Create Lesson:

Subject = Testing

Stream= Selenium

Lesson=Introduction

Create Lesson Topics:

1 lesson – multiple topic

Browser tools to find the Web element’s information:

For different browsers there are different tools to find information about web elements. These tools are helpful in debugging, authoring and profiling.

Given below are the different tools for different browsers:

1. Internet Explorer:

**Firebug:**

**Introduction:**

This is a plug in and is extremely useful in finding Id, Name, CSS etc. of the Web element. In Selenium, Web elements are located by different means. The value obtained by Firebug can be used in locators to find the particular web element.

Installation

Steps:

1. Open the Firefox web browser.

2. Navigate to the link <https://addons.mozilla.org/en-US/firefox/addon/firebug/>

3. Click on the button “Add to Firefox”.

4. Click on the pop up “Allow”.

5. Click on the pop up “Install”.

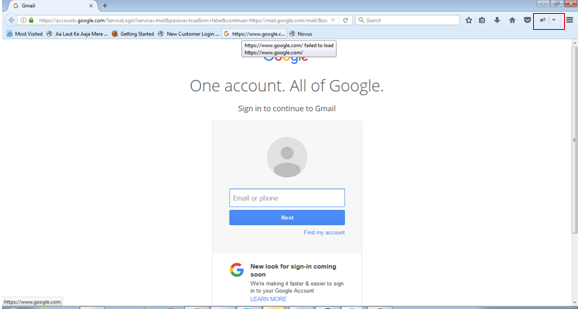
6. Click on the pop up “Ok”.

7. Need not to re launch the browser.

**Usage of the Firebug:**

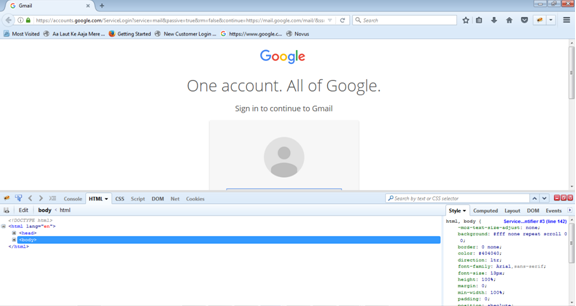
1. Launch web browser Mozilla Firefox.

2. To launch Firebug, click on the Firebug button which is on the right top of the web page, inside red rectangle.

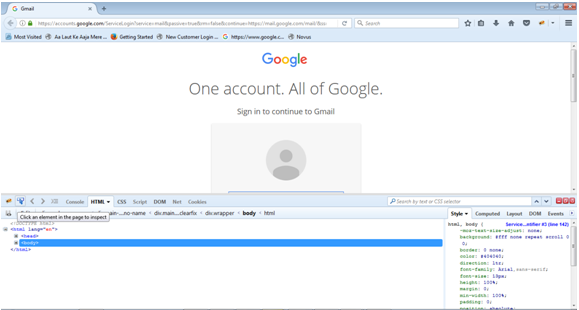


3. Another way to open Firebug is to press F12.

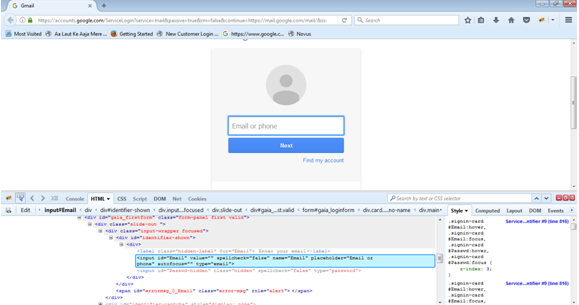
4. Firebug will be opened at the bottom, by default, of the web page.



5. To inspect the page, click on the button as shown below:



6. Hover/ put the cursor on the web element of which you want to get information like Id, Name, CSS etc.



7. HTML code will be highlighted as shown above for the corresponding web element.

8. Corresponding Id, Name or CSS of the web element can be used in our script.

In the above picture, Input is the web element.

Value of attribute “name” and “id” is “Email”

9. Similarly, for “Submit” button, value of “Id” is “next” and “name” is “signIn”.

The attribute like id, name etc can be used in finding the Web element present in the page.

Eg: How to retrieve the web element whose “id” is “next” in a web page using web Driver?

Soln: By.Id(“next”);

Here, Web element with the “id” as “next” will be returned to the caller.

**FirePath:**

**Introduction:**

FirePath is a FireBug extension. This is required to find out the XPath of an element.

**Installation:**

Steps:

1. Navigate to the URL <https://addons.mozilla.org/en-US/firefox/addon/firepath/>

2. Click on button “Add to FireFox”.

3. Click on Pop up “Allow”.

4. Click on next Pop up “Install”.

5. Click on “Restart Now”.

**Usage of the FirePath:**

1. Open FireFox.

2. Go to the URL gmail.com

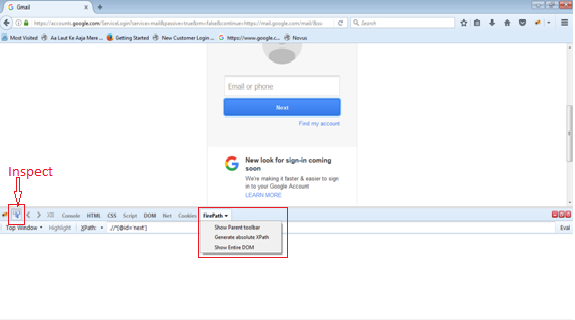
3. Open FireBug.

4. FirePath tab is at the last of the FireBug window. Refer to below picture.

5. Click on Inspect. Refer to below picture.

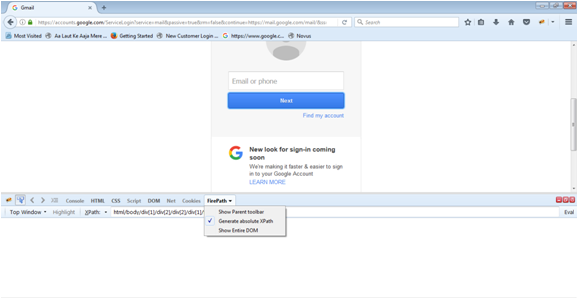
6. Put cursor on the button “Next”. Relative XPATH will be shown as “Generate absolute XPath” is unchecked:

Here, XPath= .//\*[@id='next']



7. To generate absolute path check the option “Generate absolute XPath”.

Here, XPATH = html/body/div[1]/div[2]/div[2]/div[1]/form/div[1]/div/input



# Selenium-IDE

## Introduction

### The Selenium-IDE (Integrated Development Environment) is the tool used to develop Selenium test cases through its record-and-playback functionality. It’s an easy-to-use Firefox plug-in and is generally the most efficient way to develop test cases. This is an excellent way of learning Selenium script syntax

### Pros:

1. It is easier to use.

2. It has capability to convert the test into different languages like Java, HTML etc.

3. Knowledge of programming language is not required.

Cons:

1. It does not support iteration and condition statements.

2. Does not support test script grouping.

Installation

Selenium IDE can be downloaded from Selenium HQ [download page](http://www.seleniumhq.org/download/). It can be used as a Firefox plug in.

In case, Firefox is not installed on System, it can be downloaded from [Download FireFox](http://www.mozilla.org/en-US/firefox/new).

Steps to install Selenium IDE:

1. Open web browser Firefox.

2. Navigate to URL <https://addons.mozilla.org/en-US/firefox/addon/selenium-ide/>

3. Click on Add to Firefox.

4. Click “Allow” on the pop up.

5. Click on “Install”.

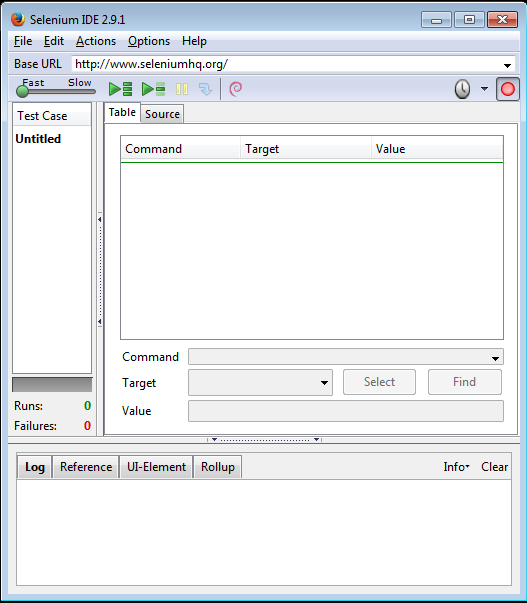
6. Click on “Restart Now”.

## Opening the Selenium IDE[¶](http://www.seleniumhq.org/docs/02_selenium_ide.jsp#opening-the-ide)

1. Go to the Tools menu.

2. Click on the option “Selenium IDE”.

3. Selenium IDE will be launched as shown below:



## Selenium IDE Features

### Menu Bar

1. File menu:

File menu contains options as shown below:

a. New Test Case: Used to create a new test case.

b. Open: Used to open an existing test case

c. Save Test Case: To save a test.

d. Save Test Case As: To save a test with name.

e. New Test Suite: To create a new test suite.

f. Open Test Suite: To open an existing test suite.

2. Edit menu: It has Undo, Cut, Copy, Paste option in it.

3. Actions menu:

a. Record: It is used to record the user actions.

b. Play Entire Test Suite: This option is used to run whole Test suite which

is consist of one or more test cases.

c. Play current test case: This option is used to run current “selected” test case.

4. Options menu:

a.

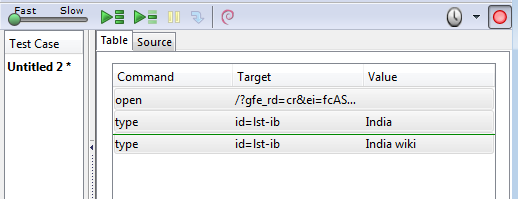
This option helps in converting Selenese commands into particular programming format. For eg. Given below is the sample of converting Selenese commands into Webdriver format.

Step:

1. Select Options -> Clipboard Format -> Java / JUnit 4 / WebDriver

2. Generate a Selenese script.

Eg:



3. Copy the Selenese command as shown below:

Type id=lst-ib India wiki

4. Open Eclipse and paste it on the java file. Output is given below.

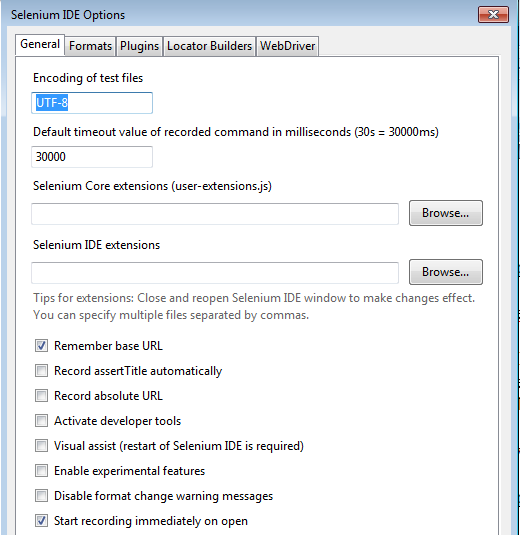
driver.findElement(By.id("lst-ib")).sendKeys("India wiki");

It will not paste a Selenese command. In fact it will paste corresponding Java code. This is the function of the Clipboard format.

By default, HTML will be generated.

b. Options -> Options

To launch this one needs to click of Options tab followed by another Options… tab. This will launch “Selenium IDE Options” window as shown below:



Few of the options given on it is explained as given below:

* Default Timeout Value: This is the time which Selenium waits before the action is performed. By default, it is 30000 ms.
* Remember base URL: If user want Selenium to remember base URL next time, this box needs to be checked
* Autostart record: If this box is checked, Selenium will start recording the actions as soon as IDE is launched.
* Locator builders: This specify the order in which locators are generated. For ex. Here id has higher precedence over name. So, if attribute id is present, Id locator will be generated by the Selenium. If id attribute is not present then name locator will be used and so on.
* Base URL Bar: TBD

5. Help menu:

In this, all the options open up the selenium documentation regarding on different topics.

### Toolbar[¶](http://www.seleniumhq.org/docs/02_selenium_ide.jsp#toolbar)

Toolbar buttons are used to run a test case, to run entire suite or to control the speed of the run. Debugging option is also provided here.

1. Picture

Playback Speed: It is used to control the speed of the execution of the run. Scroll bar towards Fast increase the speed of the execution. Scroll bar towards Slow decrease the speed of the execution.

2. Picture

Play entire Test suite: It is used to run the entire Test suite sequentially. Test suite contains more than one or more Test cases.

3. Picture

Play current test case: It is used to run a currently selected Test case.

4.

Pause/Resume: It is used to Pause and Re-start the execution of the run.

5. Step: It is used to step into each command used in the script.

6.

TestRunner Mode: Allows you to run the test case in a browser loaded with the Selenium-Core TestRunner.

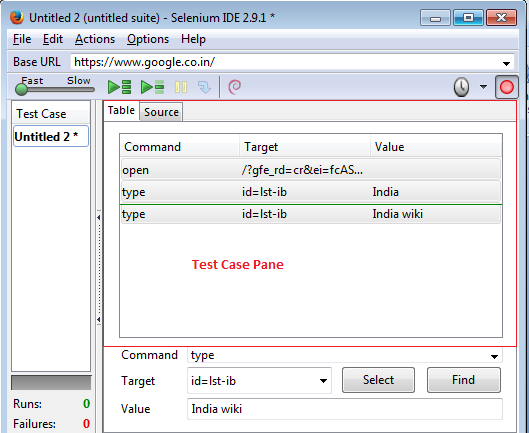
7.

Apply Rollup Rules: This advanced feature allows repetitive sequences of Selenium commands to be grouped into a single action. Detailed documentation on rollup rules can be found in the UI-Element Documentation on the Help menu.

8.

Record: This is used for the start and end of the test run. It records the user action on the GUI and saves it in a script.

### Test Case Pane



* Test case Pane contains the opened Test cases.
* If suite is opened, all the test cases will be enlisted in the pane.
* Selected test case will be bold in font.
* Passed test case is represented in Green color.
* Failed test case is represented in Red color.

**Editor:**

### This is used to display script generated during the recording. There are two tabs, Table and Source, as given below:

### 1. Table format:

### It has three columns in it as shown below. Selenese commands can be created and modified here.

1. Command: To show command like get etc. It displays all the options as user enters text here.

2. Target: Objects where action has to be performed.

3. Value: value to be assigned to the Target object.

2. Source View:

It is used to display the script in the language as preferred like in Java, Python etc. By default it is displayed in HTML. Here too, script can be edited.

Commonly Used Selenium Commands

**open**

opens a page using a URL.

**click/clickAndWait**

performs a click operation, and optionally waits for a new page to load.

**verifyTitle/assertTitle**

verifies an expected page title.

**verifyTextPresent**

verifies expected text is somewhere on the page.

**verifyElementPresent**

verifies an expected UI element, as defined by its HTML tag, is present on the page.

**verifyText**

verifies expected text and its corresponding HTML tag are present on the page.

**verifyTable**

verifies a table’s expected contents.

**waitForPageToLoad**

pauses execution until an expected new page loads. Called automatically when clickAndWait is used.

**waitForElementPresent**

pauses execution until an expected UI element, as defined by its HTML tag, is present on the page.

**Log Pane:**

PICS

When a test case or suite is run, corresponding logs are generated. These logs are especially useful when a test case is failed. It can be helpful in finding the cause of the failure.

Logs can be categorized into following:

Debug: Debug messages are displayed only when messages are filtered. It contains info about like call of function, loading etc.

Info: These are the messages which are generated with each command execution.

Warn: These messages informs about the condition which might become issue in future.

Error: These are the messages which needs immediate fix before going to the next step.

**Reference Pane:**

Whenever Selenese commands and parameters are created or modified, this Pane is displayed automatically. This pane provides the description of the command and its input parameter.

**UI – Element Pane:**

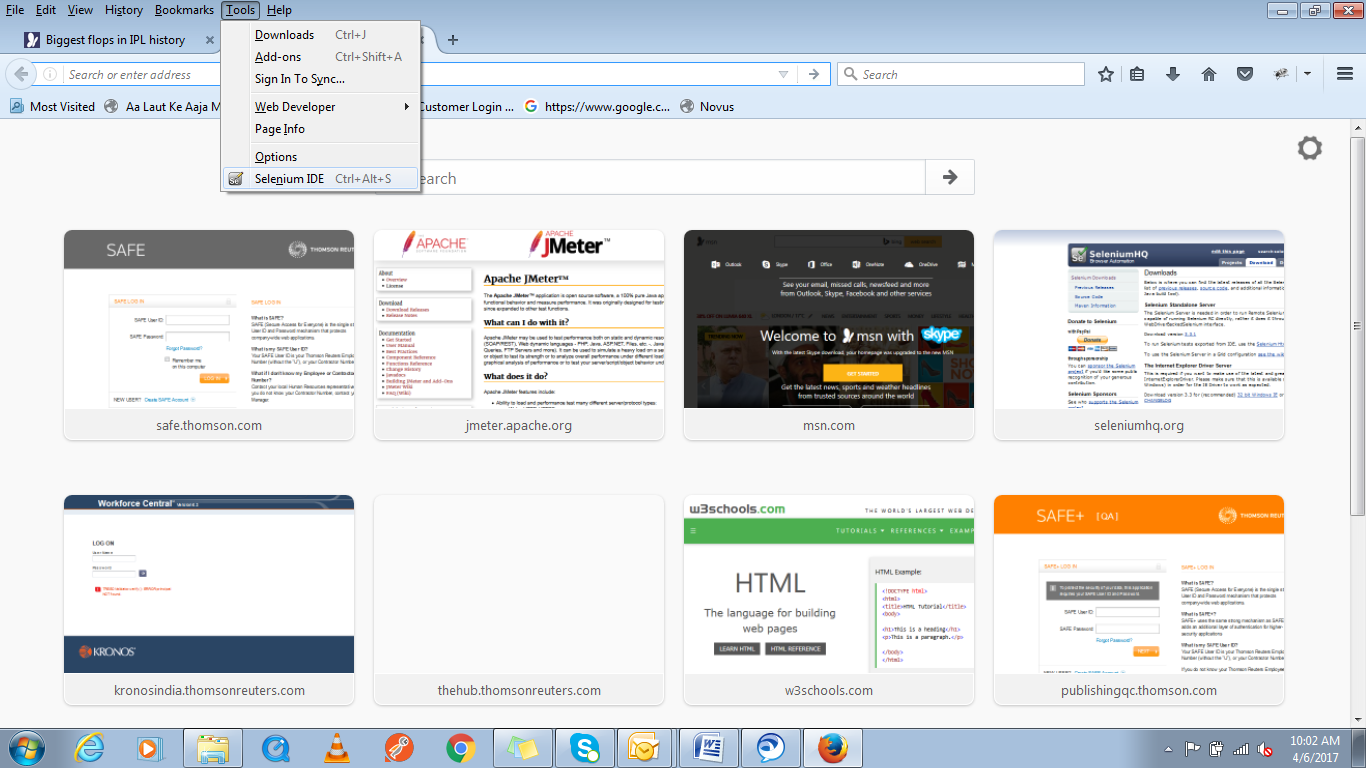
Rollup Pane:

Rollup is used to create a collection of Commands. This can be used later as and when required.

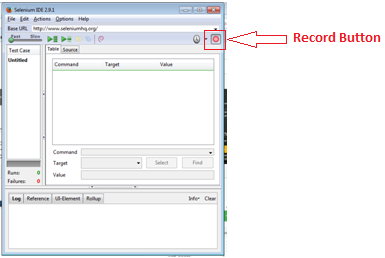
1. Steps to Create a sample Selenium test script using IDE:

Step 1: Launch web browser Mozilla Firefox.

Step 2: Go to the tab Tools, click on option “Selenium IDE”. OR press Ctrl +Alt + S.



Step 3: Toggle Record button, if recording is not started.



Step 4: Navigate URL [www.seleniumhq.org](http://www.seleniumhq.org) in Firefox. Wait until home page is not opened.

Step 5: Click on “Download” tab in Home page.

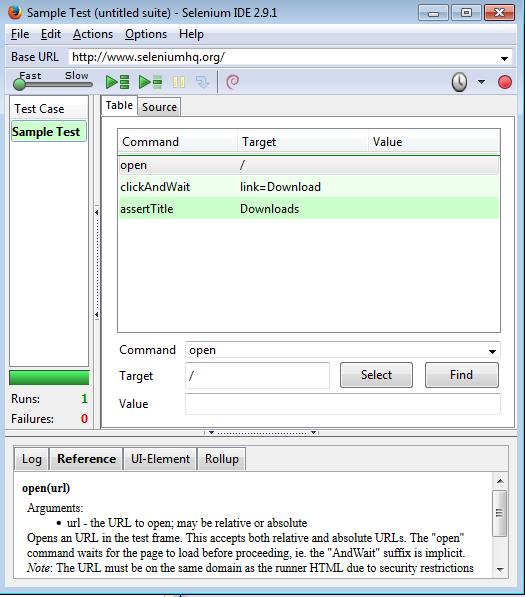
Step 6: Right click anywhere on the web page and go to the option “Show All Available Commands” and click on the “verify Title Downloads”.

Step 7: Stop recording by toggling Record button.

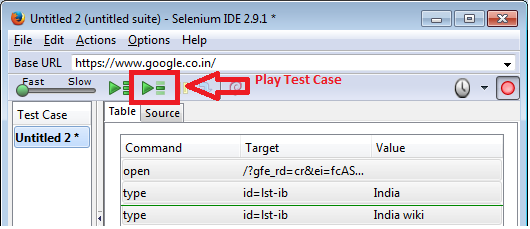
Step 8: Go to the menu File and click on “Save Test Case As…”.

Step 9: Save file as “Sample Test”.

File name with Sample Test.html will be saved.



Step 10: To play the recorded script, click on “Play current test case”.



Each statement in the script will be executed and browser will open seleniumhq home page followed by the click on Download tab.

### Selenium-Grid[¶](http://www.seleniumhq.org/docs/01_introducing_selenium.jsp#selenium-grid) (Content is incomplete)

Selenium Grid provides a mechanism to Selenium RC to scale for large test suites to run in multiple environments. Test suites can be run in parallel. Benefit is that large test suite can be run in less time by splitting suite into multiple suite. Each suite can be run on different hosts at same time. Also, suites can be run on multiple environments in parallel eg. Windows, Unix etc

**1. Introduction to WebDriver**

WebDriver is a tool which is used to automate web application. WebDriver is a part of Selenium 2. It is developed in Java. It provides well designed object oriented API’s in addition to addressing some limitations in Selenium-RC API. Also it supports dynamic web pages where elements are reloaded without being reloaded of the entire web page.

It allows test script to run against these web browsers – Internet Explorer, Google Chrome, Mozilla Firefox, Safari, GhostDriver and Opera browser. It supports six languages to develop automation script – Java, .NET, PHP, Python, Perl and Ruby.

**WebDriver Architecture:**

Refer to pic Architecture\_Webdriver.png

**3. Webdriver (Selenium 2) Vs RC (Selenium 1)**

1. Selenium Web Driver makes direct call to web browser using browser’s native support for the automation.

Qs: What is browser’s native support?

1. Selenium RC injects javascript functions into the web browser.

2. WebDriver is faster than RC (Selenium 1). WebDriver makes direct call to browser unlike RC where each and every command first goes to server and then server sends it to browser.

Pic to describe point 2.

3. WebDriver supports headless web browser HTMLUnit. It simulates Chrome, Firefox, IE and Edge web browser. Since, there is no GUI in headless browser, so execution is very fast.

Selenium RC does not support headless browser.

4. WebDriver needs not to start server before execution.

Selenium RC needs to start server before execution.

5. WebDriver supports moving of cursor.

Selenium RC does not support.

6. WebDriver supports automation of Android and IPhone.

Selenium RC does not.

7. WebDriver does notk include any built in generator to produce test report documents.

Selenium RC has built in generator to produce test report documents.

**Installation of Selenium Web Driver.**

* Download Selenium Client Driver for chrome.

1. Navigate to the link <https://chromedriver.storage.googleapis.com/index.html?path=2.29/>

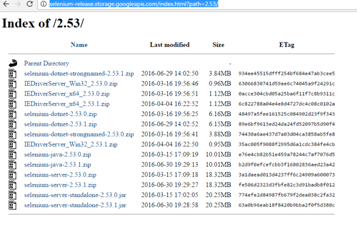
2. Click on chromedriver\_win32.zip



3. Unzip the selenium client driver to a new folder chromedriver\_win32.

4. For selenium java libraries, navigate to <http://selenium-release.storage.googleapis.com/index.html?path=2.53/>

5. Click on [selenium-java-2.53.1.zip](http://selenium-release.storage.googleapis.com/2.53/selenium-java-2.53.1.zip) to download.

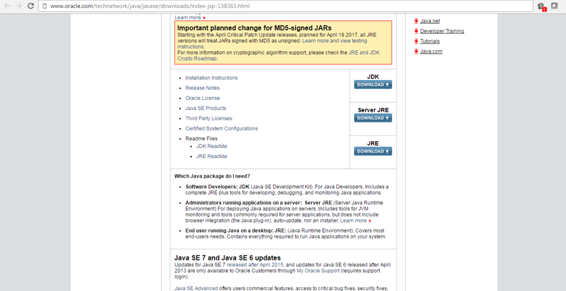


6. Unzip the zip file to selenium-java-2.53.1 folder.

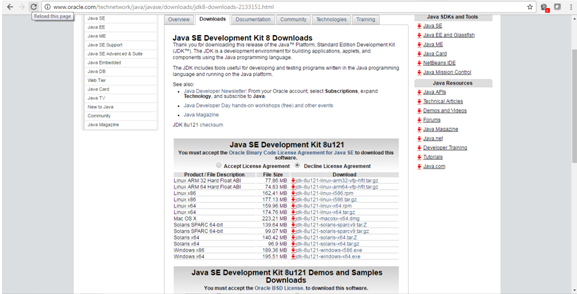
* Download of Java:

1. Navigate to the link <http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html>

2. Click on the JDK.



3. Check **Accept License Agreement**



4. Click on jdk\*\*\*\*



Installation of Java:

1. Launch the downloaded jkd exe.

2. Click Next.

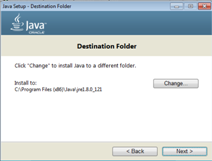


3. Select Development Tools

4. Click Next



5. Click Next



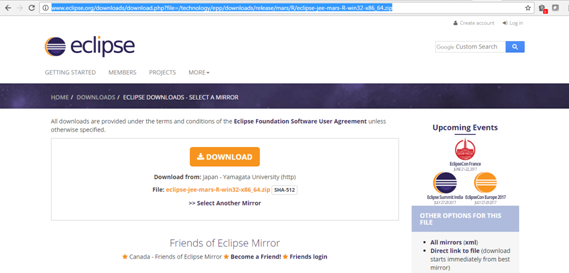
6. Click Close

Jdk is installed successfully

* Download Eclipse:

1. Navigate to the link <http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/mars/R/eclipse-jee-mars-R-win32-x86_64.zip>

2. Click on Download.



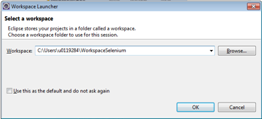
3. Unzip the zip file.



* **Writing first simple WebDriver script for Chrome:**

2. Launch the Eclipse.

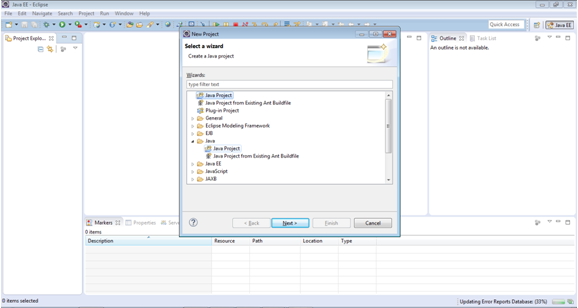
3. Give the new Workspace name.



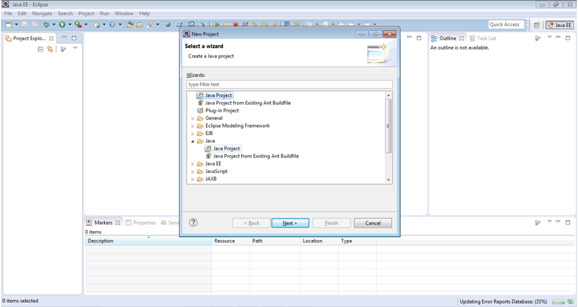
4. Click Ok

5. Navigate to menu File -> New -> Project. Click on Project

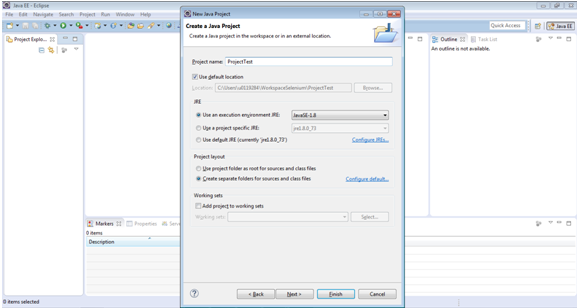
6. Select Java Project in Java.



7. Click Next

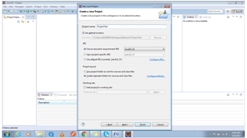


8. Give Project name as ProjectTest

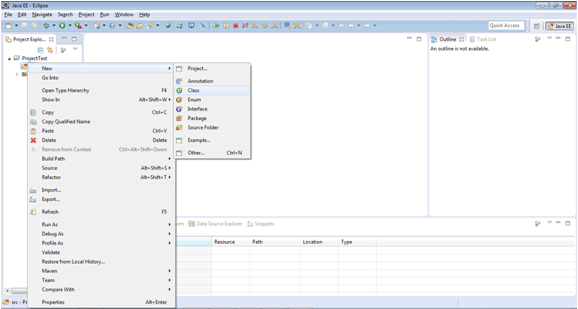


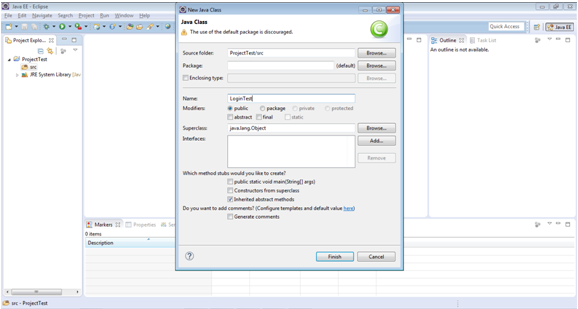
9. Click Finish

10. Expand ProjectTest



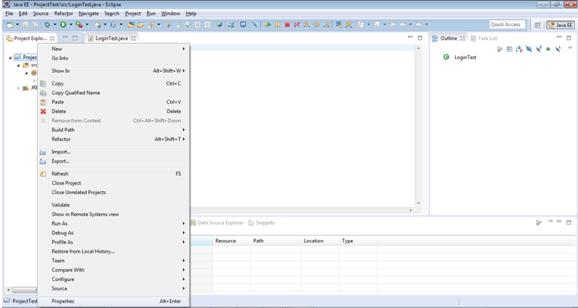
11. Right Click on src - > New -> Class



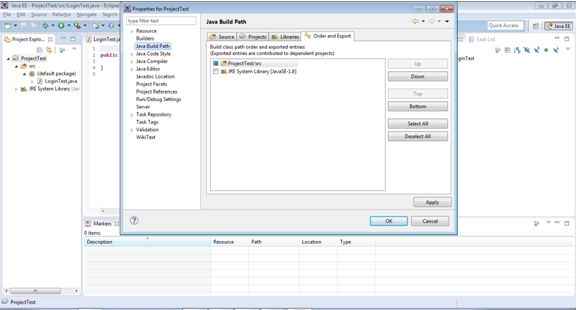
12. Give Name to the class file as LoginTest

* Include the required libraries in Java build path

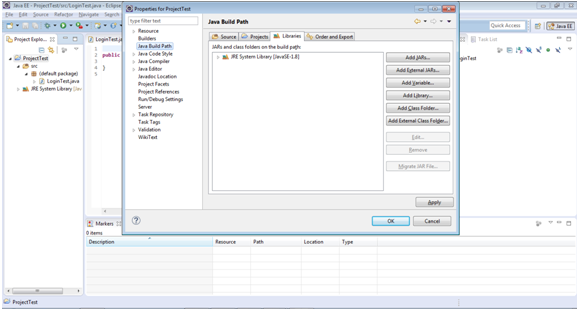
1. Righ click on ProjectTest -> Click on Properties.



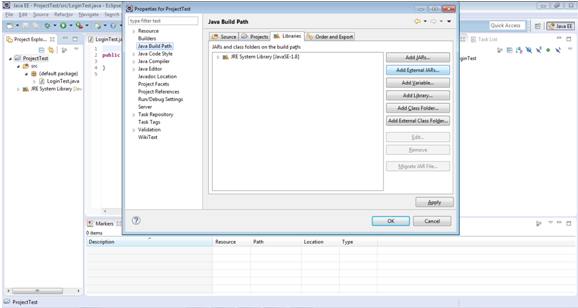
2. Click on Java Build Path



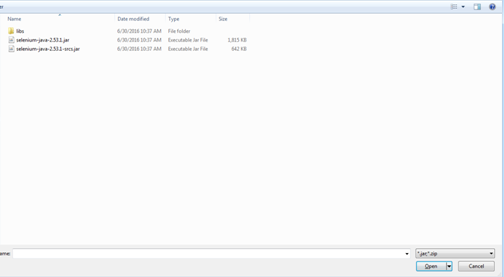
3. Go to Libraries tab.



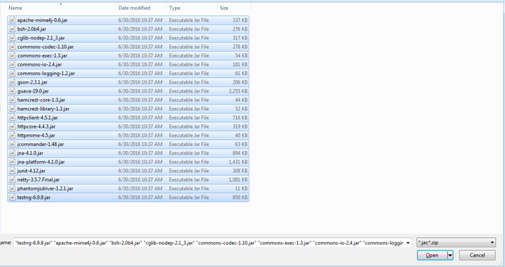
4. Click on Add External JARs…



5. Select jars selenium-java-\* from current folder.

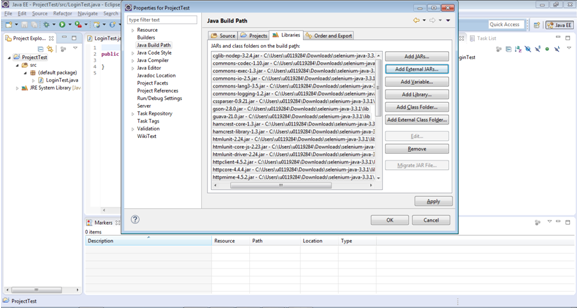


6. Select all jars from lib folder



Click on Open button.

7. Click on Ok button.



8. Copy the below script in file GoogleTest.java

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

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

**public** **class** GoogleTest {

**public** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**public** **static** WebDriver *driver*;

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

**try** {

System.***out***.println("launching chrome browser");

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.navigate().to("http://google.com");

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println(e.getMessage());

}

}

}

9. Run the script by pressing Ctrl + F11. It should launch chrome browser and then open google.com website.

**Note**: If you get an exception, “chromedriver.exe has stopped working”. It means chrome driver is incompatible to the Chrome browser. Please note that Chrome version 54 and below are compatible with the Chrome driver 2.24 and the Chrome versions above 54 need Chrome driver 2.27.

Explanation of the code line by line:

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

Explanation: WebDriver is an interface. This interface is used to refer the web browser like ChromeDriver, EdgeDriver, EventFiringWebDriver, FirefoxDriver, InternetExplorerDriver, OperaDriver, RemoteWebDriver and SafariDriver.

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

Explanation: ChromeDriver is an implementation of WebDriver for which is used to execute test cases on Chrome web browser. Instance of this Chrome driver is assigned to the interface of WebDriver.

3. *driver*.navigate().to("http://google.com");

Explanation: This statement launches website google.com.

4. driver.close();

Explanation: This close() method is used to close the current web browser.

5. System.exit(0);

Explanation: This exit(0) method is used to terminate the JVM.

What is chromedriver.exe?

ChromeDriver is a standalone server which implements WebDriver's wire protocol for Chromium. ChromeDriver is available for Chrome on Android and Chrome on Desktop (Mac, Linux, Windows and ChromeOS).

**Types of element locators:**

Following are the types:

By ID – Locate web element by ID. It is a unique attribute of web element which may or may not be present. - <div id=" Identifier1">...</div> - WebElement element = driver.findElement(By.id("Identifier1"));

By Class Name – Locate web element by Class Name. There could be more than one element with same Class Name.

–>

<div class="CLASS1"><span>Cheddar</span></div><div class="cheese"><span>Gouda</span></div>

->

List<WebElement> cheeses = driver.findElements(By.className("CLASS1"));

By Tag Name – Locating web element by Tag name.

->

<iframe src="..."></iframe>

->

WebElement frame = driver.findElement(By.tagName("iframe"));

By Name – Locating web element by attribute Name.

->

<input name="NAME1" type="text"/>

->

WebElement cheese = driver.findElement(By.name("NAME1"));

By Link Text – Locating web element by matching visible text.

->

<a href="http://www.google.com/search?q=cheese">TEXT1</a>>

->

WebElement cheese = driver.findElement(By.linkText("TEXT1"));

By Partial Link Text - Locating web element by matching partial visible text.

->

<a href="http://www.google.com/search?q=cheese">search for Link Text</a>>

->

WebElement cheese = driver.findElement(By.partialLinkText("search"));

->

By CSS - Locating web element by CSS. ->

<div id="food"><span class="dairy">milk</span><span class="dairy aged">cheese</span></div>

->

WebElement cheese = driver.findElement(By.cssSelector("#food span.dairy.aged"));

->

By XPath - Locating web element by XPath.

Using JavaScript

<input type="text" name="example" />

<INPUT type="text" name="other" />

->

List<WebElement> inputs = driver.findElements(By.xpath("//input"));

We will cover each of the following with an example later.

**Example of finding Web Elements with locators:**

“By” is a query object or locator which is passed inside Find methods to find a web element. “By” strategies are mentioned below:

1. By.xpath

* Manual steps required for automation:

Step 1: Launch Eclipse.

Step 2: Create a new class TestWithLocators.

Step 3: Launch Firefox.

Step 4: Enable Firebug.

Step 5: Go to URL <http://www.studydunes.com/homes/index>

Step 6: Click on Inspect.

Step 7: click on Class Rooms link to find the XPath of it.

Step 8: XPath = .//\*[@id='onLineT']/a as shown in FirePath. (Tip: Absolute XPath may change while development of the website. So, it is good to use relative XPath and not absolute XPath).

* Java automation script:

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

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

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

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

**public** **class** TestWithLocators {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** WebDriver *driver*;

**private** **static** String *URL* = "http://www.studydunes.com/homes/index";

**private** **static** String *classRoomsXPath* = ".//\*[@id='onLineT']/a";

**private** **static** WebElement *element* = **null**;

**private** **static** String *text* = **null**;

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.get(*URL*);

*element* = *driver*.findElement(By.*xpath*(*classRoomsXPath*));

*element*.click();

*text* = *driver*.getTitle();

**if** (*text*.contains("Enlist the Classes for Different Technologies")) {

System.***out***.println("Pass");

} **else** {

System.***out***.println("Fail");

}

*text* = *driver*.getCurrentUrl();

**if**(*text*.contains("http://www.studydunes.com/classroom/classrooms/view\_classes")){

System.***out***.println("Pass");

}

**else**{

System.***out***.println("Fail");

}

*text* = *driver*.getPageSource();

**if**(*text*.contains("studydunes")){

System.***out***.println("Pass");

}

**else**{

System.***out***.println("Fail");

}

Thread.*sleep*(2000);

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception:" + e.getMessage());

}

}

}

* Explanation of *element* = *driver*.findElement(By.*xpath*(*classRoomsXPath*)):

findElement() is a method which is used to locate element on the web page. It takes By object query as an input and returns web element if found. Otherwise exception will be thrown.

* *text* = *driver*.getTitle();

Method getTitle() returns the title of the web page.

* *text* = *driver*.getCurrentUrl();

Method getCurrentUrl() returns the current URL.

* *text* = *driver*.getPageSource();

Method getPageSource() returns the entire content of the current web page.

2. By.id

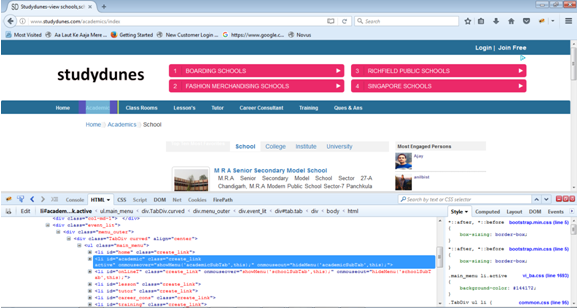
* Manual steps required for automation:

Step 1: Enable Firebug, click on Inspect.

Step 2: Click on tab Academic.

Step 3: Click on HTML section of Firebug.

Step 4: See the highlighted section of HTML having id of Academic tab. Id of Academic tab is “academic”.



* Java automation script:

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

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

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

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

**public** **class** TestWithLocators {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** WebDriver *driver*;

**private** **static** String *URL* = "http://www.studydunes.com/homes/index";

//private static String classRoomsXPath = ".//\*[@id='onLineT']/a";

**private** **static** String *academicId* = "academic";

**private** **static** WebElement *element* = **null**;

**private** **static** String *titleText* = **null**;

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.get(*URL*);

//element = driver.findElement(By.xpath(classRoomsXPath));

*element* = *driver*.findElement(By.*id*(*academicId*));

*element*.click();

*titleText* = *driver*.getTitle();

**if** (*titleText*.contains("view schools")) {

System.***out***.println("Pass");

} **else** {

System.***out***.println("Fail");

}

Thread.*sleep*(2000);

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception:" + e.getMessage());

}

}

}

3. By. tagName

Suppose we need all anchor tags which are present in a web page. Then given below is the Java script for the same.

**import** java.util.Iterator;

**import** java.util.List;

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

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

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

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

**public** **class** TestWithLocators {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** WebDriver *driver*;

**private** **static** String *URL* = "http://www.studydunes.com/homes/index";

**private** **static** String *anchortagName* = "a";

**private** **static** WebElement *element* = **null**;

**private** **static** List *elements* = **null**;

**private** **static** String *titleText* = **null**;

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.get(*URL*);

*elements* = *driver*.findElements(By.*tagName*(*anchortagName*));

*element* = *driver*.findElement(By.*tagName*(*anchortagName*));

System.***out***.println(*element*.getText());

*elements* = *driver*.findElements(By.*tagName*(*anchortagName*));

Iterator elementIterator = *elements*.iterator();

**while** (elementIterator.hasNext()) {

*element* = (WebElement) elementIterator.next();

System.***out***.println(*element*.getText());

}

Thread.*sleep*(2000);

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception:" + e.getMessage());

}

}

}

Note:

Method findElement() only returns first anchor element found in the page.

Method findElements() returns all of the anchor elements in the web page.

4. By.name

* Manual steps required for automation:

Step 1: Click on Inspect.

Step 2: Click on Text area under Ask Your Question.

Step 3: name = data[Query][description] as shown in HTML section of Firebug.

* Java automation script:

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

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

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

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

**public** **class** TestWithLocators {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** WebDriver *driver*;

**private** **static** String *URL* = "http://www.studydunes.com/homes/index";

**private** **static** String *textAreaName* = "data[Query][description]";

**private** **static** WebElement *element* = **null**;

**private** **static** String *titleText* = **null**;

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.get(*URL*);

*element* = *driver*.findElement(By.*name*(*textAreaName*));

*element*.sendKeys("Hello...");

Thread.*sleep*(10000);

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception:" + e.getMessage());

}

}

}

4. By.className

* Manual steps required for automation:

Step 1: Click on Inspect.

Step 2: Click on Text area under Ask Your Question.

Step 3: name = data[Query][description] as shown in **HTML section** of Firebug.

* Java automation script:

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

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

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

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

**public** **class** TestWithLocators {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** WebDriver *driver*;

**private** **static** String *URL* = "http://www.studydunes.com/homes/index";

**private** **static** String *className* = "ask\_your\_qust\_text\_box";

**private** **static** WebElement *element* = **null**;

**private** **static** String *text* = **null**;

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.get(*URL*);

*element* = *driver*.findElement(By.*className*(*className*));

*element*.sendKeys("Element located by Class Name.");

Thread.*sleep*(10000);

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception:" + e.getMessage());

}

}

}

4. By.linkText

* Manual steps required for automation:

Step 1: Click on Inspect.

Step 2: Click on tab Ques & Ans.

link text = Ques & Ans as shown in **HTML section** of Firebug.

* Java automation script:

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

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

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

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

**public** **class** TestWithLocators {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** WebDriver *driver*;

**private** **static** String *URL* = "http://www.studydunes.com/homes/index";

**private** **static** String *className* = "ask\_your\_qust\_text\_box";

**private** **static** String *linkText* = "Ques & Ans";

**private** **static** WebElement *element* = **null**;

**private** **static** String *text* = **null**;

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.get(*URL*);

*element* = *driver*.findElement(By.*linkText*(*linkText*));

*element*.click();

*text* = *driver*.getTitle();

**if**(*text*.contains("User can submit")){

System.***out***.println("Pass");

}

**else**{

System.***out***.println("Fail");

}

Thread.*sleep*(10000);

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception:" + e.getMessage());

}

}

}

4. By.partialLinkText

* Manual steps required for automation:

Step 1: Click on Inspect.

Step 2: Click on tab Ques & Ans.

link text = Ques & Ans as shown in **HTML section** of Firebug.

* Java automation script:

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

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

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

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

**public** **class** TestWithLocators {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** WebDriver *driver*;

**private** **static** String *URL* = "http://www.studydunes.com/homes/index";

**private** **static** String *className* = "ask\_your\_qust\_text\_box";

**private** **static** String *linkText* = "Ques";

**private** **static** WebElement *element* = **null**;

**private** **static** String *text* = **null**;

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.get(*URL*);

*element* = *driver*.findElement(By.*partialLinkText*(*linkText*));

*element*.click();

*text* = *driver*.getTitle();

**if**(*text*.contains("User can submit")){

System.***out***.println("Pass");

}

**else**{

System.***out***.println("Fail");

}

Thread.*sleep*(10000);

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception:" + e.getMessage());

}

}

}

**Navigation:**

Web navigation means visiting the website. WebDriver provides a way to navigate forward and backward the website.

* driver.navigate().to(URL)

This is similar to the driver.get(URL). It launches URL in the web browser.

* driver.navigate().back();

This method takes web browser to the previously opened URL.

* driver.navigate().forward();

This method takes browser to the next URL which is already visited.

* driver.navigate().refresh();

This method refreshes the entire web page. It is similar to pressing F5.

* Driver.quit()

It closes all windows opened by WebDriver.

**Java Script:**

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

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

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

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

**public** **class** NavigationTest {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** WebDriver *driver*;

**private** **static** String *URL* = "http://www.studydunes.com/homes/index";

**private** **static** String *classRoomsXPath* = ".//\*[@id='onLineT']/a";

**private** **static** WebElement *element* = **null**;

**private** **static** String *text* = **null**;

**private** **static** String *academicId* = "academic";

**private** **static** String *linkText* = "Ques & Ans";

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.navigate().to(*URL*);

*element* = *driver*.findElement(By.*id*(*academicId*));

*element*.click();

Thread.*sleep*(2000);

*element* = *driver*.findElement(By.*xpath*(*classRoomsXPath*));

*element*.click();

Thread.*sleep*(2000);

*element* = *driver*.findElement(By.*linkText*(*linkText*));

*element*.click();

Thread.*sleep*(2000);

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

Thread.*sleep*(2000);

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

Thread.*sleep*(2000);

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

Thread.*sleep*(2000);

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

Thread.*sleep*(2000);

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

Thread.*sleep*(2000);

*driver*.quit();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception:" + e.getMessage());

}

}

}

* **Switch to Frames and Windows:**

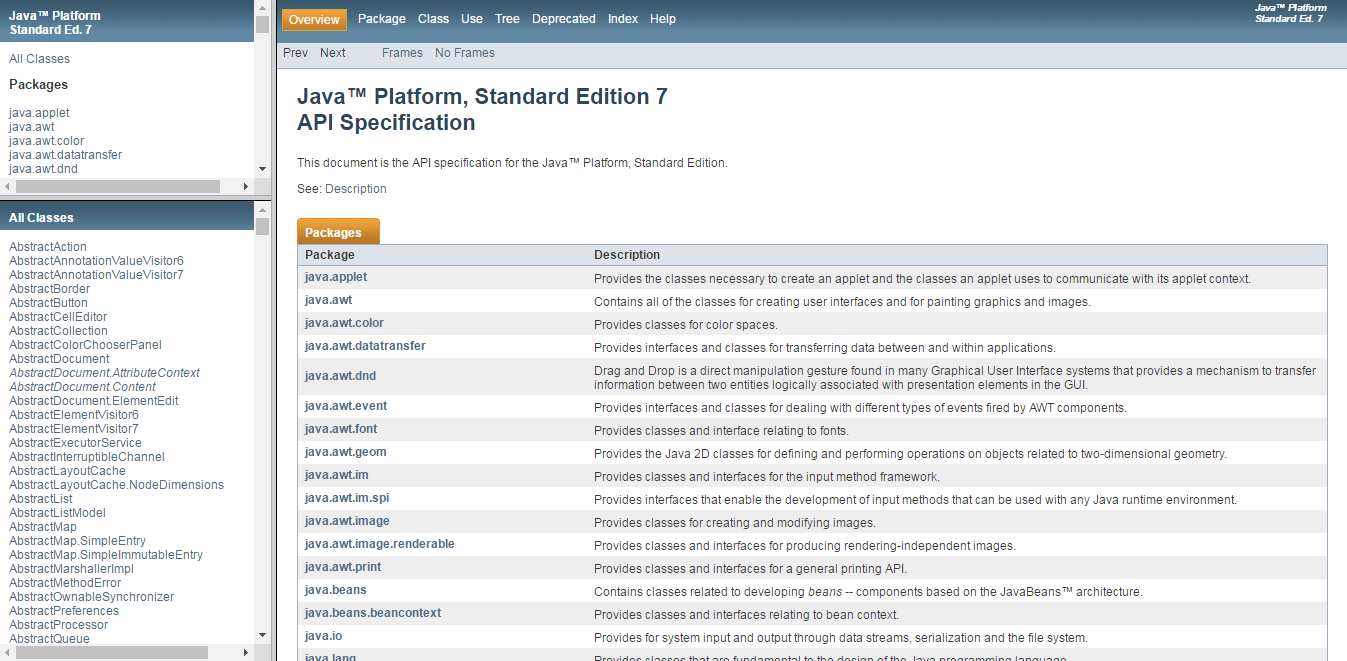
While doing operations on application, it may open multiple windows or frames (what is window or frame?). To switch to particular window or frame, it is important to know the Window or Frame handle.

What is a frame?

A frame is an HTML element that allows an external webpage to be embedded in an HTML document.

**Note:** There are two types of elements in HTML, frame and iframe. In this tutorial, we have considered frame tag. In real scenarios, it could be iframe as well. To test iframe, all we need is to replace the frame with iframe in all the examples given below.

See the below pics as an example of frames. URL is <https://docs.oracle.com/javase/7/docs/api/>



In above web page, there are 3 frames. These frames are inside element “frameset” and the attribute is “name”:

1. packageListFrame

2. packageFrame

3. classFrame

If we want to perform any action on these frames, we have to switch over these frames. If we want to switch to another frame then we have to switch to the parent window first.

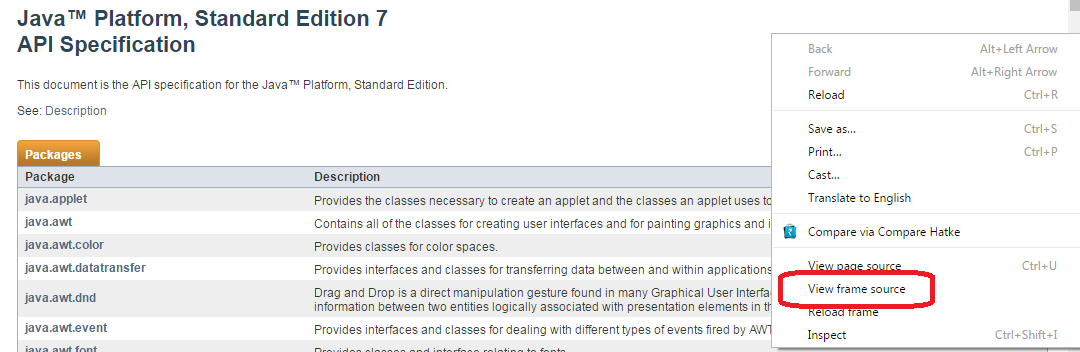
HTML code for frameset:

<frameset cols="25%,50%,25%">  
  <frame src="frame\_a.htm">  
  <frame src="frame\_b.htm">  
  <frame src="frame\_c.htm">  
</frameset>

HTML code for iframe:

<iframe src="https://www.w3schools.com"></iframe>

How to identify the frames?



**Steps to identify the frame in a web page:**

1. Right click on the web page. Eg. Any section, advertisement etc.

2. Look for the “View frame source” in the available options as shown in the above pic.

If it is present, it means that part of the HTML doc is a frame.

**How to find out if there are one or more frames or iframes in a web page?**

For this, we can write a simple program as given below for frame:

**import** java.util.Iterator;

**import** java.util.List;

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

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

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

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

**public** **class** NumberOfFrames {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** String *URL* = "https://docs.oracle.com/javase/7/docs/api/";

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

driver.navigate().to(*URL*);

List webElementList = driver.findElements(By.*tagName*("frame"));

Iterator iterator = webElementList.iterator();

System.***out***.println("Total numbers of frames: "+ webElementList.size());

**while**(iterator.hasNext()){

WebElement frameElement = (WebElement)iterator.next();

String frameName = frameElement.getAttribute("name");

System.***out***.println("Frame: "+ frameName);

}

driver.quit();

} **catch** (Exception e) {

System.***out***.println(e.getMessage());

}

}

}

**List of Switch operations:**

1. activeElement() – This method is used to test if expected element in a web page is active. It returns current active element.

2. alert() – Switch to alert;

3. defaultContent();

4. frame(int arg0);

5. frame(String arg0);

6. frame(WebElement arg0);

7. parentFrame();

8. window(arg0);

**Example of Switch operations:**

1. By attribute Name:

Frame can be switched using attribute name or id of an element frame. Name is given higher precedence over id.

Parameters: name or the id of the frame element.

Returns: driver focused on the current frame

Throws: NoSuchFrameException - If the frame is not found

Below is an example:

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import java.util.Iterator;

import java.util.List;

import java.util.Set;

import org.junit.Assert;

import org.openqa.selenium.By;

import org.openqa.selenium.JavascriptExecutor;

public class SwitchToFramesByName {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://docs.oracle.com/javase/7/docs/api/";

public static void main(String[] args){

try{

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.navigate().to(URL);

List elementList = driver.findElements(By.tagName("frame"));

Iterator iterator = elementList.iterator();

WebElement webElement1 = (WebElement) iterator.next();

WebElement webElement2 = (WebElement) iterator.next();

WebElement webElement3 = (WebElement) iterator.next();

System.out.println(webElement1.getAttribute("name"));

System.out.println(webElement2.getAttribute("name"));

System.out.println(webElement3.getAttribute("name"));

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

driver.findElement(By.linkText("java.awt")).click();

driver.switchTo().parentFrame();

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

driver.findElement(By.linkText("ActiveEvent")).click();

driver.switchTo().parentFrame();

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

String title = driver.getTitle();

Assert.assertTrue(title.contains("ActiveEvent"));

System.out.println(title.contains("ActiveEvent"));

driver.quit();

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

}

2. By Index:

If a page has three frames, the first frame would be at index 0, the second at index 1 and the third at index 2. So you can select any frame using the index.

Parameters: Zero-based index

Returns: driver focused on the current frame

Throws: If frame is not found throws NoSuchFrameException.

Below is an example:

**import** java.util.Iterator;

**import** java.util.List;

**import** org.junit.Assert;

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

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

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

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

**public** **class** SwitchToFramesById {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** String *URL* = "https://docs.oracle.com/javase/7/docs/api/";

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

driver.navigate().to(*URL*);

driver.switchTo().frame(0);

driver.findElement(By.*linkText*("java.awt")).click();

driver.switchTo().parentFrame();

driver.switchTo().frame(1);

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

driver.switchTo().parentFrame();

driver.switchTo().frame(2);

String title = driver.getTitle();

driver.quit();

} **catch** (Exception e) {

System.***out***.println(e.getMessage());

}

}

}

3. By WebElement:

To switch over to particular frame, first WebElement referring to that frame need to find out.

Parameters: frameElement - The frame element to switch to.

Returns: driver focused on the current frame.

Throws: NoSuchFrameException - If the given element is neither an iframe nor a frame element.

StaleElementReferenceException - If the WebElement has gone stale.

Below is the example:

import java.util.Iterator;

import java.util.List;

import org.junit.Assert;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class SwitchToFramesByWebElement {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://docs.oracle.com/javase/7/docs/api/";

public static void main(String[] args) {

try{

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.navigate().to(URL);

List elementList = driver.findElements(By.tagName("frame"));

Iterator iterator = elementList.iterator();

WebElement webElement1 = (WebElement) iterator.next();

WebElement webElement2 = (WebElement) iterator.next();

WebElement webElement3 = (WebElement) iterator.next();

driver.switchTo().frame(webElement3);

WebElement linkElement = driver.findElement(By.linkText("No Frames"));

linkElement.click();

String url = driver.getCurrentUrl();

Assert.assertTrue(url.contains("https://docs.oracle.com/javase/7/docs/api/overview-summary.html"));

driver.quit();

}

catch(Exception e){

System.out.println(e.getMessage());

}

}

}

**Note**: if there is frame (child frame) inside a frame (parent frame). First we need to switch to parent frame and than Child frame. We cannot switch directly to Child frame.

**Switch to Windows:**

Application may open one or more Windows when a link (or button etc) is clicked. Once WebDriver object is instantiated, unique alphanumeric id is assigned for each unique Window which is called Window handle. With the help of Window handle, all Windows can be tracked.

**HTML code to open a new Window:**

<a href="http://studydunes.com/" target="\_blank">Welcome to Studydunes</a>

If we set the target attribute to "\_blank", the link will open in a new browser window or a new tab.

Example of switch operation:

1. By GetWindowHandle() command:

import java.util.Iterator;

import java.util.List;

import org.junit.Assert;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class SwitchToWindowUsingHandle {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://docs.oracle.com/javase/7/docs/api/";

public static void main(String[] args) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.navigate().to(URL);

String currentWindow = driver.getWindowHandle();

driver.switchTo().window(currentWindow);

driver.quit();

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

}

2. By GetWindowHandles() command:

import java.util.Iterator;

import java.util.List;

import java.util.Set;

import org.junit.Assert;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class SwitchToWindowUsingHandles {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://docs.oracle.com/javase/7/docs/api/";

public static void main(String[] args) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.navigate().to(URL);

Set windowsSet = driver.getWindowHandles();

Iterator iterator = windowsSet.iterator();

String window;

while (iterator.hasNext()) {

window = (String) iterator.next();

driver.switchTo().window(window);

}

driver.quit();

} catch (Exception e) {

System.out.println(e.getMessage());

}

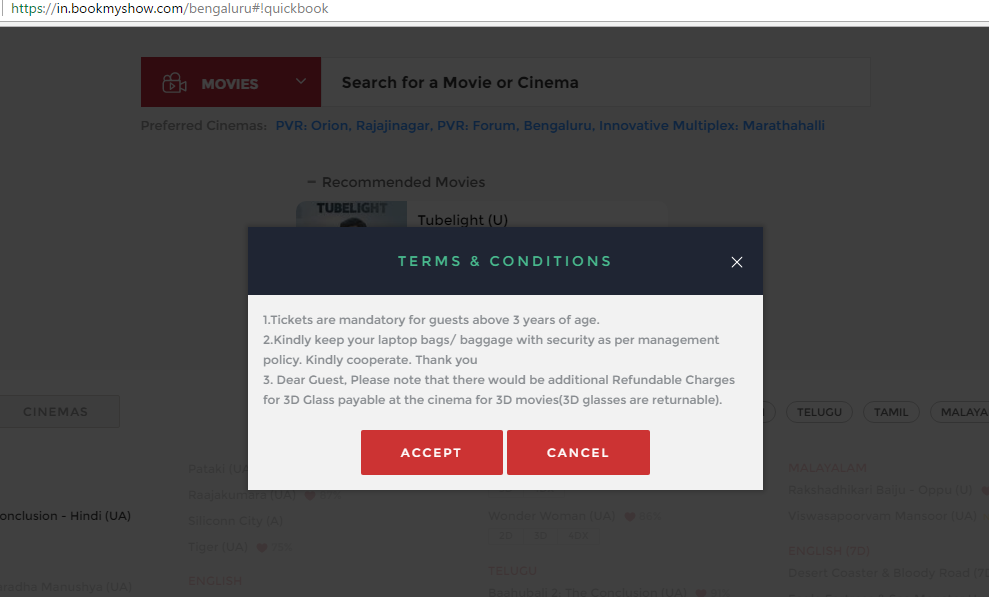
}

}

**Switch to Alert:**

Alert is prompted when information from user is needed. To proceed, user has to click on Alert.

Given below is the picture of the Alert. If we “Accept” than only we can move ahead. If we cancel, we will be on the same page.



**Javascript code of the Alert:**

Javascript provides three types of Alert: Alert box, Confirm box and Prompt box.

Javascript for Alert box: alert("This is an alert box!");

Javascript for Confirm box: window.confirm("This is a Confirm box");

Javascript for Prompt box: var person = prompt("Website Name:", "studydines");

Code snippet to switch over Alert:

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

Operations:

1. To accept the alert: alert.accept();

2. To dismiss the alert: alert.dismiss();

3. To get the text: alert.getText();

4. To write on Alert: alert.sendKeys("India");

Note: please include “import org.openqa.selenium.Alert;”, so that Class Alert can be used.

Switch to Active Element:

Active element is the currently focused element in the web page.

API driver.switchTo().activeElement() is actually a misleading name. This API does not switch focus to the web element. In fact, it returns the currently focused web element.

When we open google.com, than focus is on the text field where we can give any input to find.

Example:

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class SwitchToActiveElement {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://google.co.in";

private static WebElement element;

public static void main(String[] args){

try{

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get(URL);

element = driver.switchTo().activeElement();

element.sendKeys("One World, One Nation");

element.submit();

Thread.sleep(10000);

driver.close();

}catch(Exception e){

System.out.println("Exception: "+e.getMessage());

}

}

}

**Switch to parent frame:**

It is quite possible that one frame is inside second frame and second frame is inside third frame and so on.

Ex. Frame B is inside Frame A. Frame C is inside Frame B.

Frame A -> Frame B -> Frame C

**Switch from Frame C to Frame B:**

driver.switchTo().parentFrame();

Thus parentFrame() is used to switch to immediate parent,

**Switch from Frame C to Frame A:**

driver.switchTo().defaultContent();

Thus defaultContent() is used to switch to the main parent or parent at highest level.

**Action on Elements:**

Actions on elements are like clicking the button or entering the text in an Input button.

List of actions:

*1.* switchTo().

Please refer to the section “Switch to Frames and Windows”

2. sendKeys(""); submit(); clear(); click();

**SendKeys():**

* Meaning and Usage:

It is a method used to give value or text to an editable element. This method does not replace the text already available in the element instead it concatenates the new text to the already available text.

Example:

Send text to the Google’s search input box twice as given below:

Step 1: driver.findElement(By.name("q")).sendKeys("India")

Step 2: driver.findElement(By.name("q")).sendKeys("USA")

Google search input box value will be as given below:

Input box = IndiaUSA

Example:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

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

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

public class ActionOnElements1 {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://google.co.in";

private static WebElement element;

public static void main(String[] args) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get(URL);

WebElement inputBox = driver.findElement(By.name("q"));

inputBox.sendKeys("Clear this text");

Thread.sleep(1000);

inputBox.clear();

inputBox = driver.findElement(By.name("q"));

inputBox.sendKeys("http://studydunes.com");

inputBox.submit();

Thread.sleep(1000);

WebElement link = driver.findElement(By.partialLinkText("Studydunes:"));

link.click();

Thread.sleep(1000);

driver.close();

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage());

}

}

}

* To send special characters and keys using sendKeys():

Many a times there is the requirement in the use case to use the special characters or keys, for ex. to refresh the page using command F5. Below is the code snippet to achieve this:

driver.findElement(By.id("name")).sendKeys(Keys.F5);

List of all of the keys are given below:

**: The number pad addition key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Add);

**: The Alt key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Alt);

**: The Left arrow key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.ArrowDown);

**: The left arrow key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.ArrowLeft);

**: The right arrow key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.ArrowRight);

**: The up arrow key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.ArrowUp);

**: The Backspace key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Backspace);

**: The Cancel keystroke.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Cancel);

**: The Clear keystroke.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Clear);

**: The function key COMMAND.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Command);

**: The Control key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Control);

**: The number pad decimal separator key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Decimal);

**: The Delete key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Delete);

**: The number pad division key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Divide);

**: The Left arrow key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Down);

**: The End key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.End);

**: The Enter key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Enter);

**: The equal sign key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Equal);

**: The Escape key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Escape);

**: The function key F1.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F1);

**: The function key F10.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F10);

**: The function key F11.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F11);

**: The function key F12.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F12);

**: The function key F2.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F2);

**: The function key F3.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F3);

**: The function key F4.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F4);

**: The function key F5.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F5);

**: The function key F6.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F6);

**: The function key F7.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F7);  
 **: The function key F8.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F8);

**: The function key F9.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.F9);

**: The Help keystroke.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Help);  
**: The Home key.**

driver.FindElement(By.XPath("String")).SendKeys(Keys.Home);  
**: The Insert key.**

driver.FindElement(By.XPath("String")).SendKeys(Keys.Insert);  
**: The left arrow key.**

driver.FindElement(By.XPath("String")).SendKeys(Keys.Left);  
**: The Alt key.**

driver.FindElement(By.XPath("String")).SendKeys(Keys.LeftAlt);  
**: The Control key.**

driver.FindElement(By.XPath("String")).SendKeys(Keys.LeftControl);  
**: The Shift key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.LeftShift);

**: The function key META.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Meta);

**: The number pad multiplication key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Multiply);

: The **NULL keystroke.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Null);

**: The number pad 0 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad0);

**: The number pad 1 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad1);

**: The number pad 2 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad2);

**: The number pad 3 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad3);  
 **: The number pad 4 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad4);

**: The number pad 5 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad5);

**: The number pad 6 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad6);

**: The number pad 7 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad7);

**: The number pad 8 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad8);

**: The number pad 9 key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.NumberPad9);

**: The Page Down key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.PageDown);

**: The Page Up key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.PageUp);

**: The Pause key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Pause);

**: The Return key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Return);

**: The right arrow key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Right);

**: The semi-colon key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Semicolon);

**: The number pad thousands separator key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Separator);

**: The Shift key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Shift);

**: The Space bar key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Space);

**: The number pad subtraction key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Subtract);

**: The Tab key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Tab);

**: The up arrow key.**  
driver.FindElement(By.XPath("String")).SendKeys(Keys.Up);

We can also send the pressable keys as [Unicode PUA(Privtae User Area)](http://www.w3.org/TR/2012/WD-webdriver-20120710/#typing-keys) format . So the above samples can be rewritten as below :

sendKeys(Keys.F5) == sendKeys("\uE035")

sendKeys(Keys.PAGE\_DOWN) == sendKeys("\uE00F")

sendKeys(Keys.ARROW\_DOWN) == sendKeys("\uE015")

sendKeys(Keys.SPACE) == sendKeys("\uE00D")

sendKeys(Keys.TAB) == sendKeys("\uE004")

sendKeys(Keys.ALT) == sendKeys("\uE00A")

3. findElement(arg0); findElements(arg0); getAttribute(arg0);

Qs: What is the difference between findElement() and findElements()?

Ans: Differences are given below:

|  |  |  |
| --- | --- | --- |
|  | **findElement()** | **findElements()** |
| **On zero match** | Throws noSuchElementException | Returns a empty list |
| **On one match** | Returns WebElement | Returns the list of one WebElement |
| **On more than one match** | Returns the first appearance in DOM | Returns the list of all matching WebElements |

Example:

import java.util.Iterator;

import java.util.List;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

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

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

public class ActionOnElements2 {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "http://studydunes.com/";

private static WebElement element;

public static void main(String[] args) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get(URL);

WebElement tagElement = driver.findElement(By.tagName("a"));

System.out.println("\*\*\*\*\* Start of findElement() \*\*\*\*\*");

System.out.println(tagElement.getAttribute("href"));

System.out.println("\*\*\*\*\* End of findElement() \*\*\*\*\*");

List listOfWebElements = driver.findElements(By.tagName("a"));

Iterator iterator = listOfWebElements.iterator();

System.out.println();

System.out.println("\*\*\*\*\* Start of findElements() \*\*\*\*\*");

while(iterator.hasNext()){

WebElement element = (WebElement) iterator.next();

System.out.println(element.getAttribute("href"));

}

System.out.println("\*\*\*\*\* End of findElements() \*\*\*\*\*");

driver.close();

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage());

}

}

}

4. getCssValue(arg0);

Qs: What is CSS?

Ans: CSS is a language that describes the style of the HTML document and how elements should be displayed.

Ex:

<style>

body {

background-color: blue;

}

</style>

It means that background color of the document will be blue in color.

Qs: What is the use of getCssValue()?

Ans: API getCssValue() accepts the properties (eg. Height, width, color) as a parameter and returns the value of the property.

Eg. If color value is red for an element, API getCssValue(“color”) will return red.

Qs: where it can be used?

Ans: If there is a requirement to confirm the color change of an element when we put cursor on it, getCssValue() can be used. Example is given below:

Ex:

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

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

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

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

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

**import** org.openqa.selenium.support.ui.ExpectedConditions;

**import** org.openqa.selenium.support.ui.WebDriverWait;

**public** **class** GetCssValue {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** String *URL* = "https://google.co.in";

**private** **static** WebElement *element*;

**public** **static** WebDriver *driver*;

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

**try** {

System.***out***.println("launching chrome browser");

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.navigate().to("http://google.com");

WebElement searchButton = *driver*.findElement(By.*name*("btnK"));

String colorValueInRGB = searchButton.getCssValue("color");

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

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

action.moveToElement(searchButton).perform();

colorValueInRGB = searchButton.getCssValue("color");

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

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception: " + e.getMessage());

}

}

}

5. getLocation(); getSize();

getLocation() API is used to identify the X coordinates and Y coordinates of an element in the web page. If (X,Y) coordinates are (0,0), it means that element is on the uppermost left side of the web page.

getSize() API returns the Height and Width of the element.

import org.openqa.selenium.By;

import org.openqa.selenium.Dimension;

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.interactions.Actions;

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

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

public class GetLocationAndSize {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://google.co.in";

private static WebElement element;

public static WebDriver driver;

public static void main(String[] args) {

try {

System.out.println("launching chrome browser");

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

driver = new ChromeDriver();

driver.navigate().to("http://google.com");

WebElement searchButton = driver.findElement(By.name("btnK"));

Point searchBtnCoordinates = searchButton.getLocation();

int x = searchBtnCoordinates.getX();

int y = searchBtnCoordinates.getY();

System.out.println("Location => x = " + x + "; y = " + y);

Dimension dimensionOfSearchButton = searchButton.getSize();

int height = dimensionOfSearchButton.height;

int width = dimensionOfSearchButton.getWidth();

System.out.println("Size => Height = " + height + "; Width = " + width);

driver.close();

System.exit(0);

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage());

}

}

}

6. getScreenshotAs(arg0);

Screenshots are desirable for the visual testing and can be taken during execution of the test suites. With the help of screenshot bug can be verified or it can be ensured that layout of the web design is intact.

import java.io.File;

import org.apache.commons.io.FileUtils;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

public class ScreenShotExample {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://google.co.in";

public static WebDriver driver;

public static void main(String[] args) {

try {

System.out.println("launching chrome browser");

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

driver = new ChromeDriver();

driver.navigate().to(URL);

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

getScreenshot();

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage());

} finally {

driver.close();

System.exit(0);

}

}

public static void getScreenshot() throws Exception {

File scrFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);

FileUtils.copyFile(scrFile, new File("C:\\Users\\u0119284\\Documents\\screenshot.png"));

}

}

7. getTagName();

API getTagName() returns the Tag name of the Web element.

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

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

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

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

**public** **class** GetTagName {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** String *URL* = "https://google.co.in";

**public** **static** WebDriver *driver*;

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

**try** {

System.***out***.println("launching chrome browser");

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.navigate().to(*URL*);

WebElement searchButton = *driver*.findElement(By.*name*("btnK"));

String tagName = searchButton.getTagName();

System.***out***.println("Tag Name: "+ tagName);

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("Exception: " + e.getMessage());

}

}

}

8. getText();

API getText() returns the text of the web element.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class GetText {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://google.co.in";

public static WebDriver driver;

public static void main(String[] args) {

try {

System.out.println("launching chrome browser");

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

driver = new ChromeDriver();

driver.navigate().to(URL);

WebElement link = driver.findElement(By.partialLinkText("Gma"));

String text = link.getText();

System.out.println("Text: "+ text);

driver.close();

System.exit(0);

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage());

}

}

}

9. isDisplayed();

API isDisplayed() returns TRUE in case web element is visible otherwise FLASE if element is invisible.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class IsDisplayed {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://google.co.in";

public static WebDriver driver;

public static void main(String[] args) {

try {

System.out.println("launching chrome browser");

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

driver = new ChromeDriver();

driver.navigate().to(URL);

WebElement link = driver.findElement(By.partialLinkText("Gma"));

boolean flag = link.isDisplayed();

System.out.println("Link is displayed: "+ flag);

driver.close();

System.exit(0);

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage());

}

}

}

10. isEnabled();

API isEnabled() returns TRUE in case operation can be performed on the web element otherwise FALSE is returned if operations cannot be performed.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class IsEnabled {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "https://google.co.in";

public static WebDriver driver;

public static void main(String[] args) {

try {

System.out.println("launching chrome browser");

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

driver = new ChromeDriver();

driver.navigate().to(URL);

WebElement link = driver.findElement(By.partialLinkText("Gma"));

boolean flag = link.isEnabled();

System.out.println("Link is Enabled: "+ flag);

driver.close();

System.exit(0);

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage());

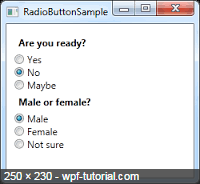
}

}

}

**Select a Radio button:**

Radio button can be selected by click() method.



To select any radio button, first get that radio button element.

For eg. in our case, we need to get “No” radio button with the help of selectors.

WebElement No = driver.findElement(By.id("RadioButtonNoId"));

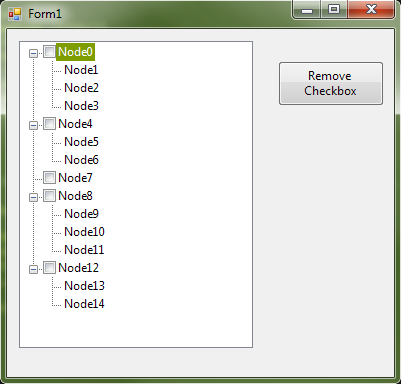
Then, click on No web element.

No.click();

This way radio button No will be selected. In case, any other radio button needs to be selected, same process need to follow.

**Select a Check Box**:

Check box can be turned on by clicking on it. To turn off, click on it again.



To select Node0 in above diagram, code is as follow:

WebElement Node0 = driver.findElement(By.id("CheckBoxNode0Id"));

Node0.click();

To deselect the Node0, after selection:

Node0.click();

Node0 will be deselected.

**Submission of a Form**:

Many a times user instead of click button, press Enter button. For eg. For Google search, most of the times user hit Enter button instead of clicking on Google Search button. On hitting Enter button, Google returns the result. So, this is called submission of Form.

In WebDriver, Form can be submit by submit() method. Below is the Java code snippet to search in a Google using submit() method.

driver.findElement(By.*name*("q"));

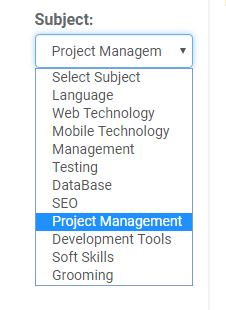
inputBox.sendKeys("http://studydunes.com");

inputBox.submit();

In above example, submit() method is used instead of click() method. Still Google will return the desired result. This is called the submission of the Form.

**Selection of options from Drop down**:

Drop down provides users to select one or multiple options to select.

****

Given below is the **HTML code** for the above Drop Down example.

<select name="…" id="LessonSubjectId">

<option value="">Select Subject</option>

<option value="1">Language</option>

<option value="2">Web Technology</option>

<option value="3">Mobile Technology</option>

<option value="4">Management</option>

<option value="5">Testing</option>

<option value="6">DataBase</option>

<option value="7">SEO</option>

<option value="8">Project Management</option>

<option value="10">Development Tools</option>

<option value="12">Soft Skills</option>

<option value="13">Grooming</option>

</select>

**Simple WebDriver example to select an option as given below:**

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

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

public class SelectOptionFromDropDown {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "http://studydunes.com/classroom/classrooms/stream\_search\_result";

public static void main(String[] args) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get(URL);

Thread.sleep(1000);

// Select by Index

WebElement sub = driver.findElement(By.id("LessonSubjectId"));

Select subject = new Select(sub);

subject.selectByIndex(1);

WebElement submit = driver.findElement(By.xpath("//\*[@id=\"home\"]/section[2]/div/table/tbody/tr/td[1]/form/button"));

submit.click();

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

}

Different ways to select/deselect options from Drop Down:

By index:

HTML code: <option value="1">Language</option>

Here, option “Language” is first in list. So, it has index as 1.

subject.selectByIndex(1);

subject.deselectByIndex(1);

By value:

HTML code: <option value="1">Language</option>

Here, value of attribute “value” is 1. So, By Value will take 1 as parm value.

subject.selectByValue(1);

subject.deselectByValue(1)

By visible text:

HTML code: <option value="1">Language</option>

Here, value of option tag is Language. So, By Visible Text will take “Language” as a parm value.

subject.selectByVisibleText(“Language”);

subject.deselectByVisibleText(“Language”);

**Selection of multiple options:**

If multiple selection of the options are supported, API isMultiple() returns TRUE.

To select multiple options:

To select multiple options, we need to select the items as many as we want. For ex if we want to select 2 options, 1: India and 2: Latest Technology, then

subject.selectByVisibleText("Inida");

subject.selectByVisibleText ("Latest Technology");

To deselect all:

API deselectAll() can be used to deselect all options.

Example:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

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

public class SelectOptionFromDropDown {

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static String URL = "http://studydunes.com/classroom/classrooms/stream\_search\_result";

public static void main(String[] args) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get(URL);

Thread.sleep(1000);

// Select by Index

WebElement sub = driver.findElement(By.id("LessonSubjectId"));

Select subject = new Select(sub);

subject.selectByIndex(1);

WebElement submit = driver.findElement(By.xpath("//\*[@id=\"home\"]/section[2]/div/table/tbody/tr/td[1]/form/button"));

submit.click();

// Select by Visible Text

sub = driver.findElement(By.id("LessonSubjectId"));

subject = new Select(sub);

subject.selectByVisibleText("Testing");

submit = driver.findElement(By.xpath("//\*[@id=\"home\"]/section[2]/div/table/tbody/tr/td[1]/form/button"));

submit.click();

// Select by Value

sub = driver.findElement(By.id("LessonSubjectId"));

subject = new Select(sub);

subject.selectByValue("8");

submit = driver.findElement(By.xpath("//\*[@id=\"home\"]/section[2]/div/table/tbody/tr/td[1]/form/button"));

submit.click();

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

}

**Advanced User Interactions API:**

**For the Advance interactions, WebDriver provides the Class Actions. Using this, keyboard and mouse events can be handled.**

**Below is the list of API’s provided by the Class Actions.**

|  |  |
| --- | --- |
| **Modifier and Type** | **Method and Description** |
| [**Action**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Action.html) | [**build**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#build--)()  Generates a composite action containing all actions so far, ready to be performed  (and resets the internal builder state, so subsequent calls to [**build()**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#build--)will contain  fresh sequences). |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**click**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#click--)()  Clicks at the current mouse location. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**click**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#click-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target)  Clicks in the middle of the given element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**clickAndHold**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#clickAndHold--)()  Clicks (without releasing) at the current mouse location. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**clickAndHold**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#clickAndHold-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target)  Clicks (without releasing) in the middle of the given element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**contextClick**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#contextClick--)()  Performs a context-click at the current mouse location. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**contextClick**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#contextClick-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target)  Performs a context-click at middle of the given element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**doubleClick**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#doubleClick--)()  Performs a double-click at the current mouse location. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**doubleClick**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#doubleClick-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target)  Performs a double-click at middle of the given element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**dragAndDrop**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#dragAndDrop-org.openqa.selenium.WebElement-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) source, [**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target)  A convenience method that performs click-and-hold at the location of the  source element, moves to the location of the target element, then releases the mouse. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**dragAndDropBy**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#dragAndDropBy-org.openqa.selenium.WebElement-int-int-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) source, int xOffset, int yOffset)  A convenience method that performs click-and-hold at the location of the  source element, moves by a given offset, then releases the mouse. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**keyDown**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#keyDown-java.lang.CharSequence-)(java.lang.CharSequence key)  Performs a modifier key press. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**keyDown**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#keyDown-org.openqa.selenium.WebElement-java.lang.CharSequence-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target, java.lang.CharSequence key)  Performs a modifier key press after focusing on an element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**keyUp**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#keyUp-java.lang.CharSequence-)(java.lang.CharSequence key)  Performs a modifier key release. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**keyUp**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#keyUp-org.openqa.selenium.WebElement-java.lang.CharSequence-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target, java.lang.CharSequence key)  Performs a modifier key release after focusing on an element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**moveByOffset**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#moveByOffset-int-int-)(int xOffset, int yOffset)  Moves the mouse from its current position (or 0,0) by the given offset. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**moveToElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#moveToElement-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target)  Moves the mouse to the middle of the element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**moveToElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#moveToElement-org.openqa.selenium.WebElement-int-int-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target, int xOffset, int yOffset)  Moves the mouse to an offset from the top-left corner of the element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**pause**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#pause-java.time.Duration-)(java.time.Duration duration) |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**pause**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#pause-long-)(long pause)  **Deprecated.** |
| void | [**perform**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#perform--)()  A convenience method for performing the actions without calling build() first. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**release**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#release--)()  Releases the depressed left mouse button at the current mouse location. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**release**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#release-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target)  Releases the depressed left mouse button, in the middle of the given element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**sendKeys**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#sendKeys-java.lang.CharSequence...-)(java.lang.CharSequence... keys)  Sends keys to the active element. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**sendKeys**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#sendKeys-org.openqa.selenium.WebElement-java.lang.CharSequence...-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target, java.lang.CharSequence... keys)  Equivalent to calling: *Actions.click(element).sendKeys(keysToSend).* This method is  different from [**WebElement.sendKeys(CharSequence...)**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html#sendKeys-java.lang.CharSequence...-)  - see[**sendKeys(CharSequence...)**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#sendKeys-java.lang.CharSequence...-) for details how. |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**tick**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#tick-org.openqa.selenium.interactions.Action-)([**Action**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Action.html) action) |
| [**Actions**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html) | [**tick**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#tick-org.openqa.selenium.interactions.Interaction...-)([**Interaction**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Interaction.html)... actions) |

**1.** [**contextClick**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#contextClick-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target):

To perform right click at the center of the web element, API contextClick() is used.

Example:

import org.openqa.selenium.Alert;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;

public class ContextClick {

public static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

public static WebDriver driver;

public static void main(String[] args) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

driver = new ChromeDriver();

driver.get("http://www.seleniumhq.org/");

WebElement Home = driver.findElement(By.linkText("Projects"));

Actions action = new Actions(driver);

action.contextClick(Home).build().perform();

Thread.sleep(2000);

driver.quit();

System.exit(0);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

}

2. [**dragAndDrop**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#dragAndDrop-org.openqa.selenium.WebElement-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) source, [**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target):

This API is used to drop one web element (source) inside another web element (target).

Example:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;

public class DragAndDrop {

public static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

public static WebDriver driver;

public static WebElement source, target;

public static void main(String[] args) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

driver = new ChromeDriver();

driver.get("http://jqueryui.com/droppable/");

Thread.sleep(3000);

driver.switchTo().frame(0);

source = driver.findElement(By.id("draggable"));

target = driver.findElement(By.id("droppable"));

Actions action = new Actions(driver);

action.dragAndDrop(source, target).build().perform();

Thread.sleep(1000);

driver.close();

System.exit(0);

} catch (Exception e) {

System.out.println(e.getMessage());

}

}

}

3. [**keyDown**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#keyDown-org.openqa.selenium.WebElement-java.lang.CharSequence-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target, java.lang.CharSequence key)

The API keyDown() method takes the modifier Keys as parameter (Shift, Alt and Control Keys - that modifies the purpose of other keys, hence the name) performed on the Web element. It is used to simulate the action of pressing a modifier key, without releasing. The expected values for the keyDown() method are - Keys.SHIFT, Keys.ALT and Keys.CONTROL only, passing key other than these results in IllegalArgumentException.

Example:

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

**import** org.openqa.selenium.Keys;

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

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

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

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

**public** **class** KeyDown {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** String *URL* = "https://google.co.in";

**public** **static** WebDriver *driver*;

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.get(*URL*);

WebElement inputBox = *driver*.findElement(By.*name*("q"));

inputBox.sendKeys("Clear this text");

Thread.*sleep*(1000);

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

// API keyDown() will press down the "Control" key available in Keyboard.

// Then, character "a" will be pressed.

// Which is actually "Ctrl + a" operation on the keyboard.

// i.e. selecting all of the content.

action.keyDown(inputBox, Keys.***CONTROL***).sendKeys("a").perform();

Thread.*sleep*(1000);

*driver*.quit();

System.*exit*(0);

} **catch** (Exception e) {

}

}

}

4. [**moveToElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/interactions/Actions.html#moveToElement-org.openqa.selenium.WebElement-)([**WebElement**](https://seleniumhq.github.io/selenium/docs/api/java/org/openqa/selenium/WebElement.html) target):

This API is used to perform “mouse hover” on any web element.

Example:

Actions actions = new Actions(driver);

WebElement menu = driver.findElement(By.linkText("MenuLink"));

actions.moveToElement(menu);

To build a series of actions:

To perform multiple actions, all the actions need to be put in a sequence we want to achieve.

Given below is the series of action user want to perform.

Step 1: Send “soarlogic” to the Google search box in capital letters.

Step 2: Select all letters of SOARLOGIC.

Step 3: Right click on it.

WebDriver code for the above requirement:

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

**import** org.openqa.selenium.Keys;

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

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

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

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

**public** **class** SeriesOfMultipleActions {

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**private** **static** String *URL* = "https://google.co.in";

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

driver.get(*URL*);

WebElement inputBox = driver.findElement(By.*name*("q"));

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

builder.click(inputBox).keyDown(Keys.***SHIFT***).sendKeys("soarogic").doubleClick().contextClick().perform();

Thread.*sleep*(2000);

driver.close();

} **catch** (Exception e) {

System.***out***.println("Exception: " + e.getMessage());

}

}

}

**Implicit and Explicit Waits:**

What is a Wait?

We have been using sleep() method to delay the execution till now. This is required in automation so that before executing next step, some time can be delayed. Sleep() method usage is not recommended as it takes exact amount of time as mentioned. This increases execution time a lot, which is undesirable in real time. Waits() are much better than Sleep() as it takes much lesser time than sleep() and is encouraged to use.

Why Wait is needed?

Due to AJAX application or Javascript or slowness of the internet, some web elements may take more time to appear on the web page. Now, this can cause WebDriver to throw an exception like Element Not Visible Exception.

Here, there is a requirement to wait some time until web element is loaded in a web page and then to proceed. To achieve this, Waits are needed.

Types of Wait:

There are two types of Wait:

1. Implicit Wait

2. Explicit Wait

1. Implicit Wait:

Idea of Implicit Wait is taken from Watir. Implict wait is set at WebDriver level and is valid till same WebDriver is used. WebDriver waits to the time set at Implicit Wait before executing next statement.

Syntax:

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

First parameter TimeOut is an integer value.

Second parameter TimeUnit could be in NANOSECONDS, MICROSECONDS, MILISECOND, SECONDS, MINUTES, DAYS, HOURS etc.

Example:

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

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

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

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

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

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

**public** **class** ImplicitWait {

**public** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**public** **static** WebDriver *driver*;

**public** **static** WebElement *source*, *target*;

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

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

*driver*.get("http://jqueryui.com/droppable/");

*driver*.manage().timeouts().implicitlyWait(3, TimeUnit.***SECONDS***);

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

*source* = *driver*.findElement(By.*id*("draggable"));

*target* = *driver*.findElement(By.*id*("droppable"));

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

action.dragAndDrop(*source*, *target*).build().perform();

Thread.*sleep*(1000);

*driver*.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println(e.getMessage());

}

}

}

**Explicit Wait:**

Explicit Wait depends on certain condition. If that condition is fulfilled, WebDriver proceeds with the execution of next statement without waiting for the time to complete. Thus, Explicit Wait is better than Implicit Wait as execution time is lesser in Explicit wait then Implicit wait. In case, condition is not fulfilled within defined period of time, ElementNotVisibleException is thrown.

Example:

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

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

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

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

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

**import** org.openqa.selenium.support.ui.WebDriverWait;

**import** org.openqa.selenium.support.ui.ExpectedConditions;

**public** **class** WebDriverWaitExample {

**public** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**public** **static** WebElement *source*, *target*;

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

**try** {

WebDriver driver;

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

driver = **new** ChromeDriver();

driver.get("http://jqueryui.com/droppable/");

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

driver = wait.until(ExpectedConditions.*frameToBeAvailableAndSwitchToIt*(0));

*source* = driver.findElement(By.*id*("draggable"));

*target* = driver.findElement(By.*id*("droppable"));

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

action.dragAndDrop(*source*, *target*).build().perform();

Thread.*sleep*(1000);

driver.close();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println(e.getMessage());

}

}

}

In the example given above, WebDriverWait waits max upto 10 seconds.

driver = wait.until(ExpectedConditions.*frameToBeAvailableAndSwitchToIt*(0));

WebDriver will keep checking in every 500 ms for the frame availability. Suppose frame is available in 1 sec then WebDriver will switch over frame immediately and will execute next statement. Thus remaining 9 seconds will be saved. This way it is better than Implicit wait.

List of Expected conditions:

1. alertIsPresent()

2. attributeContains()

3. attributeToBe()

4. attributeToBeNotEmpty()

5. elementSelectionStateToBe()

6. elementToBeClickable()

7. elementToBeSelected()

8. frameToBeAvailableAndSwitchToIt()

9. invisibilityOfAllElements()

10. invisibilityOfElementLocated()

11. invisibilityOfElementWithText()

12. not()

13. numberOfElementsToBe()

14. numberOfElementsToBeLessThan()

15. numberOfElementsToBeMoreThan()

16. numberOfwindowsToBe()

17. or()

18. presenceOfAllElementsLocatedBy()

19. presenceOfElementLocated()

20. presenceOfNestedElementLocatedBy()

21. refreshed()

22. stalenessOf()

23. textMatches()

24. textToBe()

25. textToBePresentInElement()

26. textToBePresentInElementLocated()

27. textToBePresentInElementValue()

28. titleIs()

29. urlContains()

30. urlMatches()

31. urlToBe()

32. visibilityOf()

33. visibilityOfAllElements()

34. visibilityOfAllElementsLocatedBy()

35. visibilityOfNestedElementsLocatedBy()

**Fluent Wait:**

Fluent Wait has an option to define the frequency with which WebDriver should check the availability of the web element. In the below example, value of 500 ms is set in API pollingEvery() which ensures that WebDriver will keep checking the availability of required element in every 500 ms.

Wait<WebDriver> wait = **new** FluentWait<WebDriver>(driver)

.withTimeout(10, TimeUnit.***SECONDS***)

.pollingEvery(500, TimeUnit.***MILLISECONDS***)

.ignoring(NoSuchElementException.**class**);

Method until() will return the web element once found. Else No such element exception will be thrown.

WebElement searchButton = wait.until(new Function() {

     public WebElement apply(WebDriver driver) {

return driver.findElement(By.id("searchButton"));

  }

});

* Note: Please do not mix implicit and explicit waits at same time. Doing so, may result in unpredictable wait.

**Types of Testing:**

Now a day’s automation has become more challenging as we see new technologies every day like dynamic elements or Ajax. It is important for us to understand these things before starting automation and know to how to identify these thing and test.

* Dynamic elements
  + What is a dynamic element

Dynamic elements are those which are regularly changing, moving or updating. It could be text, image, advertisement or any component.

* + How to locate

Dynamic element’s name or id may keep changing every time we open it. So next time when WebDriver open the page and look for the Dynamic element, it may throw “Element not found” error.

To avoid this, XPath of the dynamic element should be used.

* + How to test

If there is a particular pattern which is used by the dynamic element to generate the element’s name or Id, it can be tested. Otherwise, it can be tested only manually.

* Ajax tests
  + What is Ajax

AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

* + How to locate

Suppose there is a link which appears on the web page without loading the entire web page when button is clicked. Given below is the code to locate the element.

// Loop initialization.

**for** (**int** second = 0;; second++) {

// If loop reached 60 seconds then break the loop.

**if** (second >= 60) **break**;

// Search for element "link=ajaxLink" and if available then break loop.

**try** {

**if** (selenium.isElementPresent("link=ajaxLink"))

**break**;

} **catch** (Exception e) {

}

// Pause for 1 second.

Thread.*sleep*(1000);

}

* Static content:
  + What is static content?

Static content is that which does not change for eg. elements, content, layout etc.

* + How to test

The best way to test static content is to take a base snapshot which can be compared to the test snapshot taken every time when automation is run and then compare base and test snapshot.

If there is a mismatch between 2 snapshots, it means there is something broken in a web page otherwise everything is fine.

Another way to test static content is the manual testing.

* Links

Links are important to test to ensure that links are not

broken and open right page on click.

* + How to test

Link can be automated to ensure when opened have correct title and page contents.

* Functionality

Functional testing is most critical testing needed to ensure that website is working as per requirement. We may or may not pass parameters value and see if the correct response is returned or not.

* + How to test

WebDriver provides rich number of APIs to support functional testing.

**Wrapping Selenium Calls**:

It is always good to avoid redundant code. This enhances readability and maintainability. This can be done via writing utility API’s which wraps more than one WebDriver methods.

For eg. After clicking the link, it is required to wait some time before proceeding. So, for this we may write utility API which contains both the WebDriver methods.

Example:

selenium.click(elementLocator);

selenium.waitForPageToLoad(waitPeriod);

**Wrapped methods:**

/\*\*

\* Clicks and Waits for page to load.

\*

\* param elementLocator

\* param waitPeriod

\*/

**public** **void** clickAndWait(String elementLocator, String waitPeriod) {

selenium.click(elementLocator);

selenium.waitForPageToLoad(waitPeriod);

}

Also, good design handles the exception well. Before proceeding with any operation, it is good to test the element first for eg. validate if button is present or not before clicking on it. If button is not present, handle the exception gracefully.

/\*\*

\* Selenium-WebDriver -- Clicks on an element only if it is available on a page.

\*

\* param elementLocator

\*/

**public** **void** safeClick(String elementLocator) {

WebElement webElement = getDriver().findElement(By.XXXX(elementLocator));

**if**(webElement != **null**) {

selenium.click(webElement);

} **else** {

// Using the TestNG API for logging

Reporter.log("Element: " + elementLocator + ", is not available on a page - "

+ getDriver().getUrl());

}

}

**User Interface Mapping**:

Generally locators are scattered anywhere in the script or could be scattered in many other scripts which refers to the web element. If there is any change in web element later, locators are required to change accordingly which could be complicated as locators are scattered everywhere.

User interface mapping is a technique in which all of the web elements of the corresponding web page are stored at one place and each element can be mapped to meaningful name. This meaningful name is referred by locators. This will not only increase the readability but operability as well. Now, if there is any change in element later, all we need is to change at one place which will be reflected in all of the locators. This common place can be termed as repository of web elements.

Example:

**public** **void** test(){

WebElement inputBox = driver.findElement(By.*name*("q"));

inputBox.sendKeys("Google Search Input Box");

}

Given below are the steps to apply User interface mapping in above method.

Step 1:

Create a prop.properties java file and assign the meaningful name of the elements.

Google.InputBox.Name=q

Step 2:

Refer this assigned name in the locators.

**public** **void** test(){

WebElement inputBox = driver.findElement(By.*name*(Google.InputBox.Name));

inputBox.sendKeys("Google Search Input Box");

}

Step 3:

Add comments to increase readability.

**public** **void** test(){

// Set name of Search Input Box in locator to find the element.

WebElement inputBox = driver.findElement(By.*name*(Google.InputBox.Name));

inputBox.sendKeys("Google Search Input Box");

}

**Page Object Model**:

What is POM – Page Object Model?

POM is a technique in which object repository of web elements along with the methods are kept separate from the test cases.

POM = Object Repository (Web Elements) + Functionality (Application) – Test Case.

For each web page of an application there should be separate object repository.

Example:

Below sample is the test script without POM technique:

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

public class SampleWithoutPOM {

private static String URL = "http://google.com/";

private WebDriver driver;

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

private static WebElement element;

private static String title;

public static void main(String[] args) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get(URL);

// Object Repository

WebElement inputBox = driver.findElement(By.name("q"));

// Operation

inputBox.sendKeys("soarlogic.com");

inputBox.submit();

Thread.sleep(3000);

// Object Repository

WebElement link = driver.findElement(By.partialLinkText("Soarlogic"));

// Operation

link.click();

title = driver.getCurrentUrl();

// Test

if (title.contains("http://www.soarlogic.com/"))

System.out.println("Test is Passed");

else

System.out.println("Test is Failed");

} catch (Exception e) {

System.out.println("Exception: " + e.getMessage());

}

}

}

Sample with POM:

Above test script is broken into two scripts:

Script 1:

First test script “PageObjectRepositoryAndOperation.java” contains object repository and the operation for the Google home page.

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

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

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

**public** **class** PageObjectRepositoryAndOperation {

**private** WebDriver driver;

**private** String text;

**private** By byNameInputBox = By.*name*("q");

**private** By byPartialLinkTextAsSoarLogic = By.*partialLinkText*("Soarlogic");

**private** WebElement inputBox;

**public** PageObjectRepositoryAndOperation(WebDriver driver) {

**this**.driver = driver;

}

**public** **void** setTextAndSubmit(String text) **throws** Exception {

**try** {

// Object Repository

inputBox = driver.findElement(byNameInputBox);

inputBox.sendKeys(text);

inputBox.submit();

Thread.*sleep*(3000);

WebElement link = driver.findElement(byPartialLinkTextAsSoarLogic);

link.click();

} **catch** (Exception e) {

**throw** e;

}

}

}

Script 2:

Second test script “GoogleSearchTest.java” contains test cases to test Google home page.

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

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

**public** **class** GoogleSearchTest {

**private** **static** String *URL* = "http://google.com/";

**private** **static** String *driverPath* = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

**public** **static** **void** main(String[] a) {

**try** {

System.*setProperty*("webdriver.chrome.driver", *driverPath* + "chromedriver.exe");

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

driver.get(*URL*);

PageObjectRepositoryAndOperation search = **new** PageObjectRepositoryAndOperation(driver);

search.setTextAndSubmit("soarlogic.com");

String title = driver.getCurrentUrl();

// Test

**if** (title.contains("http://www.soarlogic.com/"))

System.***out***.println("Test is Passed");

**else**

System.***out***.println("Test is Failed");

driver.quit();

System.*exit*(0);

} **catch** (Exception e) {

System.***out***.println("GoogleSearchTest Exception: " + e.getMessage());

}

}

}

Benefits of POM:

1. Object Repository

For each web page corresponding repository can be constructed. Each web page, if required, can be split into multiple components where each component can have repository.

Thus, all the page objects can be stored at one place.

2. Low Maintenance

When there is no POM implemented, objects are scattered in different scripts. Thus any change in object tomorrow, eg change in Id or name of web element, requires to change all of the script wherever object is present. This is cumbersome and error prone.

Keeping all of the objects, for the particular web page, at one place require a change at one place. Thus, maintenance is very low and safe.

3. Reusable

When objects are defined at common place, it can be used in many scripts thus increases the reusability.

4. Readability

Test cases are kept differently from the object repository, thus it increases the readability of the code.

**Page Factory**:

Page Factory Model is an enhancement to the Page Object Model. It is similar to the POM but web elements are initialized using @FindyBy annotation which is much optimized.

Example:

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

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

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

**import** org.openqa.selenium.support.FindBy;

**import** org.openqa.selenium.support.PageFactory;

**public** **class** PageFactoryExample {

**private** WebDriver driver;

@FindBy(name = "q")

WebElement inputBox;

@FindBy(partialLinkText = "Soarlogic")

WebElement link;

**public** PageFactoryExample(WebDriver driver) {

**this**.driver = driver;

PageFactory.*initElements*(driver, **this**);

}

**public** **void** setTextAndSubmit(String text) **throws** Exception {

**try** {

// Object Repository

inputBox.sendKeys(text);

inputBox.submit();

Thread.*sleep*(3000);

link.click();

} **catch** (Exception e) {

**throw** e;

}

}

}

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

public class GoogleSearchTestWithPageFactory {

private static String URL = "http://google.com/";

private static String driverPath = "C:\\Users\\u0119284\\Downloads\\chromedriver\_win32\\";

public static void main(String[] a) {

try {

System.setProperty("webdriver.chrome.driver", driverPath + "chromedriver.exe");

WebDriver driver = new ChromeDriver();

driver.get(URL);

PageFactoryExample search = new PageFactoryExample(driver);

search.setTextAndSubmit("soarlogic.com");

String title = driver.getCurrentUrl();

// Test

if (title.contains("http://www.soarlogic.com/"))

System.out.println("Test is Passed");

else

System.out.println("Test is Failed");

driver.quit();

System.exit(0);

} catch (Exception e) {

System.out.println("GoogleSearchTest Exception: " + e.getMessage());

}

}

}

Annotation @FindBy can accept id, name, link text, partial link text, XPath, class name and CSS.

Web elements are initialized when API PageFactory.initElements(driver, this) is called.

**AjaxElementLocatorFactory**:

AjaxElementLocatorFactory is a lazy load concept in which elements are located only if they are used otherwise not. Benefit of this is that unnecessary time is saved from loading those web elements which are not used at all.

AjaxElementLocatorFactory accepts TimeOutvalue and if web element does not appear by this time NoSuchElementException is thrown.

**public** AjaxElementLocatorFactoryExample(WebDriver driver) {

**this**.driver = driver;

PageFactory.*initElements*(**new** AjaxElementLocatorFactory(driver, TimeoutValue), **this**);

}

TestNG - Introduction:

TestNG is a testing framework and NG stands for Next generation. It is inspired from JUnit and NUnit and provides unit testing, integration testing and functional testing etc for Java programming language.

There are 3 steps to write tests in TestNG:

Step 1: Write test cases having logic to test code. Insert TestNG annotations in your code.

Step 2: There is a file called as TestNG.xml or build.xml which contains information about the methods, Classes and tests which needs to run in TestNG.

Step 3: Run TestNG.

The concepts used in this documentation are as follows:

* A suite is represented by one XML file. It can contain one or more tests and is defined by the <suite> tag.
* A test is represented by <test> and can contain one or more TestNG classes.
* A TestNG class is a Java class that contains at least one TestNG annotation. It is represented by the <class> tag and can contain one or more test methods.
* A test method is a Java method annotated by @Test in your source.

A TestNG test can be configured by @BeforeXXX and @AfterXXX annotations which allows to perform some Java logic before and after a certain point, these points being either of the items listed above.

Annotations:

Here is a quick overview of the annotations available in TestNG along with their attributes.

|  |  |  |
| --- | --- | --- |
| **@BeforeSuite @AfterSuite @BeforeTest @AfterTest @BeforeGroups @AfterGroups @BeforeClass @AfterClass @BeforeMethod @AfterMethod** | | **Configuration information for a TestNG class:**   **@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 <test> tag is run.  **@AfterTest**: The annotated method will be run after all the test methods belonging to the classes inside the <test> 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.  **Behaviour of annotations in superclass of a TestNG class**  The annotations above will also be honored (inherited) when placed on a superclass of a TestNG class. This is useful for example to centralize test setup for multiple test classes in a common superclass.  In that case, TestNG guarantees that the "@Before" methods are executed in inheritance order (highest superclass first, then going down the inheritance chain), and the "@After" methods in reverse order (going up the inheritance chain). |
|  | alwaysRun | For before methods (beforeSuite, beforeTest, beforeTestClass and beforeTestMethod, but not beforeGroups): If set to true, this configuration method will be run regardless of what groups it belongs to.  For after methods (afterSuite, afterClass, ...): If set to true, this configuration method will be run even if one or more methods invoked previously failed or was skipped. |
|  | dependsOnGroups | The list of groups this method depends on. |
|  | dependsOnMethods | The list of methods this method depends on. |
|  | enabled | Whether methods on this class/method are enabled. |
|  | groups | The list of groups this class/method belongs to. |
|  | inheritGroups | If true, this method will belong to groups specified in the @Test annotation at the class level. |
|  | | |
| **@DataProvider** | | **Marks a method as supplying data for a test method. The annotated method must return an Object[][] where each Object[] can be assigned the parameter list of the test method. The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation.** |
|  | name | The name of this data provider. If it's not supplied, the name of this data provider will automatically be set to the name of the method. |
|  | parallel | If set to true, tests generated using this data provider are run in parallel. Default value is false. |
|  | | |
| **@Factory** | | **Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[].** |
|  | | |
| **@Listeners** | | **Defines listeners on a test class.** |
|  | value | An array of classes that extend org.testng.ITestNGListener. |
|  | | |
| **@Parameters** | | **Describes how to pass parameters to a @Test method.** |
|  | value | The list of variables used to fill the parameters of this method. |
|  | | |
| **@Test** | | **Marks a class or a method as part of the test.** |
|  | alwaysRun | If set to true, this test method will always be run even if it depends on a method that failed. |
|  | dataProvider | The name of the data provider for this test method. |
|  | dataProviderClass | The class where to look for the data provider. If not specified, the data provider will be looked on the class of the current test method or one of its base classes. If this attribute is specified, the data provider method needs to be static on the specified class. |
|  | dependsOnGroups | The list of groups this method depends on. |
|  | dependsOnMethods | The list of methods this method depends on. |
|  | description | The description for this method. |
|  | enabled | Whether methods on this class/method are enabled. |
|  | expectedExceptions | The list of exceptions that a test method is expected to throw. If no exception or a different than one on this list is thrown, this test will be marked a failure. |
|  | groups | The list of groups this class/method belongs to. |
|  | invocationCount | The number of times this method should be invoked. |
|  | invocationTimeOut | The maximum number of milliseconds this test should take for the cumulated time of all the invocationcounts. This attribute will be ignored if invocationCount is not specified. |
|  | priority | The priority for this test method. Lower priorities will be scheduled first. |
|  | successPercentage | The percentage of success expected from this method |
|  | singleThreaded | If set to true, all the methods on this test class are guaranteed to run in the same thread, even if the tests are currently being run with parallel="methods". This attribute can only be used at the class level and it will be ignored if used at the method level. Note: this attribute used to be called sequential (now deprecated). |
|  | timeOut | The maximum number of milliseconds this test should take. |
|  | threadPoolSize | The size of the thread pool for this method. The method will be invoked from multiple threads as specified by invocationCount.  Note: this attribute is ignored if invocationCount is not specified |

Invocation of TestNG:

TestNG can be invoked in many ways:

1. With a testing.xml file

2. With ant

3. Using Command line.

Example of testing.xml

1. testng.xml with Class:

|  |
| --- |
| <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >    <suite name="TestSuite" verbose="1" >  <test name="FunctionalTest" >  <classes>  <class name="com.test.Test1" />  <class name="com.test.Test2" />  </classes>  </test>    <test name="RegressionTest">  <classes>  <class name="com.test.Test3"/>  <class name="com.test.Test4"/>  </classes>  </test>  </suite> |

Here, suite name is TestSuite and contains 2 tests, FunctionalTest and RegressionTest. Test FunctionalTest and RegressionTest contains 2 classes each.

Each Class for example com.test.Test1 is a group of unit or functional etc test cases.

2. testng with Package:

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="TestSuite" verbose="1" >

<test name="RegressionTest" >

<packages>

<package name="com.test " />

</packages>

</test>

</suite>

Here, all the classes in one package, com.test, will be executed.

Testng.xml file can be run like any other java file.

Right click on the testng.xml file -> Run As -> Java Application.

TestNG – Parallel running of tests, classes and methods:

Execution time of the suite can be reduced substantially by using Parallel attribute provided by TestNG. This parallel attribute can be controlled at suite level of testng.xml.

Parallel attribute can have any of these values – tests, classes, methods or instances:

1. Parallel=”tests”

Consider the following testng.xml.

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="TestSuite" verbose="1" parallel="tests" thread-count="5">

<test name="FunctionalTest" >

<classes>

<class name="com.test.Test1" />

<class name="com.test.Test2" />

</classes>

</test>

<test name="RegressionTest">

<classes>

<class name="com.test.Test3"/>

<class name="com.test.Test4"/>

</classes>

</test>

</suite>

Here, there are 2 tests, FunctionalTest and RegressionTest. When parallel is set to tests at suite level, it means that both of the tests (FuncationalTest and RegressionTest) will be executed in parallel.

2. Parallel=”classes”

Suite will be defined as follow for the attribute parallel as classes.

<suite name="TestSuite" verbose="1" parallel="classes" thread-count="5">

There are total 2 <classes> tag in the above testng.xml file. So, these 2 classes will be executed in parallel.

3. Parallel=”methods”

Suite will be defined as follow for the attribute parallel as methods.

<suite name="TestSuite" verbose="1" parallel="methods" thread-count="5">

Here, methods are the unit test cases which will be executed in parallel.

4. Parallel=”instances”

Suite will be defined as follow for the attribute parallel as instances.

<suite name="TestSuite" verbose="1" parallel="instances" thread-count="5">

Here, instances will be executed in parallel.

**TestNG – Parameterization**:

In real time testing, it is required to test same use case with different data to ensure that functionality works fine for all of the possible scenarios. This can be achieved by using Parameterization.

There are two types of parameterization supported by TestNG:

1. Parameter annotation.

2. DataProvider annotation.

Parameter annotation:

Suite level parameters:

Below is the testng.xml with the Paramater annotaton.

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name=*"Suite1"* verbose=*"1"*>

<parameter name=*"f-name"* value=*"Studydunes"*></parameter>

<test name=*"A"*>

<classes>

<class name=*"com.test.A.A1"* />

</classes>

</test>

<test name=*"B"*>

<classes>

<class name=*"com.test.B.B1"* />

</classes>

</test>

</suite>

Here, Tag <parameter> is global for all the tests which mean that any test can use this parameter. Name of the parameter is “f-name” and values is “Studydunes”.

Example:

**package** com.test.A;

**import** org.testng.annotations.Parameters;

**import** org.testng.annotations.Test;

**public** **class** A1 {

@Parameters({ "f-name" })

@Test

**public** **void** A11(String name) {

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

}

}

When we run the suite via testng.xml, parameter “f-name” and its value is passed to the test case. To receive the value, we need to put Parameters annotation at method level. The value then is passed from “f-name” to string name of test method which can be used in logic later.

Test level parameters:

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name=*"Suite1"* verbose=*"1"*>

<test name=*"A"*>

<parameter name=*"f-name"* value=*"Studydunes"*></parameter>

<classes>

<class name=*"com.test.A.A1"* />

</classes>

</test>

<test name=*"B"*>

<classes>

<class name=*"com.test.B.B1"* />

</classes>

</test>

</suite>

In above testng.xml file, parameter “f-name” is not global and written inside test “A”. It is limited to test “A” and not others. Which means other tests like test “B” cannot use this parameter.

**DataProvider annotation:**

Many a times parameters are complex in nature and Parameter tag may be insufficient to handle to requirement for eg. Data from database, files, array of Objects etc. To tackle these kinds of data, DataProvider annotation come into picture.

**package** com.test.A;

**import** org.testng.annotations.DataProvider;

**import** org.testng.annotations.Test;

**public** **class** A2 {

@DataProvider(name = "DP")

**public** Object[][] emp() {

**return** **new** Object[][] { { "Soarlogic", 1 }, { "StudyDunes", 2 } };

}

@Test(dataProvider="DP")

**public** **void** A11(String name, **int** value) {

System.***out***.println(name+" "+value);

}

}

Explanation:

@DataProvider(name = "DP")

DP is the name of the DataProvider.

**public** Object[][] emp() {

Method emp() returns 2 dimensional array of Objects.

@Test(dataProvider="DP")

We bind dataProvider with the Test method here.

**public** **void** A11(String name, **int** value)

Here, 2 parameters are defined to take values form the DataProvider.

By default, the data provider will be looked for in the current test class or one of its base classes. If you want to put your data provider in a different class, it needs to be a static method or a class with a non-arg constructor, and you specify the class where it can be found in the dataProviderClass attribute:

**public** **class** StaticProvider {

@DataProvider(name = "DP")

**public** **static** Object[][] createData() {

**return** **new** Object[][] { { "Soarlogic", 1 }, { "StudyDunes", 2 } };

}

}

**public** **class** A2 {

@Test(dataProvider = "DP", dataProviderClass = "StaticProvider.class")

**public** **void** A11(String name, **int** value) {

System.***out***.println(name + " " + value);

}

}

Data providers can run in parallel with the attribute parallel:

|  |  |
| --- | --- |
| @DataProvider(parallel = true) | |
|  |

Parallel data providers running from an XML file share the same pool of threads, which has a size of 10 by default. You can modify this value in the <suite> tag of your XML file:

|  |  |
| --- | --- |
| <suite name="Suite1" data-provider-thread-count="20" > | |
|  |

If you want to run a few specific data providers in a different thread pool, you need to run them from a different XML file.