## Introduction to WebDriver

**Udemy tutorial reference:**

Section 1

**Selenium Introduction**

Section 2

**Setup and installation of required tools and plugins**

 After learning to create simple tests in Selenium IDE, we shall now create more powerful scripts using an advanced tool called **WebDriver**.

 What is WebDriver?

WebDriver is a web automation [framework](http://www.guru99.com/quick-test-professional-qtp-tutorial-34.html) that allows you to **execute your tests against different browsers**, not just Firefox (unlike Selenium IDE).

WebDriver also enables you to **use a programming language** in creating your test scripts (not possible in Selenium IDE).

* You can now use **conditional operations** like if-then-else or switch-case
* You can also perform **looping**like do-while.

Following programming languages are supported by WebDriver

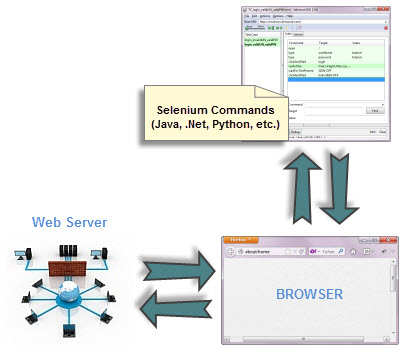
* Java, .Net, [PHP](http://www.guru99.com/php-tutorials.html), Python, [Perl](http://www.guru99.com/perl-tutorials.html), Ruby

**You do not have to know all of them. You just need to be knowledgeable in one.**However, in this tutorial, we will be using Java with Eclipse as our IDE.

### Architecture

**WebDriver's architecture is simpler than Selenium RC's**.

* It controls the browser from the OS level
* All you need are your programming language's IDE (which contains your Selenium commands) and a browser.

[](http://www.guru99.com/images/simplified_webdriver_architecture.jpg)

**WebDriver interacts with page elements in a more realistic way.** For example, if you have a disabled text box on a page you were testing, WebDriver really cannot enter any value in it just as how a real person cannot.

Selenium Core, just like other Javascript codes, can access disabled elements. In the past, Selenium testers complain that Selenium Core was able to enter values to a disabled text box in their tests. Differences in API

## Summary

* WebDriver is a tool for testing web applications **across different browsers** using different programming languages.
* You are now able to make powerful tests because WebDriver **allows you to use a programming language** of your choice in designing your tests.
* WebDriver is **faster than Selenium RC** because of its simpler architecture.
* WebDriver **directly talks to the browser** while Selenium RC needs the help of the RC Server in order to do so.
* WebDriver's API is more**concise** than Selenium RC's.
* WebDriver **can support HtmlUnit** while Selenium RC cannot.
* The only drawbacks of WebDriver are:
* It **cannot readily support new browsers**, but Selenium RC can.

It **does not have a built-in command** for automatic generation of test results

## Set up and installation of required

* Aside from a browser, you will need the following to start using WebDriver
* **Java Development Kit (JDK).** <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
* **Eclipse IDE** - <http://www.eclipse.org/downloads/>
* **Java Client Driver** - <http://seleniumhq.org/download/>
* **FireBug and FirePath –** Firefox plugins
* **Maven –** if needed
* When starting a WebDriver project in Eclipse, do not forget to import the Java Client Driver files onto your project. These files will constitute your Selenium Library.
* HTMLUnit and Firefox are the only browsers that you can automate without the use of a Driver Server.
* Each other browser has its own driver server.

## Web driver - Drivers

**Udemy tutorial reference:**

Section 6

**Selenium WebDriver -> Running Tests On Various Browsers**

**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.

|  |  |  |
| --- | --- | --- |
| **Browser** | **Name of Driver Server** | **Remarks** |
| HTMLUnit | (none) | WebDriver can drive HTMLUnit without the need of a driver server |
| Firefox | (none) | WebDriver can drive Firefox without the need of a driver server |
| 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. |

## How to use Locators in Selenium IDE

**Locators tell Selenium which GUI elements ( say Text Box, Buttons, Check Boxes etc) its needs to operate on**.  Identification of correct GUI elements is a prerequisite to create an automation script. But accurate identification of GUI elements is more difficult than it sounds. Sometimes, you end up working with incorrect GUI elements or no elements at all!  Hence, Selenium provides a number of Locators to precisely locate a GUI element

**Locating by CSS Selector**

**CSS Selectors are string patterns used to identify an element based on a combination of HTML tag, id, class, and attributes. Locating by CSS Selector is more complicated than the previous methods, but it is the most common locating strategy of advanced Selenium users because it can access even those elements that have no ID or name.**

When using this strategy, we always prefix the Target box with "css=" as will be shown on the following examples.

## Locating by XPath

XPath is the language used when locating XML (Extensible Markup Language) nodes. Since HTML can be thought of as an implementation of XML, we can also use XPath in locating HTML elements.

**Advantage:** It can access almost any element, even those without class, name, or id attributes

**Disadvantage:** It is the most complicated method of identifying elements because of too many different rules and considerations.

Fortunately, Firebug can automatically generate XPath locators. In the following example, we will access an image that cannot possibly be accessed through the methods we discussed earlier.

|  |  |
| --- | --- |
|  |  |

**Locating GUI Elements - Webdriver**

## Udemy tutorial reference:

Section 7

**Mutliple Ways To Locate Elements**

Section 8

**CSS Selectors - Advanced Locators**

Locating elements in WebDriver is done by using the "**findElement(By.locator())**" method.. Infact, it is recommended that you locate GUI elements and once successfully identified export the code to webdriver. Here is a sample code that locates an element by its id. Facebook is used as the Base URL.

|  |  |
| --- | --- |
|  | package mypackage;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  public class myclass {       public static void main(String[] args) {          WebDriver driver = new FirefoxDriver();          String baseUrl = "http://www.facebook.com";          String tagName = "";           driver.get(baseUrl);          tagName = driver.findElement(By.id("email")).getTagName();          System.out.println(tagName);          driver.close();          System.exit(0);      }  } |

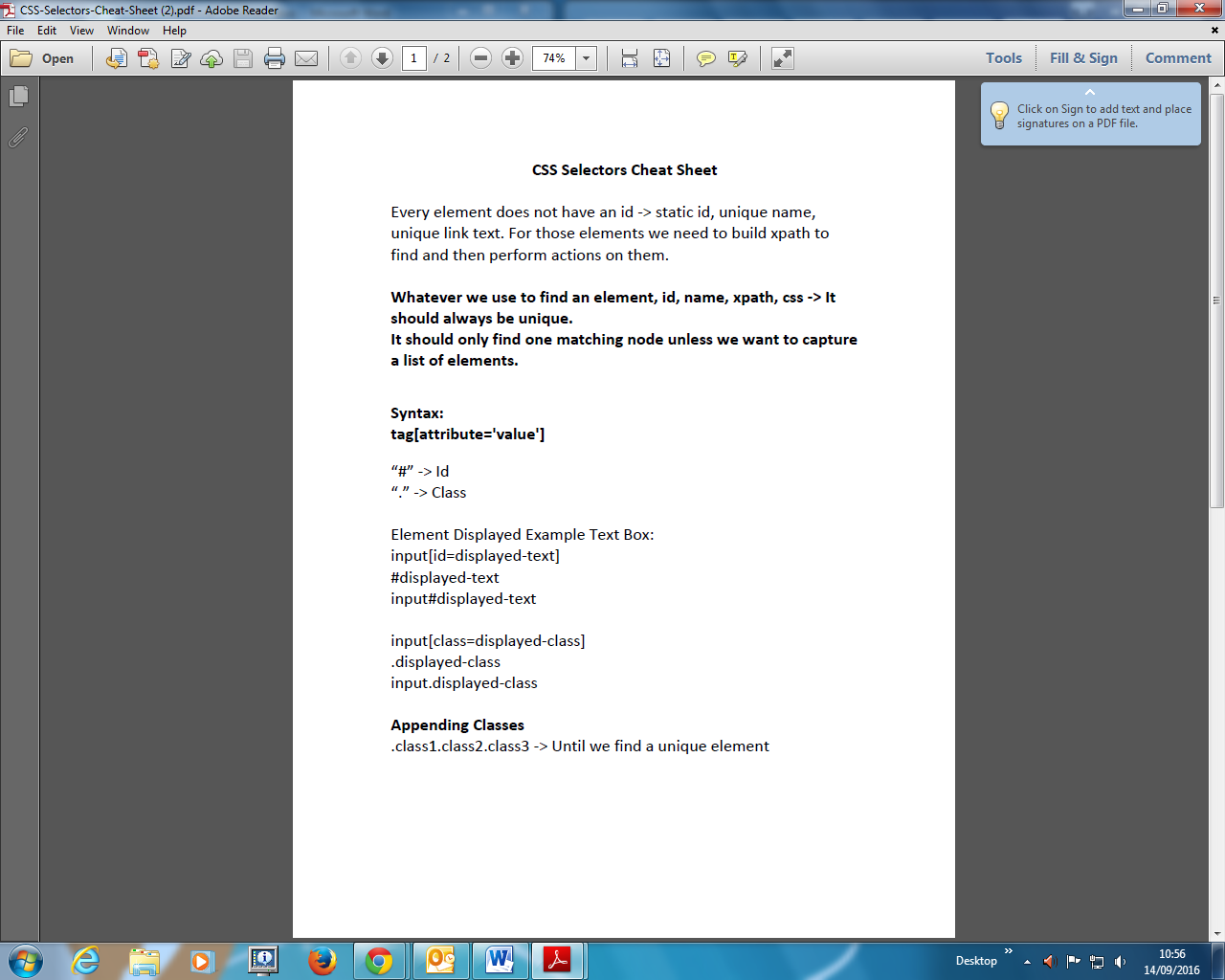
We used the **getTagName()** method to extract the tag name of that particular element whose id is "email". When run, this code should be able to correctly identify the tag name "input" and will print it out on Eclipse's Console window.

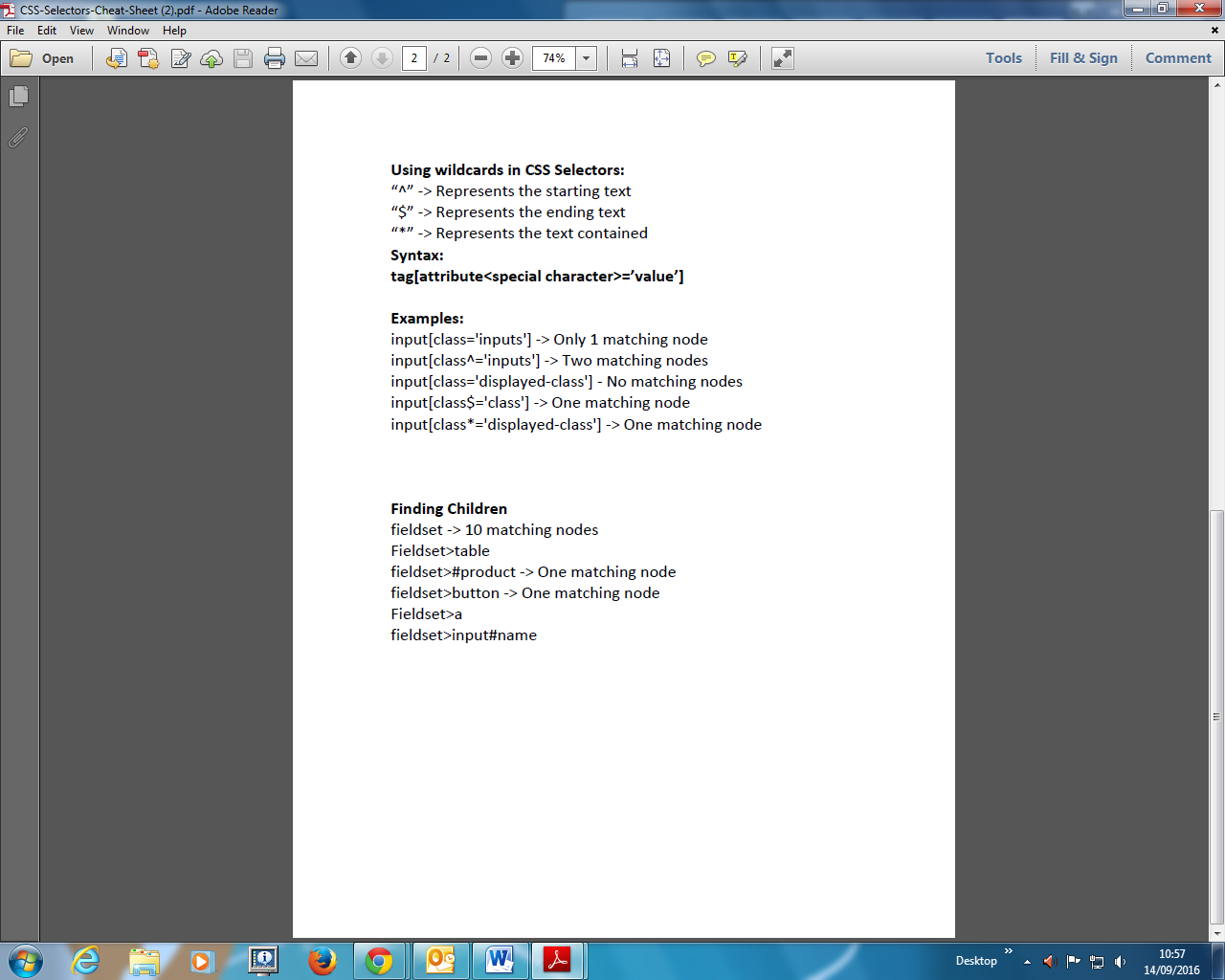
Summary for locating elements

|  |  |  |
| --- | --- | --- |
| **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/table/tbody/tr/td[2]/table/tbody/tr[2]/td[3]/form/table/tbody/tr[5]")) |

Section 8

**CSS Selectors - Advanced Locators**





Section 9

**Xpath - Advanced Locators**

## 

## 

## Common Commands

### Instantiating Web Elements

Instead of using the long "driver.findElement(By.locator())" syntax every time you will access a particular element, we can instantiate a WebElement object for it. The WebElement class is contained in the "org.openqa.selenium.\*" package.

[http://cdn.guru99.com/images/image016(2).png](http://www.guru99.com/images/image016(2).png)

### Clicking on an Element

Clicking is perhaps the most common way of interacting with web elements**. The click() method is used to simulate the clicking of any element.** The following example shows how click() was used to click on Mercury Tours'  "Sign-In" button.

[http://cdn.guru99.com/images/image017(2).png](http://www.guru99.com/images/image017(2).png)

Following things must be noted when using the click() method.

* **It does not take any parameter/argument.**
* The method **automatically waits for a new page to load** if applicable.
* The element to be clicked-on, **must be visible** (height and width must not be equal to zero).

### 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 |

**getCssValue Command**

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

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

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)”.

**getAttribute Command**

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

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

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

|  |  |
| --- | --- |
| 1  2 | WebElement element = driver.findElement(By.id("SubmitButton"));  String attValue = element.getAttribute("id"); //This will return "SubmitButton" |

**getSize Command**

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

**Command**- ***element.getSize();***

This returns the size of the element on the page.

|  |  |
| --- | --- |
| 1  2  3 | WebElement element = driver.findElement(By.id("SubmitButton"));  Dimension dimensions = element.getSize();  System.out.println(“Height :” + dimensions.height + ”Width : "+ dimensions.width); |

**getLocation Command**

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

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

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

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

### Navigate commands

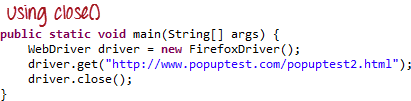
These commands allow you to refresh, go-into and switch back and forth between different web pages.

|  |  |
| --- | --- |
| **navigate().to()**  Sample usage: | * It automatically **opens a new browser window and fetches the page** that you specify inside its parentheses. * **It does exactly the same thing as the get() method.** |
| **navigate().refresh()**  Sample usage: | * Needs no parameters. * It **refreshes** the current page. |
| **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. |

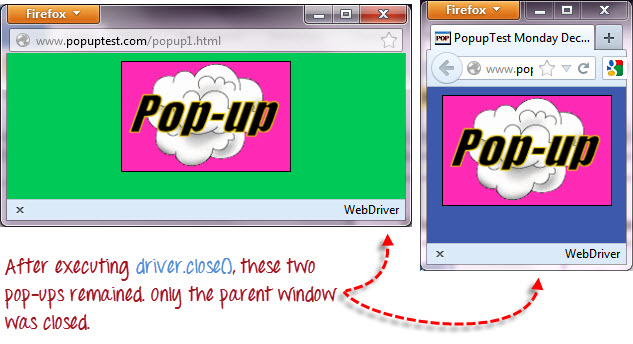
**Closing and Quitting Browser Windows**

|  |  |
| --- | --- |
| **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.** |

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.

[](http://www.guru99.com/images/image030(1).png)

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

[](http://www.guru99.com/images/image031.jpg)

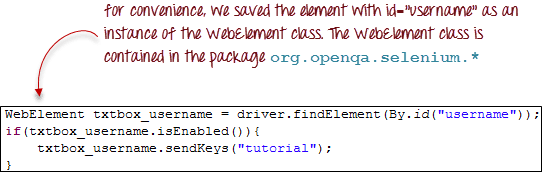
But if you use quit(), all windows will be closed - not just the parent one. Try running the code below and you will notice that the two pop-ups above will automatically be closed as well.

|  |  |
| --- | --- |
|  |  |

## 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.

[](http://www.guru99.com/images/image043(1).png)

**Example:**

package workingwithelements;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

// Please change the extension of the file to .java

// I added .txt because udemy currently does not allow to add .java files

public class ElementState {

WebDriver driver;

String baseUrl;

@Before

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl = "https://www.google.com";

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

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

driver.get(baseUrl);

}

@Test

public void test() {

WebElement e1 = driver.findElement(By.id("gs\_htif0"));

System.out.println("E1 is Enabled? " + e1.isEnabled());

WebElement e2 = driver.findElement(By.id("gs\_taif0"));

System.out.println("E2 is Enabled? " + e2.isEnabled());

WebElement e3 = driver.findElement(By.id("lst-ib"));

System.out.println("E3 is Enabled? " + e3.isEnabled());

e3.sendKeys("letskodeit");

}

@After

public void tearDown() throws Exception {

Thread.sleep(2000);

driver.quit();

}

}

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

[http://cdn.guru99.com/images/image044(1).png](http://www.guru99.com/images/image044(1).png)

**Example:**

package workingwithelements;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

// Please change the extension of the file to .java

// I added .txt because udemy currently does not allow to add .java files

public class ElementDisplayed {

WebDriver driver;

String baseUrl1;

String baseUrl2;

@Before

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl1 = "http://letskodeit.teachable.com/pages/practice";

baseUrl2 = "http://www.expedia.com";

driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

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

}

@Test

public void testLetsKodeIt() throws InterruptedException {

driver.get(baseUrl1);

WebElement textBox = driver.findElement(By.id("displayed-text"));

System.out.println("Text Box Displayed: " + textBox.isDisplayed());

Thread.sleep(3000);

WebElement hideButton = driver.findElement(By.id("hide-textbox"));

hideButton.click();

System.out.println("Clicked on hide button");

System.out.println("Text Box Displayed: " + textBox.isDisplayed());

Thread.sleep(3000);

WebElement showButton = driver.findElement(By.id("show-textbox"));

showButton.click();

System.out.println("Clicked on show button");

System.out.println("Text Box Displayed: " + textBox.isDisplayed());

}

@Test

public void testExpedia() throws InterruptedException {

driver.get(baseUrl2);

WebElement childDropdown = driver.findElement(By.id("package-1-age-select-1"));

System.out.println("Child Drpdown Displayed: " + childDropdown.isDisplayed());

}

@After

public void tearDown() throws Exception {

Thread.sleep(2000);

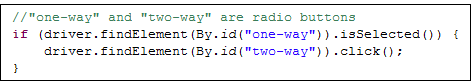
driver.quit();

}

}

Repatriation form- ecmo if Select y/n option check box and then a child check boxes will be displayed: Head CT this admission – yes then Known Hemorrhage, Ischaemia or other abnormality opened up

* **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.

[](http://www.guru99.com/images/image045.png)

**Example:**

package workingwithelements;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

// Please change the extension of the file to .java

// I added .txt because udemy currently does not allow to add .java files

public class RadioButtonsAndCheckBoxes {

WebDriver driver;

String baseUrl;

@Before

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl = "http://letskodeit.teachable.com/pages/practice";

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

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

driver.get(baseUrl);

}

@Test

public void test() throws InterruptedException {

WebElement bmwRadioBtn = driver.findElement(By.id("bmwradio"));

bmwRadioBtn.click();

Thread.sleep(2000);

WebElement benzRadioBtn = driver.findElement(By.id("benzradio"));

benzRadioBtn.click();

Thread.sleep(2000);

WebElement bmwCheckBox = driver.findElement(By.id("bmwcheck"));

bmwCheckBox.click();

Thread.sleep(2000);

WebElement benzCheckBox = driver.findElement(By.id("benzcheck"));

benzCheckBox.click();

System.out.println("BMW Radio Button is selected? " + bmwRadioBtn.isSelected());

System.out.println("Benz Radio Button is selected? " + benzRadioBtn.isSelected());

System.out.println("BMW Checkbox is selected? " + bmwCheckBox.isSelected());

System.out.println("Benz Checkbox is selected? " + benzCheckBox.isSelected());

}

@After

public void tearDown() throws Exception {

Thread.sleep(3000);

//driver.quit();

}

}

**Working with a list of Elements:**

package workingwithelements;

import java.util.List;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

// Please change the extension of the file to .java

// I added .txt because udemy currently does not allow to add .java files

public class WorkingWithElementsList {

private WebDriver driver;

private String baseUrl;

@Before

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl = "http://letskodeit.teachable.com/pages/practice";

// Maximize the browser's window

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

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

driver.get(baseUrl);

}

@Test

public void testListOfElements() throws Exception {

boolean isChecked = false;

List<WebElement> radioButtons = driver.findElements(

By.xpath("//input[contains(@type,'radio') and contains(@name,'cars')]"));

int size = radioButtons.size();

System.out.println("Size of the list: " + size);

for (int i=0; i<size; i++) {

isChecked = radioButtons.get(i).isSelected();

if (!isChecked) {

radioButtons.get(i).click();

Thread.sleep(2000);

}

}

}

@After

public void tearDown() throws Exception {

Thread.sleep(2000);

driver.quit();

}

}

## 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.

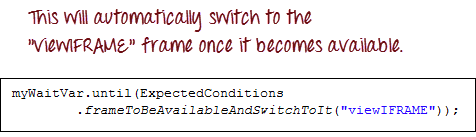
* **alertIsPresent()**- waits until an alert box is displayed.

[http://cdn.guru99.com/images/image046(1).png](http://www.guru99.com/images/image046(1).png)

* **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".

[http://cdn.guru99.com/images/image047(1).png](http://www.guru99.com/images/image047(1).png)

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

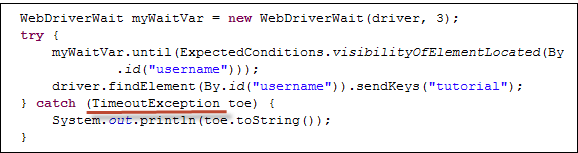
[](http://www.guru99.com/images/image048(1).png)

## 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".

[](http://www.guru99.com/images/image050(2).png)

## Summary

* To start using the WebDriver API, you must import at least these two packages.
* **org.openqa.selenium.\***
* **org.openqa.selenium.firefox.FirefoxDriver**
* The **get()** method is the equivalent of Selenium IDE's "open" command.
* Locating elements in WebDriver is done by using the **findElementBy()** method.
* The following are the available options for locating elements in WebDriver:
* By.**className**
* By.**cssSelector**
* By.**id**
* By.**linkText**
* By.**name**
* By.**partialLinkText**
* By.**tagName**
* By.**xpath**
* TheBy.cssSelector() **does not** support the **"contains"** feature.
* You can instantiate an element using the **WebElement** class.
* Clicking on an element is done by using the **click()** method.
* WebDriver provides these useful **get commands**:
* get()
* getTitle()
* getPageSource()
* getCurrentUrl()
* getText()
* WebDriver provides these useful **navigation commands**
* navigate().forward()
* navigate().back()
* navigate().to()
* navigate().refresh()
* The close() and quit() methods are used to close browser windows. **Close()** is used to close a single window; while **quit()** is used to close all windows associated to the parent window that the WebDriver object was controlling.
* You can use the **isEnabled(), isDisplayed(), isSelected(),** and a combination of  **WebDriverWait** and **ExpectedConditions** methods when verifying the state of an element. However, they do not verify if the element exists.
* When isEnabled(), isDisplayed(),or isSelected() was called while the element was not existing, WebDriver will throw a **NoSuchElementException**.
* When WebDriverWait and ExpectedConditions methods were called while the element was not existing, WebDriver would throw a **TimeoutException**.
* Thread.sleep(millsecs)throws InterruptedException

## Accessing Form Elements

**Udemy tutorial reference:**

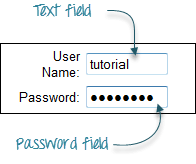
**Working with web Elements**

Section 11

**Input Box**

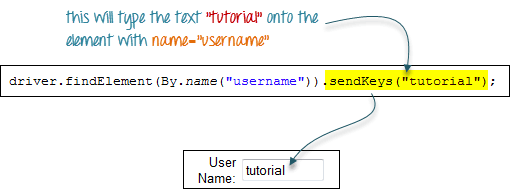
Input boxes refer to either of these two types:

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.

[](http://www.guru99.com/images/image002(1).png)

### Entering Values in Input Boxes

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

[](http://www.guru99.com/images/image003(2).png)

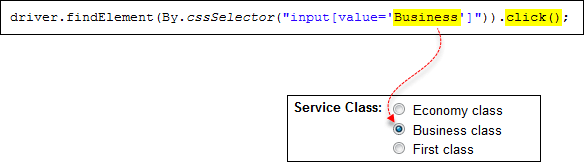
### Deleting Values in Input Boxes

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.

[http://cdn.guru99.com/images/image004(3).png](http://www.guru99.com/images/image004(3).png)

### Radio Button

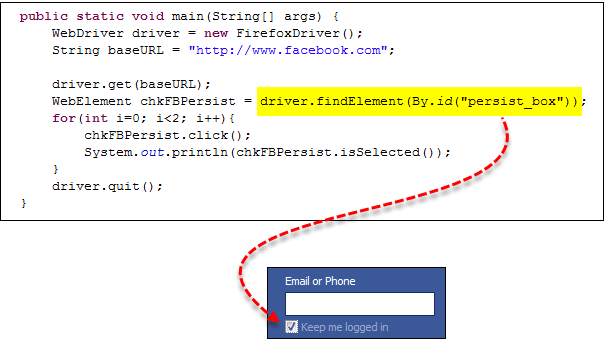
Toggling a radio button on is done using the **click()**method.

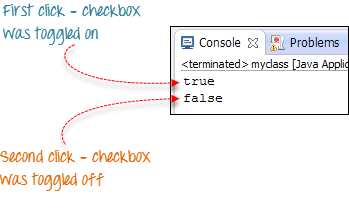
[](http://www.guru99.com/images/image005(3).png)

### 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.

[](http://www.guru99.com/images/image006(3).png)

**example**[](http://www.guru99.com/images/image007(3).png)

 package workingwithelements;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

public class RadioButtonsAndCheckBoxes {

WebDriver driver;

String baseUrl;

@Before

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl = "http://letskodeit.teachable.com/pages/practice";

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

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

driver.get(baseUrl);

}

@Test

public void test() throws InterruptedException {

WebElement bmwRadioBtn = driver.findElement(By.id("bmwradio"));

bmwRadioBtn.click();

Thread.sleep(2000);

WebElement benzRadioBtn = driver.findElement(By.id("benzradio"));

benzRadioBtn.click();

Thread.sleep(2000);

WebElement bmwCheckBox = driver.findElement(By.id("bmwcheck"));

bmwCheckBox.click();

Thread.sleep(2000);

WebElement benzCheckBox = driver.findElement(By.id("benzcheck"));

benzCheckBox.click();

System.out.println("BMW Radio Button is selected? " + bmwRadioBtn.isSelected());

System.out.println("Benz Radio Button is selected? " + benzRadioBtn.isSelected());

System.out.println("BMW Checkbox is selected? " + bmwCheckBox.isSelected());

System.out.println("Benz Checkbox is selected? " + benzCheckBox.isSelected());

}

@After

public void tearDown() throws Exception {

Thread.sleep(3000);

//driver.quit();

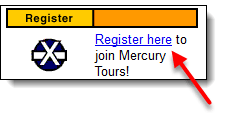
}

}

### Links

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

Consider the below link found in Mercury Tours' homepage.

[](http://www.guru99.com/images/image008(4).png)

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.

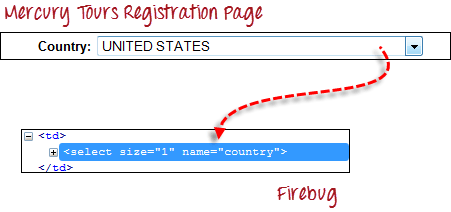
[http://cdn.guru99.com/images/image009(4).png](http://www.guru99.com/images/image009(4).png)

[http://cdn.guru99.com/images/image010(3).png](http://www.guru99.com/images/image010(3).png)

**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

[](http://www.guru99.com/images/image011(3).png)

**Step 1**

Import the "Select" package.

[http://cdn.guru99.com/images/image012(1).png](http://www.guru99.com/images/image012(1).png)

**Step 2**

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

[http://cdn.guru99.com/images/image013(3).png](http://www.guru99.com/images/image013(3).png)

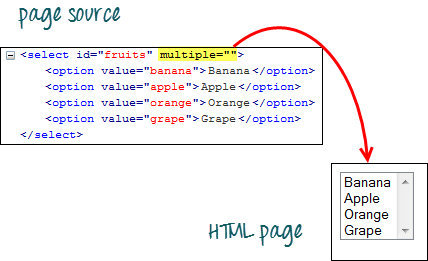
**Step 3**

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

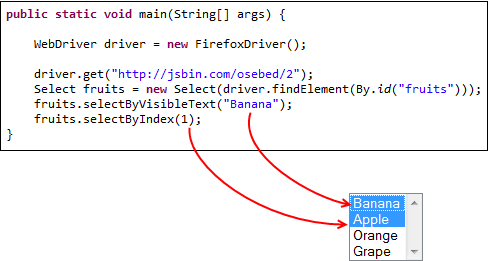
[http://cdn.guru99.com/images/image014(3).png](http://www.guru99.com/images/image014(3).png)

### Selecting Items in a Multiple SELECT element

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.

[](http://www.guru99.com/images/image015(3).png)

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

[](http://www.guru99.com/images/image016(3).png)

### Select Methods

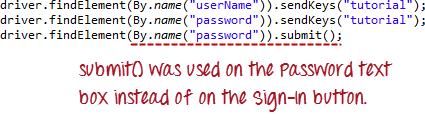
The following are the most common methods used on drop-down elements.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **selectByVisibleText()**and  **deselectByVisibleText()**  Example:  [http://cdn.guru99.com/images/image017(3).png](http://www.guru99.com/images/image017(3).png) | * Selects/deselects the option that displays the text matching the parameter. * **Parameter**: The exactly displayed text of a particular option |
| **selectByValue()** and  **deselectByValue()**  Example:  [http://cdn.guru99.com/images/image018(3).png](http://www.guru99.com/images/image018(3).png) | * 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.   [http://cdn.guru99.com/images/image019(2).png](http://www.guru99.com/images/image019(2).png) |
| **selectByIndex()** and  **deselectByIndex()**  Example:  *[http://cdn.guru99.com/images/image020(2).png](http://www.guru99.com/images/image020(2).png)* | * Selects/deselects the option at the given index. * **Parameter**: the index of the option to be selected. |
| **isMultiple()**  Example:  *[http://cdn.guru99.com/images/image021(2).png](http://www.guru99.com/images/image021(2).png)* | * Returns TRUE if the drop-down element allows multiple selections at a time; FALSE if otherwise. * **Needs parameters needed** |
| **deselectAll()**  Example:  *[http://cdn.guru99.com/images/image022(2).png](http://www.guru99.com/images/image022(2).png)* | * Clears all selected entries. This is only valid when the drop-down element supports multiple selections. * **No parameters needed** |

### 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.

[](http://www.guru99.com/images/image023(1).png)

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

## Summary

|  |  |  |
| --- | --- | --- |
| **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. |
| **Drop-Down Box** | selectByVisibleText()/  deselectByVisibleText() | selects/deselects an option by its displayed text |
| selectByValue()/  deselectByValue() | selects/deselects an option by the value of its "value" attribute |
| selectByIndex()/  deselectByIndex() | selects/deselects an option by its index |
| isMultiple() | returns TRUE if the drop-down element allows multiple selection at a time; FALSE if otherwise |
| deselectAll() | deselects all previously selected options |
| **Submit Button** | submit() |  |

* WebDriver allows selection of more than one option in a multiple SELECT element.
* To control drop-down boxes, you must first import the org.openqa.selenium.support.ui.Select package and then create a Select instance.
* 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.

**All links**

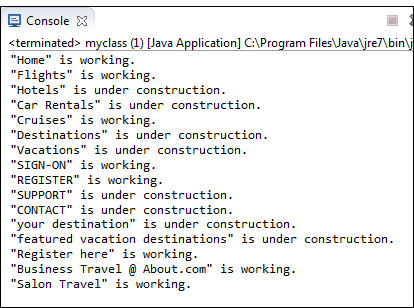
One of the common procedures in web [testing](http://www.guru99.com/software-testing.html) is to test if all the links present within the page are working. This can be conveniently done using a combination of the **Java for-each loop** and the **By.tagName("a")** method. The WebDriver code below checks each link from the Mercury Tours homepage to determine those that are working and those that are still under construction.

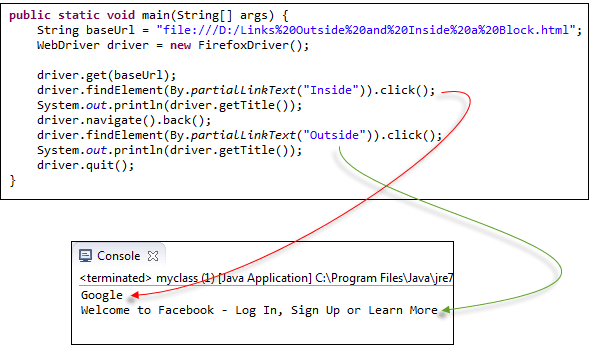
**Example test Ecmo site – burger menu options**

### //div[@id='pathway']//a[starts-with(@href, 'htt') or starts-with(@href, 'mdt')] ECMO burgers menu

|  |  |
| --- | --- |
|  | package practice\_webdriver;  import java.util.List;  import java.util.concurrent.TimeUnit;  import org.openqa.selenium.\*;  import org.openqa.selenium.firefox.FirefoxDriver;  import org.openqa.selenium.support.ui.ExpectedConditions;  import org.openqa.selenium.support.ui.WebDriverWait;    public class AllLinks {       public static void main(String[] args) {          String baseUrl = "http://newtours.demoaut.com/";  //url = http://ecmo.test.healthpathways.co.uk/patient-dashboard.php?pid=00002A          WebDriver driver = new FirefoxDriver();          String underConsTitle = "Under Construction: Mercury Tours";          driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);            driver.get(baseUrl);          List<WebElement> linkElements = driver.findElements(By.tagName("a"));  //By.xPath(“//div[@id='pathway']//a[starts-with(@href, 'htt') or starts-with(@href, 'mdt')]”)          String[] linkTexts = new String[linkElements.size()];          int i = 0;            //extract the link texts of each link element          for (WebElement e : linkElements) {              linkTexts[i] = e.getText();              i++;          }           //test each link          for (String t : linkTexts) {              driver.findElement(By.linkText(t)).click();  // move to element  WebElement heading= driver.findElement(By.xPath("//div[@id='content']/h1"));  Actions builder = new Actions(driver);          Action mouseOverHeading = builder                  .moveToElement(heading)                  .build();            String textHeading = heading.getText();          System.out.println("heading text is: " + textHeading);          mouseOverHeading.perform();          if(textHeading.equals(t){linked to that page  }else{not linked}              if (driver.getTitle().equals(underConsTitle)) {                  System.out.println("\"" + t + "\""                          + " is under construction.");              } else {                  System.out.println("\"" + t + "\""                          + " is working.");              }              driver.navigate().back();          }          driver.quit();      }  } |

The output should be similar to the one indicated below.

[](http://www.guru99.com/images/image013(1).png)

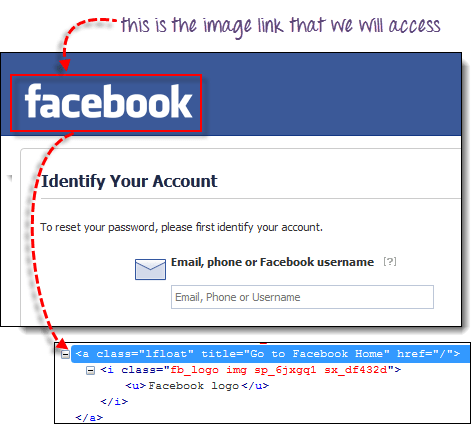
[](http://www.guru99.com/images/image016(1).png)

The output above confirms that both links were accessed successfully because their respective page titles were retrieved correctly.

### Accessing Image Links

Image links are images that act as references to other sites or sections within the same page. Since they are images, we cannot use the By.linkText() and By.partialLinkText() methods because image links basically have no link texts at all. In this case, we should resort to using either By.cssSelector or By.xpath. The first method is more preferred because of its simplicity.

In the example below, we will access the "Facebook" logo on the upper left portion of Facebook's Password Recovery page.

[](http://www.guru99.com/images/image017(1).png)

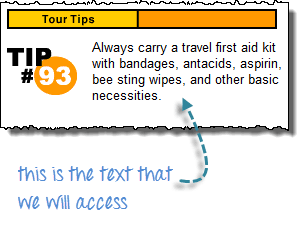
We will use By.cssSelector and the element's "title" attribute to access the image link. And then we will verify if we are taken to Facebook's homepage.

## Reading a Table

There are times when we need to access elements (usually texts) that are within HTML tables. However, it is very seldom for a web designer to provide an id or name attribute to a certain cell in the table. Therefore, we cannot use the usual methods such as "By.id()", "By.name()", or "By.cssSelector()". In this case, the most reliable option is to access them using the "By.xpath()" method.

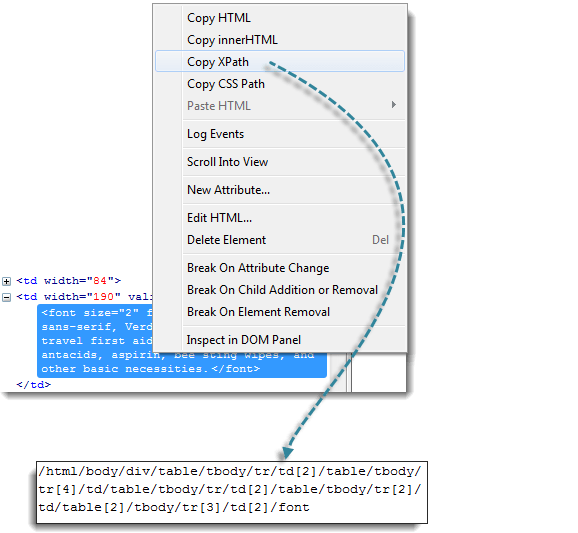
If the number or attribute of an element is extremely difficult or impossible to obtain, the quickest way to generate the XPath code is thru Firebug.

Consider the example below from Mercury Tours homepage.

[](http://www.guru99.com/images/image040.png)

**Step 1**

Use Firebug to obtain the XPath code.

[](http://www.guru99.com/images/image041.png)

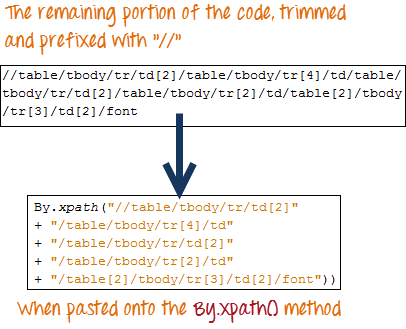
**Step 2**

Look for the first "table" parent element and delete everything to the left of it.

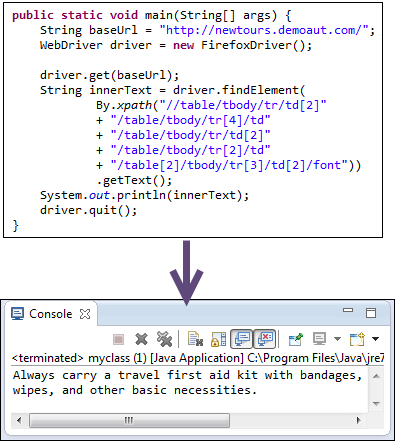
[](http://www.guru99.com/images/image042.png)

**Step 3**

Prefix the remaining portion of the code with double forward slash "//" and copy it over to your WebDriver code.

[](http://www.guru99.com/images/image043.png)

The WebDriver code below will be able to successfully retrieve the inner text of the element we are accessing.

[](http://www.guru99.com/images/image044.png)

## Summary

* Accessing links using their exact match is done using By.linkText() method.
* Accessing links using their partial match is done using By.partialLinkText() method.
* If there are multiple matches, By.linkText() and By.partialLinkText() will only select the first match.
* Pattern matching using By.linkText() and By.partialLinkText() is case-sensitive.
* The By.tagName("a") method is used to fetch all links within a page.
* Links can be accessed by the By.linkText() and By.partialLinkText() whether they are inside or outside block-level elements.
* Accessing image links are done using By.cssSelector() and By.xpath() methods.
* By.xpath() is commonly used to access table elements.

## Switching Between Windows

**Udemy tutorial reference:**

**Switch windows and Iframes**

Section 13

package switchto;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

// Please change the extension of the file to .java before using

// I have to change it to .txt because udemy does not allow uploading .java files

public class SwitchWindow {

private WebDriver driver;

private String baseUrl;

@Before

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl = "http://letskodeit.teachable.com/pages/practice";

// Maximize the browser's window

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

driver.manage().timeouts().implicitlyWait(2, TimeUnit.SECONDS);

driver.get(baseUrl);

}

@Test

public void test() throws InterruptedException {

// Get the handle

String parentHandle = driver.getWindowHandle();

System.out.println("Parent Handle: " + parentHandle);

// Find Open Window button

WebElement openWindow = driver.findElement(By.id("openwindow"));

openWindow.click();

// Get all handles

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

// Switching between handles

for (String handle: handles) {

System.out.println(handle);

if (!handle.equals(parentHandle)) {

driver.switchTo().window(handle);

Thread.sleep(2000);

WebElement searchBox = driver.findElement(By.id("search-courses"));

searchBox.sendKeys("python");

Thread.sleep(2000);

driver.close();

break;}

}

// Switch back to the parent window

driver.switchTo().window(parentHandle);

driver.findElement(By.id("name")).sendKeys("Test Successful");

}

@After

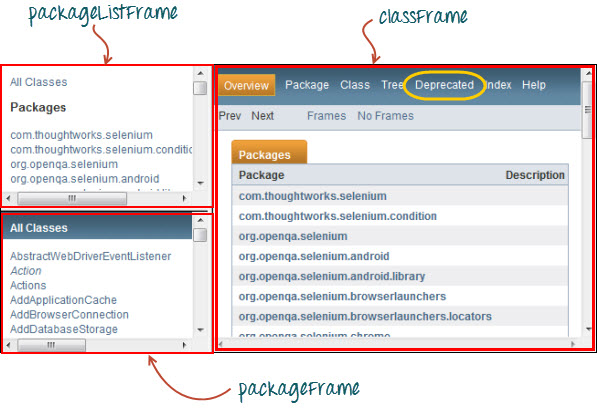
public void tearDown() throws Exception {

Thread.sleep(2000);

driver.quit();}}

**Switching Between Frames**

To access GUI elements in a Frame, we should first direct WebDriver to focus on the frame or pop-up window first before we can access elements within them. Let us take, for example, the web page <http://selenium.googlecode.com/svn/trunk/docs/api/java/index.html>

[](http://www.guru99.com/images/image033.jpg)

This page has 3 frames whose "name" attributes are indicated above. We wish to access the "Deprecated" link encircled above in yellow. In order to do that, we must first instruct WebDriver to switch to the "classFrame" frame using the **"switchTo().frame()"** method. We will use the name attribute of the frame as the parameter for the "frame()" part.

|  |  |
| --- | --- |
|  | package mypackage;   import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;   public class myclass {       public static void main(String[] args) {          WebDriver driver = new FirefoxDriver();           driver.get("http://selenium.googlecode.com/svn/trunk/docs/api/java/index.html");          driver.switchTo().frame("classFrame");          driver.findElement(By.linkText("Deprecated")).click();          driver.quit();              }  } |

## Switching Between Pop-up Windows

WebDriver allows pop-up windows like alerts to be displayed, unlike in Selenium IDE. To access the elements within the alert (such as the message it contains), we must use the **"switchTo().alert()"** method. In the code below, we will use this method to access the alert box and then retrieve its message using the **"getText()"** method, and then automatically close the alert box using the **"switchTo().alert().accept()"** method.

First,  head over to <http://jsbin.com/usidix/1> and manually click the "Go!" button there and see for yourself the message text.

[](http://www.guru99.com/images/image037(1).png)

Lets see the WebDriver code to do this-

|  |  |
| --- | --- |
|  | package mypackage;   import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;   public class myclass {       public static void main(String[] args) {          WebDriver driver = new FirefoxDriver();          String alertMessage = "";           driver.get("http://jsbin.com/usidix/1");          driver.findElement(By.cssSelector("input[value=\"Go!\"]")).click();          alertMessage = driver.switchTo().alert().getText();          driver.switchTo().alert().accept();                  System.out.println(alertMessage);          driver.quit();        }  } |

## Waits

There are two kinds of waits.

1. Implicit wait - used to set the default waiting time throughout the program
2. Explicit wait - used to set the waiting time for a particular instance only

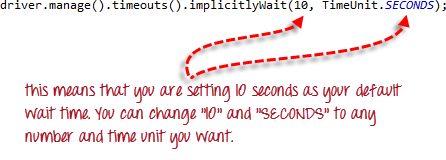
### Implicit Wait

* It is simpler to code than Explicit Waits.
* It is usually declared in the instantiation part of the code.
* You will only need one additional package to import.

To start using an implicit wait, you would have to import this package into your code.

[http://cdn.guru99.com/images/image038(1).png](http://www.guru99.com/images/image038(1).png)

Then on the instantiation part of your code, add this.

[](http://www.guru99.com/images/image039(1).png)

### Explicit Wait

**Explicit waits are done using the WebDriverWait and ExpectedCondition classes**. For the following example, we shall wait up to 10 seconds for an element whose id is "username" to become visible before proceeding to the next command. Here are the steps.

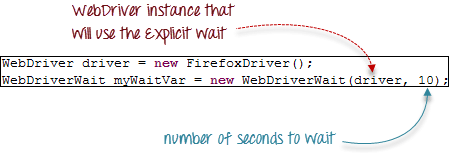
**Step 1**

Import these two packages:

**[http://cdn.guru99.com/images/image040(1).png](http://www.guru99.com/images/image040(1).png)**

**Step 2**

Declare a WebDriverWait variable. In this example, we will use "myWaitVar" as the name of the variable.

[](http://www.guru99.com/images/image041(1).png)

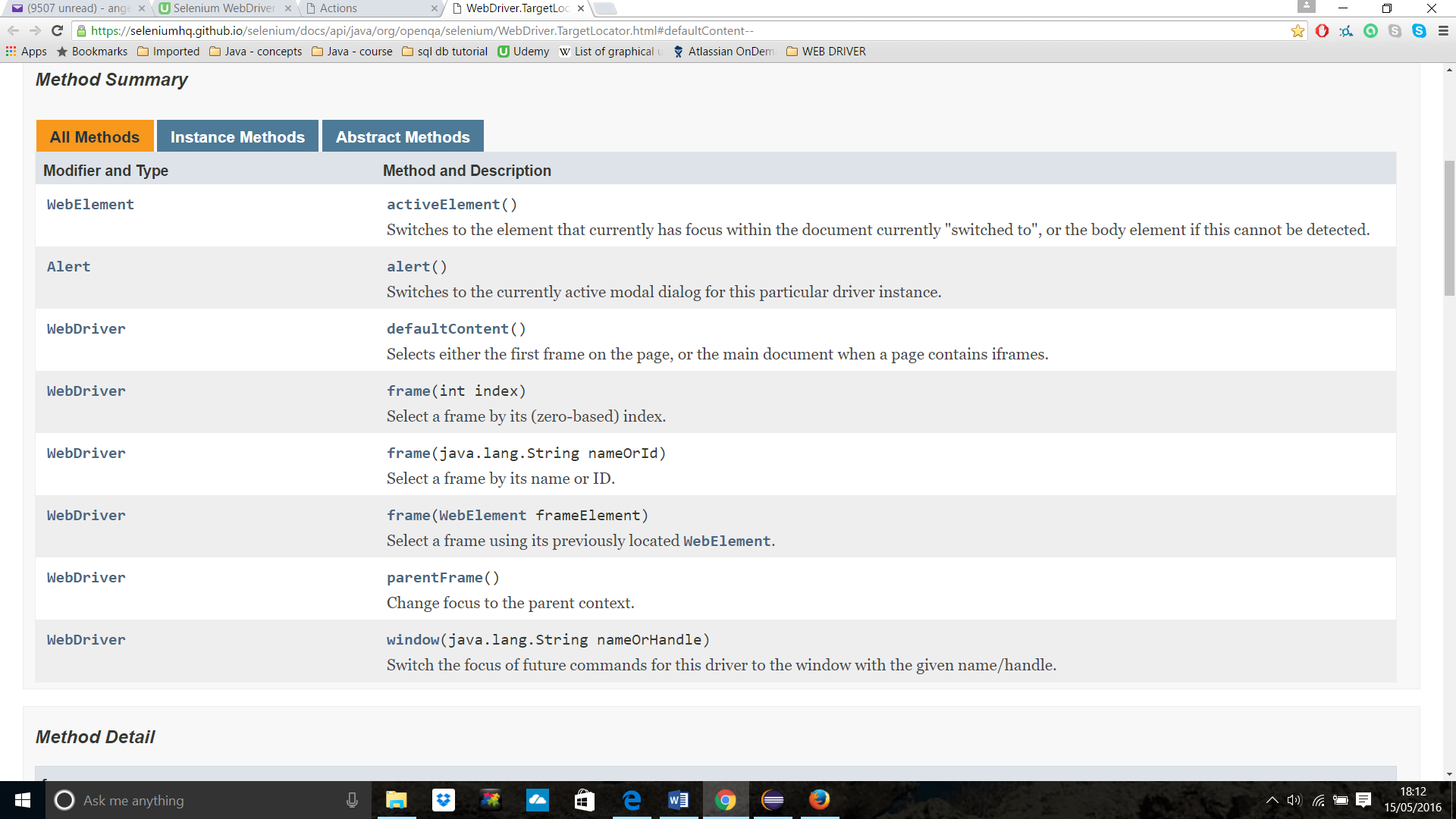
**Step 3**

Use myWaitVar with ExpectedConditions on portions where you need the explicit wait to occur. In this case, we will use explicit wait on the "username" (Mercury Tours HomePage) input before we type the text "tutorial" onto it.

[http://cdn.guru99.com/images/image042(1).png](http://www.guru99.com/images/image042(1).png)

**Summary**

* The **switchTo().frame()** and **switchTo().alert()** methods are used to direct WebDriver's focus onto a frame or alert, respectively.
* **Implicit waits** are used to set the waiting time throughout the program, while **explicit waits** are used only on specific portions.



## Handling Keyboard & Mouse Events

**Udemy tutorial reference:**

**Working with Actions Class**

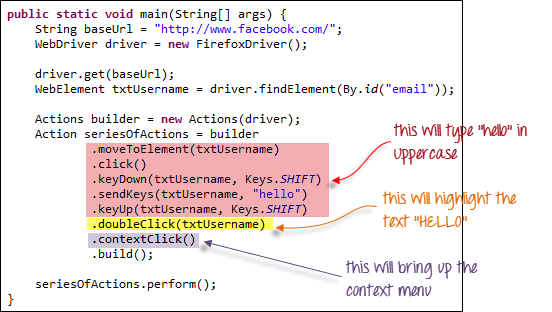
Section 14

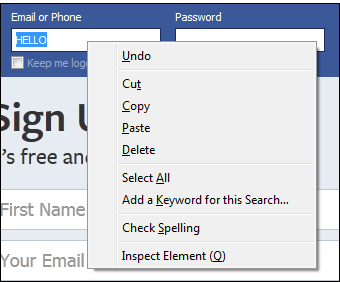
**Handling special keyboard and mouse events are done using the Advanced User Interactions API**. It contains the **Actions** and the **Action** classes that are needed when executing these events. The following are the most commonly used keyboard and mouse events provided by the Actions class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **clickAndHold()** | Clicks (without releasing) at the current mouse location. |
| **contextClick()** | Performs a context-click at the current mouse location. |
| **doubleClick()** | Performs a double-click at the current mouse location. |
| **dragAndDrop(source, target)** | Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.  **Parameters:**  *source*- element to emulate button down at.  *target*- element to move to and release the mouse at. |
| **dragAndDropBy(source, x-offset, y-offset)** | Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.  **Parameters**:  *source*- element to emulate button down at.  *xOffset*- horizontal move offset.  *yOffset*- vertical move offset. |
| **keyDown(modifier\_key)** | Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed.  **Parameters**:  modifier*\_key* - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **keyUp(modifier \_key)** | Performs a key release.  **Parameters**:  modifier*\_key* - any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **moveByOffset(x-offset, y-offset)** | Moves the mouse from its current position (or 0,0) by the given offset.  **Parameters**:  *x-offset*- horizontal offset. A negative value means moving the mouse left.  *y-offset*- vertical offset. A negative value means moving the mouse up. |
| **moveToElement(toElement)** | Moves the mouse to the middle of the element. **Parameters**:  *toElement*- element to move to. |
| **release()** | Releases the depressed left mouse button at the current mouse location |
| **sendKeys(onElement, charsequence)** | Sends a series of keystrokes onto the element. **Parameters**:  *onElement*- element that will receive the keystrokes, usually a text field  *charsequence*- any string value representing the sequence of keystrokes to be sent |

### Building a Series of Multiple Actions

**You can build a series of actions using the Action and Actions classes**. Just remember to close the series with the build() method. Consider the sample code below.

[](http://cdn.guru99.com/images/image053.png)

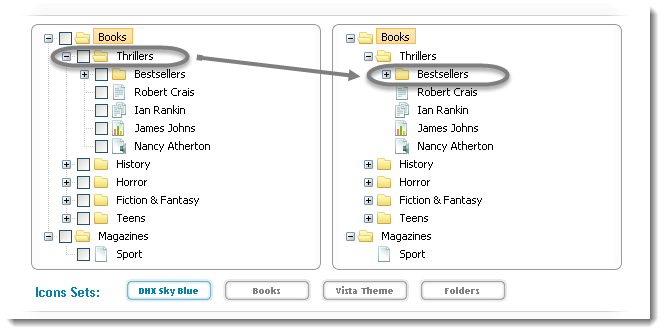
[](http://cdn.guru99.com/images/image054.png)

# Drag And Drop in Selenium Webdriver

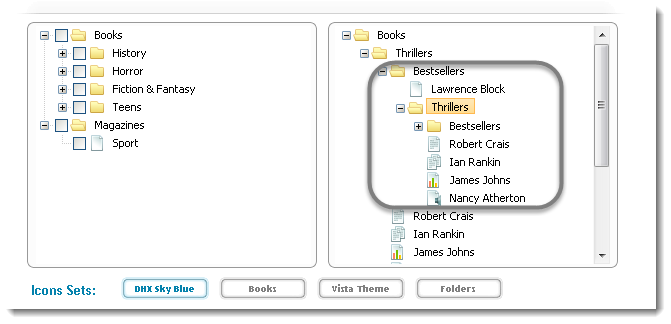
Let’s say we have a web application where we need to drag an item from one location to another location. These kinds of complex actions are not available in basic element properties. Automating rich web application is interesting, as it involves advanced user interactions. Thankfully Selenium has provided a separate “Actions” class to handle these advanced user interactions.

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

It was really very hard for me to search for a website where I can try ‘Drag n drop’ feature of WebDriver.

**Example 1:** In this example we will drag the **Thriller** folder from the left table on to the **Bestsellers** folder of the right side table. 

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

Now if you carefully look at the folder, you will notice that the Thrillers folder is been moved to Bestsellers folder. The new folder structure will look like this:

# Mouse Hover action in Selenium Webdriver

There will be situations where it is required to click on the item of the drop down menu .

**Other Coding Examples:**

**Mouse Hover Actions**

package tutorialselenium;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Actions;

public class MouseHoverActions {

private WebDriver driver;

private String baseUrl;

@Before

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl = "http://www.dhtmlx.com";

// Maximize the browser's window

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

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

}

@Test()

public void testMouseHoverActions() throws Exception {

driver.get(baseUrl);

Thread.sleep(1000);

// I have modified this xpath because the website has change since then

// All the websites are public they might change, which is actually a good thing because then

// you can practice on the changed stuff and then send me the code if you have issues completing the code

// And as always I will be more than happy to help you

WebElement mainElement = driver.findElement(By.xpath("//li[@xtitle='products']//a"));

Actions action = new Actions(driver);

action.moveToElement(mainElement).perform();

Thread.sleep(2000);

WebElement subElement = driver.findElement(By.linkText("DHTMLX Suite"));

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

//action.moveToElement(mainElement).moveToElement(subElement).click().build().perform();

}

@After

public void tearDown() throws Exception {

//driver.quit();

}

}

**Drag and Drop Actions**

package tutorialselenium;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Actions;

public class DragAndDropActions {

private WebDriver driver;

private String baseUrl;

@Before

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl = "http://www.dhtmlx.com/docs/products/dhtmlxTree/";

// Maximize the browser's window

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

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

}

@Test

public void testDragAndDrop() throws Exception {

driver.get(baseUrl);

Thread.sleep(1000);

WebElement fromElement1 = driver.findElement(By.xpath("//div[@id='treebox1']//span[text()='James Johns']"));

WebElement toElement1 = driver.findElement(By.xpath("//div[@id='treebox2']//span[text()='Bestsellers']"));

Actions action = new Actions(driver);

// Click and hold, move to element, release, build and perform

action.clickAndHold(fromElement1).perform();

Thread.sleep(1000);

action.moveToElement(toElement1).perform();

Thread.sleep(1000);

action.release(toElement1).perform();

//

// WebElement fromElement2 = driver.findElement(By.xpath("//div[@id='treebox1']//span[text()='Ian Rankin']"));

// WebElement toElement2 = driver.findElement(By.xpath("//div[@id='treebox1']//span[text()='Bestsellers']"));

//

// // Drag and drop

// action.dragAndDrop(fromElement2, toElement2).build().perform();

}

@After

public void tearDown() throws Exception {

// Thread.sleep(3000);

driver.quit();

}

}

# How to verify tooltip text with selenium webdriver using java

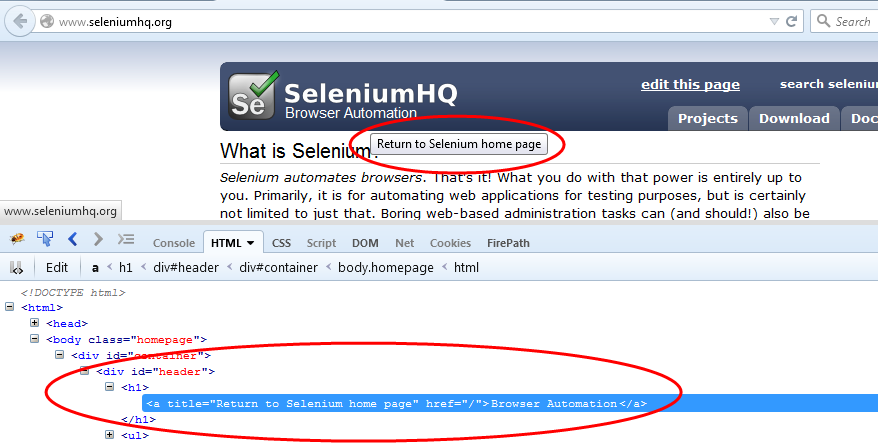
When user mouse hovers an any item (Button/link/field etc), without clicking it, and a tool tip may appear with information about the item being hovered. And Some times it may require us to check for the tooltip text.

There are different ways in showing tool tip to the user. We will look into two such different examples, one is with simple HTML and other example with Jquery ToolTip.

**Example#1:**

Let us take selenium official website to verify the tooltip in the first case.

Here when we mouse hover on the header, it has anchor tag with title attribute which is displayed as tooltip. Below is the screen shot:



In this case it is very simple to get tooltip text by using [selenium getAttribute() method.](http://seleniumeasy.com/selenium-tutorials/how-to-get-attribute-values-using-webdriver)

**Syntax:**

WebElement element = driver.findElement(By.cssSelector(".header"));

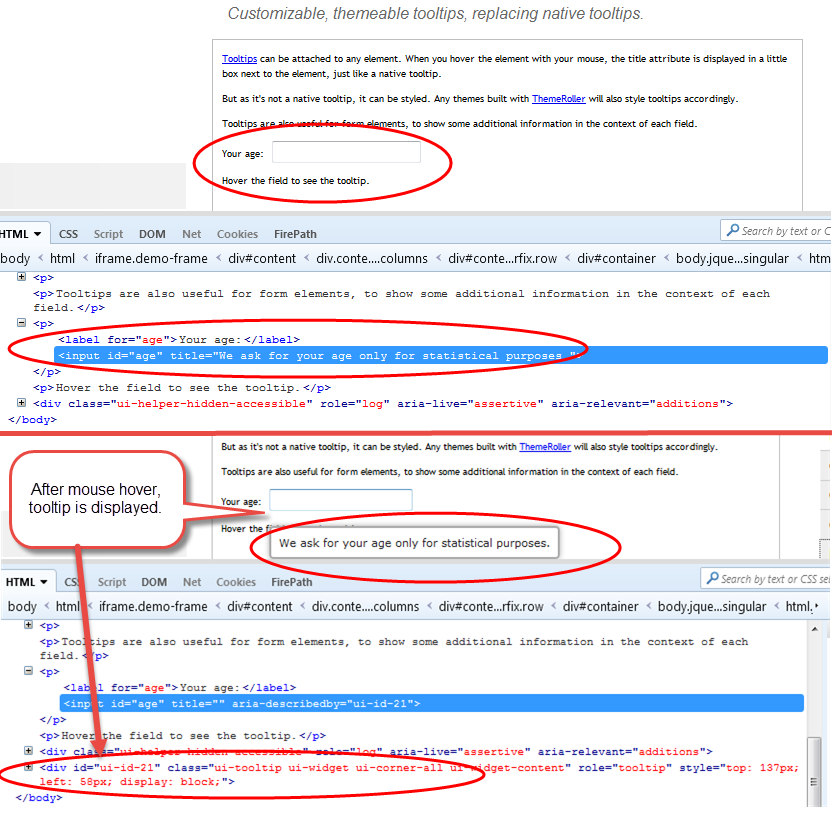
**String** toolTipText = element.getAttribute("title");

**We will follow the below steps for Case#1:**

1. Open browser  
2. Identify the element  
3. Get tool tip text by attribute  
4. compare Actual with Expected tool tip text

**Example#2:**

Now let us take JQuery example for tool tip.

Here when user mouse hover on the text field, it will display the tool tip. But when you observer the HTML, it doesn't have any title attribute. When ever user mouse hover, it will generate a div tag in which tool tip text resides. Check the below screen shot.  


So here getAttribute() will not work. To get the tool tip text here, we need to take the help of [Selenium actions class.](https://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/interactions/Actions.html)

**Syntax:**

Actions action = **new** Actions(driver);

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

actions.moveToElement(element).build().perform();

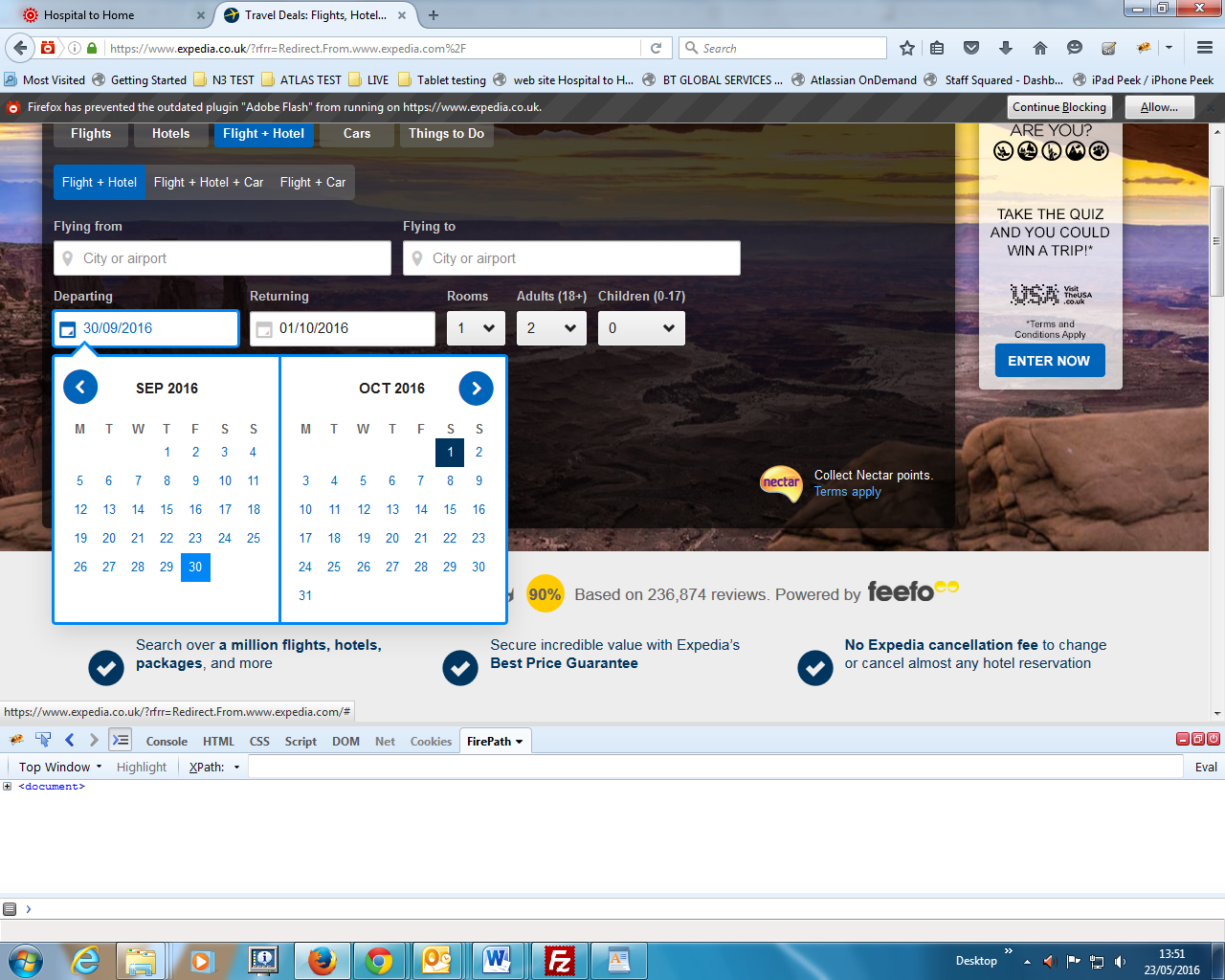
# Calendar Selection – Date pickers

**Udemy tutorial reference:**

**Selenium Webdriver - Advanced**

Section 12

When most of the websites are using advanced jQuery Datepickers instead of displaying individual dropdowns for month, day, year.



If we look at the Datepicker, it is just a like a table with set of rows and columns. To select a date, we just have to navigate to the cell where our desired date is present.

**Example Ecmo**

**package** tutorial.selenium;

**import** java.util.List;

**import** java.util.concurrent.TimeUnit;

**import** tutorial.selenium.multiSessionLockout;

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Test;

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

**import** org.openqa.selenium.JavascriptExecutor;

**import** org.openqa.selenium.Point;

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

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

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

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

**import** org.openqa.selenium.interactions.Actions;

**public** **class** CalendarSelection {

WebDriver driver;

String baseUrl;

String baseUrl2;

@Before

**public** **void** Setup() **throws** Exception {

baseUrl = "http://ecmo.test.healthpathways.co.uk/login.php";

baseUrl2 = "http://ecmo.test.healthpathways.co.uk/referral.php?pid=00003A&form=1";

}

@Test

**public** **void** testCalendarSelection1() **throws** Exception {

System.*setProperty*("webdriver.chrome.driver",

"C:\\Users\\Angelo\\Desktop\\eclipse projects\\drivers\\chromedriver.exe");

driver = **new** ChromeDriver();

// WebDriver driver=new FirefoxDriver();

// Maximize the browser's window

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

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

// static method - imported a static method login()

// http://stackoverflow.com/questions/18834005/calling-static-method-from-another-java-class

multiSessionLockout.*login*(driver);

Thread.*sleep*(2000);

// system navigates to referral form

driver.navigate().to(baseUrl2);

Thread.*sleep*(3000);

// click on create new Button to access to the referral form

WebElement createNewButton = driver.findElement(By.*xpath*("//a[@data-title='Create New']"));

createNewButton.click();

// Scroll down of 500 pixel

Thread.*sleep*(2000);

WebElement dobBox = driver.findElement(By.*xpath*("//input[@id='pd\_dob']"));

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("window.scrollTo(0,500)");

// Click on Date of Birth box

Thread.*sleep*(2000);

dobBox.click();

Point currentLocation = dobBox.getLocation();

System.***out***.println("DOB BOX = COORDINATE X : " + currentLocation.x + " ; CORDINATE Y : " + currentLocation.y);

/\*

\* other solution instead of scrolling down

\*

\* Actions action = new Actions(driver);

\* action.moveToElement(dobBox).click().perform();

\*

\*/

// This code clicks next button to find a particular month

// To enable this, enter a valid month in the while statement "while

// (!month.getText().contains("March"))"

WebElement nextButton;

WebElement month = driver.findElement(By.*xpath*(

"//div[@class='xdsoft\_datetimepicker xdsoft\_noselect xdsoft\_' and contains(@style,'display')]//div[@class='xdsoft\_label xdsoft\_month']/span"));

**do** {

nextButton = driver.findElement(By.*xpath*(

"//div[contains(@style,'display')]//button[@class='xdsoft\_next' and contains(@style,'visibility')]"));

nextButton.click();

Thread.*sleep*(1000);

} **while** (!month.getText().contains("March"));

// click on year link

Thread.*sleep*(1000);

WebElement year = driver.findElement(By.*xpath*(

"//div[@class='xdsoft\_datetimepicker xdsoft\_noselect xdsoft\_' and contains(@style,'display')]//div[@class='xdsoft\_label xdsoft\_year']/span"));

year.click();

Thread.*sleep*(1000);

// scroll locator

WebElement scroller = driver.findElement(By.*xpath*(

"//div[@class='xdsoft\_datetimepicker xdsoft\_noselect xdsoft\_' and contains(@style,'display')]//div[@class='xdsoft\_label xdsoft\_year']//div[@class='xdsoft\_scroller']"));

Point originalLocation = scroller.getLocation();

System.***out***

.println("SCROLLER = COORDINATE X : " + originalLocation.x + " ; COORDINATE Y : " + originalLocation.y);

// Actions builder = new Actions(driver); Scroll up y=+50

Thread.*sleep*(2000);

Actions action = **new** Actions(driver);

action.dragAndDropBy(scroller, 0, 50).build().perform();

Point targetLocation = scroller.getLocation();

System.***out***

.println("TARGET POINT = COORDINATE X : " + targetLocation.x + " ; COORDINATE Y : " + targetLocation.y);

// Select a year: for example 1982

Thread.*sleep*(2000);

WebElement selectedYear = driver.findElement(By.*xpath*(

"//div[@class='xdsoft\_datetimepicker xdsoft\_noselect xdsoft\_' and contains(@style,'display')]//div[@data-value='1982']"));

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

// select a cell date for example 25

WebElement calendar = driver.findElement(By

.*xpath*("//div[@class='xdsoft\_datetimepicker xdsoft\_noselect xdsoft\_' and contains(@style,'display')]"));

List<WebElement> rows = calendar.findElements(By.*xpath*(

"//div[@class='xdsoft\_datetimepicker xdsoft\_noselect xdsoft\_' and contains(@style,'display')]//td//div"));

Thread.*sleep*(2000);

**for** (WebElement cell : rows) {

**if** (cell.getText().equals("25")) {

// cell.findElement(By.linkText("25")).click();

cell.click();

**break**;

}

}

String nameScreenshot = "Test\_Passed\_Calendar\_20160524";

multiSessionLockout.*getscreenshot*(driver, nameScreenshot);

}

@After

**public** **void** tearDown() **throws** Exception {

driver.quit();

}

}

**Example in scroll down – sidebar scroll**

// Scroll down the page of 500 pixel

Thread.*sleep*(2000);

WebElement dobBox = driver.findElement(By.*xpath*("//input[@id='pd\_dob']"));

JavascriptExecutor js = (JavascriptExecutor) driver;

js.executeScript("window.scrollTo(0,500)");

// Click on Date of Birth box

Thread.*sleep*(2000);

dobBox.click();

Point currentLocation = dobBox.getLocation();

System.***out***.println("DOB BOX = COORDINATE X : " + currentLocation.x + " ; CORDINATE Y : " + currentLocation.y);

**Other example slider actions**

package actionsclass;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Actions;

public class SliderActions {

private WebDriver driver;

private String baseUrl;

@Before

public void setUp() throws Exception {

driver = new FirefoxDriver();

baseUrl = "https://jqueryui.com/slider/";

// Maximize the browser's window

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

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

}

@Test

public void testSliderActions() throws Exception {

driver.get(baseUrl);

driver.switchTo().frame(0);

Thread.sleep(3000);

// Using the actions class

WebElement element = driver.findElement(By.xpath("//div[@id='slider']/span"));

Actions action = new Actions(driver);

action.dragAndDropBy(element, 100, 0).perform();

}

@After

public void tearDown() throws Exception {

}

}

# Wait Commands

Listing out the different WebDriver **Wait** statements that can be useful for an effective scripting and can avoid using the **Thread.sleep()** commands.

## 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.

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

**FluentWait Command**

**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.

|  |  |
| --- | --- |
|  | // 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"));       }      }); |

**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>))); |

**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); |

**SetScriptTimeout Command**

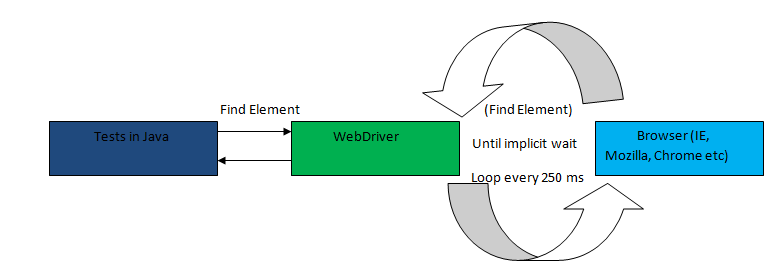
**Purpose**: Sets the amount of time to wait for an asynchronous script to finish execution before throwing an error. If the timeout is negative, then the script will be allowed to run indefinitely.

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

**Sleep 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 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.

|  |  |
| --- | --- |
| 1 | thread.sleep(1000); |

[](http://20tvni1sjxyh352kld2lslvc.wpengine.netdna-cdn.com/wp-content/uploads/2015/05/mplicitWait.png)

When the until method is called, following things happen in strictly this sequence

* ***Step 1***: In this step fluent wait captures the wait start time.
* ***Step 2***: Fluent wait checks the condition that is mentioned in the .until() method
* ***Step 3***: If the condition is not met, a thread sleep is applied with time out of the value mentioned in the .pollingEvery(250, TimeUnit.MILLISECONDS) method call. In the example above it is of 250 milliseconds.
* ***Step 4***: Once the thread sleep of step 3 expires, a check of start time is made with the current time. If the difference between wait start time, as captured in step 1, and the current time is less than time specified in .withTimeout(5000, TimeUnit.MILLISECONDS) then step 2 is repeated.

This process keeps on happening till the time either the time out expires or the condition comes out to be true.

# Selenium Automation Framework

**Udemy tutorial reference:**

**Automation Framework**

Section 15

## Introduction

Mission critical software undergoes rigorous functional tests, especially supported by automated testing frameworks. Automating these frameworks and maintaining quality software releases are critical to business performance. Enterprises often face the dilemma of balancing costs and managing resources to ensure that automation frameworks cover all the business scenarios and the applications delivered are error – free.

By implementing the appropriate automated testing framework, enterprises can significantly increase the speed and accuracy of the testing process, provide a higher return on investment (ROI) from software projects and systematically minimize risk.

## Why Framework

[](http://20tvni1sjxyh352kld2lslvc.wpengine.netdna-cdn.com/wp-content/uploads/2014/02/Framework.png)

A framework defines the organization’s way of doing things – a ‘Single Standard’. Following this standard would result in the project team achieving:

## Script-less representation of Automated tests:

[](http://20tvni1sjxyh352kld2lslvc.wpengine.netdna-cdn.com/wp-content/uploads/2014/02/Database-2.png)The testing framework should offer point-and-click interface for accessing and interacting with the application components under test—as opposed to presenting line after line of scripting. Testers should be able to visualize each step of the business scenario, view and edit test cases intuitively. This will shorten the learning curve for testers and help QA teams meet deadlines.

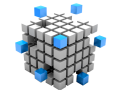
## Data Driven tests:

[](http://20tvni1sjxyh352kld2lslvc.wpengine.netdna-cdn.com/wp-content/uploads/2014/02/Excel.png)A key benefit of automating functional testing is the ability to test large volumes of data on the system quickly. But you must be able to manipulate the data sets, perform calculations, and quickly create hundreds of test iterations and permutations with minimal effort. Test Automation Frameworks must have capability to integrate with spreadsheets and provide powerful calculation features.

## Concise Reporting:

[](http://20tvni1sjxyh352kld2lslvc.wpengine.netdna-cdn.com/wp-content/uploads/2014/02/Reports.png)The ability to run high volume of tests is of little benefit if the results of the tests are not easy to understand. The framework must automatically generate reports of the test run and show the results in an easy-to-read format. The reports should provide specifics about where application failures occurred and what test data was used. Reports must present application screen shots for every step to highlight any discrepancies and provide detailed explanations of each checkpoint pass and failure. Reports must also be easily shared across the entire QA and development teams.

## Standard Scripting and Team Consistency:

[](http://20tvni1sjxyh352kld2lslvc.wpengine.netdna-cdn.com/wp-content/uploads/2014/02/Modular.png)Scripting standard should be maintained across the framework library creation, which includes business components, system communications, data check points, loggers, reporters etc. Project team should follow the defined scripting standards. Published standards across the project team pre-empt the effort involved in duplicate coding, which prevent individuals from following their own coding

standards.

## Encapsulation from Complexities:

[](http://20tvni1sjxyh352kld2lslvc.wpengine.netdna-cdn.com/wp-content/uploads/2014/02/Cabinet-1.png)Test engineers are encapsulated from the complexities and critical aspects of the code. Engineers are exposed only to the implemented libraries and tests are executed by just invoking the libraries.

## Implement and Maximize Re-Usability:

[](http://20tvni1sjxyh352kld2lslvc.wpengine.netdna-cdn.com/wp-content/uploads/2014/02/Documents.png)Establish the developed libraries across the organization/project team/product team, i.e. publish the library and provide access rights. Utilities/components shared across the team. Usage of available libraries. Minimized effort for repeated regression cycle.

Automation of test process include automation of following modules:

**Test Execution:** This is the place where a tool is used to execute the scripts.

**Report Generation:** This module concentrates on taking test execution results as input and will generate the customized reports of different levels.  
It can be summary report which provides details like PASS/FAIL status and Time taken to execute the test case.  
In detailed reports, it can list the number of exceptions cases handled, Start time and End time for each test case, test cases passed, failed status and .

**Defect Logging:** This module concentrates on automatic defect logging of defects into bug tracking tool and taking the screen shots.

**Test Case writing:** Automation of this part includes providing help for writing manual test cases as well as generating corresponding keywords into automation steps.

**Environment Specifications and Tools:**

1. Selenium Webdriver (Supports all major browsers, we use Mozilla, chrome and IE)
2. Eclipse IDE
3. Java
4. TestNG
5. AutoIT Tool (Used to handle Windows popups for Document [Uploads](http://seleniumeasy.com/tags/upload) and [Downloads](http://seleniumeasy.com/selenium-tutorials/how-to-download-a-file-with-webdriver).)
6. [Apache POI](http://seleniumeasy.com/apachepoi-tutorials) to perform operations with excel like read, write and update the excel sheet

**This Framework has the following tools:**

**1. Selenium -**Selenium is a well know open source testing framework, which is widely used for testing Web-based applications. It has different components and in that Webdriver has rendered the Selenium Remote Control obsolete, and is commonly referred to as Selenium 2.0.

Selenium Webdriver supports most of all browsers to run your test cases and many programming languages like C#, Java, Python, Ruby, .Net, Perl, PHP, etc.. to create and modify your test scripts.

**2. Eclipse IDE**: Eclipse is an integrated development environment (IDE) for Java. The Eclipse IDE is the most known product of the Eclipse Open Source project.

**3.**[**TestNG**](http://seleniumeasy.com/testng-tutorials) - Is a testing framework inspired from JUnit and NUnit. It has extended new functionalities which made it more powerful and easier than the other testing frameworks.

It supports [ReportNG (simple HTML reporting plug-in)](http://seleniumeasy.com/testng-tutorials/configuring-reportng-with-testng-to-generate-html-reports) and XLST (Graphical / Pictorial reports) plug-ins to customize or extend the default TestNG reporting style.

TestNG also provides ability to implement 'IReporter' an interface which can be implemented to generate a [Customized TestNG report](http://seleniumeasy.com/testng-tutorials/testng-customize-emailable-html-report-example) by users. It has 'generateReport()' method which will be invoked after all the suite has completed its execution and gives the report into the specified output directory.

**File Formats Used in the Framework:**

1. [*Properties file*](http://seleniumeasy.com/selenium-tutorials/read-data-from-properties-file-using-java-selenium) – We use properties file to store and retrieve the UI elements of an application or a website and data set file paths. It contains id of the elements, name, xpath or Css selector etc.
2. Excel files – Excel files are used to pass multiple sets of data to the application.
3. Xml file – Is used to execute the test scripts. Based on the package or classes or Tests mentioned in the xml file scripts will be executed.

# Introduction to Page Object Model Framework

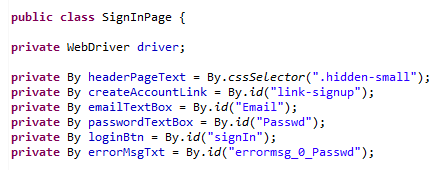
**Udemy tutorial reference:**

**Automation Framework**

Section 15

Page Object Model Framework has now a days become very popular test automation framework in the industry and many companies are using it because of its easy test maintenance and reduces the duplication of code.

The main advantage of Page Object Model is that if the UI changes for any page, it don’t require us to change any tests, we just need to change only the code within the page objects (Only at one place). Many other tools which are using selenium, are following the page object model.

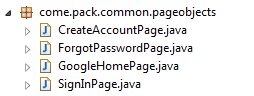


In the above screen shot, we have first identified the locators and defined it on the top after the class. In this way we can achieve readability of test scripts and we can easily identify locators and change them if needed at only one place.

Page Object model is writing all the functionalities / reusable components of a page that we want to automate in a separate class. Say now if we consider four pages as Home page, Login page, Create Account and Forgot password page etc.

As per Google Wiki [Page Object](https://code.google.com/p/selenium/wiki/PageObjects)  
***"****Within your web app’s UI there are areas that your tests interact with. A Page Object simply models these as objects within the test code. This reduces the amount of duplicated code and means that if the UI changes, the fix need only be applied in one place.****"***

For the above pages we will create classes as *HomePage.class, LoginPage.class, CreateAccountPage.class and ForgotPasswordPage.class*. In each class we will identify and write reusable methods which are specific to a page.



Here in The first page 'google home page' which will have many options like Search, Sign In, +You, Images, privacy etc links. based on the user action it navigates to respective page. Now all functionalities that we want to automate should have reusable methods/components for each page.

Now as our main page is google page we can navigate to other pages by clicking on any link from the google page. When ever we are navigating to other page, we need to return that page object. Else Return the current page object as this action doesn't navigate to a other page represented by another Page Object

**Example: Lets us take a simple login example:**

/\*\*\*

  \* Tests login functionality

  \*/

**public** **void** **loginTestCase**() {

       driver.navigate().to(URL);

       driver.findElement(By.name("signIn")).click();

       driver.findElement(By.id("username")).sendKeys("testuser");

       driver.findElement(By.id("password")).sendKeys("testpassword");

       driver.findElement(By.name("loginbtn")).click();

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

       wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("profile")));

       String Expected=driver.findElement(By.id("message")).getText();

       Assert.assertEquals(Expected, "Welcome");

}

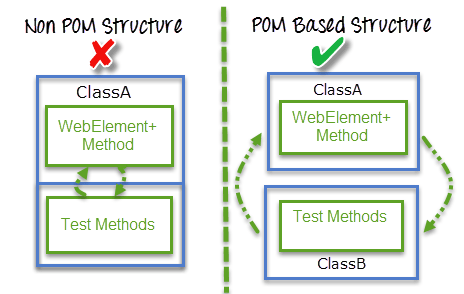
If you observe the above test, there is no separation of test and test locators. If this is the case, in future if the UI changes, it must be changed in multiple places. It will also become difficult to identify where these locators are used as the chances of locators are being used in multiple tests are more.

This is a small script. Script maintenance looks easy. But with time test suite will grow. As you add more and more lines to your code, things become tough.

The chief problem with script maintenance is that if 10 different scripts are using the same page element, with any change in that element, you need to change all 10 scripts. This is time consuming and error prone.

A better approach to script maintenance is to create a separate class file which would find web elements , fill them or verify them. This class can be reused in all the scripts using that element. In future if there is change in the web element , we need to make change in just 1 class file and not 10 different scripts.

This approach is called **Page Object Model(POM)**. It helps make code **more readable, maintainable**, and **reusable.**

[](http://cdn.guru99.com/images/AdvanceSelenium/071514_0722_PageObjectM2.png)

We will try to rewrite the above example by implementing the page object model:

/\*\*\*

\* Tests login functionality

\*/

**public** **void** **loginTestCase**() {

*// To go to home page*

homePage.gotoHomePage();

*//To click on SignIn link*

accountLoginPage = homePage.clickOnSignIn()

*//To verify if user is navigated to sign-in page*

Assert.assertTrue(accountLoginPage.verifyPage());

*//Login to the account*

accountLoginPage.userLogin(username,password);

*//To verify if user is navigated to user home page after successfull login*

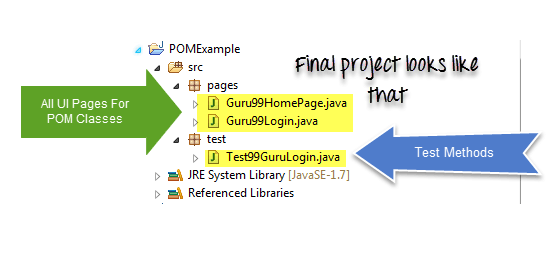
Assert.assertTrue(userHomePage.verifyPage());

}

In the above test, we have not used any locators. It is completely separated by driver.findElement 's, waits, exceptions and no static values in the code etc.We will be working only with the methods which are defined in multiple pages. Based on test, we will navigate to the required page and access those page methods.

### What is POM?

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

[](http://cdn.guru99.com/images/AdvanceSelenium/071514_0722_PageObjectM3.png)

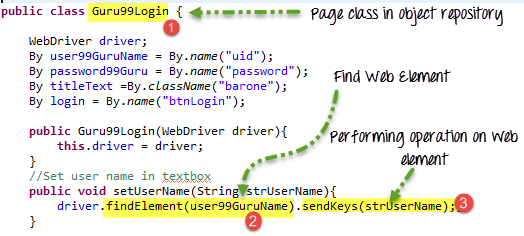
### Advantages of POM

1. Page Object Patten says operations and flows in the UI should be separated from verification. This concept makes our code cleaner and easy to understand.
2. Second benefit is the **object repository is independent of testcases**, so we can use the same object repository for a different purpose with different tools. For example, we can integrate POM with TestNG/JUnit for functional testing and at the same time with JBehave/[Cucumber](http://www.guru99.com/cucumber-tutorials.html) for acceptance testing.
3. Code becomes less and optimized because of the reusable page methods in the POM classes.
4. **Methods** get **more realistic names** which can be easily mapped with the operation happening in UI. i.e. if after clicking on the button we land on the home page, the method name will be like 'gotoHomePage()'.

### How to implement POM

Simple POM:

It's the basic structure of Page object model (POM) where all Web Elements of the **AUT** and the method that operate on these Web Elements are maintained inside a class file. Task like **verification** should be **separate** as part of Test methods.

[](http://cdn.guru99.com/images/AdvanceSelenium/071514_0722_PageObjectM4.png)

**Complete Example**

**TestCase:** Go to Guru99 Demo Site .

|  |  |
| --- | --- |
| Step 1) Go to Guru99 Demo Site  <http://cdn.guru99.com/images/AdvanceSelenium/071514_0722_PageObjectM5.png> |  |
| Step 2) In home page check text **"Guru99 Bank"** is present  <http://cdn.guru99.com/images/AdvanceSelenium/071514_0722_PageObjectM6.png> |  |
| Step 3) Login into application  <http://cdn.guru99.com/images/AdvanceSelenium/071514_0722_PageObjectM7.png> |  |
| Step 4) Verify that the Home page contains text as "Manger Id : demo"  <http://cdn.guru99.com/images/AdvanceSelenium/071514_0722_PageObjectM8.png> |  |

Here are we are dealing with 2 pages

1. Login Page
2. Home Page (shown once you login)

Accordingly we create 2 POM classes

**Guru99 Login page POM**

package pages;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

public class Guru99Login {

    WebDriver driver;

    By user99GuruName = By.name("uid");

    By password99Guru = By.name("password");

    By titleText =By.className("barone");

    By login = By.name("btnLogin");

    public Guru99Login(WebDriver driver){

        this.driver = driver;

    }

    //Set user name in textbox

    public void setUserName(String strUserName){

        driver.findElement(user99GuruName).sendKeys(strUserName);;

    }

    //Set password in password textbox

    public void setPassword(String strPassword){

         driver.findElement(password99Guru).sendKeys(strPassword);

    }

    //Click on login button

    public void clickLogin(){

            driver.findElement(login).click();

    }

    //Get the title of Login Page

    public String getLoginTitle(){

     return    driver.findElement(titleText).getText();

    }

    /\*\*

     \* This POM method will be exposed in test case to login in the application

     \* @param strUserName

     \* @param strPasword

     \* @return

     \*/

    public void loginToGuru99(String strUserName,String strPasword){

        //Fill user name

        this.setUserName(strUserName);

        //Fill password

        this.setPassword(strPasword);

        //Click Login button

        this.clickLogin();

    }

}

**Guru99 Home Page POM**

package pages;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

public class Guru99HomePage {

    WebDriver driver;

    By homePageUserName = By.xpath("//table//tr[@class='heading3']");

    public Guru99HomePage(WebDriver driver){

        this.driver = driver;

    }

    //Get the User name from Home Page

        public String getHomePageDashboardUserName(){

         return    driver.findElement(homePageUserName).getText();

        }

}

**Guru99 Simple POM Test case**

package test;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import pages.Guru99HomePage;

import pages.Guru99Login;

public class Test99GuruLogin {

    WebDriver driver;

    Guru99Login objLogin;

    Guru99HomePage objHomePage;

    @BeforeTest

    public void setup(){

        driver = new FirefoxDriver();

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

        driver.get("http://demo.guru99.com/V4/");

    }

    /\*\*

     \* This test case will login in http://demo.guru99.com/V4/

     \* Verify login page title as guru99 bank

     \* Login to application

     \* Verify the home page using Dashboard message

     \*/

    @Test(priority=0)

    public void test\_Home\_Page\_Appear\_Correct(){

        //Create Login Page object

    objLogin = new Guru99Login(driver);

    //Verify login page title

    String loginPageTitle = objLogin.getLoginTitle();

    Assert.assertTrue(loginPageTitle.toLowerCase().contains("guru99 bank"));

    //login to application

    objLogin.loginToGuru99("mgr123", "mgr!23");

    // go the next page

    objHomePage = new Guru99HomePage(driver);

    //Verify home page

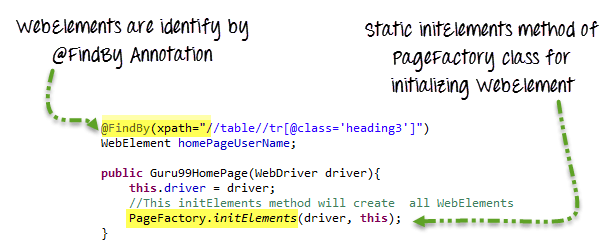
    Assert.assertTrue(objHomePage.getHomePageDashboardUserName().toLowerCase().contains("manger id : mgr123"));

    }

**What is Page Factory?**

Page Factory is an inbuilt page object model concept for Selenium WebDriver but it is more optimized. Here as well we follow the concept of separation of Page Object repository and Test methods.

Additionally with the help of PageFactory class we use annotations **@FindBy** to find WebElement. We use initElements method to initialize web elements

[](http://cdn.guru99.com/images/AdvanceSelenium/071514_0722_PageObjectM9.png)

We should initialize page objects using initElements() method from PageFactory Class as below, Once we call initElements() method, all elements will get initialized. PageFactory.initElements() static method takes the driver instance of the given class and the class type, and returns a Page Object with its fields fully initialized.

    public Guru99Login(WebDriver driver){

        this.driver = driver;

        //This initElements method will create all WebElements

        PageFactory.initElements(driver, this);

    }

We should preferably use a constructor which takes a WebDriver instance as its only argument or falling back on a no-arg constructor. An exception will be thrown if the class cannot be instantiated.

Page Factory will initialize every WebElement variable with a reference to a corresponding element on the actual web page based on “locators” defined. This is done by using **@FindBy annotations**.

#### Annotations?

In Page Factory, Annotations are used to give descriptive names for WebElements to improve code readability. And annotation **@FindBy** is used to identify Web Elements in the page.

By default, PageFactory will search for elements on the page with a matching id attribute, If that fails, then it will search by the name attribute. But as we need more control over identifying elements in the HTML page and mapping them to our Page Object fields. One way to do this is to use the @FindBy annotation, as shown in the following code:

The [@FindBy](https://selenium.googlecode.com/git/docs/api/java/org/openqa/selenium/support/FindBy.html) annotation supports all the other locators strategies that we use:  
**id, name, className, css, xpath, tagName, linkText and partialLinkText**

We can either use this annotation by specifying both "How" and "using" or by specifying any one of the location strategies (Eg: "id")

@FindBy(how = How.ID, **using** = "username")

**private** WebElement userName;

We can re-write the above one as below:

@FindBy(id="username")

**private** WebElement userName;

And To work with class name, we will define as below:

@FindBy(className=".input.username")

**private** WebElement userName;

**When we have multiple elements (list of WebElements), we can initialize them using PageFactory as below :**

@FindBy(tagName = "mylist")

**private** List<WebElement> links;

Every time when a method is called on a WebElement, the driver will first find it on the current page and then simulate the action on the WebElement. There are cases where we will be working with a basic page, and we know that we will find the element on the page every time we look for it, In such cases we can use annotation **‘@CacheLookup‘** which is another annotation in page factory

@FindBy(name="username")

@CacheLookup

**private** WebElement userName;

**Example Login page with Page Factory**

package PageFactory;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class Guru99Login {

    /\*\*

     \* All WebElements are identified by @FindBy annotation

     \*/

    WebDriver driver;

    @FindBy(name="uid")

    WebElement user99GuruName;

    @FindBy(name="password")

    WebElement password99Guru;

    @FindBy(className="barone")

    WebElement titleText;

    @FindBy(name="btnLogin")

    WebElement login;

    public Guru99Login(WebDriver driver){

        this.driver = driver;

        //This initElements method will create all WebElements

        PageFactory.initElements(driver, this);

    }

    //Set user name in textbox

    public void setUserName(String strUserName){

        user99GuruName.sendKeys(strUserName);

    }

    //Set password in password textbox

    public void setPassword(String strPassword){

    password99Guru.sendKeys(strPassword);

    }

    //Click on login button

    public void clickLogin(){

            login.click();

    }

    //Get the title of Login Page

    public String getLoginTitle(){

     return    titleText.getText();

    }

    /\*\*

     \* This POM method will be exposed in test case to login in the application

     \* @param strUserName

     \* @param strPasword

     \* @return

     \*/

    public void loginToGuru99(String strUserName,String strPasword){

        //Fill user name, ‘this’is an istance of class Guru99Login

        this.setUserName(strUserName);

        //Fill password

        this.setPassword(strPasword);

        //Click Login button

        this.clickLogin();

    }

}

**Home Page with Page Factory**

package PageFactory;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.support.FindBy;

import org.openqa.selenium.support.PageFactory;

public class Guru99HomePage {

    WebDriver driver;

    @FindBy(xpath="//table//tr[@class='heading3']")

    WebElement homePageUserName;

    public Guru99HomePage(WebDriver driver){

        this.driver = driver;

        //This initElements method will create all WebElements

        PageFactory.initElements(driver, this);

    }

    //Get the User name from Home Page

        public String getHomePageDashboardUserName(){

         return    homePageUserName.getText();

        }

}

### TestCase with Page Factory concept

package test;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import PageFactory.Guru99HomePage;

import PageFactory.Guru99Login;

public class Test99GuruLoginWithPageFactory {

    WebDriver driver;

    Guru99Login objLogin;

    Guru99HomePage objHomePage;

    @BeforeTest

    public void setup(){

        driver = new FirefoxDriver();

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

        driver.get("http://demo.guru99.com/V4/");

    }

    /\*\*

     \* This test go to http://demo.guru99.com/V4/

     \* Verify login page title as guru99 bank

     \* Login to application

     \* Verify the home page using Dashboard message

     \*/

    @Test(priority=0)

    public void test\_Home\_Page\_Appear\_Correct(){

        //Create Login Page object

    objLogin = new Guru99Login(driver);

    //Verify login page title

    String loginPageTitle = objLogin.getLoginTitle();

    Assert.assertTrue(loginPageTitle.toLowerCase().contains("guru99 bank"));

    //login to application

    objLogin.loginToGuru99("mgr123", "mgr!23");

    // go the next page

    objHomePage = new Guru99HomePage(driver);

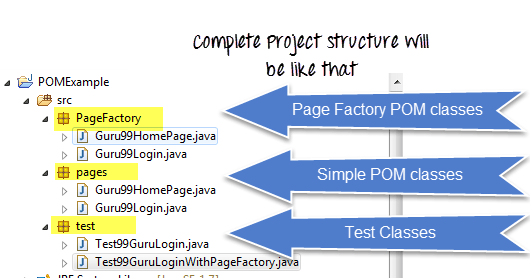
    //Verify home page

    Assert.assertTrue(objHomePage.getHomePageDashboardUserName().toLowerCase().contains("manger id : mgr123"));

    }

}

Complete Project Structure will look like the diagram:

[](http://cdn.guru99.com/images/AdvanceSelenium/071514_0722_PageObjectM10.png)

**Example**

package tutorialselenium;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import pageclasses.SearchPageFactory;

public class FrameworkTestCase {

private WebDriver driver;

private String baseUrl;

SearchPageFactory searchPage;

@Before

public void beforeClass() {

driver = new FirefoxDriver();

baseUrl = "https://www.expedia.com/";

searchPage = new SearchPageFactory(driver);

// Maximize the browser's window

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

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

driver.get(baseUrl);

}

@Test

public void test() throws Exception {

searchPage.clickFlightsTab();

searchPage.setOriginCity("New York");

searchPage.setDestinationCity("San Francisco");

searchPage.setDepartureDate("10/28/2015");

searchPage.setReturnDate("10/31/2015");

}

@After

public void afterClass() {

}

}

**Summary**

1. Page Object Model is an Object repository design pattern in Selenium WebDriver.
2. POM creates our testing code maintainable, reusable.
3. Page Factory is an optimized way to create object repository in POM concept.

**Logging Infrastructure - Using Log4j**

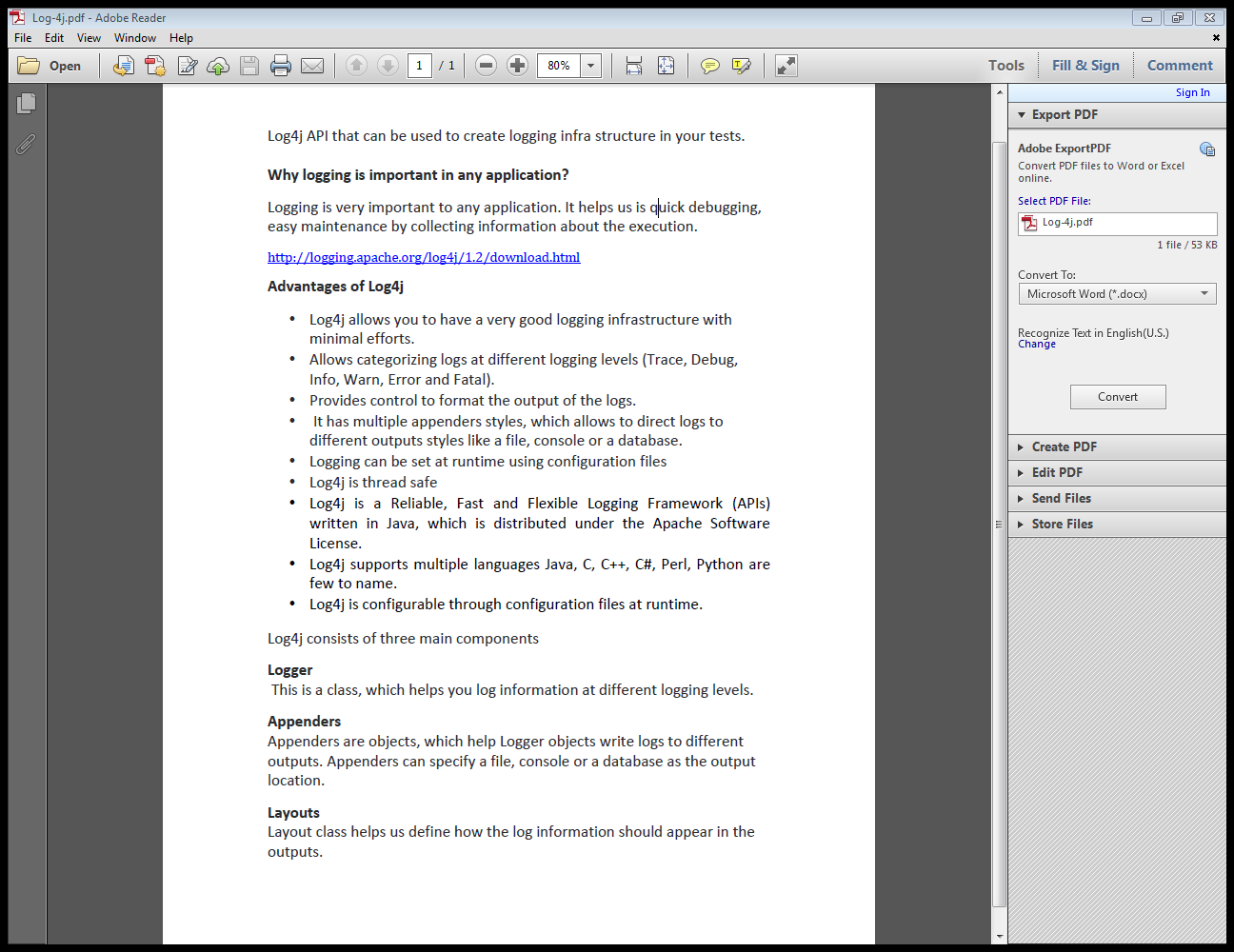
**Udemy tutorial reference:**

**Logging Infrastructure – Log4j**

Section 15

[**https://www.seleniumeasy.com/log4j-tutorials**](https://www.seleniumeasy.com/log4j-tutorials)

[**http://toolsqa.com/selenium-webdriver/log4j-introduction/**](http://toolsqa.com/selenium-webdriver/log4j-introduction/)



**Example: appender – Configuration simply layout**

package tutorialselenium;

import org.apache.log4j.\*;

import java.util.Properties;

**import org.apache.log4j.Logger;**

**import org.apache.log4j.PropertyConfigurator;**

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.junit.\*;

/\*\*

\* Trace

\* Debug

\* Info

\* Warn

\* Error

\* Fatal

\* VM Argument: -Dlog4j.debug=true

\*/

public class LoggingDemo {

static Logger log = Logger.getLogger(LoggingDemo.class);

static Appender append;

@Before

public void setUp() throws Exception {

// Define Appender and then adding an appender to log

append = new ConsoleAppender(new SimpleLayout());

log.addAppender(append);

// BasicConfigurator.configure();

Properties log4jprop = new Properties();

log4jprop.setProperty("log4j.rootlogger", "DEBUG, CA");

log4jprop.setProperty("log4j.appender.CA", "org.apache.log4j.ConsoleAppender");

log4jprop.setProperty("log4j.appender.CA.layout", "org.apache.log4j.PatternLayout");

log4jprop.setProperty("log4j.appender.CA.layout.CoversionPattern", "%d{yyyy-MM-dd} -- %-10p %c - %m%n");

PropertyConfigurator.configure(log4jprop);

log.info("Running before method");

}

@Test

public void testLog4j() throws Exception {

log.info("Running test method");

}

@After

public void tearDown() throws Exception {

log.info("Running after method");

}

}

**Example: Console Appenders – layout**

package tutorialselenium;

import org.apache.log4j.Appender;

import org.apache.log4j.ConsoleAppender;

import org.apache.log4j.Level;

import org.apache.log4j.Logger;

import org.apache.log4j.PatternLayout;

import org.junit.AfterClass;

import org.junit.BeforeClass;

import org.junit.Test;

public class DemoLogging {

static Logger log = Logger.getLogger(DemoLogging.class);

static Appender append;

@BeforeClass

public static void setUpBeforeClass() throws Exception {

final PatternLayout layout = new PatternLayout();

layout.setConversionPattern("%d{yyyy-MM-dd} --%-10p %c{1} -%m%n");

append = new ConsoleAppender(layout);

log.addAppender(append);

log.setLevel(Level.INFO);

log.info("Running before method");

}

@AfterClass

public static void tearDownAfterClass() throws Exception {

log.debug("Running after method");

}

@Test

public void test() {

log.info("Running test method");

}

}

**Example: Configuring Properties file -** log4j.properties file in the same project

# Define the root logger with appender file

log4j.rootLogger=DEBUG, FILE

log=<location>

# Define the file appender

log4j.appender.FILE=org.apache.log4j.RollingFileAppender

log4j.appender.FILE.File=${log}/log.out

# Set the maximum file size before rollover

log4j.appender.FILE.MaxFileSize=10KB

# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.ConversionPattern=%d{yyyy-MM-dd} -- %-10p %c - %m%n

# Set the logger level of File Appender to Info

log4j.appender.FILE.Threshold=Info

**--------------------------------------------------------------------------------------------------------**package tutorial.selenium;

import org.apache.log4j.\*;

/\*\*

\* Trace

\* Debug

\* Info

\* Warn

\* Error

\* Fatal

\* VM Argument: -Dlog4j.debug=true

\*/

public class LoggingDemo {

**static Logger log = Logger.getLogger(LoggingDemo.class);**

public static void main(String[] args) {

//properties file as argument

PropertyConfigurator.configure("log4j.properties");

log.info("This message is a info");

runMethod();

log.debug("This message is a debug");

}

public static void runMethod() {

log.info("This info message is from runMethod");

}

}

**Example: Test case using Logging infrastructure on test method**

package tutorial.selenium;

import java.util.concurrent.TimeUnit;

import org.apache.log4j.\*;

import org.junit.\*;

import org.openqa.selenium.\*;

import org.openqa.selenium.firefox.FirefoxDriver;

import page.classes.SearchPage;

public class UsingLog4j {

private WebDriver driver;

private String baseUrl;

//static Logger log = Logger.getLogger(UsingLog4j.class);

@Before

public void setUp() throws Exception {

PropertyConfigurator.configure("log4j.properties");

driver = new FirefoxDriver();

baseUrl = "https://www.expedia.com/";

// Maximize the browser's window

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

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

}

@Test

public void test() {

driver.get(baseUrl);

SearchPage.navigateToFlightsTab(driver);

//log.info("Navigate to flights tab");

SearchPage.fillOriginTextBox(driver, "New York");

//log.info("Enter origin city");

SearchPage.fillDestinationTextBox(driver, "Chicago");

//log.info("Enter destination city");

SearchPage.departureDateTextBox(driver).sendKeys("12/25/2014");

//log.info("Enter departure date");

SearchPage.returnDateTextBox(driver).sendKeys("12/31/2014");

//log.info("Enter return date");

}

@After

public void tearDown() throws Exception {

}

}

**Set up Logging infrastructure on page refactory class (POM)**

package page.classes;

import org.apache.log4j.Logger;

import org.openqa.selenium.\*;

public class SearchPage {

public static WebElement element = null;

**static Logger log = Logger.getLogger(SearchPage.class);**

/\*\*

\* Returns the flight origin text box element

\* @param driver

\* @return

\*/

public static WebElement originTextBox(WebDriver driver) {

element = driver.findElement(By.id("flight-origin"));

log.info("Origin text box element found");

return element;

}

/\*\*

\* Fill origin city in origin text box

\* @param driver

\* @param origin

\*/

public static void fillOriginTextBox(WebDriver driver, String origin) {

element = originTextBox(driver);

element.sendKeys(origin);

log.info("Enter origin city as " + origin);

}

/\*\*

\* Returns the flight destination text box element

\* @param driver

\* @return

\*/

public static WebElement destinationTextBox(WebDriver driver) {

element = driver.findElement(By.id("flight-destination"));

log.info("Destination text box element found");

return element;

}

/\*\*

\* Fill destination city in destination text box

\* @param driver

\* @param origin

\*/

public static void fillDestinationTextBox(WebDriver driver, String destination) {

element = destinationTextBox(driver);

element.sendKeys(destination);

log.info("Enter destination city as " + destination);

}

/\*\*

\* Returns the departure date text box element

\* @param driver

\* @return

\*/

public static WebElement departureDateTextBox(WebDriver driver) {

element = driver.findElement(By.id("flight-departing"));

return element;

}

/\*\*

\* Returns the return date text box element

\* @param driver

\* @return

\*/

public static WebElement returnDateTextBox(WebDriver driver) {

element = driver.findElement(By.id("flight-returning"));

return element;

}

/\*\*

\* Returns the search button box element

\* @param driver

\* @return

\*/

public static WebElement searchButton(WebDriver driver) {

element = driver.findElement(By.id("search-button"));

return element;

}

/\*\*

\* Click on search button

\* @param driver

\*/

public static void clickOnSearchButton(WebDriver driver) {

element = searchButton(driver);

element.click();

}

/\*\*

\* Navigate to flights tab

\* @param driver

\*/

public static void navigateToFlightsTab(WebDriver driver) {

driver.findElement(By.id("header-history")).click();

element = driver.findElement(By.id("tab-flight-tab"));

element.click();

}

}

# TestNG Framework

**Udemy tutorial reference:**

**TestNG Framework**

Section 16

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

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

TestNG eliminates most of the limitations of the older framework and gives the developer the ability to write more flexible and powerful tests with help of easy annotations, grouping, sequencing & parametrizing.  
   
**Benefits of TestNG**

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

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

## Test Case Writing

Writing a test in TestNG is quite simple and basically involves following steps:

**Step 1** – Write the business logic of the test

**Step 2** – Insert TestNG annotations in the code

**Step 3** – Add the information about your test (e.g. the class names, methods names, groups names etc…) in a testng.xml file

**Step 4** – Run TestNG

## Annotations in TestNG

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

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

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

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

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

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

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

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

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

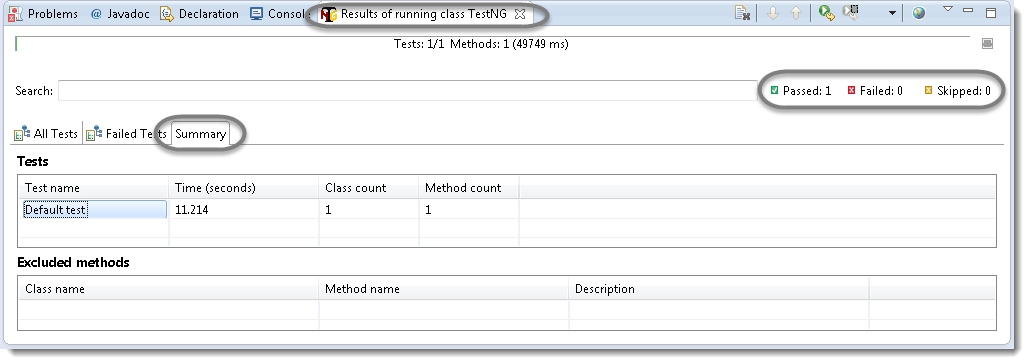
**@AfterMethod**: The annotated method will be run after each test method.

**@Test**: The annotated method is a part of a test case.  
   
**Benefits of using Annotations**

1. It identifies the methods it is interested in by looking up annotations. Hence method names are not restricted to any pattern or format.
2. We can pass additional parameters to annotations.
3. Annotations are strongly typed, so the compiler will flag any mistakes right away.
4. Test classes no longer need to extend anything (such as Test Case, for JUnit 3).

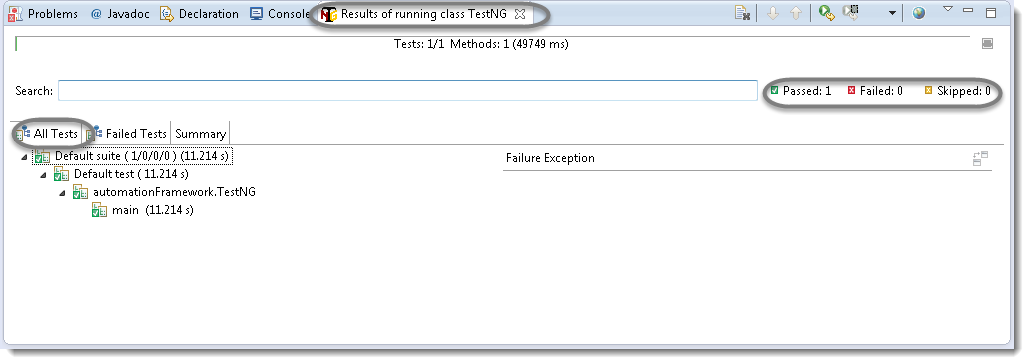
 **Results of running the Testng Test Case**

Give it few minutes to complete the execution, once it is finished the results will look like this in the **TestNg Result**window.



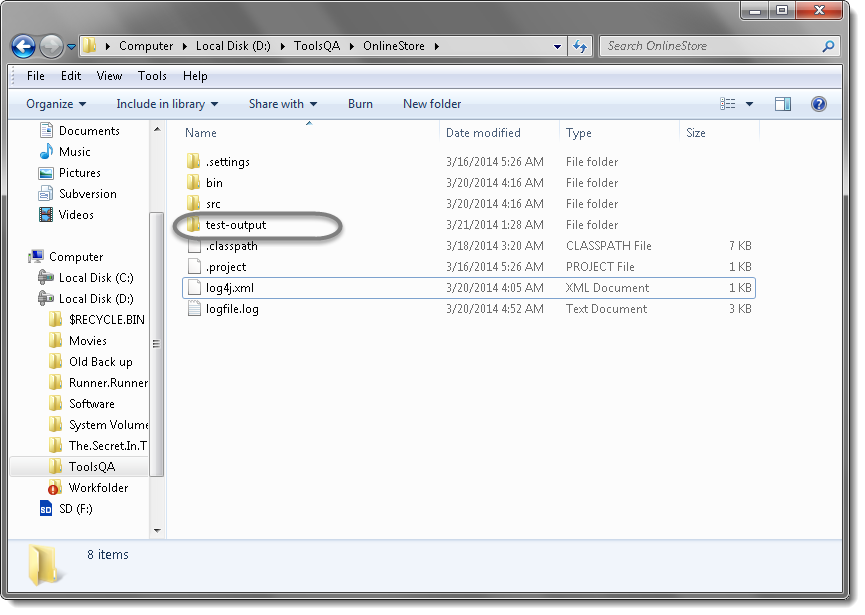
It displayed ‘passed : 1’. This means test is successful and  Passed.

There are 3 sub tabs. “All Tests”, “Failed Tests” and “Summary”. Just click “All Tests” to see what is there.

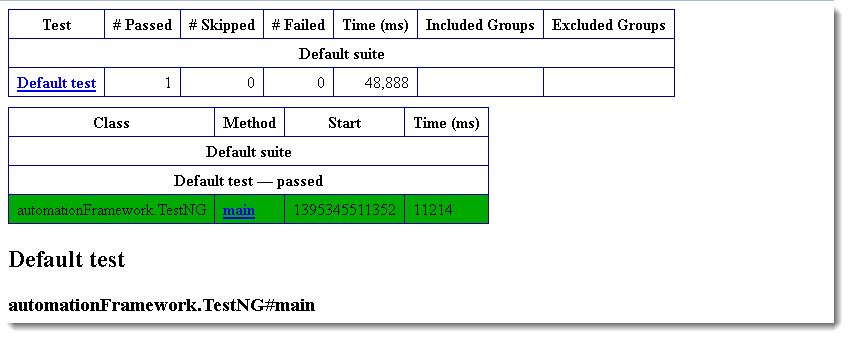


As you see, there is information of which test cases are executed and their duration. Take look to other tabs. Better than Junit right?

TestNG also produce HTML reports. To access those reports go to the**Project** directory and open **test-output** folder.

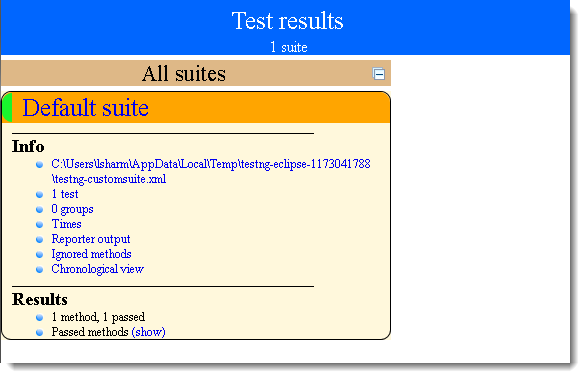


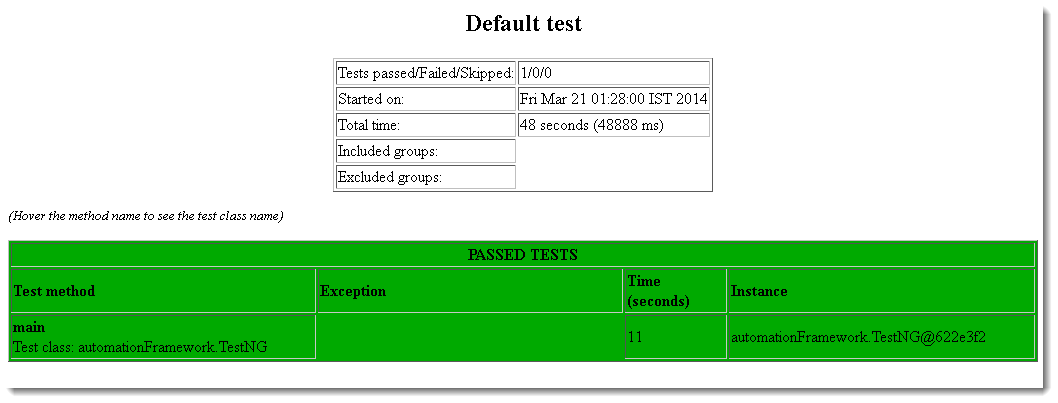
Open ‘**emailable-report.html**‘, as this is a html report open it with browser.



TestNG also produce ‘**index.html**‘ report and it resides in the same **test-output** folder.

This reports gives the link to all the different component of the TestNG reports like **Groups** &**Reporter Output**. On clicking these will display detailed descriptions of execution. In the advance chapter of TestNG we will go though each of the TestNG topics.





# Test suite

In any project, you will end up to a place where you need to execute so many test cases on a run. Running a set of test cases together is call executing a **Test Suite**. Those test cases can be dependent to each other or may have to be executed in a certain order. TestNg gives us the capability to manage our test execution.

In TestNG framework, we need to create **testng.xml** file to create and handle multiple test classes. This is the xml file where you will configure your test run, set test dependency, include or exclude any test, method, class or package and set priority etc.

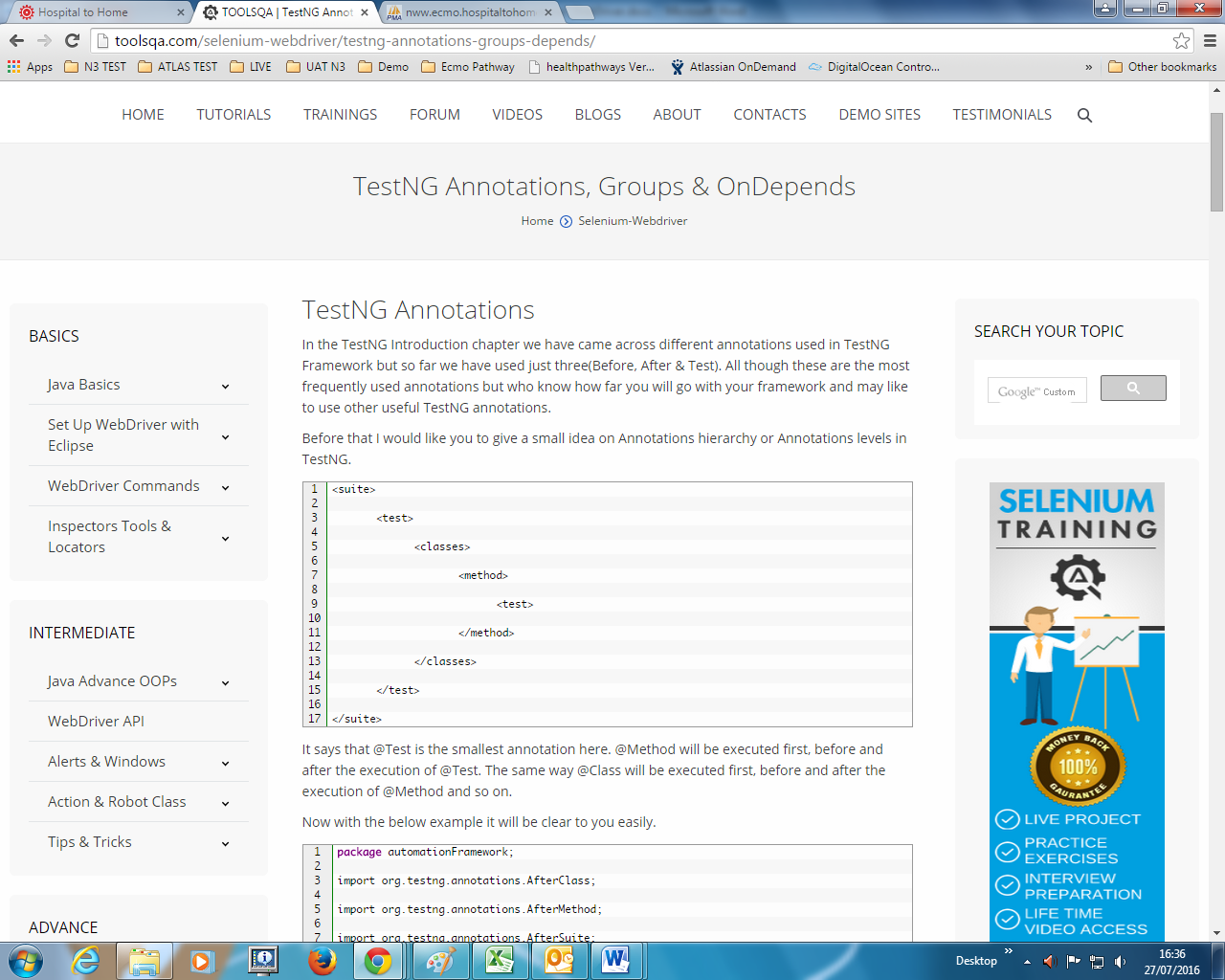
**How to do it…**

**TestNG Annotations**

In the TestNG Introduction chapter we have came across different annotations used in TestNG Framework but so far we have used just three (Before, After & Test).

All though these are the most frequently used annotations but who know how far you will go with your framework and may like to use other useful TestNG annotations.

Before that I would like you to give a small idea on Annotations hierarchy or Annotations levels in TestNG.



It says that @Test is the smallest annotation here. @Method will be executed first, before and after the execution of @Test. The same way @Class will be executed first, before and after the execution of @Method and so on.

**Test Case Grouping**

‘**Groups**‘is one more annotation of TestNG which can be used in the execution of multiple tests. Let’s say you have hundred tests of class vehicle and in it ten method of car, ten method of scooter and so on. You probably like to run all the scooter tests together in a batch. And you want all to be in a single test suite. With the help of grouping you can easily overcome this situation.

**How to do it…**

1) Create two methods for Car, two methods for Scooter and one method in conjunction with Car & Sedan Car.

2) Group them separately with using (groups = { ” Group Name” })

package automationFramework;

import org.testng.annotations.Test;

public class Grouping {

  @Test (groups = { "Car" })

  public void Car1() {

  System.out.println("Batch Car - Test car 1");

  }

  @Test (groups = { "Car" })

  public void Car2() {

  System.out.println("Batch Car - Test car 2");

  }

  @Test (groups = { "Scooter" })

  public void Scooter1() {

  System.out.println("Batch Scooter - Test scooter 1");

  }

  @Test (groups = { "Scooter" })

  public void Scooter2() {

  System.out.println("Batch Scooter - Test scooter 2");

  }

  @Test (groups = { "Car", "Sedan Car" })

  public void Sedan1() {

  System.out.println("Batch Sedan Car - Test sedan 1");

  }

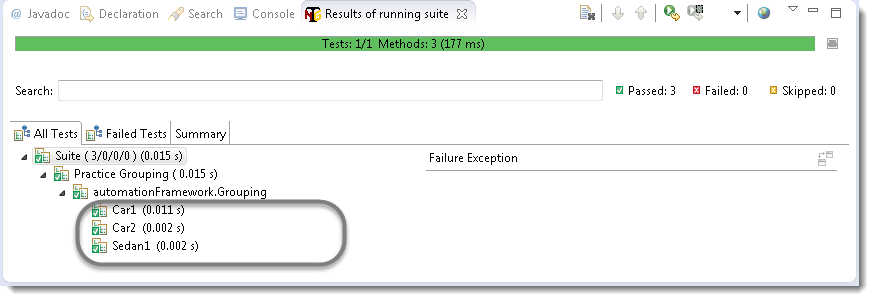
}

|  |  |
| --- | --- |
|  |  |

3) Create a testng xml like this:

|  |  |
| --- | --- |
|  | <suite name="Suite">        <test name="Practice Grouping">            <groups>        <run>    <include name="Car" />        </run>    </groups>    <classes>        <class name="automationFramework.Grouping" />    </classes>        </test>    </suite> |

 4) Run the test by right click on the testng.xml file and select **Run As** > **TestNG Suite.**Output will be like this in TestNg console:



**Dependent Test**

Sometimes, you may need to invoke methods in a Test case in a particular order or you want to share some data and state between methods. This kind of dependency is supported by TestNG as it supports the declaration of explicit dependencies between test methods.

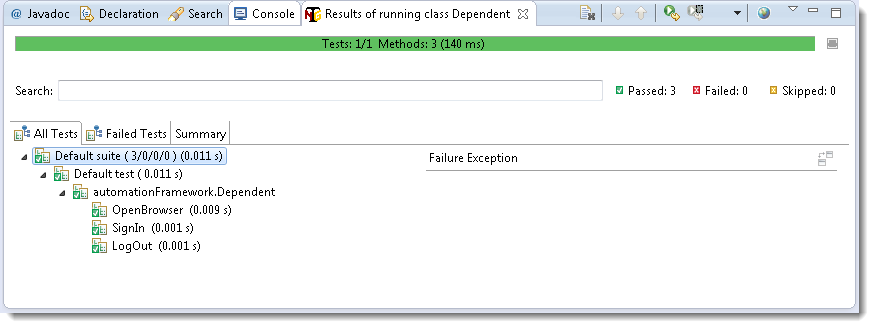
TestNG allows you to specify dependencies either with:

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

Take a look over the below example:

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

The output will be like this:



## Multiple Tests

There will be situations when you want to put number of tests under a single test class and like to run all in single shot.

With the help of TestNG ‘**@Test**‘ annotations we can execute multiple tests in single TestNG file.

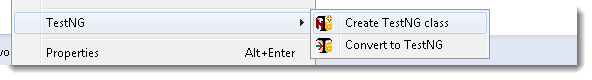
Take an example of four different tests under one testng class and print the test sequence on the console.

**How to do it…**

1) Press **Ctrl+N**, select “**TestNG Class**” under **TestNG**category and click **Next**.

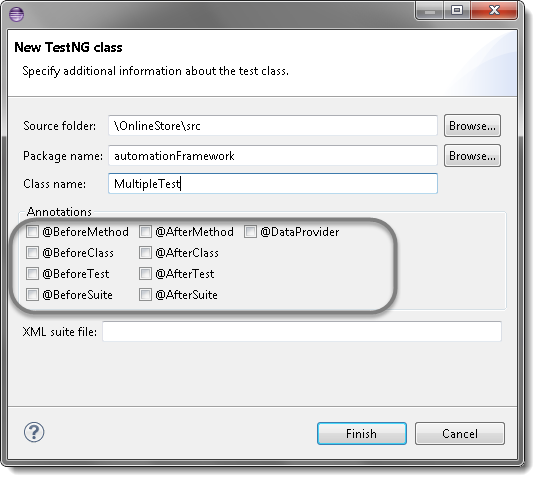
**Or**

Right click on Test Case folder, go to **TestNG**and select “**TestNG Class**“.



2) If your project is set up and you have selected the Test Case folder before creating TestNG class then the source folder and the package name will be pre-populated on the form. Set class name as **TestNG**.

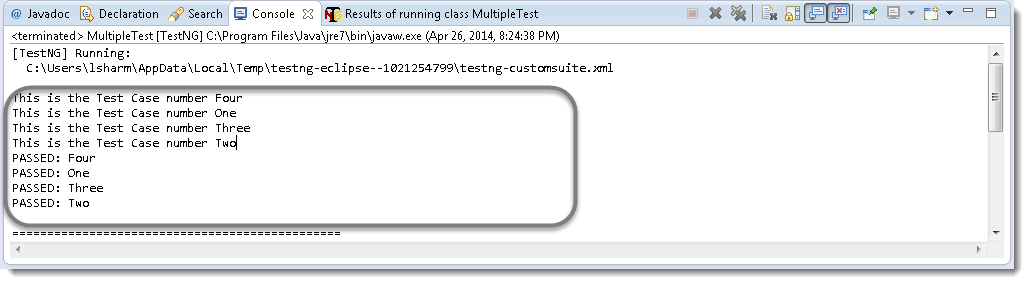
Leave rest of the settings untouched, do not check for “**@BeforeMethod**”, “**@AfterMethod**” for now and click **Finish**. That’s it.



3) By default a new class will have only one @Test method. Add two more methods by yourself and put your code accordingly in methods. Code will look like:

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

This will enable you to execute all four tests with just one testng class. Take a look on the output.



**Attention:**By default, methods annotated by @Test are executed alphabetically. Take a look over the next topic to see how to prioritize @Test.

**Sequencing & Prioritizing**

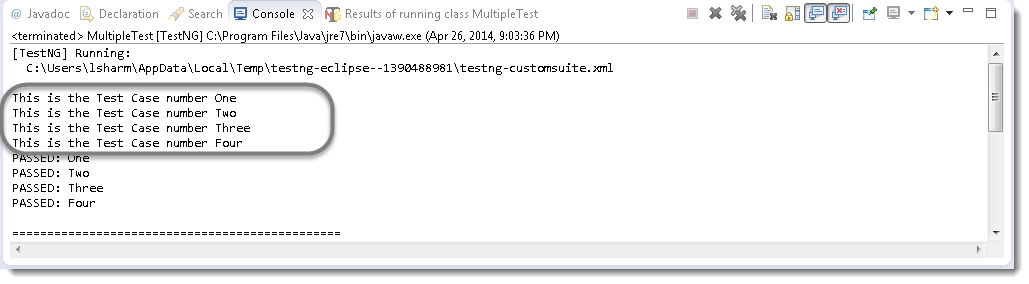
You need to use the ‘**priority**‘ parameter, if you want the methods to be executed in your order. **Parameters** are keywords that modify the annotation’s function.

Let’s take the same above example and execute all @Test methods in right order. Simply assign priority to all @Test methods starting from 0 (Zero).

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

**Note:**TestNG will execute the @Test annotation with the lowest priority value up to the largest.

Output of the above:



## TestNG Reporters

TestNG is a Framework and so far we have already seen the many different powerful features of TestNG. It almost gives you all the important things you are required to complete the Framework.

## TestNG Reporter Logs

TestNG also gives us the logging facility for the test. For example during the running of test case user wants some information to be logged in the console. Information could be any detail depends upon the purpose. Keeping this in mind that we are using Selenium for testing, we need the information which helps the User to understand the test steps or any failure during the test case execution. With the help of TestNG Logs it is possible to enable logging during the Selenium test case execution.

In selenium there are two types of logging. **High level** logging and **Low level**logging. In low level logging you try to produce logs for the every step you take or every action you make in your automation script.

In high level logging you just try to capture main events of your test.

Everybody has their own style of logging. Talking about Log4j logging, that’s why we do not mix log4j logging with testng logging.

We’d perform [**low level logging with log4j**](http://toolsqa.com/selenium-webdriver/log4j-logging/) and **high level logging with testng** reporter logs.

How to do it…

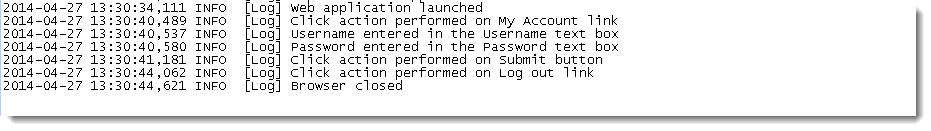
1) Write a test case for Sign In application and implement Log4j logging on every step.

2) Insert Reporter logs on the main events of the test.

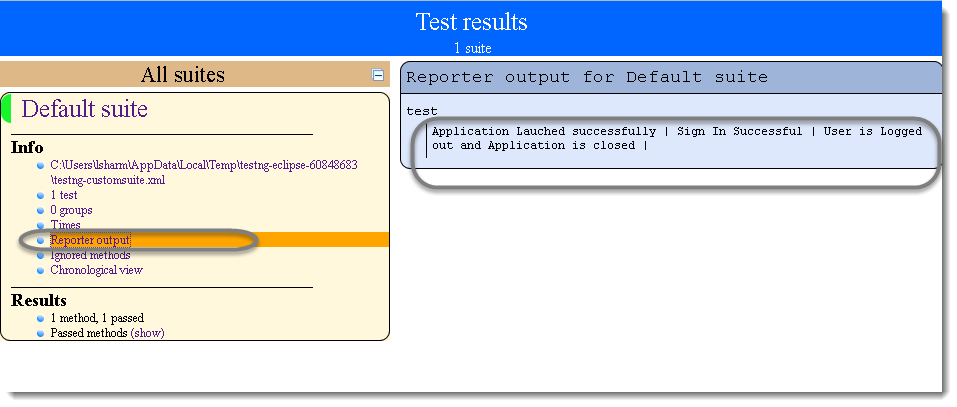
|  |  |
| --- | --- |
|  | package automationFramework;   import java.util.concurrent.TimeUnit;   import org.apache.log4j.Logger;   import org.apache.log4j.xml.DOMConfigurator;   import org.openqa.selenium.By;   import org.openqa.selenium.WebDriver;   import org.openqa.selenium.firefox.FirefoxDriver;   import org.testng.Reporter;   import org.testng.annotations.Test;   import utility.Log;    public class ReporterLogs {    private static WebDriver driver;    private static Logger Log = Logger.getLogger(Log.class.getName());       @Test    public static void test() {    DOMConfigurator.configure("log4j.xml");           driver = new FirefoxDriver();           Log.info("New driver instantiated");           driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);           Log.info("Implicit wait applied on the driver for 10 seconds");           driver.get("http://www.store.demoqa.com");           Log.info("Web application launched");           // Our first step is complete, so we produce a main event log here for our reports.           Reporter.log("Application Lauched successfully | ");           driver.findElement(By.xpath(".//\*[@id='account']/a")).click();            Log.info("Click action performed on My Account link");           driver.findElement(By.id("log")).sendKeys("testuser\_1");           Log.info("Username entered in the Username text box");           driver.findElement(By.id("pwd")).sendKeys("Test@123");           Log.info("Password entered in the Password text box");           driver.findElement(By.id("login")).click();           Log.info("Click action performed on Submit button");           // Here we are done with our Second main event           Reporter.log("Sign In Successful | " );           driver.findElement(By.id("account\_logout"));           Log.info("Click action performed on Log out link");           driver.quit();           Log.info("Browser closed");           // This is the third main event           Reporter.log("User is Logged out and Application is closed | ");    }   } |

3) Run the test by right click on the test case script and select **Run As** > **TestNG Test**.

Your Log4j logging output will look like this:



But your Reporters log will look like this:



Log4j logging will help you to report a bug or steps taken during the test, on the other hand reporters log will help you to share the test status with leadership.

As leadership is just interested in the test results, not the test steps.

## TestNG Parameters

Everybody knows the importance of Parameterization in testing and in automation testing. It allows us to automatically run a test case multiple times with different input and validation values.

As Selenium Webdriver is more an automated testing framework than a ready-to-use tool, you will have to put in some effort to support data driven testing in your automated tests. I usually prefer to use Microsoft Excel as the format for storing my parameters but so many of my followers have requested to write an article on TestNG Data Provider.

TestNG again gives us another interesting feature called **TestNG Parameters**.

TestNG lets you pass parameters directly to your test methods with your testng.xml.

**How to do it…**

Let me take a very simple example of LogIn application, where the username and password is required to clear the authentication.

1) Create a test on my demo [OnlineStore](http://www.store.demoqa.com/)application to perform LogIn which takes the two string argument as username & password.

2) Provide Username & Password as parameter using TestNG Annotation.

package automationFramework;

 import java.util.concurrent.TimeUnit;

 import org.openqa.selenium.By;

 import org.openqa.selenium.WebDriver;

 import org.openqa.selenium.firefox.FirefoxDriver;

 import org.testng.annotations.Test;

 import org.testng.annotations.Parameters;

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

3) The parameter would be passed values from testng.xml which we will see in the next step.

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

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

# Multi Browser, Cross Browser & Parallel Testing using TestNG

When the time comes to turn your site from mock-up to something fully functional, you’ll want to make sure that it works great for everyone visiting your site whether they’re using Internet Explorer, Firefox, or any other browser. Testing your website with multiple combinations of browsers is known as **Cross Browser** testing.

Your site will look different in different browsers. That’s because browsers understand some code slightly differently. Your designer should be testing to make sure that your site works well in all modern browsers. But as a tester we need to make sure that functionality should at least tested on Internet Explorer, Firefox, Safari & Google Chrome browser.

## Multi Browser Testing using Selenium TestNG

In every project it is required to perform multi-browser testing to make sure that the functionality is working as expected with every browser to give equal user experience to all of the wide range of audience. It takes a considerable time to test everything on every browser and when we have used automation to reduce the testing efforts then why don’t we perform the multi-browser testing using automation. TestNG gives us functionality to perform same test on different browsers in a simple and easy way.

**How to do it**

1) Create your Script to test a LogIn application using TestNG class.

2) Pass ‘Browser Type’ as parameters using TestNG annotations to the before method of the TestNG class. This method will launch only the browser, which will be provided as parameter.

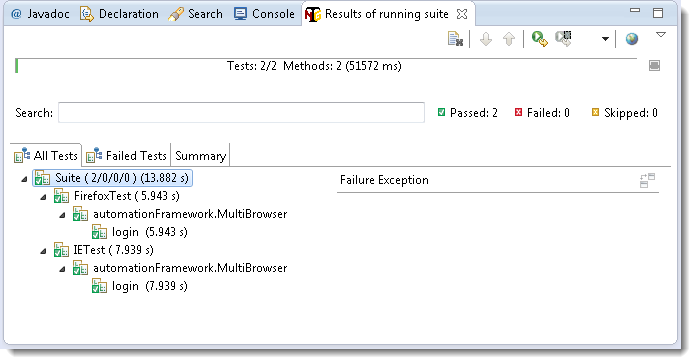
|  |  |
| --- | --- |
|  | package automationFramework;   import org.openqa.selenium.By;   import org.openqa.selenium.WebDriver;   import org.openqa.selenium.firefox.FirefoxDriver;   import org.openqa.selenium.ie.InternetExplorerDriver;   import org.testng.annotations.AfterClass;   import org.testng.annotations.BeforeClass;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

3)Create a TestNG XML for running your test. Configure the TestNG XML for passing parameters i.e. to tell which browser should be used for Running the Test.

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?>   <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">   <suite name="Suite" parallel="none">    <test name="FirefoxTest">    <parameter name="browser" value="firefox" />    <classes>    <class name="automationFramework.MultiBrowser" />    </classes>    </test>    <test name="IETest">    <parameter name="browser" value="ie" />    <classes>    <class name="automationFramework.MultiBrowser" />    </classes>    </test>   </suite> |

**Note:**You can set any number of Browsers here and just for the example purpose I have set up only two main browsers.

4) Now it’s time to run the xml. Run the test by right click on the testng.xml file and select **Run As**> **TestNG Suite**.

****

**Note:**TestNg will execute the test one by one. You may like to perform parallel tests, next topic will cover that.

**Parallel Tests using TestNG**

Using the feature provided by TestNG for Parallel Executions, just take the above example for Sign In application with two different browsers. This time all we want is to execute test in both browsers simultaneously.

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

|  |  |
| --- | --- |
|  | <?xml version="1.0" encoding="UTF-8"?>   <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">   <suite name="Suite" parallel="tests">    <test name="FirefoxTest">    <parameter name="browser" value="firefox" />    <classes>    <class name="automationFramework.MultiBrowser" />    </classes>    </test>    <test name="IETest">    <parameter name="browser" value="ie" />    <classes>    <class name="automationFramework.MultiBrowser" />    </classes>    </test>   </suite> |

**Note:**You may see some intermittent issues using parallel testing. I will not recommend you this rather run one by one only.

**Data Driven Testing**

**Udemy tutorial reference:**

**Data driven Testing**

Section 19

A key benefit of automating functional testing is the ability to test large volumes of data on the system quickly. But you must be able to manipulate the data sets, perform calculations, and quickly create hundreds of test iterations and permutations with minimal effort. Test Automation Frameworks must have capability to integrate with spreadsheets and provide powerful calculation features.

|  |  |
| --- | --- |
|  | Apache POI (Excel) – Data Driven Testing  Most commercial automated software tools on the market support some sort of data driven testing, which allows you to automatically run a test case multiple times with different input and validation values. As Selenium Webdriver is more an automated testing framework than a ready-to-use tool, you will have to put in some effort to support data driven testing in your automated tests. I usually prefer to use Microsoft Excel as the format for storing my parameters. An additional advantage of using Excel is that you can easily outsource the test data administration to someone other than yourself, someone who might have better knowledge of the test cases that need to be run and the parameters required to execute them. |
|  |  |

**Example Tutorial (Sec. 19) - Reading Data from Excel File**

We need a way to open this Excel sheet and read data from it within our Selenium test script. For this purpose, we use the Apache POI library, which allows us to read, create and edit Microsoft Office-documents using Java.

The classes and methods we are going to use to read data from Excel sheet are located in the org.apache.poi.hssf.usermodel package.

package datadriven;

import java.io.FileInputStream;

import org.apache.poi.xssf.usermodel.XSSFCell;

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

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

public class ExcelRead {

// URL to get the binary - http://poi.apache.org/download.html

// Binary Name - poi-bin-3.11-beta2-20140822.zip

// Extract the binary

// Add all the jars from the location you extracted to the build path

// Also add all the jars from lib, do not add the jar file of log4j

// Also add all the jars from ooxml-lib

// Only works for Excel 2007+

public static void main(String[] args) {

XSSFWorkbook ExcelWBook;

XSSFSheet ExcelWSheet;

XSSFCell Cell;

// Location of the Excel file

String path = "//Users//atomar//Documents//workspace\_personal//SeleniumWD2Tutorial//src//utilities//ExcelRead.xlsx";

String sheetName = "Sheet1";

try {

FileInputStream ExcelFile = new FileInputStream(path);

ExcelWBook = new XSSFWorkbook(ExcelFile);

ExcelWSheet = ExcelWBook.getSheet(sheetName);

Cell = ExcelWSheet.getRow(1).getCell(2);

String cellData = Cell.getStringCellValue();

System.out.println("Cell Data: " + cellData);

} catch (Exception e) {

e.printStackTrace();

}

}

}

## How to do it…

* 1. [Download JAR files](http://toolsqa.com/selenium-webdriver/download-apache-poi/) of Apache POI  and [Add Jars](http://toolsqa.com/selenium-webdriver/add-apache-poi-jars/) to your project library. You can download it from [here](http://poi.apache.org/). That’s all about configuration of Apache POI with eclipse. Now you are ready to write your test.
  2. Create a ‘[**New Package**](http://toolsqa.com/selenium-webdriver/configure-eclipse-with-selenium-webdriver/#package)‘ file and name it as **‘DataDriven’** , by right click on the Project and select **New** > **Package. Place the ‘UsingExcel’ file test case into it**

package datadriven;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Assert;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

import utilities.Constants;

import utilities.ExcelUtility;

public class UsingExcel {

private WebDriver driver;

@BeforeClass

public void setUp() throws Exception {

driver = new FirefoxDriver();

// Maximize the browser's window

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

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

driver.get(Constants.URL);

driver.findElement(By.xpath("//span[text()='Learn Now']")).click();

// Tell the code about the location of Excel file

ExcelUtility.setExcelFile(Constants.File\_Path + Constants.File\_Name, "LoginTests");

}

@DataProvider(name = "loginData")

public Object[][] dataProvider() {

Object[][] testData = ExcelUtility.getTestData("Invalid\_Login");

return testData;

}

@Test(dataProvider="loginData")

public void testUsingExcel(String username, String password) throws Exception {

// Click login button

driver.findElement(By.xpath("//div[@id='navbar']//a[contains(text(),'Login')]")).click();

Thread.sleep(2000);

// Enter username

driver.findElement(By.id("user\_email")).sendKeys(username);

// Enter password

driver.findElement(By.id("user\_password")).sendKeys(password);

// Click Login button

driver.findElement(By.name("commit")).click();

Thread.sleep(2000);

// Find if error messages exist

boolean result = driver.findElements(By.xpath("//form[@id='new\_user']//div[3]")).size() != 0;

Assert.assertTrue(result);

}

@AfterClass

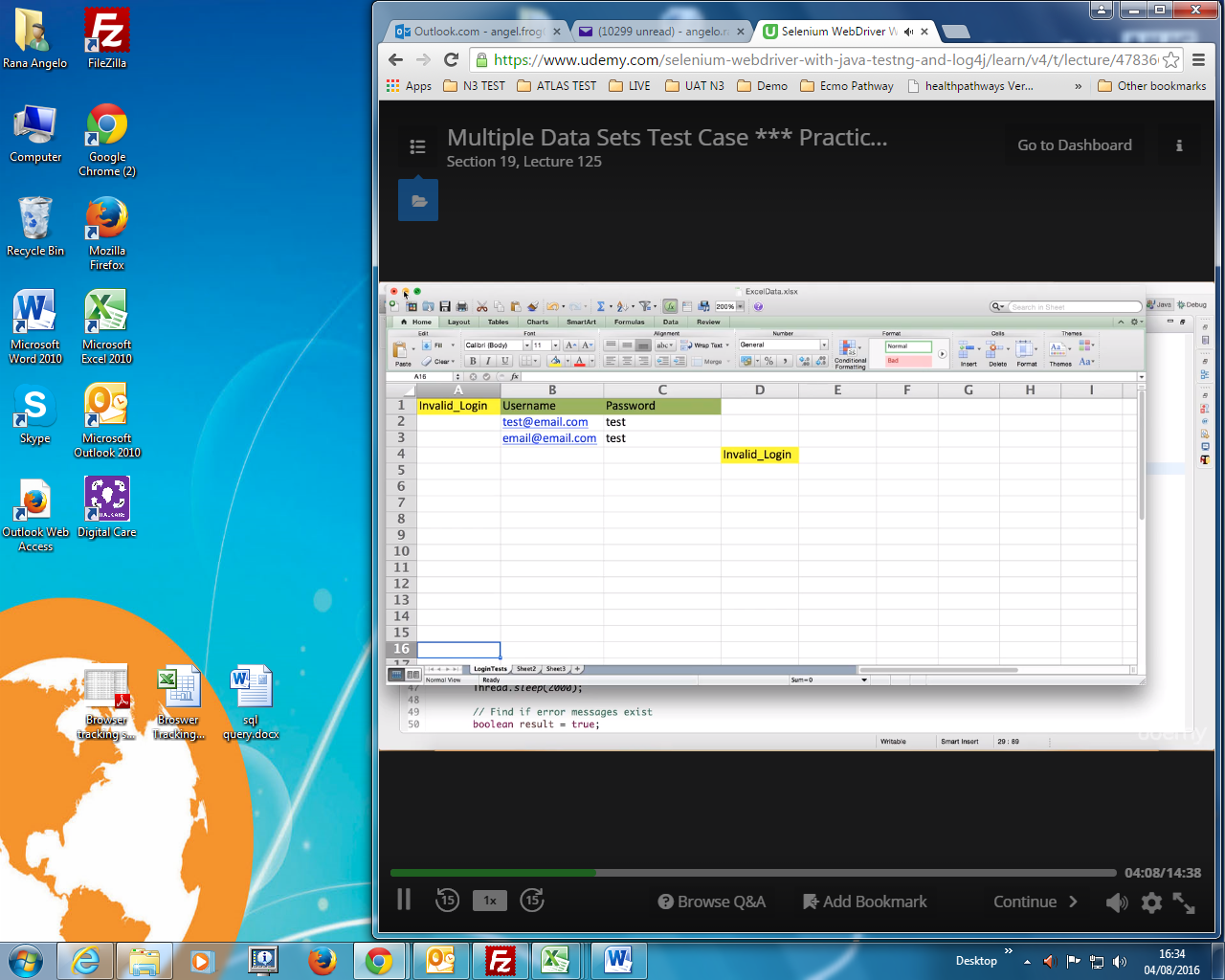
public void tearDown() throws Exception {

//driver.quit();

}

}

* 1. Place an **Excel** file in the utilities package location and save it as **ExcelData.xlsx**. Fill the data in the excel like below image:



* 1. Add two constant variables in the **Constants** class.

package utilities;

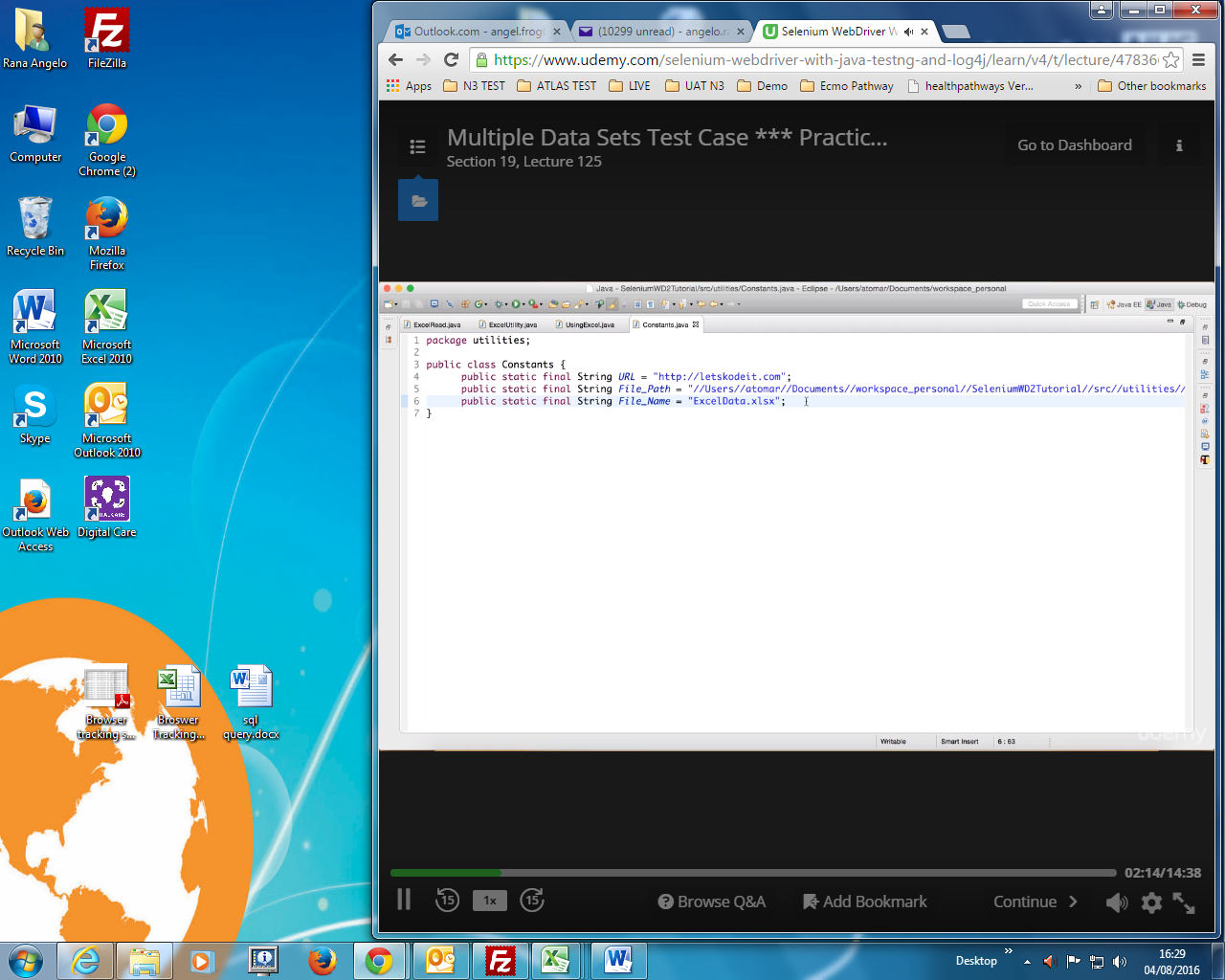
public class Constants {

public static final String URL = "http://letskodeit.com";

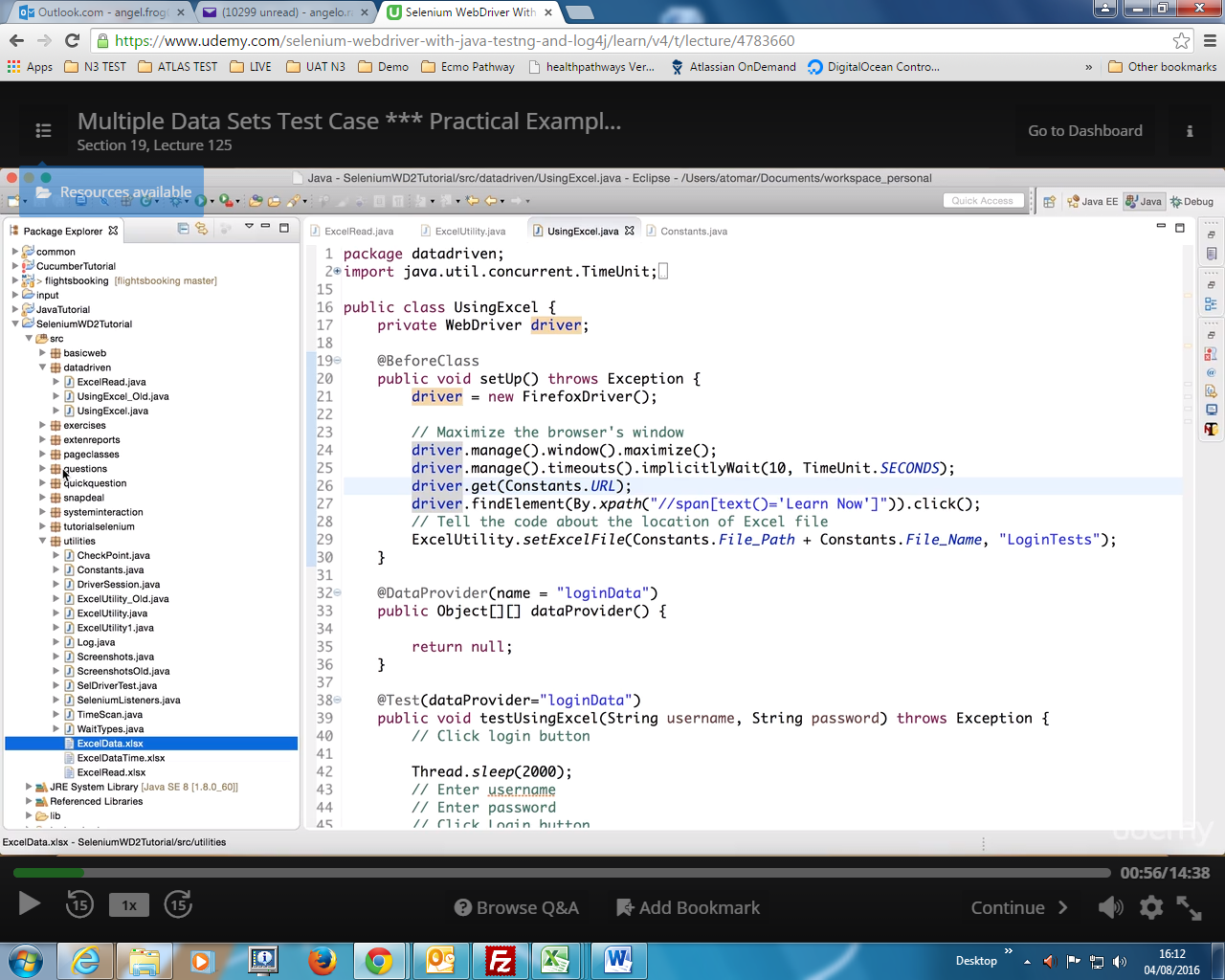
public static final String File\_Path = "//Users//atomar//Documents//workspace\_personal//SeleniumWD2Tutorial//src//utilities//";

public static final String File\_Name = "ExcelData.xlsx";

}



* 1. Create a ‘[**New Class**](http://toolsqa.com/selenium-webdriver/configure-eclipse-with-selenium-webdriver/#Class)‘ file, by right click on the ‘**utilities**‘ Package and select **New** > **Class**and name it as ‘**ExcelUtility**‘**.** First we will write basic read/write methods.



**Maven**

**Udemy tutorial reference:**

**Build Management with Maven**

Section 24

## What is Maven?

Maven is a **build tool** and it performs task just like Ant which is again a diffirent build Tool. It is a software **project management tool** which provides new concept of **project object model (POM).** Maven allows the developer to automate the process of the creation of the initial folder structure, performing the compilation and testing and the packaging and deployment of the final product. It cuts down the good number of steps in build process and it makes it one step process to do a build.

Only using Maven would not have any significance in Selenium life. This tool has to integrate with Jenkins as well to give any benifits to Selenium Automation. You may face N number of issues with installing and in setting up Maven & Jenkins which would be difficult to resolve.

## Why Maven is Used?

It cuts down the tasks in build process. To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly.

Maven increases reusability and takes care of most of build related tasks. It helps in bypassing steps like adding jars to the project library, building reports, executing Junits test cases, creating Jar, War Ear files for the project deployment and many more. A very significant aspect of Maven is the use of repositories to manage jar files.

**Maven Local Repository**

This is the place where Maven stores all the project jars files or libraries or dependencies. By deault the folder name is ‘***.m2***‘ and by default the location in windows 7 is ‘***Libraries\Documents\.m2***‘.

### Maven Central Repository

Maven central repository is the default location ‘***http://mvnrepository.com/***‘ for Maven to download all the project dependency libraries. Fo any library required in the project, Maven first look in to the .m2 folderof Local Repository, if it does not find the required libarary then it looks in Central Repository and download the libabry in to local repository.

### Dependency Keyword

Dependencies are the libraries, which are required by the project.

For example Log4j jars, Apache Poi jars, Selenium Jars etc. Dependencies are mentioned in the Maven pom.xml file like this:

|  |  |
| --- | --- |
| 1  2  3  4  5 | <dependency>     <groupId>org.seleniumhq.selenium</groupId>     <artifactId>selenium-java</artifactId>     <version>2.43.1</version>     </dependency> |

### Surefire Plugin

The Surefire Plugin is used during the test phase of the build lifecycle to execute the unit tests of an application. It generates reports in 2 different file formats like plain text file, xml files and html files as well. Even if you are using TestNG or Junits framework for reporting, this plugin is must to use, as it helps Maven to identify tests.

**Maven POM**

POM is Project Object Model XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on.

There are two ways to install Maven. One which is little difficult than the other is installing Maven through command line. And the other easy way out is installing Maven with in Eclipse. In this chapter I will show how to install Maven from Eclipse IDE and in case you face any issue during installation, you are most welcome to go for the other way which is the next chapter.

<http://toolsqa.com/java/maven/how-to-install-maven-on-windows/>

# Database Testing using Selenium: MySQL

**Udemy tutorial reference:**

[**https://www.tutorialspoint.com/jdbc/index.htm**](https://www.tutorialspoint.com/jdbc/index.htm)

**Database Testing**

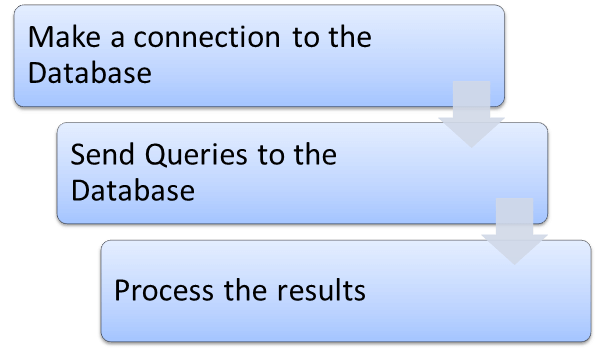
Section 26

Selenium Webdriver is limited to testing your applications using Browser. To use Selenium Webdriver for Database Testing you need to use the JDBC ("Java Database Connectivity").

JDBC (Java Database Connectivity) is a SQL level API that allows you to execute SQL statements. It is responsible for the connectivity between the Java Programming language and a wide range of databases. The JDBC API provides the following classes and interfaces

* Driver Manager
* Driver
* Connection
* Statement
* ResultSet
* SQLException

In order to test your Database using Selenium, you need to observe the following 3 steps1. Make a connection to the Database2. Send Queries to the Database3. Process the results

[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes1.png)

### 1) Make a connection to the Database

In order to make a connection to the database the syntax is

DriverManager.getConnection(URL, "userid", "password" )

Here,

* Userid is the username configured in the database
* Password of the configured user
* URL is of format jdbc:< dbtype>://ipaddress:portnumber/db\_name"
* <dbtype>- The driver for the database you are trying to connect. To connect to oracle database this value will be "oracle"

For connecting to database with name "emp" in MYSQL URL will bejdbc:mysql://localhost:3036/emp

And the code to create connection looks like

Connection con = DriverManager.getConnection(dbUrl,username,password);

You also need to load the JDBC Driver using the code

Class.forName("com.mysql.jdbc.Driver");

### 2) Send Queries to the Database

Once connection is made, you need to execute queries.

You can use the Statement Object to send queries.

Statement stmt = con.createStatement();

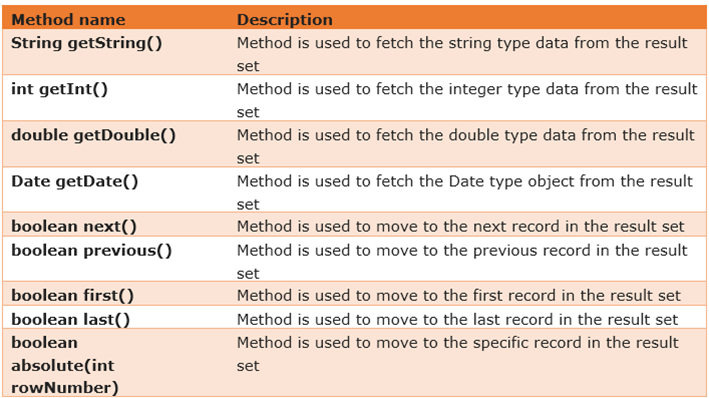
Once the statement object is created use the executeQuery method to execute the SQL queries

stmt.executeQuery(select \* from employee;);

### 3) Process the results

Results from the executed query are stored in the ResultSet Object.

Java provides loads of advance methods to process the results. Few of the methods are listed below

[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes2.png)

### Example of Database Testing with Selenium

**Step 1)** Install [MySQL Server](http://dev.mysql.com/downloads/mysql/)and [MySQL Workbench](http://dev.mysql.com/downloads/workbench/)

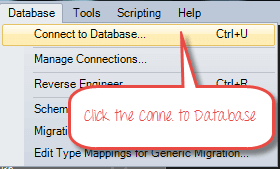
While installing MySQL Server, please note the database

* Username
* Password
* Port Number

It will be required in further steps.

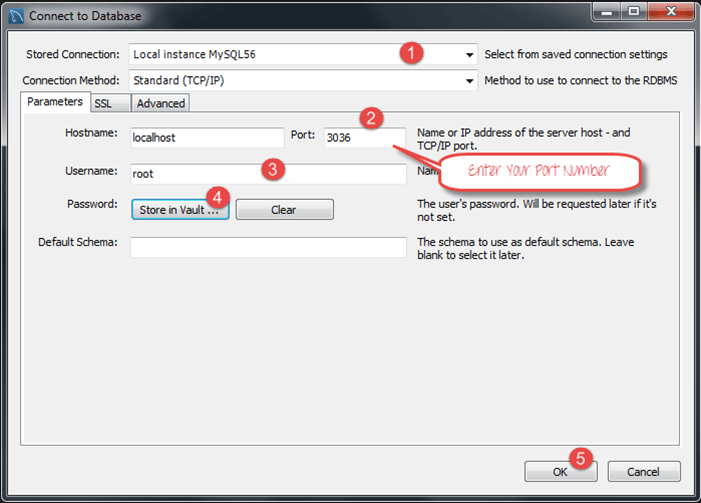
MySQL Workbench makes it easy to administer the database without the need to code SQL. Though, you can also use the MySQL Terminal to interact with the database.

**Step 2)** In MySQL WorkBench, connect to your MySQL Server

[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes3.png)

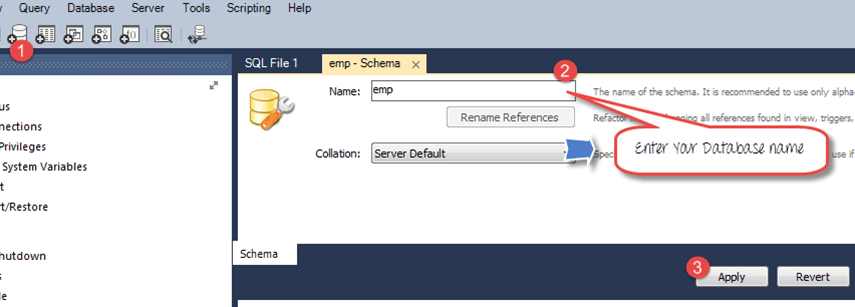
In the next screen,

1. Select Local Instance of MySQL
2. Enter Port Number
3. Enter Username
4. Enter Password
5. Click OK

[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes4.png)

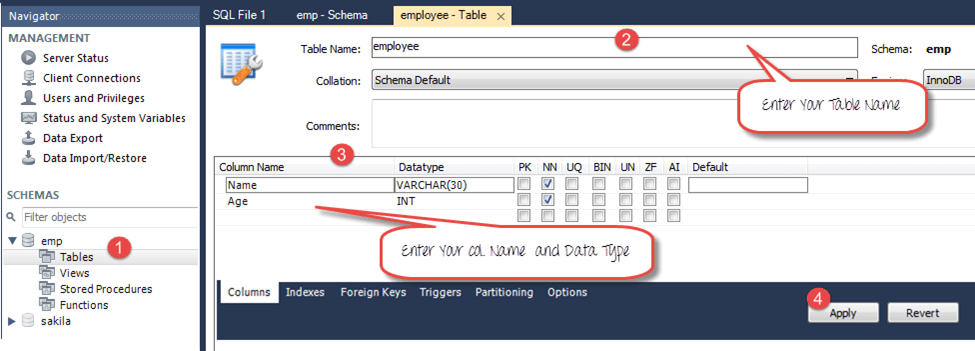
**Step 3)** To Create Database,

1. Click create Schema Button
2. Enter Name of Schema/Database
3. Click Apply

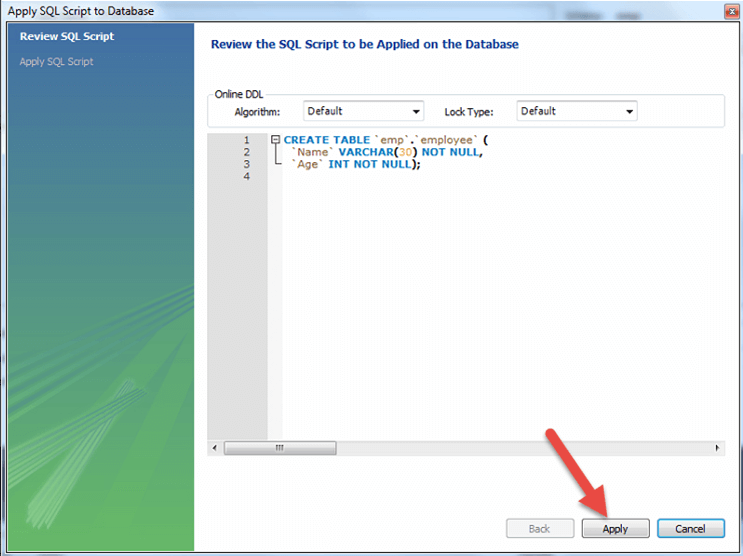
[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes5.png)

**Step 4)** In the navigator menu,

1. Click on Tables, beneath the emp database
2. Enter Table name as employee
3. Enter Fields as Name and Age
4. Click Apply

[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes6.png)

You will see the following pop-up. Click Apply

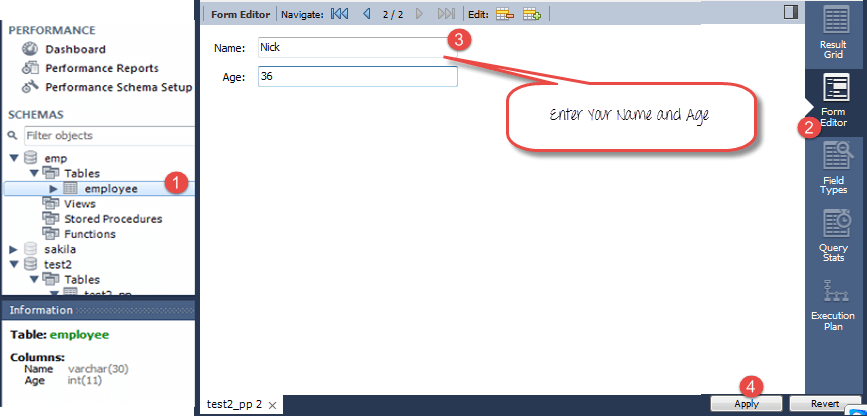
[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes7.png)

**Step 5)** We will create following data

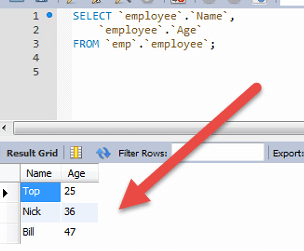
|  |  |
| --- | --- |
| **Name** | **Age** |
| Top | 25 |
| Nick | 36 |
| Bill | 47 |

To create data into the Table

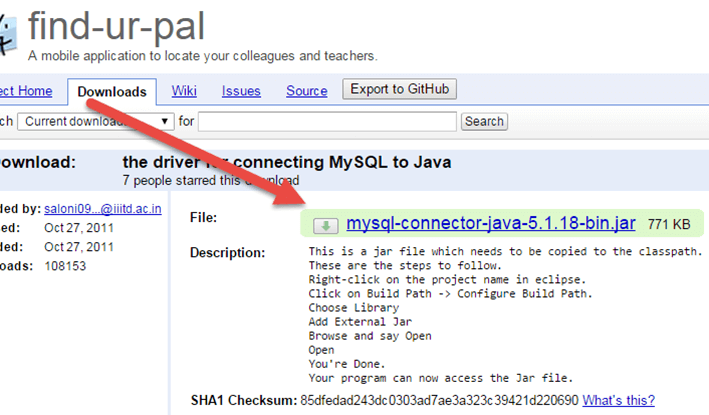
1. In navigator, select the employee table
2. In right pane, click Form Editor
3. Enter Name and Age
4. Click Apply

[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes8.png)

Repeat the process until all data is created

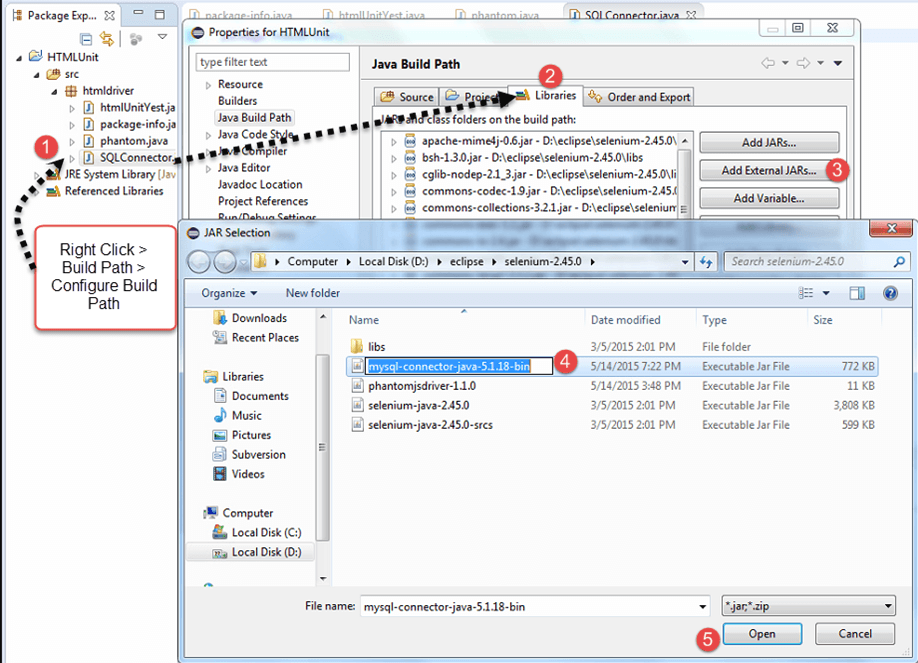
[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes9.png)

**Step 6)**Download the MySQL JDBC connector [here](https://code.google.com/p/find-ur-pal/downloads/detail?name=mysql-connector-java-5.1.18-bin.jar&)

[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes10.png)

**Step 7)**Add the downloaded Jar to your Project

1. Right click on your Java File. Then click on Build Pathà Configure build path
2. Select the libraries
3. Click on add external JARs
4. You can see MySQL connector java in your library
5. Click on open to add it to the project

[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes11.png)

**Step 8)**Copy the following code into the editor

Package htmldriver;

import java.sql.Connection;

import java.sql.Statement;

import java.sql.ResultSet;

import java.sql.DriverManager;

import java.sql.SQLException;

public class SQLConnector {

public static void main(String[] args) throws ClassNotFoundException, SQLException {

//Connection URL Syntax: "jdbc:mysql://ipaddress:portnumber/db\_name"

String dbUrl = "jdbc:mysql://localhost:3036/emp";

//Database Username

String username = "root";

//Database Password

String password = "guru99";

//Query to Execute

String query = "select \* from employee;";

//Load mysql jdbc driver

Class.forName("com.mysql.jdbc.Driver");

//Create Connection to DB

Connection con = DriverManager.getConnection(dbUrl,username,password);

//Create Statement Object

Statement stmt = con.createStatement();

// Execute the SQL Query. Store results in ResultSet

ResultSet rs= stmt.executeQuery(query);

// While Loop to iterate through all data and print results

while (rs.next()){

String myName = rs.getString(1);

String myAge = rs.getString(2);

System. out.println(myName+" "+myAge);

}

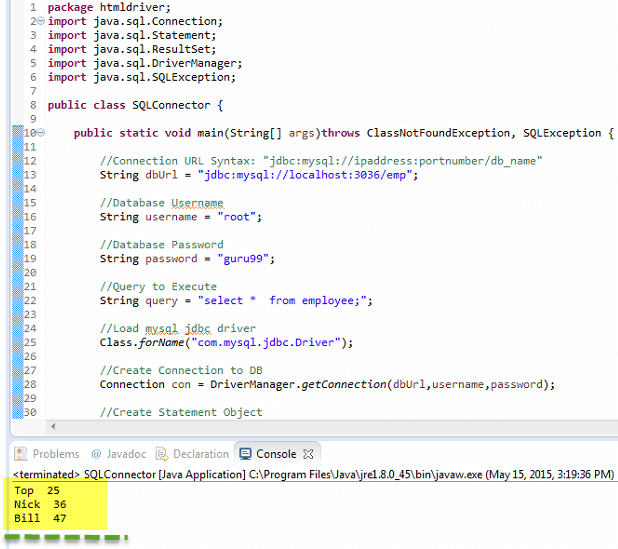
// closing DB Connection

con.close();

}

}

**Step 8)**Execute the code, and check the output

[](http://cdn.guru99.com/images/6-2015/052615_1013_DatabaseTes12.png)

# Database Testing using Selenium: MongoDB

**Udemy tutorial reference:**

<http://www.unityjdbc.com/doc/mongo/mongo_java_jdbc.php#getdatabase>

<https://docs.mongodb.com/manual/reference/operator/query-comparison/#comparison-query-operators>

# How to Query MongoDB

## Overview

MongoDB is an open source [document-oriented database](https://en.wikipedia.org/wiki/Document-oriented_database) that stores data as [BSON](http://en.wikipedia.org/wiki/BSON) (a binary-encoded serialization of [Java Script Object Notation (JSON)](http://en.wikipedia.org/wiki/JSON)) documents and supports dynamic data with no fixed schemas rather than relational tables. Interacting with MongoDB (a [NoSQL](https://en.wikipedia.org/wiki/NoSQL) system) is not done using SQL but rather using query APIs for various languages.

**This article will cover how to access MongoDB using the Java driver.**

## Step-by-Step JDBC Configuration

1. Download Java Driver
2. Put Driver in Classpath
3. Establish A Connection
4. Get Database
5. Get Collections
6. Getting A Single Document
7. Getting Multiple Documents
8. Inserting Documents
9. Save, Update, Find, Delete

## Download MongoDBJava Driver

The first thing you will need to do is download the MongoDB Java Driver. [Click here to download](http://central.maven.org/maven2/org/mongodb/mongo-java-driver/).

## Put Driver in Classpath

Once you've downloaded the Java driver, put the JAR in your Java CLASSPATH. A common location is in the <JRE\_HOME>\lib\ext folder of your Java JRE installation.

## Establish A Connection

To make a connection, we will need to use the MongoClient class instance. The MongoClient class is thread safe which means only one instance is needed even if multiple threads are being used..

Note: You will need to know the IP address and port of the database you wish to connect to. If the database does not exist, MongoDB will create it for you.

Use the following code to make your connection.

import com.mongodb.MongoClient;

import com.mongodb.MongoException;

import com.mongodb.WriteConcern;

import com.mongodb.DB;

import com.mongodb.DBCollection;

import com.mongodb.BasicDBObject;

import com.mongodb.DBObject;

import com.mongodb.DBCursor;

import com.mongodb.ServerAddress;

MongoClient mongoClient = new MongoClient( "localhost" , 27017 );

DB db = mongoClient.getDB( "mydb" );

### User authentication

If you are running MongoDB in secure mode you will need user authentication to complete the connection. If so, use the following sample code:

MongoClient mongoClient = new MongoClient();

DB db = mongoClient.getDB("test");

boolean auth = db.authenticate(myUserName, myPassword);

To clean up your resources and dispose of an instance call:

mongoClient.close();

## Get Database

Once a connection is established you can then use the getDB() method to get a database by name.

DB db = mongoClient.getDB( "mydb" );

To display all databases:

List dbs = mongo.getDatabaseNames();

for(String db : dbs)

{System.out.println(db);}

## Getting Collections

Getting collections is a key step to being able to query and manipulate data. You can retrieve a colleciton by name usinggetCollection().

DBCollection coll = db.getCollection("testCollection");

To retrieve a list of collection names in the database:

Set colls = db.getCollectionNames();

for (String s : colls)

{System.out.println(s);}

## Getting A Single Document

If you want to retrieve a single document from a collection you can use the find() method.

**For Example**: If you want to find the document for which the value of "i" is 52, we can use the following:

BasicDBObject query = new BasicDBObject("i", 52);

cursor = coll.find(query);

while(cursor.hasNext())

{ System.out.println(cursor.next());}

cursor.close();

Note that $ operators used within MongoDB documentation are represented as DBObjects when using the Java driver. Example:

db.things.find({j: {$ne: 3}, k: {$gt: 10} });

## Getting Multiple Documents

Multiple documents can be retrieved using a condition built from BasicDBObject. This example will return all documents where i > 52.

query = new BasicDBObject("i", new BasicDBObject("$gt", 52)); // e.g. find all where i > 52

cursor = coll.find(query);

## Inserting Documents

Once you have your collection object you can then choose to insert documents by byuilding a BasicDBObject.

The following sample code is an example of a JSON document.

{

"name" : "MongoDB",

"type" : "database",

"count" : 1,

"info" : {

x : 203,

y : 102

}

}

To build this using the BasicDBObject you need to use the insert() method to add it to the collection.

BasicDBObject doc = new BasicDBObject("name", "MongoDB").

append("type", "database").

append("count", 1).

append("info", new BasicDBObject("x", 203).append("y", 102));

coll.insert(doc);

## Save, Update, Find, & Delete

The following examples demonstrate how to save, update, find, and delete documents.

**Save (Insert) a document**

DBCollection col = db.getCollection("mycol");

BasicDBObject doc = new BasicDBObject();

doc.put("name", "Joe Smith");

doc.put("age", 25);

doc.put("createdDate", new Date());

col.insert(document);

**Update a document:**

DBCollection col = db.getCollection("mycol");

BasicDBObject query = new BasicDBObject();

query.put("name", "MongoDB");

BasicDBObject newDocument = new BasicDBObject();

newDocument.put("name", "MongoDB-updated");

BasicDBObject updateObj = new BasicDBObject();

updateObj.put("$set", newDocument);

col.update(query, updateObj);

**Perform a query and list matching documents:**

DBCollection col = db.getCollection("mycol");

BasicDBObject searchQuery = new BasicDBObject();

searchQuery.put("name", "Joe Smith"); // Find documents with Joe Smith in name field

DBCursor cursor = table.find(searchQuery);

while (cursor.hasNext())

{System.out.println(cursor.next());}

**Delete documents that match a given criteria:**

DBCollection table = db.getCollection("user");

BasicDBObject searchQuery = new BasicDBObject();

searchQuery.put("name", "Joe Smith"); // Find documents with Joe Smith in name field

table.remove(searchQuery); // Delete those matching documents

**Example query document with MongoDB**

<http://www.mkyong.com/mongodb/java-mongodb-query-document/>

## Test Data

Insert 5 dummy documents for testing.

{ "\_id" : { "$oid" : "id"} , "number" : 1 , "name" : "mkyong-1"}

{ "\_id" : { "$oid" : "id"} , "number" : 2 , "name" : "mkyong-2"}

{ "\_id" : { "$oid" : "id"} , "number" : 3 , "name" : "mkyong-3"}

{ "\_id" : { "$oid" : "id"} , "number" : 4 , "name" : "mkyong-4"}

{ "\_id" : { "$oid" : "id"} , "number" : 5 , "name" : "mkyong-5"}

## 1. Find() examples

1.1 Get first matched document only.

DBObject doc = collection.findOne();

System.out.println(dbObject);

Output

{ "\_id" : { "$oid" : "id"} , "number" : 1 , "name" : "mkyong-1"}

1.2 Get all matched documents.

DBCursor cursor = collection.find();

while(cursor.hasNext()) {

System.out.println(cursor.next());

}

Output

{ "\_id" : { "$oid" : "id"} , "number" : 1 , "name" : "mkyong-1"}

{ "\_id" : { "$oid" : "id"} , "number" : 2 , "name" : "mkyong-2"}

{ "\_id" : { "$oid" : "id"} , "number" : 3 , "name" : "mkyong-3"}

{ "\_id" : { "$oid" : "id"} , "number" : 4 , "name" : "mkyong-4"}

{ "\_id" : { "$oid" : "id"} , "number" : 5 , "name" : "mkyong-5"}

1.3 Get single field from matched document.

BasicDBObject allQuery = new BasicDBObject();

BasicDBObject fields = new BasicDBObject();

fields.put("name", 1);

DBCursor cursor = collection.find(allQuery, fields);

while (cursor.hasNext()) {

System.out.println(cursor.next());

}

Output

{ "\_id" : { "$oid" : "id"} , "name" : "mkyong-1"}

{ "\_id" : { "$oid" : "id"} , "name" : "mkyong-2"}

{ "\_id" : { "$oid" : "id"} , "name" : "mkyong-3"}

{ "\_id" : { "$oid" : "id"} , "name" : "mkyong-4"}

{ "\_id" : { "$oid" : "id"} , "name" : "mkyong-5"}

## 2. Find() and Comparison

2.1 Get all documents where number = 5.

BasicDBObject whereQuery = new BasicDBObject();

whereQuery.put("number", 5);

DBCursor cursor = collection.find(whereQuery);

while(cursor.hasNext()) {

System.out.println(cursor.next());

}

Output

{ "\_id" : { "$oid" : "id"} , "number" : 5 , "name" : "mkyong-5"}

2.2 $in example – Get documents where number in 2, 4 and 5.

BasicDBObject inQuery = new BasicDBObject();

List<Integer> list = new ArrayList<Integer>();

list.add(2);

list.add(4);

list.add(5);

inQuery.put("number", new BasicDBObject("$in", list));

DBCursor cursor = collection.find(inQuery);

while(cursor.hasNext()) {

System.out.println(cursor.next());

}

Output

{ "\_id" : { "$oid" : "id"} , "number" : 2 , "name" : "mkyong-2"}

{ "\_id" : { "$oid" : "id"} , "number" : 4 , "name" : "mkyong-4"}

{ "\_id" : { "$oid" : "id"} , "number" : 5 , "name" : "mkyong-5"}

2.3 $gt $lt example – Get documents where 5 > number > 2 .

BasicDBObject gtQuery = new BasicDBObject();

gtQuery.put("number", new BasicDBObject("$gt", 2).append("$lt", 5));

DBCursor cursor = collection.find(gtQuery);

while(cursor.hasNext()) {

System.out.println(cursor.next());

}

Output

{ "\_id" : { "$oid" : "id"} , "number" : 3 , "name" : "mkyong-3"}

{ "\_id" : { "$oid" : "id"} , "number" : 4 , "name" : "mkyong-4"}

2.4 $ne example – Get documents where number != 4 .

BasicDBObject neQuery = new BasicDBObject();

neQuery.put("number", new BasicDBObject("$ne", 4));

DBCursor cursor = collection.find(neQuery);

while(cursor.hasNext()) {

System.out.println(cursor.next());

}

Output

{ "\_id" : { "$oid" : "id"} , "number" : 1 , "name" : "mkyong-1"}

{ "\_id" : { "$oid" : "id"} , "number" : 2 , "name" : "mkyong-2"}

{ "\_id" : { "$oid" : "id"} , "number" : 3 , "name" : "mkyong-3"}

{ "\_id" : { "$oid" : "id"} , "number" : 5 , "name" : "mkyong-5"}

## 3. find() and Logical

3.1 $and example – get documents where number = 2 and name = 'mkyong-2'.

BasicDBObject andQuery = new BasicDBObject();

List<BasicDBObject> obj = new ArrayList<BasicDBObject>();

obj.add(new BasicDBObject("number", 2));

obj.add(new BasicDBObject("name", "mkyong-2"));

andQuery.put("$and", obj);

System.out.println(andQuery.toString());

DBCursor cursor = collection.find(andQuery);

while (cursor.hasNext()) {

System.out.println(cursor.next());

}

Output

{ "$and" : [ { "number" : 2} , { "name" : "mkyong-2"}]}

{ "\_id" : { "$oid" : "id"} , "number" : 2 , "name" : "mkyong-2"}

## 4. find() and Regex

Find document with regular expression pattern.

4.1 $regex example – get documents where name like pattern 'Mky.\*-[1-3]', case insensitive.

BasicDBObject regexQuery = new BasicDBObject();

regexQuery.put("name",

new BasicDBObject("$regex", "Mky.\*-[1-3]")

.append("$options", "i"));

System.out.println(regexQuery.toString());

DBCursor cursor = collection.find(regexQuery);

while (cursor.hasNext()) {

System.out.println(cursor.next());

}

Output

{ "name" : { "$regex" : "Mky.\*-[1-3]" , "$options" : "i"}}

{ "\_id" : { "$oid" : "515ad59e3004c89329c7b259"} , "number" : 1 , "name" : "mkyong-1"}

{ "\_id" : { "$oid" : "515ad59e3004c89329c7b25a"} , "number" : 2 , "name" : "mkyong-2"}

{ "\_id" : { "$oid" : "515ad59e3004c89329c7b25b"} , "number" : 3 , "name" : "mkyong-3"}

Read this [MongoDB operator documentation](http://docs.mongodb.org/manual/reference/operators/) for complete set of query operators supported in MongoDB.

## 5. Full Example

package com.mkyong.core;

import java.net.UnknownHostException;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.List;

import com.mongodb.BasicDBObject;

import com.mongodb.DB;

import com.mongodb.DBCollection;

import com.mongodb.DBCursor;

import com.mongodb.DBObject;

import com.mongodb.Mongo;

import com.mongodb.MongoException;

public class QueryApp {

public static void insertDummyDocuments(DBCollection collection) {

List<DBObject> list = new ArrayList<DBObject>();

Calendar cal = Calendar.getInstance();

for (int i = 1; i <= 5; i++) {

BasicDBObject data = new BasicDBObject();

data.append("number", i);

data.append("name", "mkyong-" + i);

// data.append("date", cal.getTime());

// +1 day

cal.add(Calendar.DATE, 1);

list.add(data);

}

collection.insert(list);

}

public static void main(String[] args) {

try {

Mongo mongo = new Mongo("localhost", 27017);

DB db = mongo.getDB("yourdb");

// get a single collection

DBCollection collection = db.getCollection("dummyColl");

insertDummyDocuments(collection);

System.out.println("1. Find first matched document");

DBObject dbObject = collection.findOne();

System.out.println(dbObject);

System.out.println("\n1. Find all matched documents");

DBCursor cursor = collection.find();

while (cursor.hasNext()) {

System.out.println(cursor.next());

}

System.out.println("\n1. Get 'name' field only");

BasicDBObject allQuery = new BasicDBObject();

BasicDBObject fields = new BasicDBObject();

fields.put("name", 1);

DBCursor cursor2 = collection.find(allQuery, fields);

while (cursor2.hasNext()) {

System.out.println(cursor2.next());

}

System.out.println("\n2. Find where number = 5");

BasicDBObject whereQuery = new BasicDBObject();

whereQuery.put("number", 5);

DBCursor cursor3 = collection.find(whereQuery);

while (cursor3.hasNext()) {

System.out.println(cursor3.next());

}

System.out.println("\n2. Find where number in 2,4 and 5");

BasicDBObject inQuery = new BasicDBObject();

List<Integer> list = new ArrayList<Integer>();

list.add(2);

list.add(4);

list.add(5);

inQuery.put("number", new BasicDBObject("$in", list));

DBCursor cursor4 = collection.find(inQuery);

while (cursor4.hasNext()) {

System.out.println(cursor4.next());

}

System.out.println("\n2. Find where 5 > number > 2");

BasicDBObject gtQuery = new BasicDBObject();

gtQuery.put("number", new BasicDBObject("$gt", 2).append("$lt", 5));

DBCursor cursor5 = collection.find(gtQuery);

while (cursor5.hasNext()) {

System.out.println(cursor5.next());

}

System.out.println("\n2. Find where number != 4");

BasicDBObject neQuery = new BasicDBObject();

neQuery.put("number", new BasicDBObject("$ne", 4));

DBCursor cursor6 = collection.find(neQuery);

while (cursor6.hasNext()) {

System.out.println(cursor6.next());

}

System.out.println("\n3. Find when number = 2 and name = 'mkyong-2' example");

BasicDBObject andQuery = new BasicDBObject();

List<BasicDBObject> obj = new ArrayList<BasicDBObject>();

obj.add(new BasicDBObject("number", 2));

obj.add(new BasicDBObject("name", "mkyong-2"));

andQuery.put("$and", obj);

System.out.println(andQuery.toString());

DBCursor cursor7 = collection.find(andQuery);

while (cursor7.hasNext()) {

System.out.println(cursor7.next());

}

System.out.println("\n4. Find where name = 'Mky.\*-[1-3]', case sensitive example");

BasicDBObject regexQuery = new BasicDBObject();

regexQuery.put("name",

new BasicDBObject("$regex", "Mky.\*-[1-3]")

.append("$options", "i"));

System.out.println(regexQuery.toString());

DBCursor cursor8 = collection.find(regexQuery);

while (cursor8.hasNext()) {

System.out.println(cursor8.next());

}

collection.drop();

System.out.println("Done");

} catch (UnknownHostException e) {

e.printStackTrace();

} catch (MongoException e) {

e.printStackTrace();

}

}

}

**Example of data validation GUI/Back end test**

package tutorialselenium;

import org.bson.Document;

import org.junit.Assert;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

import com.mongodb.BasicDBObject;

import com.mongodb.MongoClient;

import com.mongodb.client.FindIterable;

import com.mongodb.client.MongoCollection;

import com.mongodb.client.MongoDatabase;

// Download Driver - http://mongodb.github.io/mongo-java-driver/3.0/driver/getting-started/installation-guide/

public class DatabaseTestingMongoDB {

MongoClient mongoClient = null;

MongoDatabase db = null;

@BeforeClass

public void beforeClass() {

try {

// STEP 1: Connect to database

mongoClient = new MongoClient("localhost", 27017);

db = mongoClient.getDatabase("users");

System.out.println("Connect to database successfully");

} catch (Exception e) {

System.err.println(e.getClass().getName() + ": " + e.getMessage());

}

}

@Test

public void test() throws Exception {

try {

// STEP 2: Get Collection

MongoCollection<Document> table = db.getCollection("user\_info");

// STEP 3: Extract Data

BasicDBObject searchQuery = new BasicDBObject();

searchQuery.put("last\_name", "Scott");

FindIterable<Document> cursor = table.find(searchQuery);

// STEP 4: Iterate over data

for (Document obj : cursor) {

String lastName = obj.getString("last\_name");

System.out.println(lastName);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println(obj.toString());

// STEP 5: compare expected results in the GUI with the results by sql query

Assert.assertEquals(expected, "");

}

} catch (Exception e) {

// Handle errors for Class.forName

e.printStackTrace();

}

}

}

**Udemy tutorial reference:**

**Continuous Integration - Jenkins**

Section 25

**Udemy tutorial reference:**

**BDD - Cucumber**

Section 15