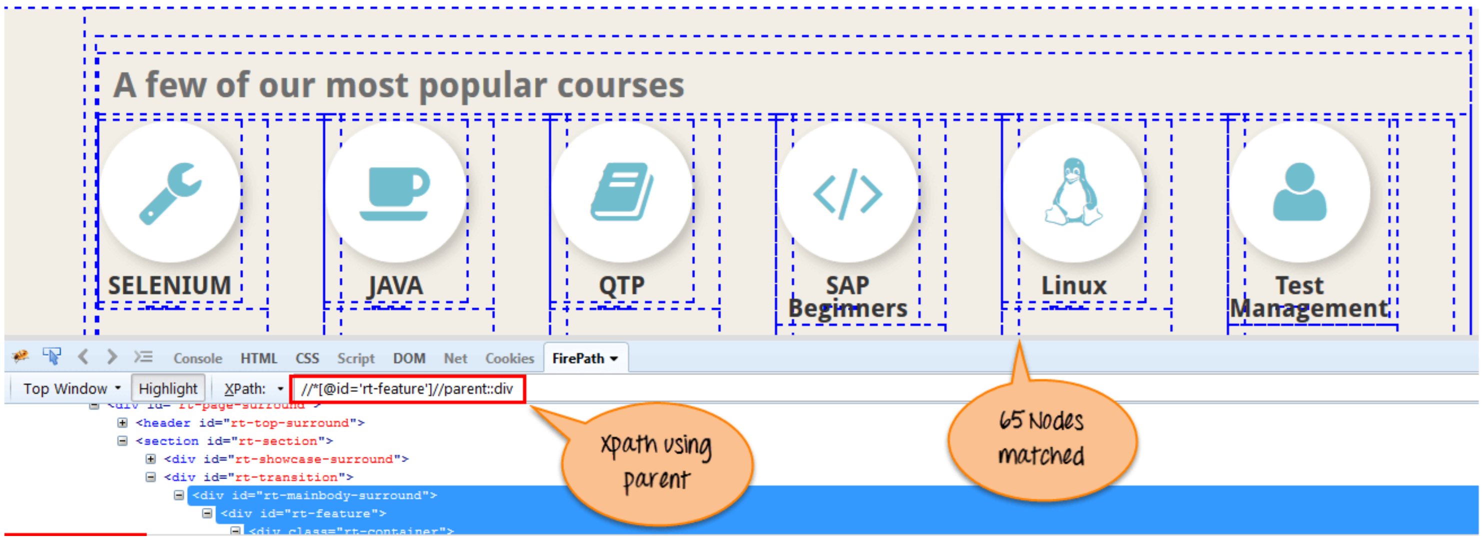
```
    <tr>
    <tr>
        <td/>
      <td>
          <input type="submit" value="LOGIN" name="btnLogin"/>
          <input type="reset" value="RESET" name="btnReset"/>
        </td>
    </tr>
  </tbody>
</table>
</form>
```

1 matching node

One input nodes matching by using "following-sibling" axis.

f) Parent: Selects the parent of the current node as shown in the below screen.

```
Xpath=//*[@id='rt-feature']//parent::div
```


A few of our most popular courses
SELENIUM
JAVA
QTP
SAP Beginners
Linux
Test Management
Console HTML CSS Script DOM Net Cookies FirePath
Top Window Highlight XPath:
Xpath using parent
65 Nodes matched

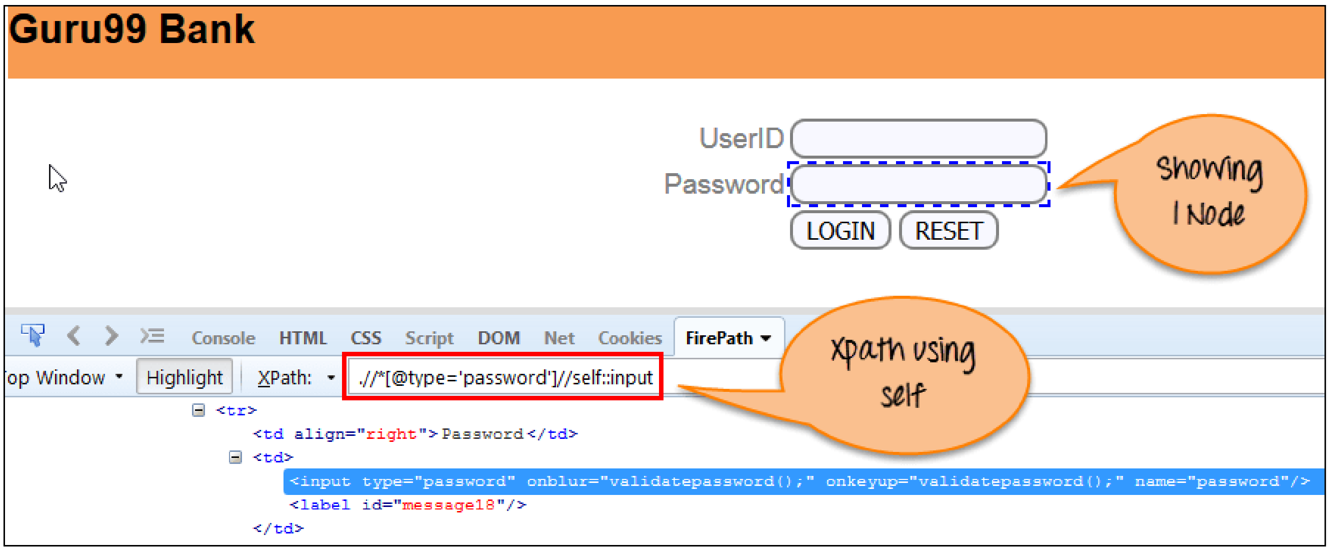
```
//*[@id='rt-feature']//parent::div
<header id="rt-top-surround">
<section id="rt-section">
<div id="rt-showcase-surround">
<div id="rt-transition">
<div id="rt-mainbody-surround">
<div id="rt-feature">
<div class="rt-container">
```

There are 65 "div" nodes matching by using "parent" axis. If you want to focus on any particular element then you can use the below XPath:

```
Xpath=//*[@id='rt-feature']//parent::div[1]
```

You can change the XPath according to the requirement by putting [1],[2]………..and so on.

g) Self: Selects the current node or 'self' means it indicates the node itself as shown in the below screen.


Guru99 Bank
UserID
Password
LOGIN
RESET
Showing 1 Node
Console HTML CSS Script DOM Net Cookies FirePath
op Window Highlight XPath:
Xpath using self

```
.//*[@type='password']//self::input
<tr>
    <td align="right">Password</td>
    <td>
        <input type="password" onblur="validatepassword();" onkeyup="validatepassword();" name="password"/>
        <label id="message18"/>
    </td>
```

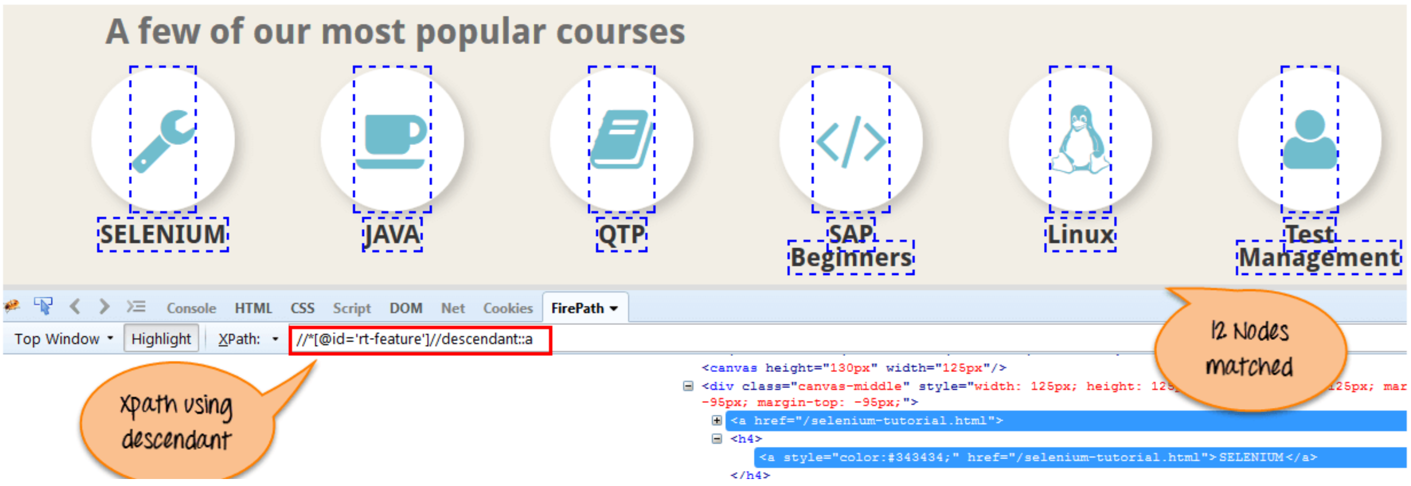
One node matching by using "self " axis. It always finds only one node as it represents self-element.

```
Xpath =//*[@type='password']//self::input
```

h) Descendant: Selects the descendants of the current node as shown in the below screen.

In the below expression, it identifies all the element descendants to current element ('Main body surround' frame element) which means down under the node (child node , grandchild node, etc.).

```
Xpath=//*[@id='rt-feature']//descendant::a
```


A few of our most popular courses
SELENIUM
JAVA
QTP
SAP Beginners
Linux
Test Management
Console HTML CSS Script DOM Net Cookies FirePath
Top Window Highlight XPath:
//*[@id='rt-feature']//descendant::a
Xpath using descendant
12 Nodes matched

```
<canvas height="130px" width="125px"/>
<div class="canvas-middle" style="width: 125px; height: 125              125px; mar
-95px; margin-top: -95px;">
  <a href="/selenium-tutorial.html">
  <h4>
    <a style="color:#343434;" href="/selenium-tutorial.html">SELENIUM</a>
  </h4>
```

There are 12 "link" nodes matching by using "descendant" axis. If you want to focus on any particular element then you can use the below XPath:

```
Xpath=//*[@id='rt-feature']//descendant::a[1]
```

You can change the XPath according to the requirement by putting [1],[2]………..and so on.

CSS Selectors

I: Simple

Direct child

A direct child in XPATH is defined by the use of a "/", while on CSS, it's defined using ">"

Examples:

```
//div/a

css=div>a
```

Child or subchild;

If an element could be inside another or one it's childs, it's defined in XPATH using "//" and in CSS just by a whitespace.

Examples:

```
//div//a

css=diva
```

Id

An element's id in XPATH is defined using: "`[@id='example']`" and in CSS using: "`#`"

Examples:

```
//div[@id='example']//a

css=div#example a
```

Class

For class, things are pretty similar in XPATH: "`[@class='example']`" while in CSS it's just "."

Examples:

```
//div[@class='example']//a

css=div.examplea
```

II: Advanced

Next sibling

This is useful for navigating lists of elements, such as forms or ul items. The next sibling will tell selenium to find the next adjacent element on the page that's inside the same parent. Let's show an example using a form to select the field after username.

```
</input></input>
```

Let's write a css selector that will choose the input field after "username". This will select the "alias" input, or will select a different element if the form is reordered.

```
css=forminput.username+input
```

Attribute values

If you don't care about the ordering of child elements, you can use an attribute selector in selenium to choose elements based on any attribute value. A good example would be choosing the 'username' element of the form without adding a class.

```
</input></input></input></input>
```

We can easily select the username element without adding a class or an id to the element.

```
css=forminput[name='username']
```

We can even chain filters to be more specific with our selections.

```
css=input[name='continue'][type='button']
```

Here Selenium will act on the input field with name="continue" and type="button"

Choosing a specific match

CSS selectors in Selenium allow us to navigate lists with more finess that the above methods. If we have a ul and we want to select its fourth li element without regard to any other elements, we should use nth-child or nth-of-type.

- `<p>Heading</p>`
- Cat
- Dog
- Car
- Goat

If we want to select the fourth li element (Goat) in this list, we can use the nth-of-type, which will find the fourth li in the list.

```
css=ul#recordlist li:nth-of-type(4)
```

On the other hand, if we want to get the fourth element only if it is a li element, we can use a filtered nth-child which will select (Car) in this case.

```
css=ul#recordlist li:nth-child(4)
```

Note, if you don't specify a child type for nth-child it will allow you to select the fourth child without regard to type. This may be useful in testing css layout in selenium.

```
css=ul#recordlist *:nth-child(4)
```

Sub-string matches

CSS in Selenium has an interesting feature of allowing partial string matches using ^=, $=, or *=. I'll define them, then show an example of each:

^= Match a prefix
`css=a[id^='id_prefix_']`

A link with an "id" that starts with the text "id_prefix_"

$= Match a suffix
`css=a[id$='_id_sufix']`

A link with an "id" that ends with the text "_id_sufix"

*= Match a substring
`css=a[id*='id_pattern']`

A link with an "id" that contains the text "id_pattern"

Matching by inner text

And last, one of the more useful pseudo-classes, :contains() will match elements with the desired text block:
`css=a:contains('Log Out')`

This will find the log out button on your page no matter where it's located. This is by far my favorite CSS selector and I find it greatly simplifies a lot of my test code.

Strings and Randomly Generated Ids

We can also use string matchers to locate elements:

<div id="123_randomId"> <div id="randomId_456"> <div id="123_pattern_randomId">
To select the first div element, we would use **^=** which means 'starts with': css="div[id^='123']"
To select second div element, we would use **$=** which means 'ends with': css="div[id$='456']"
To select the last div element we would use ***=** which means 'sub-string' css="div[id*='_pattern_']"
We can also use the 'contains' css="div:contains(_pattern_)"

Attribute NOT contain a specific value

In WebDriver, how do you find elements whose attribute contain values which you don't want to select? This CSS selector example shows how NOT to select by specific attribute value

Suppose you have many elements which share the same attribute and attribute value, but some of those elements have other variables appended to the value. e.g:

```
<div class="day past calendar-day-2017-02-13 calendar-dow-1 unavailable">
<div class="day today calendar-day-2017-02-14 calendar-dow-2 unavailable">
<div class="day calendar-day-2017-02-15 calendar-dow-3">
<div class="day calendar-day-2017-02-16 calendar-dow-4">
```

In the above snippet, we want to select an available day (i.e. the two last divs)

As can be seen, all four divs contain “calendar-day-” but the first two also contain “unavailable” which we don’t want.

The CSS selector for Not selecting the first two divs is

```
css = "div[class*=calendar-day-]:not([class*='unavailable'])"
```

Synchronization
Page Load Timeout

Sets the amount of time to wait for a page load to complete

Negative value means indefinite wait time

Implicit Wait:
Specifies the waiting time for element not immediately visible and ZERO (0) by default

driver.manage().timeouts(). pageLoadTimeout(30, TimeUnit.SECONDS);
driver.manager().timeouts().implicitlyWait(10,TimeUnits.SECONDS);

Explicit Wait:
Waiting for a certain condition

Poor alternative –**Thread.sleep(1000);**
•Recommended –**WebDriverWaitclass**

EXAMPLE :
WebDriverWaitwait = new WebDriverWait(driver, TIME_OUT);

wait.until(ExpectedConditions.method);

ExpectedConditionsclass:

Modifier and Type	Method and Description
`static ExpectedCondition<Alert>`	`alertIsPresent()`
`static ExpectedCondition<java.lang.Boolean>`	`and(ExpectedCondition<?>... conditions)` An expectation with the logical and condition of the given list of conditions.
`static ExpectedCondition<java.lang.Boolean>`	`attributeContains(By locator, String attribute, String value)` An expectation for checking WebElement with given locator has attribute which contains specific value
`static ExpectedCondition<java.lang.Boolean>`	`attributeContains(WebElement element, String attribute, String value)` An expectation for checking WebElement with given locator has attribute which contains specific value
`static ExpectedCondition<java.lang.Boolean>`	`attributeToBe(By locator,String attribute, String value)` An expectation for checking WebElement with given locator has attribute with a specific value
`static ExpectedCondition<java.lang.Boolean>`	`attributeToBe(WebElement element, String attribute, String value)` An expectation for checking given WebElement has attribute with a specific value
`static ExpectedCondition<java.lang.Boolean>`	`attributeToBeNotEmpty(WebElement element, String attribute)` An expectation for checking WebElement any non empty value for given attribute
`static ExpectedCondition<java.lang.`	`elementSelectionStateToBe(By locator, boolean selected)`

Boolean>	
static ExpectedCondition<java.lang.Boolean>	elementSelectionStateToBe(WebElement element, boolean selected) An expectation for checking if the given element is selected.
static ExpectedCondition<WebElement>	elementToBeClickable(By locator) An expectation for checking an element is visible and enabled such that you can click it.
static ExpectedCondition<WebElement>	elementToBeClickable(WebElement element) An expectation for checking an element is visible and enabled such that you can click it.
static ExpectedCondition<java.lang.Boolean>	elementToBeSelected(By locator)
static ExpectedCondition<java.lang.Boolean>	elementToBeSelected(WebElement element) An expectation for checking if the given element is selected.
static ExpectedCondition<WebDriver>	frameToBeAvailableAndSwitchToIt(By locator) An expectation for checking whether the given frame is available to switch to.
static ExpectedCondition<WebDriver>	frameToBeAvailableAndSwitchToIt(int frameLocator) An expectation for checking whether the given frame is available to switch to.
static ExpectedCondition<WebDriver>	frameToBeAvailableAndSwitchToIt(String frameLocator) An expectation for checking whether the given frame is available to switch to.
static ExpectedCondition<WebDriver>	frameToBeAvailableAndSwitchToIt(WebElement frameLocator) An expectation for checking whether the given frame is available to switch to.
static ExpectedCondition<java.lang.Boolean>	invisibilityOf(WebElement element) An expectation for checking the element to be invisible
static ExpectedCondition<java.lang.Boolean>	invisibilityOfAllElements(java.util.List<WebElement> elements) An expectation for checking all elements from given list to be invisible
static ExpectedCondition<java.lang.Boolean>	invisibilityOfElementLocated(By locator) An expectation for checking that an element is either invisible or not present on the DOM.
static ExpectedCondition<java.lang.Boolean>	invisibilityOfElementWithText(By locator, java.lang.String text) An expectation for checking that an element with text is either invisible or not present on the DOM.
static ExpectedCondition<java.lang.Boolean>	javaScriptThrowsNoExceptions(java.lang.String javaScript) An expectation to check if js executable.
static ExpectedCondition<java.lang.	jsReturnsValue(java.lang.String javaScript)

Object>	An expectation for String value from javascript
static ExpectedCondition<java.lang.Boolean>	not(ExpectedCondition<?> condition) An expectation with the logical opposite condition of the given condition.
static ExpectedCondition<java.util.List<WebElement>>	numberOfElementsToBe(By locator, java.lang.Integer number) An expectation for checking number of WebElements with given locator
static ExpectedCondition<java.util.List<WebElement>>	numberOfElementsToBeLessThan(By locator, java.lang.Integer number) An expectation for checking number of WebElements with given locator being less than defined number
static ExpectedCondition<java.util.List<WebElement>>	numberOfElementsToBeMoreThan(By locator, java.lang.Integer number) An expectation for checking number of WebElements with given locator
static ExpectedCondition<java.lang.Boolean>	numberOfwindowsToBe(int expectedNumberOfWindows) Deprecated. please use numberOfWindowsToBe(int) instead
static ExpectedCondition<java.lang.Boolean>	numberOfWindowsToBe(int expectedNumberOfWindows)
static ExpectedCondition<java.lang.Boolean>	or(ExpectedCondition<?>... conditions) An expectation with the logical or condition of the given list of conditions.
static ExpectedCondition<java.util.List<WebElement>>	presenceOfAllElementsLocatedBy(By locator) An expectation for checking that there is at least one element present on a web page.
static ExpectedCondition<WebElement>	presenceOfElementLocated(By locator) An expectation for checking that an element is present on the DOM of a page.
static ExpectedCondition<WebElement>	presenceOfNestedElementLocatedBy(By locator, By childLocator) An expectation for checking child WebElement as a part of parent element to present
static ExpectedCondition<WebElement>	presenceOfNestedElementLocatedBy(WebElement element, By childLocator) An expectation for checking child WebElement as a part of parent element to be present
static ExpectedCondition<java.util.List<WebElement>>	presenceOfNestedElementsLocatedBy(By parent, By childLocator) An expectation for checking child WebElement as a part of parent element to present
static <T> ExpectedCondition<T>	refreshed(ExpectedCondition<T> condition) Wrapper for a condition, which allows for elements to update by redrawing.
static ExpectedCondition<java.lang.Boolean>	stalenessOf(WebElement element) Wait until an element is no longer attached to the DOM.
static ExpectedCondition<java.lang.Boolean>	textMatches(By locator, java.util.regex.Pattern pattern) An expectation for checking WebElement with given

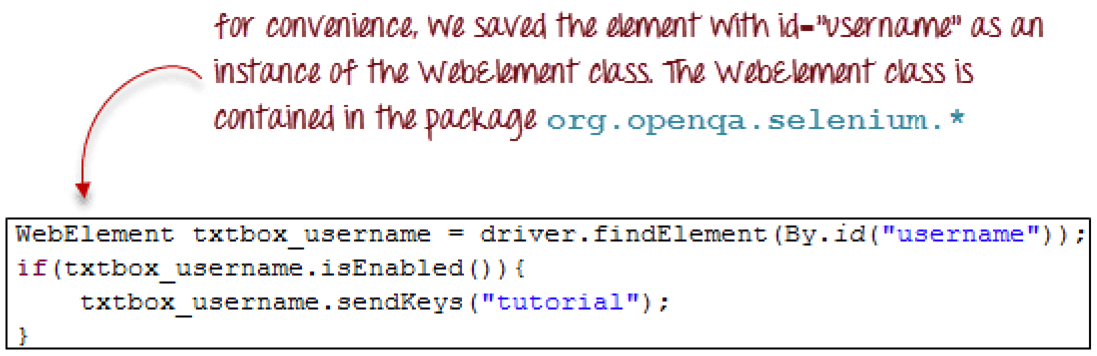
	locator has text with a value as a part of it
static ExpectedCondition<java.lang.Boolean>	textToBe(By locator, java.lang.String value) An expectation for checking WebElement with given locator has specific text
static ExpectedCondition<java.lang.Boolean>	textToBePresentInElement(By locator, java.lang.String text) Deprecated. Use textToBePresentInElementLocated(By, String) instead
static ExpectedCondition<java.lang.Boolean>	textToBePresentInElement(WebElement element, java.lang.String text) An expectation for checking if the given text is present in the specified element.
static ExpectedCondition<java.lang.Boolean>	textToBePresentInElementLocated(By locator, java.lang.String text) An expectation for checking if the given text is present in the element that matches the given locator.
static ExpectedCondition<java.lang.Boolean>	textToBePresentInElementValue(By locator, java.lang.String text) An expectation for checking if the given text is present in the specified elements value attribute.
static ExpectedCondition<java.lang.Boolean>	textToBePresentInElementValue(WebElement element, java.lang.String text) An expectation for checking if the given text is present in the specified elements value attribute.
static ExpectedCondition<java.lang.Boolean>	titleContains(java.lang.String title) An expectation for checking that the title contains a case-sensitive substring
static ExpectedCondition<java.lang.Boolean>	titleIs(java.lang.String title) An expectation for checking the title of a page.
static ExpectedCondition<java.lang.Boolean>	urlContains(java.lang.String fraction) An expectation for the URL of the current page to contain specific text.
static ExpectedCondition<java.lang.Boolean>	urlMatches(java.lang.String regex) Expectation for the URL to match a specific regular expression
static ExpectedCondition<java.lang.Boolean>	urlToBe(java.lang.String url) An expectation for the URL of the current page to be a specific url.
static ExpectedCondition<WebElement>	visibilityOf(WebElement element) An expectation for checking that an element, known to be present on the DOM of a page, is visible.
static ExpectedCondition<java.util.List<WebElement>>	visibilityOfAllElements(java.util.List<WebElement> elements) An expectation for checking that all elements present on the web page that match the locator are visible.
static ExpectedCondition<java.util.List<WebElement>>	visibilityOfAllElementsLocatedBy(By locator) An expectation for checking that all elements present on the web page that match the locator are visible.
static ExpectedCondition<WebElement	visibilityOfElementLocated(By locator) An expectation for checking that an element is present

>	on the DOM of a page and visible.
static ExpectedCondition<java.util.List<WebElement>>	visibilityOfNestedElementsLocatedBy(By parent, By childLocator) An expectation for checking child WebElement as a part of parent element to be visible
static ExpectedCondition<java.util.List<WebElement>>	visibilityOfNestedElementsLocatedBy(WebElement element, By childLocator) An expectation for checking child WebElement as a part of parent element to be visible

Conditions

Following methods are used in conditional and looping operations --

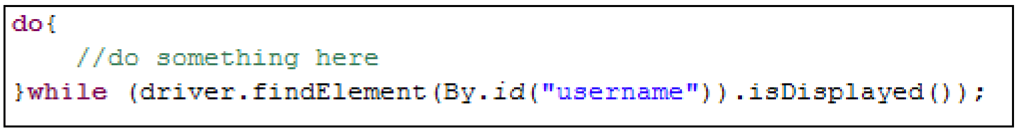
- **isEnabled()** is used when you want to verify whether a certain element is enabled or not before executing a command.



for convenience, we saved the element with id="username" as an instance of the WebElement class. The WebElement class is contained in the package org.openqa.selenium.*

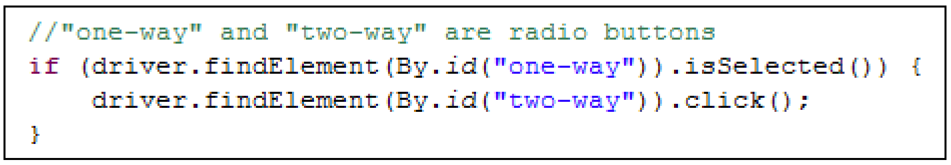
```
WebElement txtbox_username = driver.findElement(By.id("username"));
if(txtbox_username.isEnabled()){
    txtbox_username.sendKeys("tutorial");
}
```

- **isDisplayed()** is used when you want to verify whether a certain element is displayed or not before executing a command.



```
do{
    //do something here
}while (driver.findElement(By.id("username")).isDisplayed());
```

- **isSelected()** is used when you want to verify whether a certain **check box, radio button, or option in a drop-down box** is selected. It does not work on other elements.



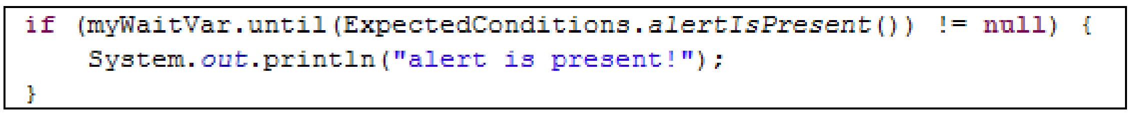
```
//"one-way" and "two-way" are radio buttons
if (driver.findElement(By.id("one-way")).isSelected()) {
    driver.findElement(By.id("two-way")).click();
}
```

Using ExpectedConditions

The ExpectedConditions class offers a wider set of conditions that you can use in conjunction with WebDriverWait's until() method.

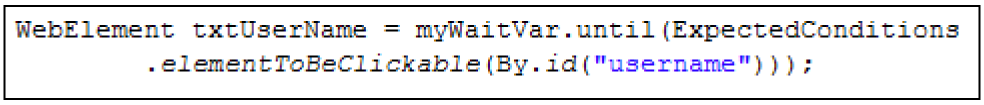
Below are some of the most common ExpectedConditions methods.

- **alertIsPresent()** - waits until an alert box is displayed.



```
if (myWaitVar.until(ExpectedConditions.alertIsPresent()) != null) {
    System.out.println("alert is present!");
}
```

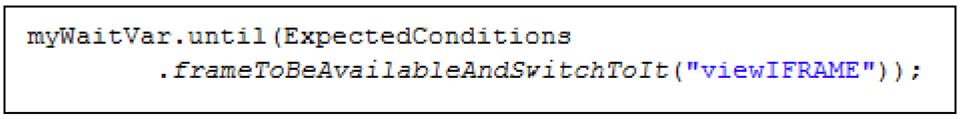
- **elementToBeClickable()** - Waits until an element is visible and, at the same time, enabled. The sample code below will wait until the element with id="username" to become visible and enabled first before assigning that element as a WebElement variable named "txtUserName".



```
WebElement txtUserName = myWaitVar.until(ExpectedConditions
        .elementToBeClickable(By.id("username")));
```

- **frameToBeAvailableAndSwitchToIt()** - Waits until the given frame is already available, and then automatically switches to it.

This will automatically switch to the
"viewIFRAME" frame once it becomes available.



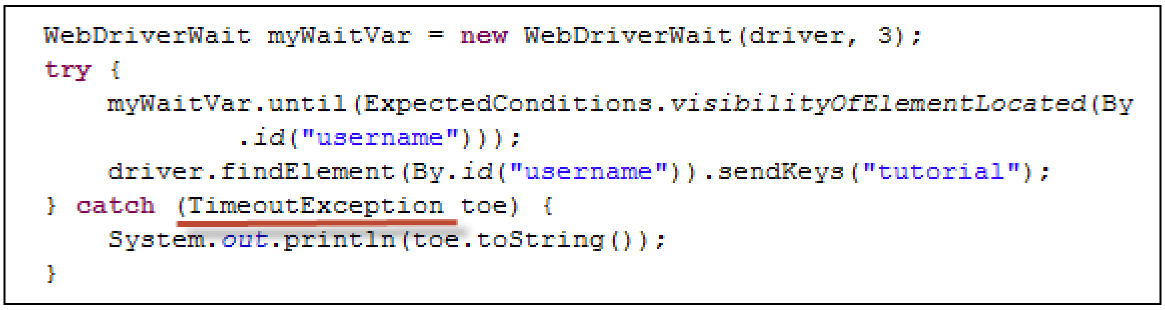
```
myWaitVar.until(ExpectedConditions
        .frameToBeAvailableAndSwitchToIt("viewIFRAME"));
```

Catching Exceptions

When using isEnabled(), isDisplayed(), and isSelected(), WebDriver assumes that the element already exists on the page. Otherwise, it will throw a **NoSuchElementException**. To avoid this, we should use a try-catch block so that the program will not be interrupted.

```
WebElement txtbox_username = driver.findElement(By.id("username"));
try{
        if(txtbox_username.isEnabled()){
            txtbox_username.sendKeys("tutorial");
        }
    }

catch(NoSuchElementException nsee){
            System.out.println(nsee.toString());
 }
```

If you use explicit waits, the type of exception that you should catch is the "TimeoutException".



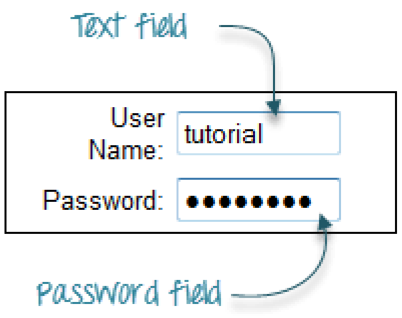
```
WebDriverWait myWaitVar = new WebDriverWait(driver, 3);
try {
    myWaitVar.until(ExpectedConditions.visibilityOfElementLocated(By
            .id("username")));
    driver.findElement(By.id("username")).sendKeys("tutorial");
} catch (TimeoutException toe) {
    System.out.println(toe.toString());
}
```

Accessing Form Elements

Input Box

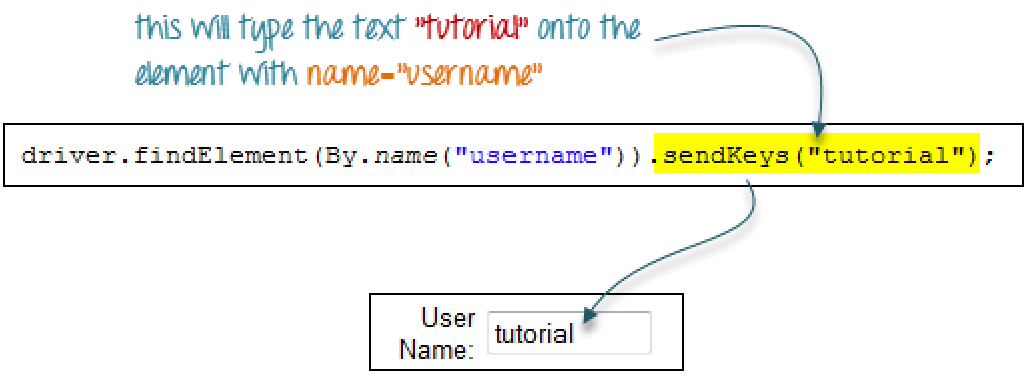
Input boxes refer to either of these two types:

1. **Text Fields**- text boxes that accept typed values and show them as they are.
2. **Password Fields**- text boxes that accept typed values but mask them as a series of special characters (commonly dots and asterisks) to avoid sensitive values to be displayed.



Entering Values in Input Boxes

The **sendKeys()** method is used to enter values into input boxes.


this will type the text "tutorial" onto the element with name="username"
User Name:
tutorial

```
driver.findElement(By.name("username")).sendKeys("tutorial");
```

Deleting Values in Input Boxes

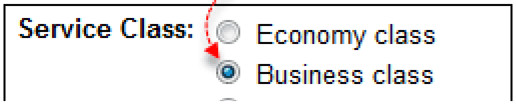
The **clear()** method is used to delete the text in an input box. **This method does not need any parameter**. The code snippet below will clear out the text "tutorial" in the User Name text box.

```
driver.findElement(By.name("userName")).clear();
```

Radio Button

Toggling a radio button on is done using the **click()** method.

```
driver.findElement(By.cssSelector("input[value='Business']")).click();
```

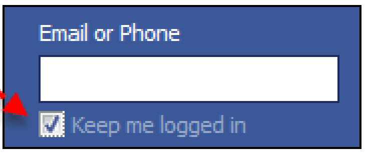


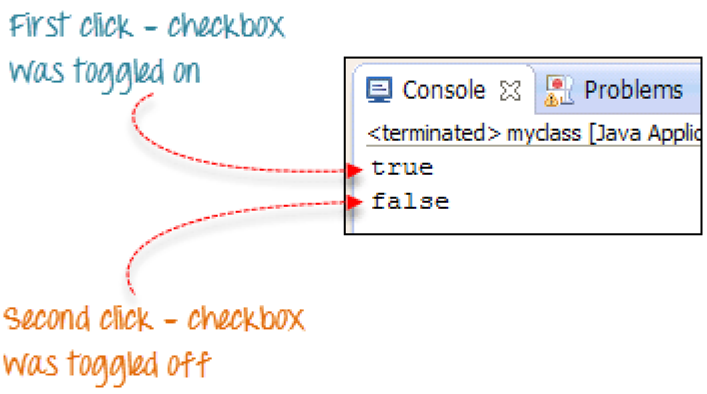
Check Box

Toggling a check box on/off is also done using the **click()** method.

The code below will click on Facebook's "Keep me logged in" check box twice and then output the result as TRUE when it is toggled on, and FALSE if it is toggled off.

```
public static void main(String[] args) {
    WebDriver driver = new FirefoxDriver();
    String baseURL = "http://www.facebook.com";

    driver.get(baseURL);
    WebElement chkFBPersist = driver.findElement(By.id("persist_box"));
    for(int i=0; i<2; i++){
        chkFBPersist.click();
        System.out.println(chkFBPersist.isSelected());
    }
    driver.quit();
}
```



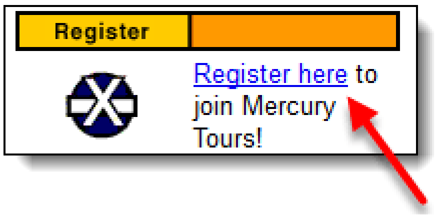

First click - checkbox was toggled on
Console
Problems
<terminated> myclass [Java Applic
Second click - checkbox was toggled off

```
true
false
```

Links

Links also are accessed by using the **click()** method.

Consider the below link found in Mercury Tours' homepage.

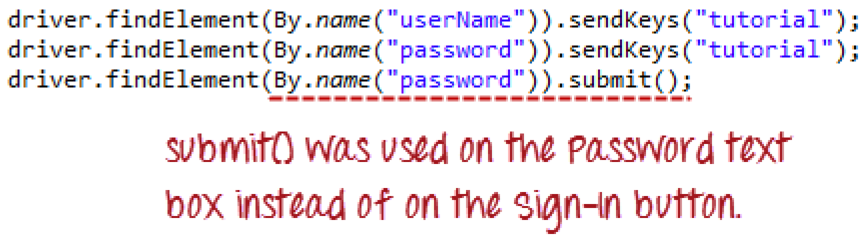


You can access this link using linkText() or partialLinkText() together with click(). Either of the two lines below will be able to access the "Register here" link shown above.

Submitting a Form

The **submit()** method is used to submit a form. This is an alternative to clicking the form's submit button.

You can use submit() on any element within the form, not just on the submit button itself.



```
driver.findElement(By.name("userName")).sendKeys("tutorial");
driver.findElement(By.name("password")).sendKeys("tutorial");
driver.findElement(By.name("password")).submit();
```

submit() was used on the password text box instead of on the sign-in button.

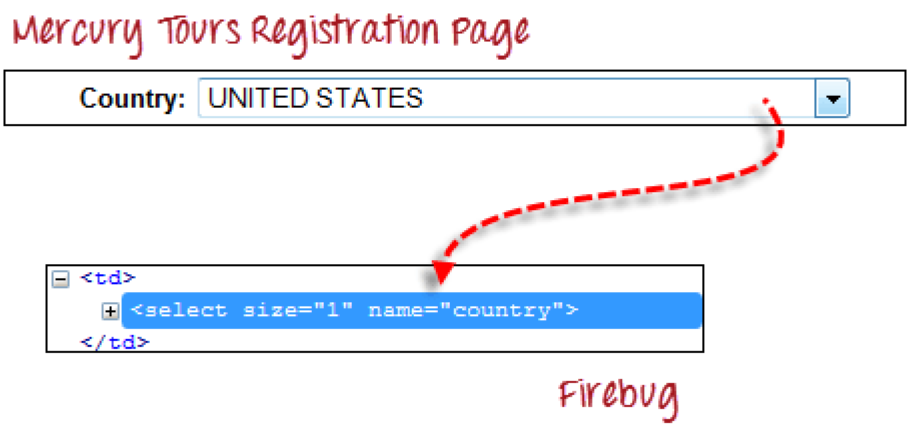
When submit() is used, WebDriver will look up the DOM to know which form the element belongs to, and then trigger its submit function.

Drop-Down Box

Before we can control drop-down boxes, we must do following two things:

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

As an example, go to Mercury Tours' Registration page (http://newtours.demoaut.com/mercuryregister.php) and notice the "Country" drop-down box there.



```
<td>
  <select size="1" name="country">
</td>
```

Step 1

Import the "Select" package.

```
import org.openqa.selenium.support.ui.Select;
```

Step 2

Declare the drop-down element as an instance of the Select class. In the example below, we named this instance as "drpCountry".

```
Select drpCountry = new Select(driver.findElement(By.name("country")));
```

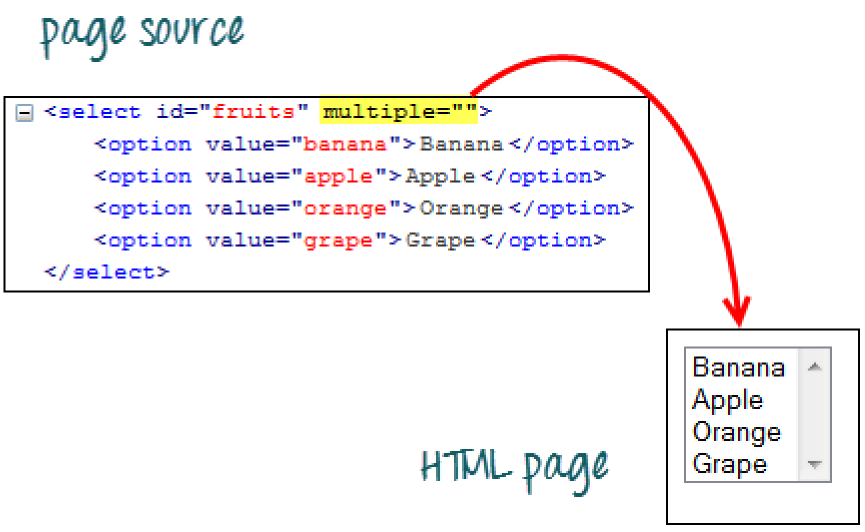
Step 3

We can now start controlling "drpCountry" by using any of the available Select methods. The sample code below will select the option "ANTARCTICA."

```
drpCountry.selectByVisibleText("ANTARCTICA");
```

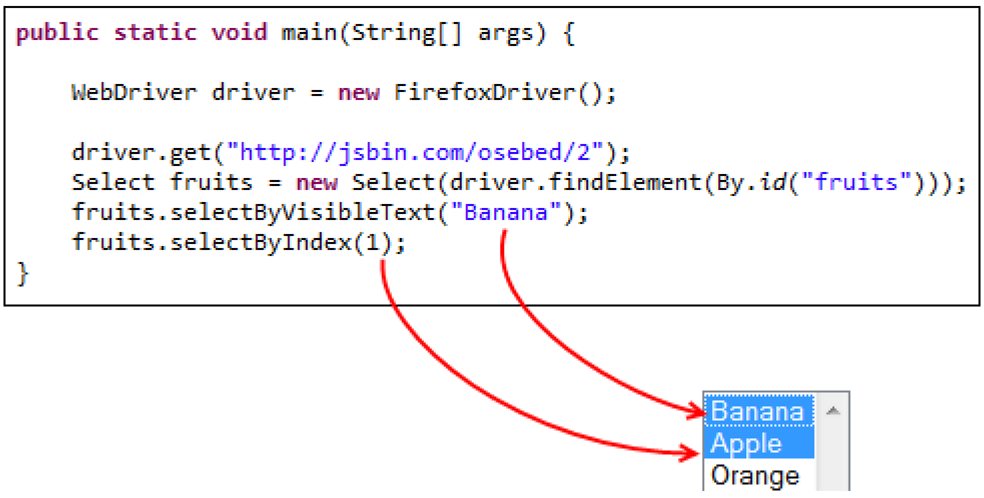
Selecting Items in a Multiple SELECT elements

We can also use the **selectByVisibleText()** method in selecting multiple options in a multi SELECT element. As an example, we will take http://jsbin.com/osebed/2 as the base URL. It contains a drop-down box that allows multiple selections at a time.



```
<select id="fruits" multiple="">
    <option value="banana">Banana</option>
    <option value="apple">Apple</option>
    <option value="orange">Orange</option>
    <option value="grape">Grape</option>
</select>
```

The code below will select the first two options using the selectByVisibleText() method.

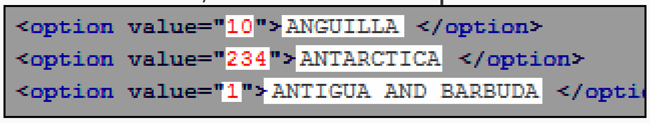


```
public static void main(String[] args) {

    WebDriver driver = new FirefoxDriver();

    driver.get("http://jsbin.com/osebed/2");
    Select fruits = new Select(driver.findElement(By.id("fruits")));
    fruits.selectByVisibleText("Banana");
    fruits.selectByIndex(1);
}
```

Select Methods

The following are the most common methods used on drop-down elements.

Method	Description
selectByVisibleText() and**deselectByVisibleText()** *Example:* `drpCountry.selectByVisibleText("ANTARCTICA");`	• Selects/deselects the option that displays the text matching the parameter. • **Parameter**: The exactly displayed text of a particular option

selectByValue() and**deselectByValue()** *Example:* `drpCountry.selectByValue("234");`	• Selects/deselects the option whose "value" attribute matches the specified parameter.
	• **Parameter**: value of the "value" attribute
	• Remember that not all drop-down options have the same text and "value", like in the example below.
	
	`<option value="10">ANGUILLA </option>`
	`<option value="234">ANTARCTICA </option>`
	`<option value="1">ANTIGUA AND BARBUDA </opti`

selectByIndex() and**deselectByIndex()** *Example:* `drpCountry.selectByIndex(0);`	• Selects/deselects the option at the given index. • **Parameter**: the index of the option to be selected.
isMultiple() *Example:* `if (drpCountry.isMultiple()) {` `//do something here` `}`	• Returns TRUE if the drop-down element allows multiple selections at a time; FALSE if otherwise. • **Needs parameters needed**
deselectAll() *Example:*`drpCountry.deselectAll();`	• Clears all selected entries. This is only valid when the drop-down element supports multiple selections. • **No parameters needed**
`java.util.List<WebElement> type is`	•

webelement **getAllSelectedOptions**()	
	•

Summary

- The table below summarizes the commands to access each type of element discussed above.

Element	**Command**	**Description**
Input Box	*sendKeys()*	used to enter values onto text boxes
	clear()	used to clear text boxes of its current value
Check Box,Radio Button,	*click()*	used to toggle the element on/off
Links	*click()*	used to click on the link and wait for page load to complete before proceeding to the next command.
Submit Button	*submit()*	

- WebDriver allows selection of more than one option in a multiple SELECT element.
- You can use the submit() method on any element within the form. WebDriver will automatically trigger the submit function of the form where that element belongs to.

WebElement Commands.
First, we need to understand what is a web element?
It is the smallest possible HTML entity that helps to form a fully functional web page. In Webdriver terminology, it is a Java class which represents an object that holds the reference to an HTML element.
e.g. ***<br/>*** is a Web element or an HTML element. Let's consider a more complex example.
<div id='employee'>…</div>
In the above HTML code, the whole ***div*** including ***opening and closing tag*** is forming an HTML element. Now let's see the Selenium Webdriver commands associated with the web elements.
4.1- Clear Command.
void clear( ) – This method will clear the value of any text type element. It doesn't allow any parameter and its return type is also void.
Command – ***element.clear();***
This method doesn't impact any HTML element other than the field of text type. Example – INPUT and TEXTAREA elements.
Java

```
WebElement user=driver.findElement(By.id("User"));
user.clear();

//Or use the following code.

driver.findElement(By.id("User")).clear();
```

4.2- SendKeys Command.
void sendKeys(CharSequence… keysToSend) – This method types in the values passed in the argument into the associated web element object. It allows a CharSequence as a parameter, and its return type is void.
Command – ***element.sendKeys("enter text");***
This method supports elements like INPUT and TEXTAREA elements.
Java

```
WebElement user=driver.findElement(By.id("User"));
user.sendKeys("TechBeamers");

//Or follow the below coding style.

driver.findElement(By.id("User")).sendKeys("TechBeamers");
```

4.3- Click Command.
void click( ) – This method performs the click action at the web element. It doesn't accept any parameter, and its return type is void.
Command – ***element.click();***
It is one of the most common methods which interacts with the web elements like links, checkboxes, buttons, etc.
Java

```
WebElement link=driver.findElement(By.linkText("TechBeamers"));
link.click();

//Or try to use the below code.

driver.findElement(By.linkText("TechBeamers")).click();
```

Back to top
4.4- IsDisplayed Command.
boolean isDisplayed( ) – This method checks the visibility of a web element. It doesn't allow any parameters but its return type is a boolean value.
Command – ***element.isDisplayed();***

This method will return true if the element is present and visible on the page. It'll throw a *NoSuchElementFound*exception if the element is not available. If it is available but kept as hidden, then the method will return false.
Java

```
WebElement user=driver.findElement(By.id("User"));
booleanis_displayed=user.isDisplayed();

//Or write the code in the below style.

booleanis_displayed=driver.findElement(By.id("User")).isDisplayed();
```

4.5- IsEnabled Command.
boolean isEnabled( ) – This method return true/false depending on the state of the element (Enabled or not).
Command – *element.isEnabled();*
This method will usually return true for all items except those which are intentionally disabled.
Java

```
WebElement user=driver.findElement(By.id("User"));
booleanis_enabled=user.isEnabled();

//Or write the code in the below style.

booleanis_enabled=driver.findElement(By.id("User")).isEnabled();

//Or choose the following way to write code.

WebElement user=driver.findElement(By.id("User"));
booleanis_enabled=element.isEnabled();

// Test if the text input field is enabled. If true then enter the value.
if(is_enabled){
  user.sendKeys("TechBeamers");
}
```

4.6- IsSelected Command.
boolean isSelected( ) – This method tests if a element is active or selected. It doesn't allow any parameter, but it does return a boolean status.
Command – *element.isSelected();*
This method is only applicable for input elements like Checkboxes, Radio Button, and Select Options. It'll return true when it finds the element is currently selected or checked.
Java

```
WebElement lang=driver.findElement(By.id("Language"));
booleanis_selected=lang.isSelected();

//Or you can re-write the code as below.

booleanis_selected=driver.findElement(By.id("Language")).isSelected();
```

Back to top
4.7- Submit Command.
void submit( ) – This method initiates the submission of an HTML form. It doesn't allow any parameter and its return type is void.
Command – *element.submit();*
If the method is leading the current page to change, then it would wait until the new page gets loaded.

Java

```
WebElement submit_form=driver.findElement(By.id("Submit"));
submit_form.submit();

//Or write the code in the below style.

driver.findElement(By.id("Submit")).submit();
```

4.8 – GetText Command.

String getText( ) – This method will give you the innerText value of the element. You must note that the innerText property is CSS aware and the getText() method would return a blank if the text is hidden.

Command – *element.getText();*

This method provides the innerText of the associated element. The output includes the sub-elements and leaves any leading or trailing whitespace aside.

Java

```
WebElement link=driver.findElement(By.xpath("MyLink"));
Stringstrlink=link.getText();
```

4.9- GetTagName Command.

String getTagName( ) – This method provides the tag name of the associated element. It doesn't allow to pass any parameter and returns a String object.

Command – *element.getTagName();*

Don't confuse this method to return the attribute name. It'll return the tag name i.e. "input" of the element. e.g. <input name="Password"/>.

Java

```
WebElement pass=driver.findElement(By.id("Password"));
StringstrTag=pass.getTagName();

//Or style the code as below.

StringstrTag=driver.findElement(By.id("Password")).getTagName();
```

Back to top

4.10- GetCssValue Command.

String getCssValue(String propertyName) – This method provides the value of the CSS property belonging to the given element.

Command – *element.getCssValue("font-size");*

This method may return an unpredictable value in a cross-browser setup. Like, you would have set the "background-color" property to "green" but the method could return "#008000" as its value.

Java

```
WebElement name=driver.findElement(By.id("Name"));
StringstrAlign=name.getCssValue("text-align");
```

4.11- GetSize Command.

Dimension getSize( ) – This method returns the height and width of the given element. It doesn't allow to pass any parameter, but its return type is the Dimension object.

Command – *element.getSize();*

This methods provides the size of the element on the web page.

Java

```
WebElement user=driver.findElement(By.id("User"));
Dimension dim=user.getSize();
```

```
System.out.println(“Height#”+dim.height+”Width#"+dim.width);
```

4.12- GetLocation Command.

Point getLocation( ) – This method locate the location of the element on the page. It doesn’t allow to pass any parameter, but its return type is the Point object.

Command – *element.getLocation();*

This method provides the Point class object. You can easily read the X and Y coordinates of a specific element.

Java

```
WebElement name=driver.findElement(By.id("Name"));
Point point=name.getLocation();
StringstrLine=System.getProperty("line.separator");
System.out.println("X cordinate# "+point.x+strLine+"Y cordinate# "+point.y);
```

Switching Between frames to main page and vice versa

driver.switchTo().defaultContent(); → switching from anyframe to the mainpage.
driver.switchTo().frame(“ID or Name”) -> switching from main page to the particular page
driver.switchTo().frame(WebElement e) -> takes webelement as a parameter
driver.switchTo().parentFrame(); -> from subframe to the parent frame or a frame to the main page.

Switching to alert

driver.switchTo().alert().accept();	Alert al=driver.switchTo().alert();
driver.switchTo().alert().sendKeys(“Hello Saurabh”)	al.sendKeys(“Hello ”);
driver.switchTo().alert().dismiss();	al.accept();
The above methods are stored in the **Alert Class**	al.dismiss();

Switching to Window

Each window maintains an unique window handle identifier which consist of alphanumeric character.
String parent = driver.getWindowHandle(); → for current windowhandle
Set <String> Browsers = driver.getWindowHandles(); → for the other window
driver.switchTo().window(handle)

Handling multiple browsers / Working with multiple browsers

```
WebDriver driver = new FirefoxDriver();
driver.get("file:///C:/Users/gcreddy/Desktop/HTMLExamples/LoginPage.html");
String parent = driver.getWindowHandle();
System.out.println(parent);
driver.findElement(By.linkText("Sign In")).click();
Set <String> Browsers = driver.getWindowHandles();
for (String i: Browsers){
if (! i.equals(parent)){   // if it’s not parent handle then perform the operation in other
window(i.e new window)
 driver.switchTo().window(i);
 System.out.println(driver.getCurrentUrl());
}
}
driver.switchTo().window(parent);  //now switch back to the main handle
 System.out.println(driver.getCurrentUrl());
```

Action class

Any actions to be performed on the webelement of Mouse movement events. So for that the Actions class is used.
Actions abc=new Actions(driver);
 abc.click().build().perform();

build().perform(); is necessary for every Action reference variable.

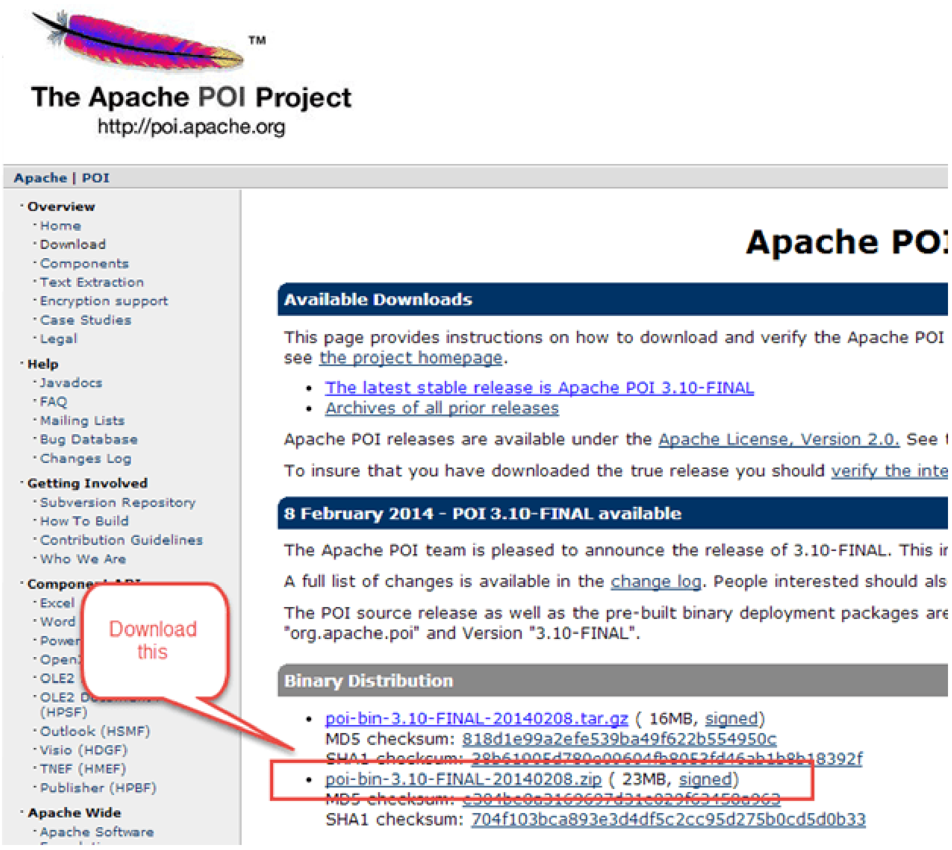
Modifier and Type	Method and Description
`Action`	`build()` Generates a composite action containing all actions so far, ready to be performed (and resets the internal builder state, so subsequent calls to `build()` will contain fresh sequences).
`Actions`	`click()` Clicks at the current mouse location.
`Actions`	`click(WebElement target)` Clicks in the middle of the given element.
`Actions`	`clickAndHold()` Clicks (without releasing) at the current mouse location.
`Actions`	`clickAndHold(WebElement target)` Clicks (without releasing) in the middle of the given element.
`Actions`	`contextClick()` Performs a context-click(RIGHT CLICK) at the current mouse location.
`Actions`	`contextClick(WebElement target)` Performs a context-click(RIGHT CLICK) at middle of the given element.
`Actions`	`doubleClick()` Performs a double-click at the current mouse location.
`Actions`	`doubleClick(WebElement target)` Performs a double-click at middle of the given element.
`Actions`	`dragAndDrop(WebElement source, WebElement target)` A convenience method that performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.
`Actions`	`dragAndDropBy(WebElement source, int xOffset, int yOffset)` A convenience method that performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.
`Actions`	`keyDown(java.lang.CharSequence key)` Performs a modifier key press.
`Actions`	`keyDown(WebElement target, java.lang.CharSequence key)` Performs a modifier key press after focusing on an element.
`Actions`	`keyUp(java.lang.CharSequence key)` Performs a modifier key release.
`Actions`	`keyUp(WebElement target, java.lang.CharSequence key)` Performs a modifier key release after focusing on an element.
`Actions`	`moveByOffset(int xOffset, int yOffset)` Moves the mouse from its current position (or 0,0) by the given offset.
`Actions`	`moveToElement(WebElement target)` Moves the mouse to the middle of the element.
`Actions`	`moveToElement(WebElement target, int xOffset, int yOffset)` Moves the mouse to an offset from the top-left corner of the element.
`Actions`	`pause(java.time.Duration duration)`
`Actions`	`pause(long pause)` **Deprecated.**

void	perform() A convenience method for performing the actions without calling build() first.
Actions	release() Releases the depressed left mouse button at the current mouse location.
Actions	release(WebElement target) Releases the depressed left mouse button, in the middle of the given element.
Actions	sendKeys(java.lang.CharSequence... keys) Sends keys to the active element.
Actions	sendKeys(WebElement target, java.lang.CharSequence... keys) Equivalent to calling: *Actions.click(element).sendKeys(keysToSend)*. This method is different from **WebElement.sendKeys(CharSequence...)** - see **sendKeys(CharSequence...)** for details how.
Actions	tick(Action action)
Actions	tick(Interaction... actions)

```
//WebElement to which the keyboard actions are performed
WebElement    textBoxElement    =    driver.findElement(By    Locator    of
textBoxElement);

//Creating object of Actions class
Actions builder = new Actions(driver);

//Generating an action to type a text in CAPS
Action         typeInCAPS         =         builder.keyDown(textBoxElement,
Keys.SHIFT).sendKeys(textBoxElement, "artOfTesting")
                   .keyUp(textBoxElement, Keys.SHIFT)
            .build();

//Performing the typeInCAPS action
typeInCAPS.perform();
```

APACHE POI: To read and write data from the Excel to the script and script to the excel.

Or you can simply download the latest version POI jars from http://poi.apache.org/download.html & download poi-bin-3.10-FINAL-20140208.zip



Apache POI

Available Downloads

This page provides instructions on how to download and verify the Apache POI
see the project homepage.

- The latest stable release is Apache POI 3.10-FINAL
- Archives of all prior releases

Apache POI releases are available under the Apache License, Version 2.0. See

To insure that you have downloaded the true release you should verify the inte

8 February 2014 - POI 3.10-FINAL available

The Apache POI team is pleased to announce the release of 3.10-FINAL. This i

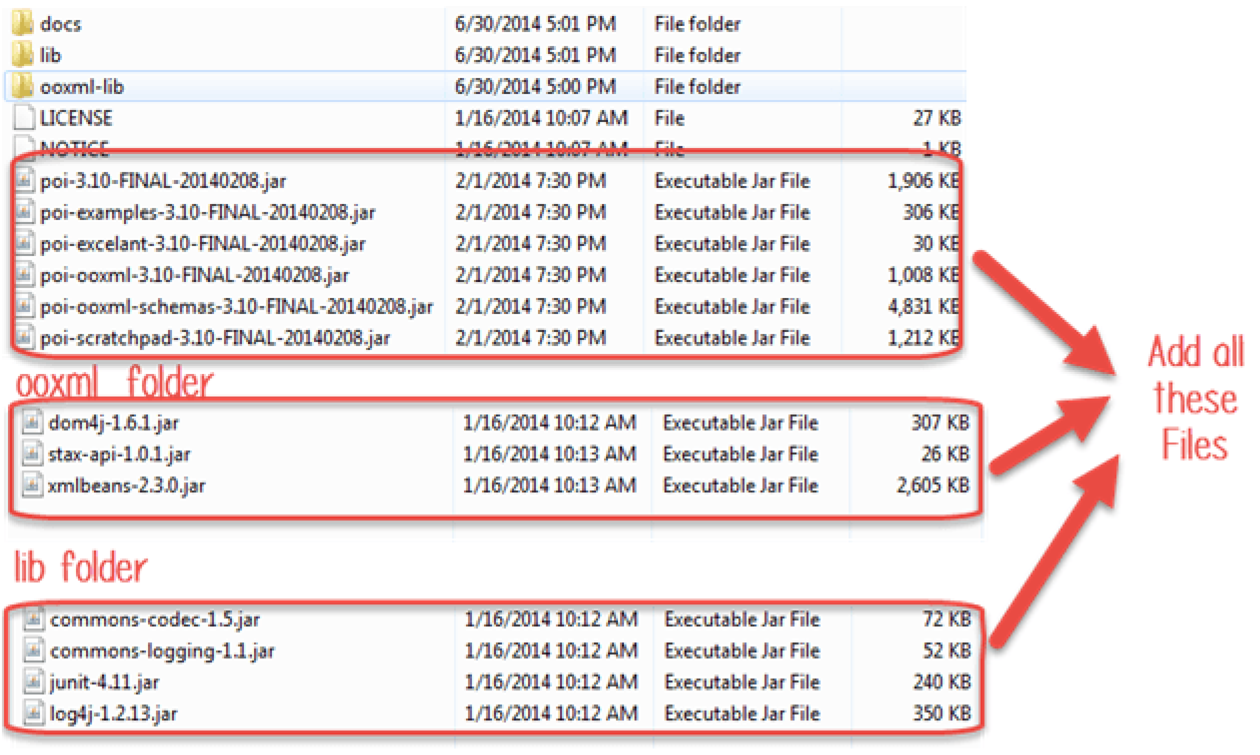
A full list of changes is available in the change log. People interested should als

The POI source release as well as the pre-built binary deployment packages are "org.apache.poi" and Version "3.10-FINAL".

Binary Distribution

- poi-bin-3.10-FINAL-20140208.tar.gz (16MB, signed)
  MD5 checksum: 818d1e99a2efe539ba49f622b554950c
  SHA1 checksum: 28b61005d780c00604fb8053fd46ab1b8b18392f
- poi-bin-3.10-FINAL-20140208.zip (23MB, signed)
  MD5 checksum: c304bc0e3169697d31c029f63450e963
  SHA1 checksum: 704f103bca893e3d4df5c2cc95d275b0cd5d0b33

When you download the zip file for this jar, you need to unzip it and add these all jars to the class path of your project.



docs	6/30/2014 5:01 PM	File folder	
lib	6/30/2014 5:01 PM	File folder	
ooxml-lib	6/30/2014 5:00 PM	File folder	
LICENSE	1/16/2014 10:07 AM	File	27 KB
NOTICE	1/16/2014 10:07 AM	File	1 KB
poi-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	1,906 KB
poi-examples-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	306 KB
poi-excelant-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	30 KB
poi-ooxml-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	1,008 KB
poi-ooxml-schemas-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	4,831 KB
poi-scratchpad-3.10-FINAL-20140208.jar	2/1/2014 7:30 PM	Executable Jar File	1,212 KB

ooxml folder

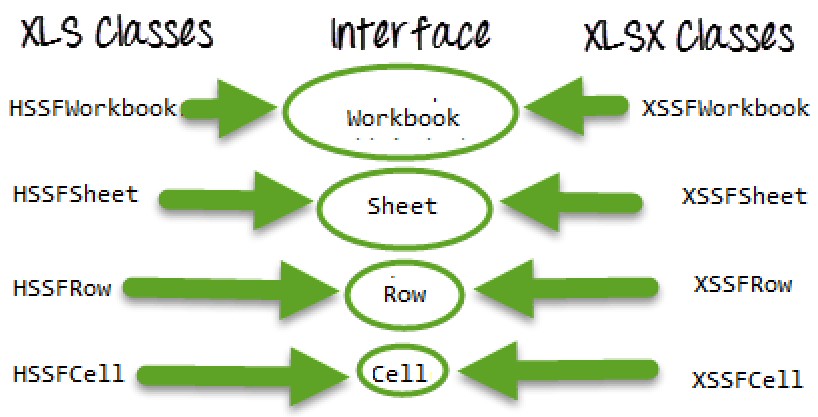
dom4j-1.6.1.jar	1/16/2014 10:12 AM	Executable Jar File	307 KB
stax-api-1.0.1.jar	1/16/2014 10:13 AM	Executable Jar File	26 KB
xmlbeans-2.3.0.jar	1/16/2014 10:13 AM	Executable Jar File	2,605 KB

lib folder

commons-codec-1.5.jar	1/16/2014 10:12 AM	Executable Jar File	72 KB
commons-logging-1.1.jar	1/16/2014 10:12 AM	Executable Jar File	52 KB
junit-4.11.jar	1/16/2014 10:12 AM	Executable Jar File	240 KB
log4j-1.2.13.jar	1/16/2014 10:12 AM	Executable Jar File	350 KB

Add all these Files

Classes and Interfaces in POI:



Following is a list of different Java Interfaces and classes in**POI**for reading**XLS**and**XLSX**file-

This will load the full workbook(complete excelsheet is known as the workbook)

- **Workbook**: XSSFWorkbook and HSSFWorkbook classes implement this interface.
- **XSSFWorkbook**: Is a class representation of XLSX file.
- **HSSFWorkbook**: Is a class representation of XLS file.

- **Sheet**: XSSFSheet and HSSFSheet classes implement this interface.
- **XSSFSheet**: Is a class representing a sheet in an XLSX file.
- **HSSFSheet**: Is a class representing a sheet in an XLS file.
-
- **Row**: XSSFRow and HSSFRow classes implement this interface.
- **XSSFRow**: Is a class representing a row in the sheet of XLSX file.
- **HSSFRow**: Is a class representing a row in the sheet of XLS file.

-
- **Cell**: XSSFCell and HSSFCell classes implement this interface.
- **XSSFCell**: Is a class representing a cell in a row of XLSX file.
- **HSSFCell:** Is a class representing a cell in a row of XLS file.

Read/Write operation-

For our example, we will consider below given Excel file format

Username1	Password1	username1@gmail.com
Username2	Password2	username2@gmail.com
Username3	Password3	username3@gmail.com
Username4	Password4	username4@gmail.com
Username5	Password5	username5@gmail.com
Username6	Password6	username6@gmail.com
Username7	Password7	username7@gmail.com
Username8	Password8	username8@gmail.com
Username9	Password9	username9@gmail.com

Writing thedata in excel:

```
package readexcel;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
importjava.io.IOException;

import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

publicclass WriteExcel {

	publicstaticvoid main(String[] args) throws Exception {
		// TODO Auto-generated method stub
		File file=new File("D:\\Saurabh\\Java\\POI ExcelData\\TestData.xlsx");
		FileInputStream fin=new FileInputStream(file);
		XSSFWorkbook wb=new XSSFWorkbook(fin);
		XSSFSheet sheet1=wb.getSheetAt(0);

		for(inti=0;i<sheet1.getLastRowNum();;i++) {
		String s="username"+(i+1)+"@gmail.com";
		sheet1.getRow(i).createCell(2).setCellValue(s);
		System.out.println("The "+s+" is written");
		}
		FileOutputStream fout=new FileOutputStream(file);
		wb.write(fout);
	}
}
```

Reading the data in excel

```
package readexcel;

import java.io.File;
import java.io.FileInputStream;
importjava.io.FileNotFoundException;
```

```
importorg.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

publicclass ReadFromExcel {

	publicstaticvoid main(String[] args) throws Exception {
		// TODO Auto-generated method stub
		File file = new File("D:\\Saurabh\\Java\\POI Excel 
Data\\TestData.xlsx");
		System.out.println(" " + file.exists());
		FileInputStream fis = new FileInputStream(file);
		XSSFWorkbook wb = new XSSFWorkbook(fis); // for xlsx file
		// HSSFWorkbook wb1=new HSSFWorkbook(fis); // for xls file
		XSSFSheet sheet1 = wb.getSheet("Sheet1");
		for (inti = 0; i<sheet1.getLastRowNum(); i++) {
			for (intj = 0; j< 2; j++) {
				String data = 
sheet1.getRow(i).getCell(j).getStringCellValue();
				System.out.print(data + "  ");
			}
			System.out.println();
		}
		System.out.println("Get Last Row number : " + sheet1.getLastRowNum());
	}
}
```