Contents

[Selenium Introduction 5](#_Toc101443931)

[Benefits of Selenium 5](#_Toc101443932)

[Limitations of Selenium 6](#_Toc101443933)

[History of Selenium 6](#_Toc101443934)

[Components of Selenium 7](#_Toc101443935)

[Selenium IDE 7](#_Toc101443936)

[Selenium RC 7](#_Toc101443937)

[Selenium Grid 7](#_Toc101443938)

[Selenium WebDriver (Selenium 2.0) 8](#_Toc101443939)

[Selenium 3 8](#_Toc101443940)

[Selenium 4 9](#_Toc101443941)

[Selenium Web driver Setup 12](#_Toc101443942)

[Drivers & Launching Browsers 12](#_Toc101443943)

[Selenium compatibility with Safari Browser: 14](#_Toc101443944)

[Launch Edge browser using Microsoft Edge Driver 16](#_Toc101443945)

[Failed To Launch IE Driver Using Selenium WebDriver 17](#_Toc101443946)

[Browser Commands 18](#_Toc101443947)

[Open Webpage: 18](#_Toc101443948)

[Closing Browser 19](#_Toc101443949)

[Get commands: 20](#_Toc101443950)

[Maximize Browser: 21](#_Toc101443951)

[Navigation commands: 21](#_Toc101443952)

[Browser size 21](#_Toc101443953)

[Refresh Page: 22](#_Toc101443954)

[Web Elements 22](#_Toc101443955)

[Locating Browser Elements 22](#_Toc101443956)

[Methods to Find Elements: 23](#_Toc101443957)

[Find Element 23](#_Toc101443958)

[Find Elements 23](#_Toc101443959)

[Locators: 24](#_Toc101443960)

[Dynamic Web elements 25](#_Toc101443961)

[Interacting with Text Boxes/Edit Box/Text Area 32](#_Toc101443962)

[Text Area: 32](#_Toc101443963)

[Interacting with Buttons 32](#_Toc101443964)

[Interacting with Images/ Image Link/Image Button/Hyperlink 33](#_Toc101443965)

[Interacting with Radio buttons/ Check boxes 38](#_Toc101443966)

[Interacting with Drop down / List box/ Combo box 39](#_Toc101443967)

[Interacting with Web table 47](#_Toc101443968)

[Handling Static Web Table: 48](#_Toc101443969)

[Handling Dynamic Web Table 49](#_Toc101443970)

[Interacting with Date & Time Picker 50](#_Toc101443971)

[Alert, Frames, Windows & Mouse Actions 50](#_Toc101443972)

[Handling Keyboard & Mouse Events 50](#_Toc101443973)

[Actions and Action in Selenium: 50](#_Toc101443974)

[Actions Class Methods 51](#_Toc101443975)

[Keyboard Events Using Actions Class 52](#_Toc101443976)

[Mouse Events Using Actions Class 53](#_Toc101443977)

[Mouse Hover Action 54](#_Toc101443978)

[Interacting With Slider 55](#_Toc101443979)

[Interacting With Tool tip 55](#_Toc101443980)

[Handling Alerts 56](#_Toc101443981)

[Handling Windows 58](#_Toc101443982)

[Handling Frames 61](#_Toc101443983)

[Handling Nested iFrames 63](#_Toc101443984)

[Handling Authentication Pop up 64](#_Toc101443985)

[Handling OS Pop ups 65](#_Toc101443986)

[Auto IT 65](#_Toc101443987)

[Robot Class 68](#_Toc101443988)

[Sikuli 68](#_Toc101443989)

[Page Loads and Time Outs 69](#_Toc101443990)

[Page Load Time Out 69](#_Toc101443991)

[Set Script Load timeout 69](#_Toc101443992)

[Implicit Wait 70](#_Toc101443993)

[Explicit Wait 71](#_Toc101443994)

[Fluent Wait 77](#_Toc101443995)

[Take Screenshots 78](#_Toc101443996)

[Full Screenshot 79](#_Toc101443997)

[Partial Screenshot 79](#_Toc101443998)

[Java Script Executor 80](#_Toc101443999)

[Data Driven Testing 83](#_Toc101444000)

[APACHI POI 84](#_Toc101444001)

[Read Excel with Apache POI 85](#_Toc101444002)

[Write Excel with Apache POI 87](#_Toc101444003)

[Reading CSV Files 87](#_Toc101444004)

[TESTNG 87](#_Toc101444005)

[Installation of TESTNG 87](#_Toc101444006)

[TESNG XML 90](#_Toc101444007)

[TESTNG Execution 93](#_Toc101444008)

[Parallel Execution 96](#_Toc101444009)

[Cross Browser Testing 96](#_Toc101444010)

[ASSERTIONS 97](#_Toc101444011)

[Hard Asserts 97](#_Toc101444012)

[Soft Assert 98](#_Toc101444013)

[TESTNG Annotations 98](#_Toc101444014)

[Annotations along with Attributes/Properties 100](#_Toc101444015)

[Enable /Disable 100](#_Toc101444016)

[Always Run 100](#_Toc101444017)

[Depends on Methods 100](#_Toc101444018)

[Depends on Groups 100](#_Toc101444019)

[Groups 101](#_Toc101444020)

[Inherit Groups 101](#_Toc101444021)

[Description 101](#_Toc101444022)

[Priority 101](#_Toc101444023)

[Invocation Count 102](#_Toc101444024)

[Invocation Timeout 102](#_Toc101444025)

[Skip Failed Invocations 102](#_Toc101444026)

[Thread Pool size 102](#_Toc101444027)

[Time Out 103](#_Toc101444028)

[Expected Exceptions 103](#_Toc101444029)

[Annotations to Parameterize 103](#_Toc101444030)

[@Parameter 104](#_Toc101444031)

[@Data Providers – Data Driven 104](#_Toc101444032)

[@Factory 106](#_Toc101444033)

[Analyse the Result of TestNG 107](#_Toc101444034)

[RE-RUN Failed Tests 108](#_Toc101444035)

[Retry Analyzer 109](#_Toc101444036)

[Listeners 110](#_Toc101444037)

[WebDriver Listeners 110](#_Toc101444038)

[TestNG Listeners 110](#_Toc101444039)

[ISuiteListener 110](#_Toc101444040)

[ITestListener 110](#_Toc101444041)

[IInvokedMethodListener 111](#_Toc101444042)

[Implementation 111](#_Toc101444043)

[Page Object Model 114](#_Toc101444044)

[Page Object Model without Page Factory 115](#_Toc101444045)

[Page Object Model with Page Factory 115](#_Toc101444046)

[Selenium Grid 116](#_Toc101444047)

[Headless browser Testing 119](#_Toc101444048)

[Exceptions in Selenium 119](#_Toc101444049)

[Cookies in Selenium 124](#_Toc101444050)

[Extent Reports 126](#_Toc101444051)

[MAVEN 127](#_Toc101444052)

[MAVEN Repository 128](#_Toc101444053)

[Proxy Settings 130](#_Toc101444054)

[POM.xml 132](#_Toc101444055)

[Maven Lifecycle 133](#_Toc101444056)

[Maven Commands 136](#_Toc101444057)

[Jenkins Integration 137](#_Toc101444058)

[Properties File 137](#_Toc101444059)

[Log Files 137](#_Toc101444060)

[GIT Integration 137](#_Toc101444061)

[Selenium Cucumber 138](#_Toc101444062)

[Behavior Driven Development 138](#_Toc101444063)

[Selenium & Cucumber Automation Challenges 138](#_Toc101444064)

[Set up 139](#_Toc101444065)

[Feature File 140](#_Toc101444066)

[Gherkin Keywords 142](#_Toc101444067)

[Feature 142](#_Toc101444068)

[Background 142](#_Toc101444069)

[Scenario 142](#_Toc101444070)

[Given 143](#_Toc101444071)

[When 143](#_Toc101444072)

[Then 143](#_Toc101444073)

[And 144](#_Toc101444074)

[But 144](#_Toc101444075)

[\* 144](#_Toc101444076)

[Step definition 144](#_Toc101444077)

[Test Runner 145](#_Toc101444078)

[Cucumber Options 145](#_Toc101444079)

[JUNIT Test Runner 147](#_Toc101444080)

[TESTNG Test Runner 148](#_Toc101444081)

[SELENIUM FRAMEWORK 150](#_Toc101444082)

# Selenium Introduction

Selenium is a free (open source) automated testing suite for web applications across different browsers, platforms and programming languages. Using selenium, we can automate the functional tests and easily integrate them with Maven, Jenkins and other build automation and continuous integration tools.

### Benefits of Selenium

* It’s an open source, there is no licensing cost for its usage.
* It has a large user base and helping communities
* It has multi-browser like Chrome, Fire Fox, Internet Explorer, Opera and Safari. and platform compatibility
* It has active repository developments
* It supports multiple language implementations like Java, C#, Ruby and Python

### Limitations of Selenium

Some of the limitation of Selenium are-

* Selenium does not provide desktop application automation support.
* Web Services - REST or SOAP cannot be automated using selenium.
* Selenium web driver requires programming language requirement for script creation.
* For performing common tasks required in automation like logging, reading writing to external files, we have to rely on external libraries.

### History of Selenium

Primarily, Selenium was created by **Jason Huggins** in **2004**. An engineer at Thought Works, he was working on a web application that required frequent testing. Having realized that the repetitious [Manual Testing](https://www.guru99.com/manual-testing.html) of their application was becoming more and more inefficient, he created a[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)program that would automatically control the browser's actions. He named this program as the "**JavaScriptTestRunner**."

Seeing potential in this idea to help automate other web applications, he made JavaScriptRunner open-source, which was later re-named as Selenium Core.

**Same Origin policy** prohibits JavaScript code from accessing elements from a domain that is different from where it was launched. Example, the HTML code in www.google.com uses a JavaScript program "randomScript.js". The same origin policy will only allow randomScript.js to access pages within google.com such as google.com/mail, google.com/login, or google.com/signup. However, it cannot access pages from different sites such as yahoo.com/search or guru99.com because they belong to different domains.

Unfortunately, testers using Selenium Core had to install the whole application under test and the web server on their own local computers because of the restrictions imposed by the **same origin policy.** Therefore, another Thought Work’s engineer, Paul Hammant, decided to create a server that will act as an HTTP proxy to "trick" the browser into believing that Selenium Core and the web application being tested come from the same domain. This system is known as the **Selenium Remote Control or Selenium 1.**

**Selenium Grid** was developed by **Patrick Lightbody** to address the need of minimizing test execution times as much as possible. He initially called the system "Hosted QA." It was capable of capturing browser screenshots during significant stages, and sending out Selenium commands to different machines simultaneously.

**Shinya Kasatani**of Japan created **Selenium IDE**, a Firefox extension that can automate the browser through a record-and-playback feature. He came up with this idea to further increase the speed in creating test cases. He donated Selenium IDE to the Selenium Project in **2006**.

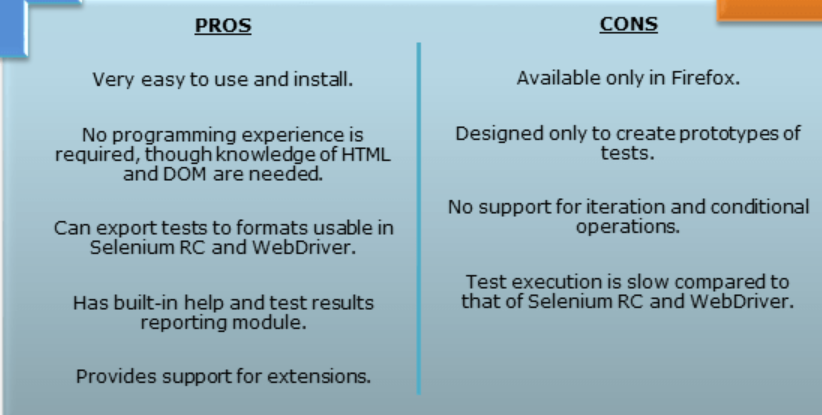
**Simon Stewart**created WebDriver circa **2006** when browsers and web applications were becoming more powerful and more restrictive with JavaScript programs like Selenium Core. The first cross-platform testing framework could control the browser from the OS level**.**

In **2008**, the whole Selenium Team decided to merge WebDriver and Selenium RC to form a more powerful tool called **Selenium 2**, with **WebDriver being the core**

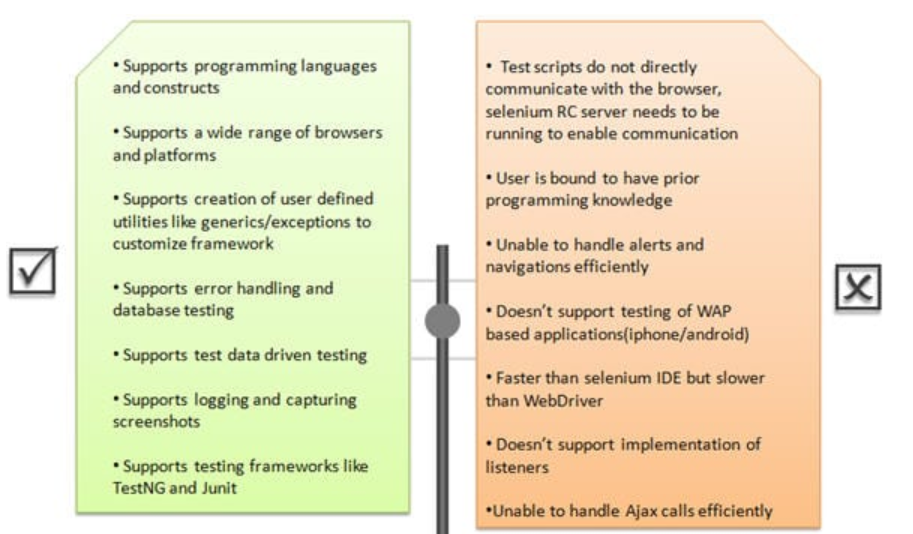
### Components of Selenium

Selenium Suite comprises of the following four components-

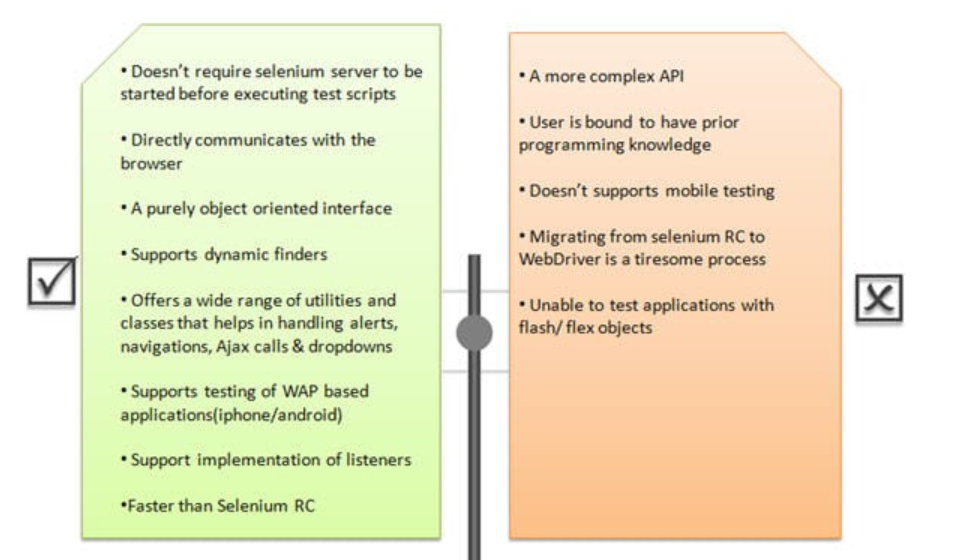
Selenium IDE - Selenium IDE is an add-on of Firefox browser that provides record and play back functionality. Its use is limited and test scripts generated are not very robust and portable.



Selenium RC - Selenium Remote Control (RC) is officially deprecated by Selenium. It used to require an additional server for running the automation scripts and had many limitations.



Selenium Grid**-** Selenium Grid is also an important part of Selenium Suite. It helps in distributed running of selenium tests in parallel across multiple remote machines.

Selenium WebDriver (Selenium 2.0) - WebDriver makes direct calls to the Web browser and It utilizes the browser’s native compatibility to automation without using any peripheral entity.

Minimum Java version required is below 1.8

* Support Firefox browser version below 47.0
* It supports IE Versions 7, 8, 9, 10 and 11. Version 11 requires additional configuration.
* It supports Head Less Browser with HtmlWebDriver
* Last 2.x version was v2.53.0. This was a stable build means it has no major issues.
* Selenium 2 supported selenium core for backward compatibility.
* supported programming languages are: Java, C#, PHP, Pearl, Ruby, Python

Selenium 3 is an advanced version of Selenium 2.

 Minimum Java version required is 1.8+

* Firefox is fully supported at version 47.0.1 or earlier. Gecko driver provides support for later versions of Firefox. Firefox 47.0.1 and earlier versions would not need GeckoDriver
* The Selenium RC APIs are only available via the legacy-rc package. Selenium RC core library won’t be supported.
* Selenium 3.0 does not have HeadLess Driver (HtmlWebDriver)
* Official support for IE browser requires 9 or above version.
* No need to maximize the browser using driver.manage().window().maximize(); By default it will maximize the window.
* Selenium WebDriver has become the W3C Standard. The W3C standard will encourage compatibility across different software implementations of WebDriver API.
* All of the major browser vendors now support WebDriver specification and provide the necessary features along with the browser. For example, Microsoft came with EdgeDriver, and Apple supports the SafariDriver implementation.
* One of the major differences introduced in Selenium 3 was the introduction of the Appium project. The mobile-testing features that were part of Selenium 2 are now moved into a separate project named Appium. Appium is an open source mobile-automation framework for testing native, hybrid, and web mobile apps on iOS and Android platforms using the JSON-Wire protocol with Selenium WebDriver. Appium replaces the iPhoneDriver and AndroidDriver APIs in Selenium 2 that were used to test mobile web applications.

Selenium 4: After an extensive alpha and beta period to get everything right, Selenium 4 has now been officially released On October 13, 2021. It is available for Java, .Net, Python, Ruby, and Javascript.

A significant change under the hood for WebDriver is the complete W3C compliance of the WebDriver APIs. This standardization will eliminate the need for encoding and decoding the API requests by the JSON wire protocol in Selenium 3 and earlier versions for communication between browsers and test scripts. This means the WebDriver will now interact directly with the target browser.

**Features of Selenium 4**

Deprecation of Desired Capabilities

Desired Capabilities were primarily used in the test scripts to define the test environment (browser name, version, operating system) for execution on the Selenium Grid.

In Selenium 4, capabilities objects are replaced with Options. This means testers now need to create an Options object, set test requirements, and pass the object to the Driver constructor.

Listed below are the Options objects to be used going forward for defining browser-specific capabilities:

* Firefox – FirefoxOptions
* Chrome – ChromeOptions
* Internet Explorer (IE) – InternetExplorerOptions
* Microsoft Edge – EdgeOptions
* Safari – SafariOptions

|  |
| --- |
| private void setChromeDriver() {  ChromeOptions options = new ChromeOptions();  List<String> chromeOptions = new ArrayList<>();  LoggingPreferences logPrefs = new LoggingPreferences();  chromeOptions.add(String.format("--proxy-server=%s", proxies.getCurrentProxyModel().getIp()));  chromeOptions.add(String.format("--user-agent=%s", userAgent.randomUA()));  chromeOptions.add("--mute-audio");  logPrefs.enable(LogType.BROWSER, Level.ALL);  logPrefs.enable(LogType.PERFORMANCE, Level.INFO);  options.setCapability(CapabilityType.LOGGING\_PREFS, logPrefs);  options.addArguments(chromeOptions);  options.setBinary(this.driverLocation);  options.setHeadless(true);  // options.setProxy(this.proxies.getCurrentProxy());  options.setCapability("proxy", this.proxies.getCurrentProxy());  this.webDriver = new ChromeDriver(options);  Log.WINFO(this.workerName, this.workerColor, "Chrome Driver Set.");  } |

Relative Locators in Selenium 4

These are also known as Friendly Locators, and this functionality is being added to find out the element which is present nearby to other web element or, we can say that it can find the web elements based on GUI location.

There are five locators added in Selenium 4:

* below(): Web element located below for the specified element.
* toLeftOf() : Target web element which is present to the left of specified element.
* toRightOf(): Target web element which is presented to the right of a specified element.
* above(): Web element located above for the specified element.
* near() : Target web element which is away(approx. 50 pixels) from the specified element.

Note: All the above relative locators method support “withTagName” method.

|  |
| --- |
| WebElement book;  book = driver.Findelement(RelativeLocators.withTagName(“li”).toLeftOf(By.id(“pid1”))  .below(By.id(“pid2”)));  String id1=book.getAttribute (“id1”); |

Better Window/Tab Management in Selenium 4

There are several instances in test automation wherein one might need to open a particular link in a new tab or window to perform certain actions.If the user wants to access two applications in the same browser, the user can now do this

|  |
| --- |
| driver.get(https://www.google.com/);  driver.switchTo().newWindow(WindowType.WINDOW);  driver.navigate().to(https://www.crmpro.com/); |

If the user wants to open 2 URLs in two different tabs at the same time, the user can do that with the Selenium 4.

|  |
| --- |
| driver.get(https://www.google.com/);  driver.switchTo().newWindow(WindowType.TAB);  driver.navigate().to(https://www.crmpro.com/); |

Capture screenshot of specific web element:

Earlier, users can take a screenshot of the entire page as there was no provision to take the screenshot of the specific web element. But with Selenium 4, users can take the screenshot of a specific web element.

|  |
| --- |
| WebElement logo=driver.Findelement(By.xpath(“//div[@id=’divLogo’]//img”));  File file=logo.getScreenshotAs(OutputType.FILE);  File destFile =new File(“logo.png”);  FileUtils.copyFile(file,destfile); |

Modifications in the Actions Class

• click(WebElement) has replaced moveToElement(onElement).click() {}

• clickAndHold(WebElement) has replaced moveToElement(onElement).clickAndHold()

• contextClick(WebElement) has replaced moveToElement(onElement).contextClick()

• doubleClick(WebElement) has replaced moveToElement(element).doubleClick()

• release() which was ealier a part of org.openqa.selenium.interactions.ButtonReleaseAction class has now been moved to Action class as per new update of Selenium 4.0

|  |
| --- |
| Actions action = new Actions(driver);  action. click(WebElement); |

# Selenium Web driver Setup

Step.1. Configuring Java on machine: Download latest version of Java Development Kit (JDK).

Step.2. Download eclipse or any Java IDE of your choice

Step.3. Create Selenium Maven Project with Eclipse

Step 4. Add  Selenium and TestNG JAR files in pom.xml

|  |
| --- |
| <dependency>  <groupId>org.seleniumhq.selenium</groupId>  <artifactId>selenium-java</artifactId>  <version>3.141.59</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.testng/testng -->  <dependency>  <groupId>org.testng</groupId>  <artifactId>testng</artifactId>  <version>6.14.3</version>  <scope>test</scope>  </dependency> |

# ****Drivers & Launching Browsers****

Drivers download link: <https://www.seleniumhq.org/download/>

<https://phantomjs.org/download>

'WebDriver' is an interface and we are creating a reference variable 'driver' of type WebDriver, instantiated using Specified driver class listed below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Driver Type** | **Driver Name** | **Latest  version** | **Developed  by** | **Selenium  Version** | **Selenium Snippet** |
| HTML Unit Driver | Available along with Selenium server standalone jar file |  | Selenium | Selenium2 | Headless Browser Testing which means there is no GUI  WebDriver driver = new HtmlUnitDriver(); |
| htmlunit-driver-2.36.0.jar | 2.36.x | Selenium | Selenium 3 | WebDriver driver = new HtmlUnitDriver(); |
| PhantomJS Driver | PhantomJs.exe | 2.1.2 | Third Party | Selenium  2 & 3 | Headless Browser Testing but can capture screenshots. Test will run in the background  File src = newFile("E:\\path\\phantomjs.exe");  System.setProperty("**phantomjs.binary.path**", src.getAbsolutePath());  WebDriver driver = newPhantomJSDriver(); [or]  Capabilities caps = new DesiredCapabilities(); ((DesiredCapabilities) caps).setJavascriptEnabled(true);  ((DesiredCapabilities) caps).setCapability(“takesScreenshot”, true); ((DesiredCapabilities) caps).setCapability( PhantomJSDriverService.PHANTOMJS\_EXECUTABLE\_PATH\_PROPERTY, “//Users//Path//phantomjs” );  driver = new PhantomJSDriver(caps); |
| Mozilla Firefox Driver | NA | NA | NA | Selenium 2 | WebDriver driver = new FirefoxDriver(); |
| GeckoDriver.exe | 0.25.0 | Mozilla | Selenium 3 | System.setProperty("**webdriver.gecko.driver**","D://path//geckodriver.exe");    WebDriver driver = new FirefoxDriver(); |
| Chrome Driver | chromedriver.exe | 77.0.x.x | Chromium team | Selenium  2 &3 | System.setProperty("**webdriver.chrome.driver**", "E:\\chromedriver.exe");  WebDriver driver = new ChromeDriver(); |
| Internet Explorer Driver | IEDriverServer.exe | 3.14.0 | Selenium hq | Selenium  2 &3 | System.setProperty("**webdriver.ie.driver**","E\\IEDriverServer.exe");  DesiredCapabilities capabilities = DesiredCapabilities.internetExplorer();  capabilities.setCapability (InternetExplorerDriver.INTRODUCE\_FLAKINESS\_BY\_IGNORING\_SECURITY\_DOMAINS,true);  WebDriver driver = new InternetxplorerDriver(capabilities); |
| Opera Driver | operadriver.exe | 76.0.3809.132 | Opera |  | System.setProperty("**webdriver.opera.driver**", "E:\\operadriver.exe");  WebDriver driver = newOperaDriver(); |

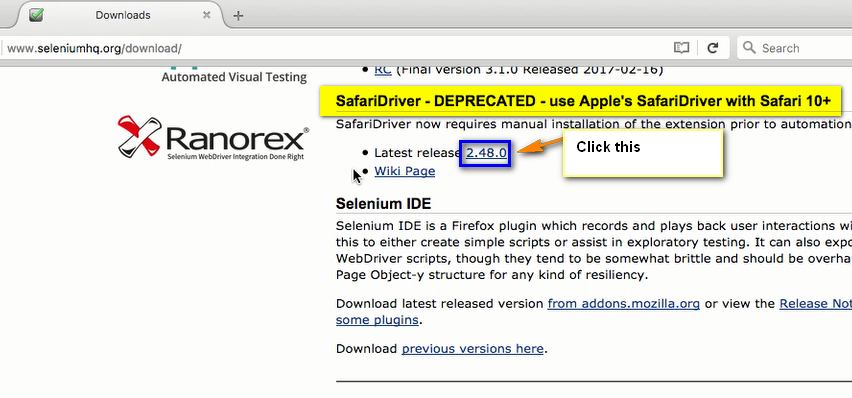
### Selenium compatibility with Safari Browser:

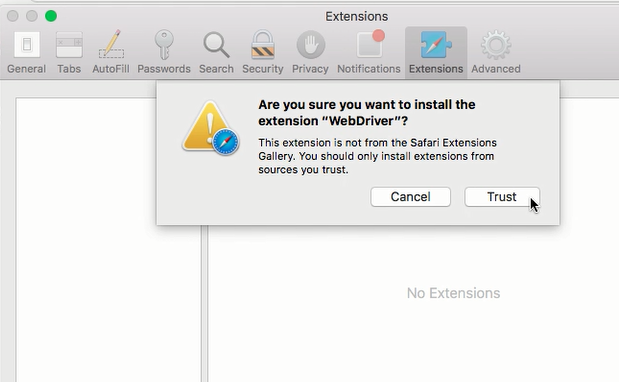
1. Safari browser that can be used on the windows machines till date is Safari 5. we cannot use the latest browser from version 6 to 10+ on windows machines.
2. Selenium also deprecated SafariDriver which can be used to run the Selenium Automation scripts
3. Selenium has provided 'Selenium WebDriver' extension for Safari browser. This 'Selenium WebDriver' will work only on the Safari Browsers having the browser version of 10 and above.
4. It is now, not possible to run the automation scripts on Windows machines

Hence, from the above points, we need the below System configuration and plug-ins for running the Selenium Automation scripts on safari browser:

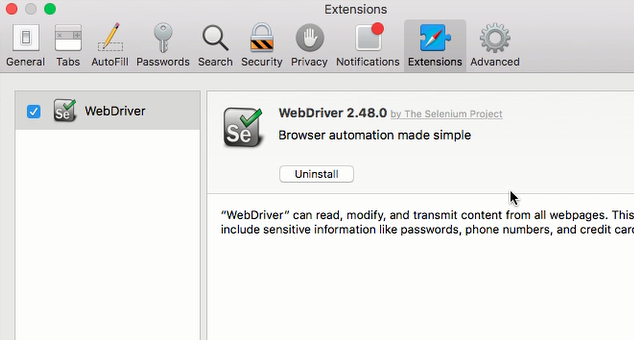
* Safari 10+ Browser
* Macintosh macOS Sierra Operating System
* 'Selenium WebDriver' Safari browser extension

**Practical execution of Selenium Scripts on Safari browser:**  
  
Using the above configuration, lets practically execute the Selenium Automation scripts on Safari Browser:

Step 1: we have to install 'Selenium WebDriver' extension for Safari browser. Go to Seleniumhq.org > downloads page and click on the latest release link as shown below:

Step 2: Observe that the 'Selenium WebDriver' extension for Safari browser will be downloaded and select to open it as shown below:Step 3: Click on 'Trust' button on the displayed pop-up window as shown below:

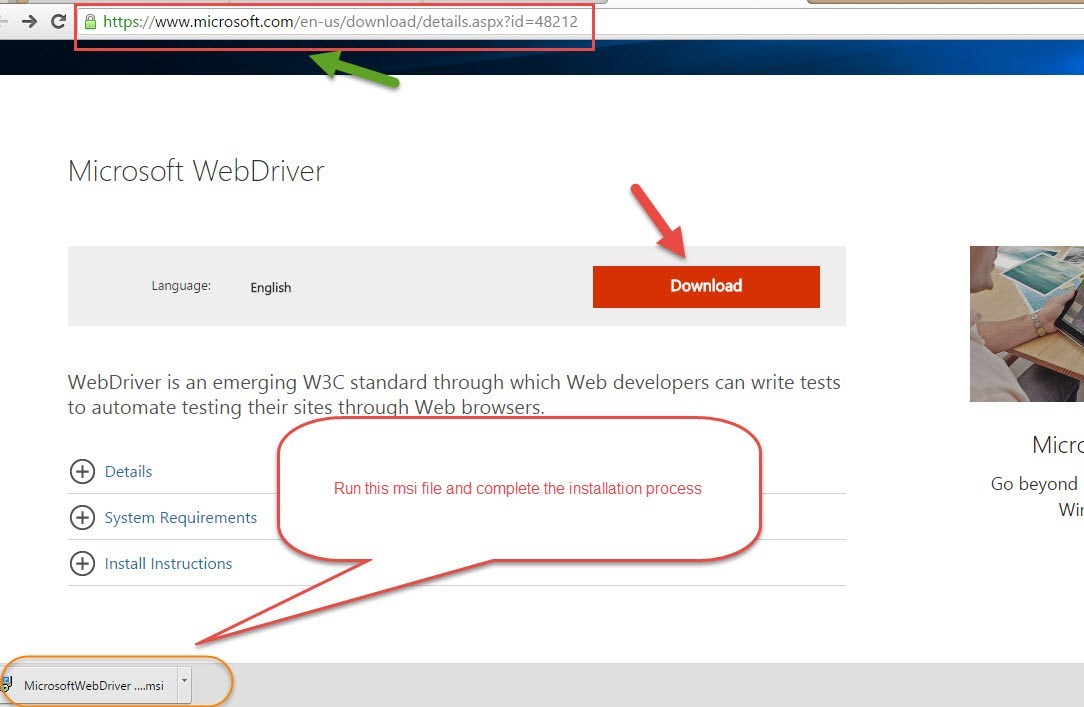
Step 4: observe that 'Selenium WebDriver' Extension is displayed as installed as shown below:

[](https://2.bp.blogspot.com/-CAqgRBe5DNs/WR_aAEBoujI/AAAAAAAAhqg/2LOnp0YrNrItTg-3Us1BdbDLiyHBh6siQCLcB/s1600/s23.png)

Step 5: write java code to launch SafariDriver

|  |
| --- |
| WebDriver driver = new SafariDriver(); |

### Launch Edge browser using Microsoft Edge Driver

Step 1- You have to *download Microsoft webdriver* and you have to install the same.

Step 2- Once you finish the installation part you will get .exe file in **Program files or Program file x86> Microsoft WebDriver > MicrosoftWebdriver.exe**

Step 3- write java code to launch Edge session.

|  |
| --- |
| // Set the driver path  System.setProperty("webdriver.edge.driver","WebDriver exe path in your machine");    //Start Edge Session  WebDriver driver=new EdgeDriver(); |

### Failed To Launch IE Driver Using Selenium WebDriver

**Error 1:** Exception in thread “main” org.openqa.selenium.remote.SessionNotFoundException: Unexpected error launching Internet Explorer. Protected Mode settings are not the same for all zones. Enable Protected Mode must be set to the same value (enabled or disabled) for all zones. (WARNING: The server did not provide any stacktrace information).

**Solution:**To fix this error, we need to enable protected mode for all zones.

Follow the below steps to enable protected mode for all zones.

Open Internet Explorer

Go to Tools menu – Internet Options

Select Security Tab

In Select a zone to view or change security settings choose Internet and select the check box of Enable protected mode (requires restarting Internet Explorer).

Similarly, select the check box of Enable protected mode (requires restarting Internet Explorer) for other three such as Local Intranet, Trusted sites, and Restricted sites.

**Error 2 :** Exception in thread “main” org.openqa.selenium.remote.SessionNotFoundException: Unexpected error launching Internet Explorer. Browser zoom level was set to 200%. It should be set to 100% (WARNING: The server did not provide any stacktrace information).

**Solution:**To fix this error, we need to set Zoom level of Internet Explorer browser to 100%

Follow the below steps to set Zoom level to 100%

Open Internet Explorer

To to View – Zoom – Select 100%

**IllegalStateException:**

Exception in thread "main" java.lang.IllegalStateException: The path  
to the chromedriver executable must be set by the  
webdriver.chrome.driver system property;

System.setProperty("webdriver.chrome.driver","\*absolute path to executable\*");

Exception in thread "main" org.openqa.selenium.remote.UnreachableBrowserException: Could not start a new session. Possible causes are invalid address of the remote server or browser start-up failure.

|  |
| --- |
| DesiredCapabilities capabilities = DesiredCapabilities.chrome();    capabilities.setCapability("chrome.binary", "C:\Users\Dell\AppData\Local\Google\Chrome\Application\chrome.exe");    WebDriver driver = new ChromeDriver(capabilities); |

### Browser Commands

#### **Open Webpage**:

The first step for automate an application, is navigate to that application using url. It can be done in different ways

1. Using get(“give URL”).  
2. Using navigate().to(“give URL”).

3. Using JavaScriptExecutor

**get():**

We can open the browser by using get("") a non-static method present in defined Driver class

|  |
| --- |
| driver.get("https://chercher.tech"); |

Open the saved webpage in a browser and copy the address in the address bar, this will have a protocol as the file

|  |
| --- |
| driver.get("file:///C:/Users/path/selenium.html"); |

[**Exception**](https://chercher.tech/java/exceptions-selenium-webdriver)in thread "main" org.openqa.selenium.WebDriverException: f.QueryInterface is not a function Command duration or timeout: 94 milliseconds  
  
**Solution**:  
Whenever we pass the website address to the get() method. It first of checks for the protocol in the address, but in the above program, we did not mention any the protocol.

**navigate().to():**

Both methods are works for navigating but using navigate.().to() we will get more features -  
1. driver.navigate().back().

2. driver.navigate().forward().

3. driver.navigate().refresh().

Code for navigate to url using navigate().to().

|  |
| --- |
| driver.navigate().to("http://www.facebook.com"); |

**JavaScriptExecutor**

Do you know that we can open a URL without using any browser as well?  It is frequently asked interview question as well. Let’s learn it.

Let’s perform some steps first:

1. Open a browser.
2. Press F12.
3. Switch to Console tab.
4. Type ” window.location=’https://www.redbus.in’ “(exclude double quotes) and hit Enter key.
5. You will notice that redbus website is loaded.

|  |
| --- |
| JavascriptExecutor js= (JavascriptExecutor) driver;  String script = "window.location = \'http://facebook.com'";  js.executeScript(script); |

#### Closing Browser

**close()**

close() is a webdriver command which closes the browser window which is currently in focus.

During the automation process, if there are more than one browser window opened, then the close() command will close only the current browser window which is having focus at that time. The remaining browser windows will not be closed. The following code can be used to close the current browser window

|  |
| --- |
| driver.close(); |

**quit()**

quit() is a webdriver command which calls the driver.dispose method, which in turn closes all the browser windows and terminates the WebDriver session. If we do not use quit() at the end of program, the WebDriver session will not be closed properly and the files will not be cleared off memory. This may result in memory leak errors.

The following code can be used to close all the browser windows:

|  |
| --- |
| driver.quit(); |

If the Automation process opens only a single browser window, the close() and quit() commands work in the same way. Both will differ in their functionality when there are more than one browser window opened during Automation.

Once quit() is used, then the driver object becomes not-reachable, so you cannot perform any operation on the driver object. Close() browser make the driver object still avail to the user.

**close a tab**

You can close the current tab using the close() method in [selenium](https://chercher.tech/java/index-selenium-webdriver) but after closing the tab make sure you are using **switchTo().Wi**ndow() command to get the [control of other windows](https://chercher.tech/java/handle-multiple-windows-tabs-selenium-webdriver)

#### Get commands:

1. In WebDriver, this method fetches the title of the current web page. It accepts no parameter and returns a String.

|  |
| --- |
| String Title = driver.getTitle(); |

1. In WebDriver, this method fetches the string representing the Current URL of the current web page. It accepts nothing as parameter and returns a String value.

|  |
| --- |
| String CurrentUrl = driver.getCurrentUrl(); |

1. In WebDriver, this method returns the source code of the current web page loaded on the current browser. It accepts nothing as parameter and returns a String value.

|  |
| --- |
| String PageSource = driver.getPageSource(); |

#### Maximize Browser:

Most of the time, when you open the firefox browser with selenium, the browser will be in minimized mode, but in a few scenarios, we might want to maximize the window. we can maximize the browser window with the help of maximize() method present in the window class

Note: Not required to do this explicitly from selenium3

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

#### Navigation commands:

1. In WebDriver, this method loads a new web page in the existing browser window. It accepts String as parameter and returns void.

|  |
| --- |
| driver.navigate().to("url"); |

1. In WebDriver, this method enables the web browser to click on the **forward** button in the existing browser window. It neither accepts anything nor returns anything.

|  |
| --- |
| driver.navigate().forward(); |

1. In WebDriver, this method enables the web browser to click on the **back** button in the existing browser window. It neither accepts anything nor returns anything.

|  |
| --- |
| driver.navigate().back(); |

1. In WebDriver, this method refresh/reloads the current web page in the existing browser window. It neither accepts anything nor returns anything.

|  |
| --- |
| driver.navigate().refresh(); |

#### ****Browser size****

**Get Browser Size:**

We can get the size of the browser window by using the getSize() method present in the Dimension class, and it returns the Dimension Class type value.

|  |
| --- |
| Dimension size = driver.manage().window().getSize();  System.out.println("The size of the window : "+size); |

The Output of the program:  
The size of the window : (1024, 546)

If you don't want value with x and y instead of that, if you need to get height or width, then use the following way.

|  |
| --- |
| int height = driver.manage().window().getSize().height;  int width =driver.manage().window().getSize().width;  (or)  int height1 = driver.manage().window().getSize().getHeight();  int width2 = driver.manage().window().getSize().getWidth(); |

**Set Browser Size:**

We can set the size of the browser window by using the setSize() method present in the window class. It accepts the point class constructor as the argument.

|  |
| --- |
| driver.manage().window().setSize(new Dimension(200, 200)); |

#### ****Refresh Page:****

1. Using SendKeys

|  |
| --- |
| driver.findElement(By.tageName("body")).sendKeys(Keys.F5); |

|  |
| --- |
| driver.findElement(By.id("firstname-placeholder")).sendKeys("uE035"); |

1. Using navigate.refresh()

|  |
| --- |
| driver.navigate().refresh(); |

1. Using navigate.to()

|  |
| --- |
| driver.navigate().to(driver.getCurrentUrl()); |

1. Using get()

|  |
| --- |
| driver.get(driver.getCurrentUrl()); |

# ****Web Elements****

Elements are the different components that are present on web pages. The most common elements we notice while browsing are:

* Text boxes/Edit Box/Text Area
* Buttons
* Images/ Image Link/Image Button
* Hyperlinks
* Radio buttons/ Check boxes
* Text area/ Error messages
* Drop down box/ List box/ Combo box
* Web Table/ HTML Table
* Frame

### **Locating Browser Elements**

Every element on a web page will have attributes (properties). Elements can have more than one attribute and most of these attributes will be unique for different elements.

### Methods to Find Elements:

#### Find Element

If we want to perform any operation on webelement, we have to find them using locators, but locators find them on the browser, to make those elements available to selenium we have to find them with selenium Method.  
  
Selenium provides method 'findElement()'. This method finds the element with the help of By class methods(nothing but locators).

|  |
| --- |
| WebElement example = driver.findElement(By.locator("locator")); |

* Use this method to access any single element on the web page.
* It returns the first matching element of the specified locator.
* It throws a [NoSuchElementException](https://chercher.tech/java/exceptions-selenium-webdriver" \l "NoSuchElementException" \t "_blank" \o "exceptions-selenium-webdriver#NoSuchElementException) exception when it fails to locate the element(when an element is not there).
* If you are in page level, findElement() will look for the element in page level; it will not look for the element which is present inside a Frame and vice versa

#### Find Elements

We can use the [findElement()](https://chercher.tech/java/locators-find-elements-selenium-webdriver" \l "find-element" \t "_blank" \o "locators-find-elements-selenium-webdriver#find-element) method to find a single element, but sometime we may want to perform an operation on more than one element, in those cases, we cannot write [findElement()](https://chercher.tech/java/find-element" \t "_blank" \o "find-element) method for all the elements.  
  
In selenium we have a findElements() method which can return multiple elements matching with given locator,   findElements() returns List<WebElement>.  
  
Note : findElements() returns List<WebElement> not WebElements

|  |
| --- |
| List<WebElement> elems = driver.findElements(By.locator("locator")); |

List operations:

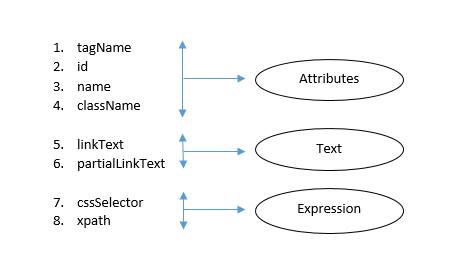
|  |
| --- |
| for (int i = 0; i < elems.size(); i++) {  elems.get(i).click();  }  (or)  for(WebElement element:elems)  {  System.out.println(element.getText());  } |

* It gives back the whole list of all(not all but all the elements which are loaded by that time) the elements matching the specified locator.
* If the element doesn’t exist or not available on the page, then the return value will be an empty list.
* It will not throw 'NoSuchElementFoundException' in case element is not present
* User will be allowed to perform only list operations on the return value; we have to iterate through each item to perform webelement operations
* If you are in page level, findElements() will look for the element in page level; it will not look for the element which is present inside a Frame and vice versa

### Locators:

There are multiple ways to uniquely identify a web element within the web page.

Selenium WebDriver uses **8** element locators



The locators are:

1. **By.id**

|  |
| --- |
| driver.findElement(By.id(“id value”)); |

1. **By.name**

|  |
| --- |
| driver.findElement(By.name(“name value/locator name”)); |

1. **By.className**

|  |
| --- |
| By.className(“class name value”) |

1. **By.tagName**

|  |
| --- |
| By.tagName(“tag name value”) |

1. **By.linkText**

|  |
| --- |
| By.linkText(“Link Text Value”) |

1. **By.partialLinkText**

|  |
| --- |
| By.partialLinkText(“Partial Link Text Value”) |

1. **By.css**

|  |
| --- |
| By.cssSelector(“value”) |

1. **By.xpath**

|  |
| --- |
| By.xpath(“xpath value”) |

### Dynamic Web elements

Web elements frequently changes for each build/page reload. Like ‘bttn-3455-textE1’ are change to ‘bttn-3456-textE1’

**Handling Dynamic Web elements:**

1. **Element with Index:** Each web element are located in different hierarchical level and we can search specific element from similar elements within hierarchical by using level index.
2. **Absolute XPath**: Absolute XPath Method uses complete path from the Root Element to the desire element. Expression either begin with the '/' or the root node. Normally it is not considered as good practice to use absolute XPath however it solve the Dynamic element problem.
3. **Relative XPath:** The relative XPath expressions are a lot more compact and use forward double slashes '//'. These XPaths can select the elements at any location that matches the selection criteria and doesn't necessarily begin with root node.

**Single Attribute:**

|  |
| --- |
| //<HTML tag>[@attribute\_name='attribute\_value']  or  //\*[@attribute\_name='attribute\_value']  E.g.  //input[@id='Email']  or  //\*[@id='Email'] |

**Multiple Attribute:**

|  |
| --- |
| //<HTML tag>[@attribute\_name1='attribute\_value1'][@attribute\_name2='attribute\_value2]    or   //\*[@attribute\_name1='attribute\_value1'][@attribute\_name2='attribute\_value2]  E.g.  //input[@id='Email'][@name='Email']  or  //\*[@id='Email'][@name='Email'] |

**Using AND/OR:**

|  |
| --- |
| //<HTML tag>[@attribute\_name1='attribute\_value1' and @attribute\_name2='attribute\_value2]  or  //\*[@attribute\_name1='attribute\_value1' or @attribute\_name2='attribute\_value2]  E.g.  //input[@id='Email' and @name='Email']    or   //\*[@id='Email' or @name='Email'] |

**contains():**It is used to identify an element, when we are familiar with some part of the attributes value of an element.

|  |
| --- |
| //<HTML tag>[contains(@attribute\_name,'attribute\_value')]  or   //\*[contains(@attribute\_name,'attribute\_value')]  E.g.  //input[contains(@id,'Email')]    or  //\*[contains(@id,'Email') |

**starts-with():**It is used to identify an element, when we are familiar with the attributes value (starting with the specified text) of an element.

|  |
| --- |
| //<HTML tag>[starts-with(@attribute\_name,'attribute\_value')]    or   //\*[starts-with(@attribute\_name,'attribute\_value')]  E.g.  //input[starts-with(@id,'Ema')]    or   //\*[starts-with(@id,'Ema')] |

**text(): This mechanism is used to locate an element based on the text available on a webpage**

|  |
| --- |
| **//\*[text()='New look for sign-in coming soon']** |

**last(): Selects the last element (of mentioned type) out of all input element present**

|  |
| --- |
| **findElement(By.xpath("(//input[@type='text'])[last()]"))**  **findElement(By.xpath("(//input[@type='text'])[last()-1]"))** |

**position(): Selects the element out of all input element present depending on the position number provided**

**In below given xpath, [@type=’text’] will locate text field and function [position()=2] will locate text filed which is located on 2nd position from the top.**

|  |
| --- |
| **findElement(By.xpath("(//input[@type='text'])[position()=2]"))**  **or**  **findElement(By.xpath("(//input[@type='text'])[2]"))** |

**Finding elements using index**

**By providing the index position in the square brackets, we could move to the nth element. Based on the below xpath, we could identify the Last Name field.**

|  |
| --- |
| **findElement(By.xpath("//label[2]"))** |

**following:**

**By using this we could select everything on the web page after the closing tag of the current node.**

**To identify the input field of type text after the FirstName field, we need to use the below xpath.**

|  |
| --- |
| **//\*[@id='FirstName']/following::input[@type='text']** |

**preceding: Selects all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes**

**To identify the input field of type text before the LastName field, we need to use the below xpath.**

|  |
| --- |
| **//\*[@id='LastName']//preceding::input[@type='text']"** |

**child:**

**select all the nodes that are Child nodes of given expression**

**It will select all the nodes that are Child nodes of table.**

|  |
| --- |
| **//child::table** |

**parent:**

**parent node may be either the root node or an element node.**

**The root node has no parent; therefore, when the context node is the root node, the parent axis is empty. For all other element nodes the parent axis contains one node.**

|  |
| --- |
| **parent::node()**  **The below example will selects the parent node of the input tag of Id='email'. : //input[@id='email']/parent::\***  **or**  **//input[@id='email']/..** |

**following-sibling**

**The following-sibling axis selects those nodes that are siblings of the context node (that  
is, the context node and its sibling nodes share a parent node) and which occur later in  
document order than the context node.**

|  |
| --- |
| **//select[@id='month']/following-sibling::\***  **or //select[@id='month']/following-sibling::select/** |

**preceding Sibling**

**The preceding-sibling axis selects those nodes which are siblings of the context node (that is, the context node and its sibling nodes share a parent node) and which occur earlier in document order than the context node.**

|  |
| --- |
| **//select[@id='day']/preceding-sibling::select/**  **or**  **//select[@id='day']/preceding-sibling::\*** |

**Ancestor**

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

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

|  |
| --- |
| //\*[text()='Enterprise Testing']//ancestor::div |

**Descendant**

Selects the descendants of the current node as shown in the below screen.

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

|  |
| --- |
| ***//nav[@class=’fusion-main-menu’]//\*[@id=’menu-main’]//descendant::li*** |

**Evaluate Xpath/CSS in chrome - elements:**

1. Press F12 to open up Chrome DevTools.
2. Elements panel should be opened by default.
3. Press Ctrl + F to enable DOM searching in the panel.
4. Type in XPath or CSS selectors to evaluate.
5. If there are matched elements, they will be highlighted in DOM.  
   However, if there are matching strings inside DOM, they will be considered as valid results as well. For example, CSS selector header should match everything (inline CSS, scripts etc.) that contains the word header, instead of match only elements.

**Evaluate Xpath/CSS in chrome /Firefox- console:**

1. Press F12 to open up Chrome DevTools.
2. Open console panel, press command in control “>”
3. I f you want to write XPath write with in $x(“XPath Expressions”) and hit enter key.
4. If you want to write CSS write with in $$(“CSS Selector”) and hit enter key.
5. It will return result in an array and you can view count of matching elements as well.
6. Expand array and mouse hover on result, It will highlight web element on web page.

**Evaluate Xpath/CSS in Firefox :**

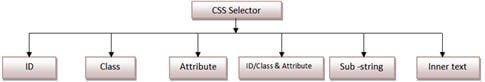
Firefox browser has also its in built developer tools but it is not advance as much as chrome DevTools.

1. Install Firebug and Firepath extension and restart your browser.
2. Press F12. It will open dev tool.
3. Go to HTML panel and press ctrl +F. It will open DOM search box. Now type XPath or CSS expressions as we do for Chrome.

**cssSelector:**

When we don't have an option to choose Id or Name, we should prefer using CSS locators as the best alternative.  
CSS is "Cascading Style Sheets" and it is defined to display HTML in structured and colorful styles are applied to webpage.

* CSS has more Advantage than Xpath
* CSS is much more faster and simpler than the Xpath.
* In IE Xpath works very slow, where as Css works faster when compared to Xpath



**ID Selector**

**#**– Symbolic notation

 ID attribute whose value is defined as “Email”.

|  |
| --- |
| tagName[attributename=attributeValue]  E.g.  input[id=email] or input#email  input[name=email][type=text] |

**Class Selector**

**.**– Symbolic notation

Has a Class attribute whose value is defined as “submitbtn”

|  |
| --- |
| tagName.attributeClassName  E.g.  input.submitbtn |

**ID and Attribute**

|  |
| --- |
| *input#Passwd[type='password'][name='Passwd']”.* |

**Match a prefix**

It is used to correspond to the string with the help of a matching prefix.

**^**– Symbolic notation to match a string using prefix.

|  |
| --- |
| *input#Passwd[name^='Pass']*  input[id^='ema'] |

**Match a suffix**

It is used to correspond to the string with the help of a matching suffix.

**#**– Symbolic notation to match a string using suffix.

|  |
| --- |
| input[id$='mail']  *input#Passwd[name$='wd']* |

**Match a substring**

It is used to correspond to the string with the help of a matching substring.

**\***– Symbolic notation to match a string using sub string.

|  |
| --- |
| *input#Passwd[name\*='wd']*  input[id\*='mai'] |

**Inner text**

 tag representing the hyperlink has a text enclosed within.

|  |
| --- |
| Input:contains(text present on web page) |

**Child Selectors**

|  |
| --- |
| parentLocator>childLocator  E.g.  div#buttonDiv>button  or  div#buttonDiv button |

Nth child

|  |
| --- |
| parentLocator childLocator:nth-of-type(n)  E.g.  ul#automation li:nth-of-type(n) |

### Interacting with Text Boxes/Edit Box/Text Area

* **sendKeys()** method is used to enter text into the textbox
* **clear()** method is used to clear the text in the textbox

The sendKeys() method used to pass the Keyboard keys or text into editable elements (text bar, text area, button) without replacing the previously available content. Using multiple sendkeys to a particular field, selenium appends all of the text one by one.

|  |
| --- |
| driver.findElement(By.id(“Email”)).sendKeys(“India”);  driver.findElement(By.id(“Email”)).clear(); |

**Checking if the text box is editable**

we check text field is editable, if yes then try to enter string

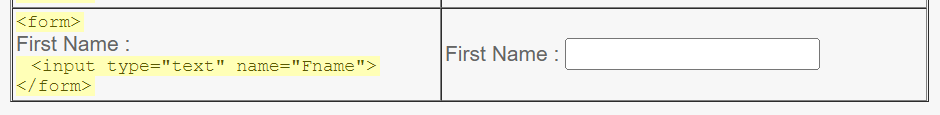
|  |
| --- |
| driver.findElement(By.ElementLocator(“value”).isEnabled(); |

**HTML component**

**Text Box /Edit Box:**

|  |
| --- |
| <input type="text" id="fname" name="fname"> |

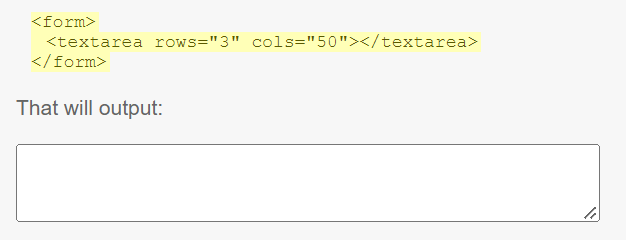
The <input> tag specifies an input field where the user can enter data. The <input> element can be displayed in several ways, depending on the type attribute.



### Text Area:

|  |
| --- |
| <textarea id="w3review" name="w3review" rows="4" cols="50"> data here </textarea> |

The <textarea> tag defines a multi-line text input control.



### Interacting with Buttons

There are three types of buttons:

* submit — Submits the current form data. (This is default.)
* reset — Resets data in the current form.
* button — Just a button. Its effects must be controlled by something else (that is, with [JavaScript](https://html.com/javascript/)).

**submit()**

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

|  |
| --- |
| driver.findElement(By.id("SubmitLogin")).submit(); |

**Click()**

used to click on the button and wait for page load to complete before proceeding to the next command.

|  |
| --- |
| driver.findElement(By.id("SubmitLogin")).click(); |

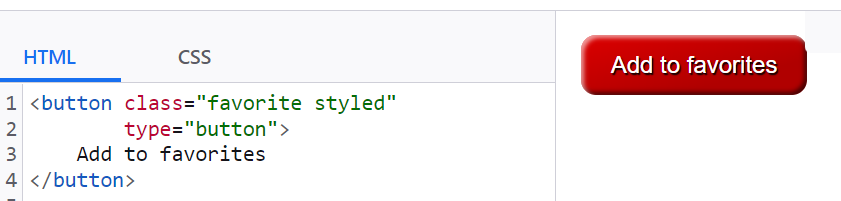
**HTML Component**

|  |
| --- |
| <button type="button">Click Me!</button> |

The <button> tag defines a clickable button.

|  |
| --- |
| <input type="submit" value="Another Button" /> |

|  |
| --- |
| <input type="button" onclick="alert('Hello World!')" value="Click Me!"> |



### Interacting with Images/ Image Link/Image Button/Hyperlink

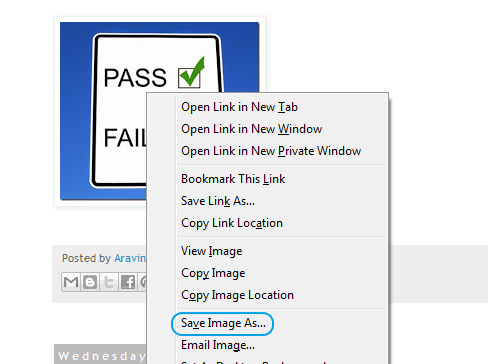
If we are testing an image, we might want to:

1. Download the image
2. Upload the image
3. Click on the image link
4. Retrieve the image title, etc.
5. **Download the image**

* For saving Image from web page In selenium webdriver, We have to perform bellow given actions.
* Right click on Image
* Select "Save Image As" option from mouse right click context menu.
* Enter file name In save Image dialog
* Press Save button.

Right click on Image using contextClick() method of Actions class

As you know, "Save Image As" option will display when you right click on any Image. We will use contextClick() method of WebDriver Actions class to right click on Image as shown In bellow Image.



|  |
| --- |
| //Locate Image  WebElement Image =driver.findElement(By.xpath("//img[@border='0']"));  //Right click on Image using contextClick() method.  Actions action= new Actions(driver);  action.contextClick(Image).build().perform();  //To perform press Ctrl + v keyboard button action.  action.sendKeys(Keys.CONTROL, "v").build().perform();  Thread.sleep(3000);  Robot robot = new Robot();  // To press D key.  robot.keyPress(KeyEvent.VK\_D);  // To press : key.  robot.keyPress(KeyEvent.VK\_SHIFT);  robot.keyPress(KeyEvent.VK\_SEMICOLON);  robot.keyRelease(KeyEvent.VK\_SHIFT);  // To press \ key.  robot.keyPress(KeyEvent.VK\_BACK\_SLASH);  // To press "test" key one by one.  robot.keyPress(KeyEvent.VK\_T);  robot.keyPress(KeyEvent.VK\_E);  robot.keyPress(KeyEvent.VK\_S);  robot.keyPress(KeyEvent.VK\_T);  // To press Save button.  robot.keyPress(KeyEvent.VK\_ENTER); |

Method 2:

Find the image element in a certain way (I'm using cssSelector)

Get the src attribute

Create a java URL

Create a BufferedImage using ImageIO java class

Use ImageIO to save the image with a preferred image extension and a location

|  |
| --- |
| WebElement logo = driver.findElement(By.cssSelector(".forum-logo"));  String logoSRC = logo.getAttribute("src");  URL imageURL = new URL(logoSRC);  BufferedImage saveImage = ImageIO.read(imageURL);  ImageIO.write(saveImage, "png", new File("logo-forum.png")); |

1. **Upload the image**

We have following options to upload files in a Remote Selenium WebDriver:

SendKeys

Robot Class

AutoIT tool

Jacob API

**SendKeys** method. It directly applies to input tags which have an attribute as type=’file’.

|  |
| --- |
| WebElement addFile = driver.findElement(By.xpath(".//input[@type='file']"));  addFile.sendKeys("/Users/neeraj.kumar/Desktop/c1.jpeg"); |

**Robot class** is an AWT class package in Java. This is also a very good option to choose for the Upload file in selenium. This will help to automate a Windows-based alert or pop up, print pop up or native Windows screen. This is independent of the Operating System.

|  |
| --- |
| public void fileUpload (String path) {  StringSelection strSelection = new StringSelection(path);  Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();  clipboard.setContents(strSelection, null);  Robot robot = new Robot();  robot.delay(300);  robot.keyPress(KeyEvent.VK\_ENTER);  robot.keyRelease(KeyEvent.VK\_ENTER);  robot.keyPress(KeyEvent.VK\_CONTROL);  robot.keyPress(KeyEvent.VK\_V);  robot.keyRelease(KeyEvent.VK\_V);  robot.keyRelease(KeyEvent.VK\_CONTROL);  robot.keyPress(KeyEvent.VK\_ENTER);  robot.delay(200);  robot.keyRelease(KeyEvent.VK\_ENTER);  } |

**AutoIT** is an external automation tool and not provided by the Selenium community. Initially, AutoIT was used to automate native Windows related pop-ups, however, a drawback of using AutoIT is that it creates .exe file and runs only on Windows. It is not advisable to use AutoIT for file uploads.

|  |
| --- |
| WinWaitActive("File Upload");  Send("/Users/neeraj.kumar/Desktop/c1.jpeg");  Send("{ENTER}") |

Jacob provides the API technique to upload files using Selenium. Again, to perform a file upload in Selenium WebDriver using Jacob API you would need a .dll file. That means it won’t work for a Mac or Linux operating system.

1. **Click()**

used to click on the link and wait for page load to complete before proceeding to the next command.

|  |
| --- |
| driver.findElement(By.id("SubmitLogin")).click(); |

1. **Retrieve the image title**

|  |
| --- |
| WebElement Div\_elem=Driver.findElement(By.id("soandso"));  String Div\_text = Div\_elem.getText();  String img\_text=Div\_elem.findElement(By.tagName("img")).getAttribute("alt"); |

**HTML Component**

**Image:**

|  |
| --- |
| <img src="pic\_trulli.jpg" alt="Italian Trulli"> |
| <input type="image" src="img\_submit.gif" alt="Submit" width="48" height="48"> |

<input> elements of type image are used to create graphical submit buttons, i.e. submit buttons that take the form of an image rather than text.

**Image Link:**

An image as a link

|  |
| --- |
| <a href="default.asp"><img src="smiley.gif" alt="HTML tutorial" style="width:42px;height:42px;"></a> |

**Image Map:**

Image maps are single images with clickable areas.

|  |
| --- |
| <img src="org.png" alt="Staff Organisation Chart" usemap="#Map">  <map name="Map" id="Map">  <area  shape="rect"  coords="176,14,323,58"  href="…"  alt="Chairman"  >  <area  shape="rect"  coords="81,75,226,114"  href="…"  alt="Secretary"  >  </map> |

We have added the attribute usemap to the img tag. Then within the map tag we have an area tag. Each of these areas creates a clickable area that acts as a button.

**Image Button:** Embed Image in HTML Button

|  |
| --- |
| <button type="button"> <img src="https://bit.ly/3BF9FD7" height ="80" width="100" /></button> |

**HyperLink:**

|  |
| --- |
| <a href="url">link text</a> |

### Interacting with Radio buttons/ Check boxes

Using Click() method in Selenium we can perform the action on the Radio button and on Checkbox.

Before performing the click even on the Radio buttons or checkboxes we will have to verify

* If Radio button or Checkbox is displayed on the webpage
* If Radio button or Checkbox is enabled on the webpage
* Check the default selection of the Radio button or Checkbox

Above mentioined verification can be done using predefined menthods in Selenium

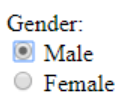
* isDisplayed()- this returns a Boolean value, if it returns true then said radio button is present on the webpage or it returns False.
* isEnabled() - this returns a Boolean value, if it returns true then said radio button is enabled on the webpage or it returns False
* isSelected() - this returns a Boolean value, if it returns true then said radio button is selected or it returns False

|  |
| --- |
| driver.findElement(By.id("ccc")).click();  driver.findElement(By.id("ccc")). isDisplayed ();  driver.findElement(By.id("ccc")). isEnabled ();  driver.findElement(By.id("ccc")). isSelected (); |

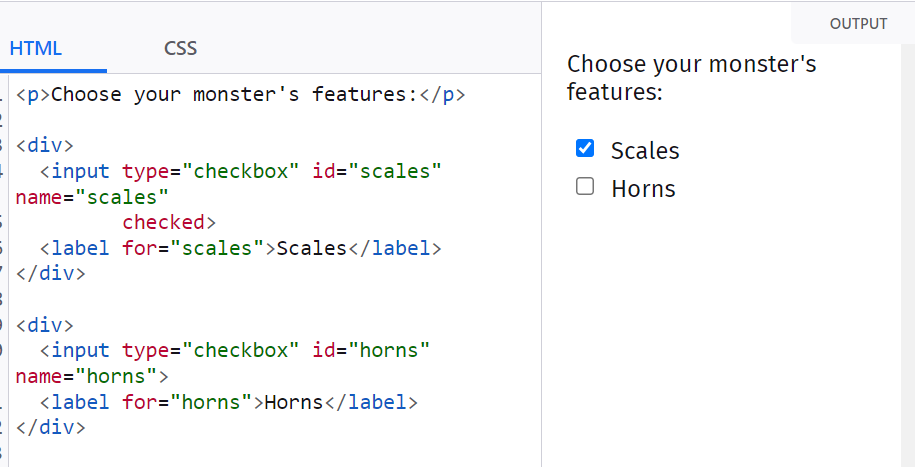
**HTML Component:**

**Radio Button:**

|  |
| --- |
| <form>  <label>  Gender:  </label> <br>  <input type="radio" id="gender" name="gender" value="male"/> Male  <br>  <input type="radio" id="gender" name="gender" value="female"/> Female <br/>  </form> |



**CheckBox:**



### Interacting with Drop down / List box/ Combo box

To perform any action, the first task to do is to identify the element group. Generally, while working with [*Selenium*](https://www.edureka.co/testing-with-selenium-webdriver), you might have to select some values from the drop-down list and also perform other activities and validate them.

HTML

|  |
| --- |
| <label for="cars">Choose a car:</label>  <select name="cars" id="cars">   <option value="volvo">Volvo</option>   <option value="saab">Saab</option>   <option value="mercedes">Mercedes</option>   <option value="audi">Audi</option> </select> |

|  |
| --- |
| <div class="dropdown">   <button onclick="myFunction()" class="dropbtn">Dropdown</button>   <div id="myDropdown" class="dropdown-content">     <a href="#">Link 1</a>     <a href="#">Link 2</a>     <a href="#">Link 3</a>   </div> </div> |

**Different ways to select the dropdowns**

* 1. Select Class
  2. Click method
  3. [Actions class](http://chercher.tech/java/mouse-keyboard-actions-class-selenium-webdriver) click methods
  4. Sendkeys method
  5. Javascript method

**Drop down with Select tag use Select Class :**

The *Select*class is a *[Webdriver](https://www.edureka.co/blog/selenium-tutorial" \t "_blank)*class which basically provides the implementation of the HTML SELECT tag. A Select tag provides the helper methods with select and deselect options.

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

1. **selectByVisibleText:** It takes a parameter of String which is one of the *value of Select element*and it returns nothing.
2. **selectByIndex:**It takes the integer parameter which is the index value of *Select element* and it returns nothing.
3. **selectByValue:**It takes a String parameter which is one of the values of *Select element* and it does not return anything.
4. **getOptions:** This method helps to get all the options belonging to the Select tag. It takes no parameter and returns *List<WebElements>*.

|  |
| --- |
| Select oSelect =new Select(driver.findElement(By.id("search-box")));  oSelect.selectByVisibleText("Blog");  oSelect.selectByIndex(2);  oSelect.selectByValue("Selenium Certification training");  List <WebElement> elementCount = oSelect.getOptions(); |

Multiple Options Select:

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

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

**isMultiple:** method returns True if the dropdown is multi-value dropdown otherwise it returns false for single value dropdown

|  |
| --- |
| boolean singleOrMultiple = oSelect.isMultiple(); |

**getAttribute:** If we try to get the attribute value that doesn't exists for the tag, it will return null value.

|  |
| --- |
| String singleOrMulti = dropdownElement.getAttribute("multiple");  // compare the result  if (singleOrMulti != null) {  System.out.println("Dropdown is Multi value dropdown");  }else{  System.out.println("Dropdown is Single value dropdown");  } |

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

1. deselectAll(): It clears all selected entries. This is only valid when the drop-down element supports multiple selections.
2. deselectByIndex: It deselects the option at the given index
3. deselectByValue : This method helps in deselecting the option whose “*value*” attribute matches the specific parameter.
4. deselectByVisibletext : **This** method helps in deselecting the option that displays the text matching the parameter.

|  |
| --- |
| oSelect.*deselectAll()*;  oSelect.*deselectByIndex(2);*   oSelect.*deselectByValue(“13”);*   oSelect.deselectByVisibletext *(“text”);* |

###### **Get Methods**

**getOptions()** - method return all the options present in dropdown irrespective of whether selected or not

|  |
| --- |
| WebElement dropdownElementMulti = driver.findElement(By.xpath("//select[@id='second']"));  Select dropdownMulti = new Select(dropdownElementMulti);  // get all the options and store it in list  List<WebElement> allElementsMulti = dropdownMulti.getOptions(); |

getText() – To get the value for that option

|  |
| --- |
| String dropdownOptionValue = element.getText(); |

**getFirstSelectedOption()** - method provides us option element which is selected currently.

|  |
| --- |
| WebElement dropdownElement = driver.findElement(By.xpath("vvvv"));  // create object for Select class  Select dropdown = new Select(dropdownElement);  // select 1st selected option from the dropdown options  WebElement selectedOption = dropdown.getFirstSelectedOption(); |

**getAllSelectedOptions()** *-* method provides us list of elements which are selected currently in Multi-Selection Box field.

|  |
| --- |
| Select dropdownMulti = new Select(dropdownElementMulti);  List<WebElement> selectedOptions = dropdownMulti.getAllSelectedOptions(); |

###### Check Particular Option is present in the dropdown or not ?

* Find the dropdown using [findElement()](http://chercher.tech/java/find-elements-in-selenium" \t "_blank" \o "find-elements-in-selenium) Method
* Create an object for Select class passing above found element
* use getOptions() method present in Select class object and store values(it returns webelement in list) in List
* iterate the list to get the text from the Options one by one
* compare the result text with the expected result.

|  |
| --- |
| WebElement dropdownElement = driver.findElement(By.xpath("//select[@id='animals']"));  // create object for Select class  Select dropdown = new Select(dropdownElement);  // get all the options and store it in list  List allElements = dropdown.getOptions();  System.out.println("Values present in Single Value Dropdown");  for (WebElement element : allElements) {  // iterate over each element and print the text  String dropdownOptionValue = element.getText();  if (dropdownOptionValue.equals("Avatar")) {  System.out.println("Avatar Option is present though it is not a cat family :)");  }  } |

###### Check Elements in drop down are sorted

We have a few ways to verify whether options are sorted or not in dropdowns with webdriver.

* For loop [ basic]
* addAll()
* TreeSet
* stream().collect()

For loop

* Create a List tempList variable
* While iterating the option in the dropdown, add values to tempList (along with originalList)
* Now sort the tempList, sorting of tempList will not affect the originalList because we have created two different objects
* [Compare the two Lists](http://chercher.tech/java/testng-assertions-selenium-webdriver)

|  |
| --- |
| WebElement element = driver.findElement(By.xpath("//select[@id='animals']"));  Select se = new Select(element);  List<String> originalList = new ArrayList();  List<String> tempList = new ArrayList();  for (WebElement e : se.getOptions()) {  originalList.add(e.getText());  tempList.add(e.getText());  }  Collections.sort(tempList);  Assert.assertEquals(tempList, originalList); |

**TreeSet**

* We can use the TreeSet for verification dropdown option order; we have to create a TreeSet object using the list as a parameter for its constructor.
* Compare the values of the treeSet and the List using [Assert methods](http://chercher.tech/java/testng-assertions-selenium-webdriver).

|  |
| --- |
| WebElement element = driver.findElement(By.xpath("//select[@id='animals']"));  Select se = new Select(element);  List<String> originalList = new ArrayList();  for (WebElement e : se.getOptions()) {  originalList.add(e.getText());  }  Set<String> treeset = new TreeSet(originalList);  Assert.assertEquals(treeset, originalList); |

**stream().collect()**

We can use the stream() method along with the collect() method to create new List object with elements

|  |
| --- |
| WebElement element = driver.findElement(By.xpath("//select[@id='animals']"));  Select se = new Select(element);  List<String> originalList = new ArrayList();  for (WebElement e : se.getOptions()) {  originalList.add(e.getText());  }  List<String> tempList = originalList.stream().collect(Collectors.toList());  Collections.sort(tempList);  Assert.assertEquals(tempList, originalList); |

###### Compare values of two drop downs

**Scenario 1: dropdowns which have the same values but in same order.**

* Find the dropdown Element using [FindElement method in selenium](http://chercher.tech/java/locators-find-elements-selenium-webdriver" \l "element" \t "_blank)
* Create an object for Select class
* We have to get all the values present in the dropdown to compare the values. So use getOptions() methods from Select class object
* getOptions() method will return the list of options/values present in the dropdown. The order of the dropdown purely depends on the developer of that dropdown
* Store the list of values using [ArrayList](http://chercher.tech/java/arraylist" \t "_blank) in a variable originalListElements
* Iterate the option that you have received and get the text from all the options and store in a List originalList
* Now perform all the above steps to the target dropdown to which you want to compare the values. Store the list of text values in targetList
* Compare the originalList and targetList using [assertions present in TestNG](http://chercher.tech/java/testng-assertions-selenium-webdriver)

|  |
| --- |
| // original dropdown  WebElement originalDropdown = driver.findElement(By.cssSelector("select#first"));  Select original = new Select(originalDropdown);  List<WebElement> originalListElements = original.getOptions();  List<String> originalList = new ArrayList<String>();  for (WebElement webElement : originalListElements) {  originalList.add(webElement.getText());  }  // target dropdown  WebElement targetDropdown = driver.findElement(By.id("order-same"));  Select target = new Select(targetDropdown);  List<WebElement> targetListElements = target.getOptions();  List<String> targetList = new ArrayList<String>();  for (WebElement webElement : targetListElements) {  targetList.add(webElement.getText());  }  Assert.assertEquals(originalList, targetList); |

**Scenario 2: dropdowns which have the same values but in different/random order**.

* Follow above steps and get originalList & targetList
* Now let's compare and see if both dropdowns have values in the same order.
* First of all, verify both dropdowns have the same number of option, if not same fail it.
* Compare both the list by iterating a list, as both have the same size so that you can pick any list for iteration.
* Remove the elements from the target list if you are iterating original, vice versa.
* After the iteration is over, you will have targetList as empty because you have removed all the matching value if targetList has any value then that value did not match with originalList

|  |
| --- |
| Assert.assertEquals(originalList.size(), targetList.size());  for (String string : originalList) {  targetList.remove(string);  }  Assert.assertTrue(targetList.isEmpty()); |

**Custom dropdowns use click**

[The Custom dropdowns](http://chercher.tech/java/dropdown-select-class-in-selenium-webdriver#custom) are built using different technologies like bootstrap, flask, django so on, but what we receive at our browser is none other than HTML code

Dropdowns are formed using td, button, span, option, submenu tags, so in those cases, we cannot use our Select class from the selenium webdriver.

If you try to use Select class on this custom dropdown, selenium webdriver throws UnexpectedTagNameException

To overcome the [custom dropdown](http://chercher.tech/java/dropdown-select-class-in-selenium-webdriver#custom) issue, we have to combine selenium methods to select a value from dropdowns which are not formed by using select class.

* Click the Dropdown
* Click the option or the value

|  |
| --- |
| // find the dropdown using xpath  WebElement dropdownElement = driver.findElement(By.xpath("//button[@id='custom']"));  // click the dropdown  dropdownElement.click();  // find and click the dropdown element  driver.findElement(By.xpath("//b[contains(text(),'submenu')]")).click(); |

**Actions Class to select an option from the dropdown**

Actions class can be used to select a dropdown value in selenium webdriver, we have to click the dropdown first and then option from the dropdown

|  |
| --- |
| Actions act = new Actions(driver);  // click the dropdown  act.click(driver.findElement(By.xpath("//select[@id='first']")))  // click the option  .click(driver.findElement(By.xpath("//option[text()='Bing']")))  // perform the operation  .build().perform(); |

**Select dropdown option with Sendkeys**

We can use [sendkeys](http://chercher.tech/java/sendkeys-click-clear-submit-selenium-webdriver" \o "sendkeys)method to select an option from a dropdown, first of all, we have to find the dropdown element and use sendkeys method to select the option.  
  
We have to pass the exact text present in the dropdown option, if we do not send exact text, selenium chooses the almost exact match option which is in the top.

For example: If you want to choose option "22" but if you are sending "2" in [sendkeys](http://chercher.tech/java/sendkeys-click-clear-submit-selenium-webdriver" \o "sendkeys), selenium tries to set "2" only but if there is no option "2" but if there is "21" which is topper than option "22", selenium selects "21".

|  |
| --- |
| driver.findElement(By.xpath("//select[@id='first']")).sendKeys("Bing"); |

**JavaScript method to select the dropdown option**

We can use the [JavascriptExecutor](http://chercher.tech/java/javascript-executor-selenium-webdriver" \t "_blank" \o "javascript-executor-selenium-webdriver) method to select an option from the dropdown with the help of JavaScriptExecutor. We have to set the dropdown option using the "value" property of the element.

|  |
| --- |
| // find the dropdown using xpath  WebElement dropdownElement = driver.findElement(By.xpath("//btn]"));  //cast driver object to JavaScriptExecutor  JavascriptExecutor jse = (JavascriptExecutor) driver;  // set the dropdown value t0 'Bing' using javascript  jse.executeScript("arguments[0].value='Bing'", dropdownElement); |

### Interacting with Web table

A table is made of rows and columns. When we create a table for a web page, that is called as a web table. In HTML, table is created using **<table>** tag. Web table is a HTML structure for creating rows and columns on a Web page.

A web table can consists below parts:

Header(s): It is created using <th> tag.

Row(s):It is created using <tr> tag.

Columns(s):It is created using <td> tag.

We can categorized web tables in two parts:

1. **Static web table:**  we know the number of rows and columns in advance and it does not change Eg. Table of months, Table of days etc.
2. **Dynamic table:** Number of rows and columns will be dynamic. It will be keep on increasing or decreasing based on data. For Eg: Sales table, Student table.

#### Handling Static Web Table:

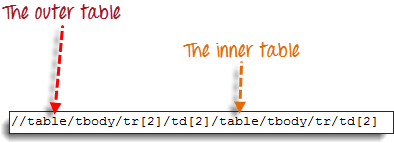
It is very seldom for a web designer to provide an id or name attribute to a certain cell in the table. Therefore, we cannot use the usual methods such as "By.id()", "By.name()", or "By.cssSelector()". In this case, the most reliable option is to access them using the "By.xpath()"

Xpath to access text in 2 row 3 column

|  |
| --- |
| String innerText = driver.findElement(By.xpath("//table/tbody/tr[2]/td[3]")).getText(); |

**Accessing Nested Web tables:**

Nested tables are tables located within another table



**Print All Headers of Web table**

Headers of web able is also a row. Header is created using <th> with <tr>. Generally first row of table is header. We can retrieve header name and print them using below code:

|  |
| --- |
| List allHeadersOfTable= driver.findElements(By.xpath("//table[@name='BookTable']/tbody/tr/th"));  System.out.println("Total headers found: "+allHeadersOfTable.size());  for(WebElement header:allHeadersOfTable)  System.out.println(header.getText()); |

**Print Number of rows in table**

Since total number of rows includes header row, so we need to exclude header row from total number of rows to get actual number of data row

|  |
| --- |
| List allRows= driver.findElements(By.xpath("//table[@name='BookTable']/tbody/tr"));  System.out.println("Total data rows found in table:"+ (allRows.size()-1)); |

**Print Number of columns in table**

|  |
| --- |
| List allColumns = driver.findElements(By.xpath("//table[@name='BookTable']/tbody/tr/td"));  System.out.println("Number of columns :"+allColumns.size()); |

#### Handling Dynamic Web Table

When the table is dynamic in nature, we cannot predict its number of rows and columns.

**To Fetch number of rows and columns from Dynamic WebTable**

|  |
| --- |
| //No.of Columns  List col = driver.findElements(By.xpath(".//\*[@id=\"leftcontainer\"]/table/thead/tr/th"));  //No.of rows  List rows = driver.findElements(By.xpath(".//\*[@id='leftcontainer']/table/tbody/tr/td[1]")); |
| //To locate table.  WebElement mytable = driver.findElement(By.xpath(".//\*[@id=xxx']/table/tbody"));  //To locate rows of table.  List<WebElement> rows\_table = mytable.findElements(By.tagName("tr"));  //To calculate no of rows In table.  int rows\_count = rows\_table.size();    //Loop will execute till the last row of table.  for (int row=0; row<rows\_count; row++){  //To locate columns(cells) of that specific row.  List<WebElement> Columns\_row = rows\_table.get(row).findElements(By.tagName("td"));  //To calculate no of columns(cells) In that specific row.  int columns\_count = Columns\_row.size(); |

|  |  |  |
| --- | --- | --- |
| **Automation Tool** | **Licensing** | **Market response** |
| Selenium | Free | In |
| QTP | Paid | Out |

For example all you know is the Text ‘Licensing’ for QTP in the above example

|  |
| --- |
| String sColValue = "Licensing";  String sRowValue = “QTP”  int colNum, rowNum;  for (int i=1;i<=columns\_count;i++){  String sValue = null;  sValue = driver.findElement(By.xpath(".//\*[@id='post-2924']/div/table/tbody/tr[1]/th["+i+"]")).getText();  if(sValue.equalsIgnoreCase(sColValue)){  colNum = i;    }  for (int j=2;j<=rows\_count;j++){  String sValue2= driver.findElement(By.xpath(".//\*[@id='post-2924']/div/table/tbody/tr["+j+"]/td[1]")).getText();  if(sValue2.equalsIgnoreCase(sRowValue)){  rowNum=j;  }  }  String sCellValue = driver.findElement(By.xpath(".//\*[@id='content']/table/tbody/tr["+rowNum+"]/td["+colNum+"]")).getText();  System.out.println(sCellValue); |

### Interacting with Date & Time Picker

# ****Alert, Frames, Windows & Mouse Actions****

### Handling Keyboard & Mouse Events

In order to perform any operations on the web application like double click, right click,  drag and drop or clicking on multiple elements with the help of the control key, actions class is a necessity. Handling special keyboard and mouse events are done using the Advanced User Interactions API. It contains the Actions and the Action classes that are needed when executing these events.

#### Actions and Action in Selenium:

**Actions** class is based on builder design pattern which builds a composite actions with the aggregation of Selenium WebDriver, where webdriver is only used to identify the presence of web elements on web application

 **Action** interface is only used to represent the single user interaction i.e to perform the series of action items build by Actions class.

To handle these actions class in Selenium, you must follow a proper format

* Create an object of the Actions class ‘action‘
* Focus on the element using [WebDriver](https://www.edureka.co/blog/selenium-webdriver-tutorial): action.moveToElement(element).build().perform();
* build().perform() is used to compile and execute the actions class.
* Use the different methods under the actions class to perform various operations like click(), drag and drop and so on.

|  |
| --- |
| Actions actions = new Actions(driver);  //Series of actions  actions.keyDown(element, Keys.SHIFT).sendKeys(“Text\_In\_UpperCase”).keyUp(Keys.SHIFT);  //build actions class object  Action action = actions.build();  //build method always returns “Action type object” so we need to create reference of Action Interface and hold all builder's Actions.  //perform the actions sequence using perform() method of Action Interface.  action.perform(); |

(Or)

In short we can use

|  |
| --- |
| Actions actions = new Actions(driver);  actions.keyDown(element,   Keys.SHIFT).sendKeys(“Text\_In\_UpperCase”).keyUp(Keys.SHIFT).build().perform(); |

#### Actions Class Methods

There are basically two methods which help in working with the actions in [Selenium](https://www.edureka.co/blog/selenium-webdriver-tutorial)*,*namely:

* Keyboard interface
* Mouse interface

##### ****Keyboard Events Using Actions Class****

**sendKeys(CharSequence keys):**  This method sends keys to an active element. There is already a *sendKeys()* method covered and used in various sample codes is *WebElement.sendKeys(CharSequence)* used to type text. This method also performs the same action to type text in the element. However, there are two ways in which it differs:

1. The release of modifier keys included in this call does not happen. In other words, any modifier key like Shift, Ctrl, etc. which gets pressed in this method, does not get released after sending keys. We have to release them using the *keyUp()* method explicitly.
2. There is no attempt to re-focus the element. So *sendKeys(Keys.TAB)* for switching elements should work.

**sendKeys(WebElementTarget, CharSequence keys):**This method first focuses on the target web element and then performs the same action as sendKeys(CharSequence keys). This is equivalent to calling: Actions.click(element).sendKeys(keysToSend)

|  |
| --- |
| Actions builder = new Actions(driver);  WebElement element = driver.findElement(By.id("search-inp"));  builder.moveToElement(act).build().perform();  builder.sendKeys(element, "Selenium") |

**keyDown(*CharSequence key*):** Sends a key press without release it. Subsequent actions may assume it as pressed.

Example: Keys.ALT, Keys.SHIFT, or Keys.CONTROL);

***keyDown(WebElement target, CharSequence key):***

This method first focuses on the target element and then performs the same the action as keyDown(java.lang.CharSequence key) So, this method performs actions equivalent to: Actions.click(element).keyDown(theKey);

|  |
| --- |
| builder.keyDown(search, Keys.SHIFT) |

**keyUp(*CharSequence key*)**: Performs a key release. Here, the assumption is that modifier keys passed in the method are in pressed state. So, if keyUp() invokes without having modifier key pressed, then the behavior is unpredictable.

(Or) sendKeys(Keys.NULL) is used to release the modifier key

***keyUp(WebElement target, CharSequence key):***This method first focuses on the target web element and then performs the same action as keyUp(java.lang.CharSequence key). So, actions performed are equivalent to Actions.click(element).keyUp(theKey);

|  |
| --- |
| builder.keyUp(search, Keys.SHIFT) |

##### ****Mouse Events Using Actions Class****

**click():** Clicks on the element.

|  |
| --- |
| Actions builder = new Actions(driver);  WebElement link = driver.findElement(By.cssSelector("#sdds"));  builder.moveToElement(link).click().build().perform(); |

**doubleClick ():** Double clicks on the element. Double-click is a frequently used user action. The most common use of double click happens in File Explorer E.g. In File Explorer, any Folder or File in a folder can be opened by double-clicking on it.

Similarly, on any webpage, there might be some elements that require a double click to invoke an action on them

|  |
| --- |
| Actions actions = new Actions(driver);  actions.doubleClick(driver.findElement(By.id("doubleClickBtn"))).perform(); |

**contextClick()** : Performs a context-click (right-click) on the element. when a user tries to click the***Right mouse button*** on a website or a web element to look at its context menu.

|  |
| --- |
| //Instantiate Action Class  Actions actions = new Actions(driver);    //Retrieve WebElement to perform right click  WebElement btnElement = driver.findElement(By.id("rightClickBtn"));    //Right Click the button to display Context Menu&nbsp;  actions.contextClick(btnElement).perform(); |

**clickAndHold()**: Clicks at the present mouse location without releasing.

**release():** Releases the pressed left mouse button at the existing mouse location.

|  |
| --- |
| Actions builder = new Actions(driver);  builder.clickAndHold(someElement)  .moveToElement(otherElement)  .release(otherElement)  .build()  .perform(); |

**Drag and Drop:** This is an action performed with a ***mouse*** when a user moves ***(drags)*** a web element and then places **(drops)** it into an alternate area.

Here are the methods Actions class provides for Drag-Drop action:

1. dragAndDrop(WebElementsource, WebElement target)
2. dragAndDropBy(WebElementsource, int xOffset, int yOffset)

***dragAndDrop(WebElement source, WebElement target):*** This method performs left click, hold the click to hold the source element, moves to the location of the target element and then releases the mouse click.

|  |
| --- |
| Actions builder = new Actions(driver);  WebElement from = driver.findElement(By.id("draggable"));  WebElement to = driver.findElement(By.id("droppable"));  //Perform drag and drop  builder.dragAndDrop(from, to).perform(); |

***dragAndDropBy(WebElement source, int xOffset, int yOffset):*** This method clicks & holds the source element and moves by a given offset, then releases the mouse. Offsets are defined by x & y.

* xOffset is horizontal movement
* yOffset is a vertical movement

Two parameter values i.e. xOffset and yOffset are values in pixel.

For Example, if a xOffset value is set as 50, it means an object needs to be dragged and dropped by 50 pixels offset horizontal direction. Similarly, if a yOffset value is set as 50, it means an object needs to be dragged and dropped by 50 pixels offset vertical direction.

|  |
| --- |
| Actions builder = new Actions(driver);  WebElement from = driver.findElement(By.id("draggable"));  WebElement to = driver.findElement(By.id("droppable"));  //Here, getting x and y offset to drop source object on target object location  //First, get x and y offset for from object  int xOffset1 = from.getLocation().getX();  int yOffset1 = from.getLocation().getY();  //Secondly, get x and y offset for to object  int xOffset = to.getLocation().getX();  int yOffset = to.getLocation().getY();  //Find the xOffset and yOffset difference to find x and y offset needed in which from object required to dragged and dropped  xOffset =(xOffset-xOffset1)+10;  yOffset=(yOffset-yOffset1)+20;  //Perform dragAndDropBy  builder.dragAndDropBy(from, xOffset,yOffset).perform(); |

##### Mouse Hover Action

A mouse hover is also called as hover. Mouse hover action is basically an action where a user places a mouse over a designated area like a hyperlink. It can cause some event to get triggered.moving the mouse over an element on web page displays some pop-up windows or maybe description boxes.

 Here are the methods Actions class has provided for Mouse Hover action:

1. moveToElement(WebElement target)
2. moveToElement(WebElement target, int xOffset, int yOffset)

**moveToElement(WebElement target):** Moves the mouse to the middle of the element.

|  |
| --- |
| Actions actions = new Actions(driver);  WebElement menuOption = driver.findElement(By.xpath(".//div/hjhj)]"));  //Mouse hover menuOption 'Music'  actions.moveToElement(menuOption).perform(); |

##### Interacting With Slider

**moveToElement(WebElement target, int xOffset, int yOffset): Moves the mouse to an offset from the top-left corner of the element*.***

In the earlier method which we saw, the mouse gets moved to the middle of the element. Usually, we need to move the mouse on any element on a web page to display tooltip or maybe to display submenu as seen in the above example. In such common situations, it doesn’t matter even if the mouse is moved to the middle of the element by default as long as it displays desired element like tooltip or submenu, etc.

There are some cases where to perform some actions on an element, it is mandatory to move the mouse to a specific location on the element only

Let’s consider ***Slider*** which is used to select some range. This can be selected by moving the slider thumb or clicking on a specific point on the slider element. To perform this action through automation script, we can take help of this particular method.

|  |
| --- |
| Actions actions = new Actions(driver);  WebElement slider = driver.findElement(By.id("slider"));  actions.moveToElement(slider,50,0).perform();  slider.click(); |

##### Interacting With Tool tip

In many web pages, it’s very common that when hovered over some link, text, some text or sometimes image gets displayed. This is called a tooltip of web element. The purpose of a tooltip is to provide some hint to the user about the object. At many instances, it is required to verify if this text description being displayed as expected. For this above reasons, it is required to retrieve the text inside the tooltip and verify the text.

This can be done in different ways depending upon how tooltip is getting inserted in the HTML. Let’s consider the following cases:

* Case One: When the tooltip is available in the ‘title’ attribute. Here, we can retrieve tooltip from By strategy
* Case Two: When the tooltip is available in ‘div’. Here, we can retrieve tooltip using Actions class method

**Tooltip is available in the ‘title’ attribute**

ToolTip gets displayed when hovered over an HTML object. This can also be seen from the Developer tool of the browser, tooltip text is seen as a value set in ‘title’ attribute.

In most cases, it is pretty straight forward to get the tooltip text. All you need to do it to locate the web-element for which tooltip needs to be retrieved.

|  |
| --- |
| WebElement ageTextBox = driver.findElement(By.id("age"));  //Get title attribute value  String tooltipText = ageTextBox.getAttribute("title"); |

**Tooltip is available in ‘div’**

This time we try to mimic the same behavior as any other user does on the website. Hover over the object and try to read the description of the tooltip. And this can only be done with the help of Actions Class in Selenium.

|  |
| --- |
| Actions actions = new Actions(driver);  WebElement element = driver.findElement(By.id("tooltipDemo"));  // Use action class to mouse hover  actions.moveToElement(element).perform();  WebElement toolTip = driver.findElement(By.cssSelector(".tooltiptext"));  // To get the tool tip text and assert  String toolTipText = toolTip.getText(); |

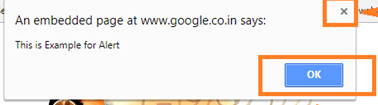
### Handling Alerts

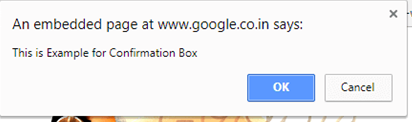
Alerts on the webpage used to get the attention of the user to perform some operation on the alert or the webpage; sometimes, alert expects input from the user. All the alerts are formed using javascript inside the HTML page;

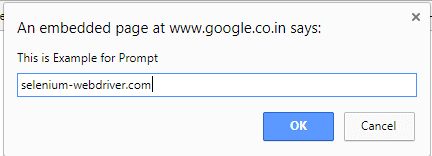
**Types of Alerts**

There are three kinds of alerts in HTML.

* Alert
* Confirmation dialog
* Prompt

**Alert: **

**Confirmation box: **

**Prompt: **

Alert takes the focus away from the current window and forces the browser to read the message. Alert prevents the user from accessing other parts of the web page until the box is closed.

Properties of Alerts with respect to Selenium :

* We cannot identify alerts using inspect tools
* We cannot write [xpaths](http://chercher.tech/java/relative-xpath-selenium-webdriver" \t "_blank) for alerts
* It is not a Window
* We cannot handle alerts using [javaScript Executor](http://chercher.tech/java/javascript-executor-selenium-webdriver" \t "_blank)

We can consider as not alert if any of the above property mismatches.

When an alert is present on the webpage, we cannot proceed further without handling the popup, and if we try to perform any operation throws [UnhandledAlertException](http://chercher.tech/java/exceptions-selenium-webdriver" \t "_blank" \o "exceptions-selenium-webdriver).

**Alert Class Methods:**  
  
We can handle alerts using **switchTo().alert()** method present in [selenium](http://chercher.tech/java/index-selenium-webdriver), with the help of this alert() we can handle the popup.

|  |
| --- |
| Alert ale = driver.switchTo().alert(); |

We can perform below operations on the pop-up.

1. **Accept** the popUp by clicking OK button

|  |
| --- |
| ale.accept(); |

1. **Dismiss** the popUp by clicking the 'X' icon

|  |
| --- |
| ale.dismiss(); |

1. Get popUp text by **getText** method

|  |
| --- |
| ale.getText(); |

1. Send a text to popUp by **sendKeys** (Applicable for Prompt only)

|  |
| --- |
| ale.sendKeys("test Text"); |

We can use [Explicit wait / WebdriverWait](https://chercher.tech/java/explicit-wait-selenium) to check whether the alert is there or not, alertIsPresent() method wait for the alert to be present till the given timeout, once it reaches a timeout, and if the alert is not present, then it throws [TimeOutException](http://chercher.tech/java/exceptions-selenium-webdriver" \t "_blank" \o "exceptions-selenium-webdriver) if the alert is present before the timeout it proceeds with remaining code.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 2 /\*timeout in seconds\*/);  try {  wait.until(ExpectedConditions.alertIsPresent());  presenceOfAlert = true;  } catch (TimeoutException e) {  presenceOfAlert = false;  } |

**Screenshot of an Alert:**

[Taking a screenshot](http://chercher.tech/java/take-screenshot-selenium-webdriver) without handling alert is not possible, but you can take a screenshot using the robot class or any other external class present in java.  
  
What you cannot do is, you cannot use the driver instance for any operation without handling the alert.  
  
Actually, the below code copies the screenshot to the clipboard.

|  |
| --- |
| // Press the key combination of (Windows + PrintScreen) by using Robot Class  Robot rb = new Robot();  rb.keyPress(KeyEvent.VK\_WINDOWS);  rb.keyPress(KeyEvent.VK\_PRINTSCREEN);  rb.keyRelease(KeyEvent.VK\_PRINTSCREEN);  rb.keyRelease(KeyEvent.VK\_WINDOWS); |

So using the [screenshot method in selenium](http://chercher.tech/java/take-screenshot-selenium-webdriver), we cannot capture the screenshot of javascript alert

### Handling Windows

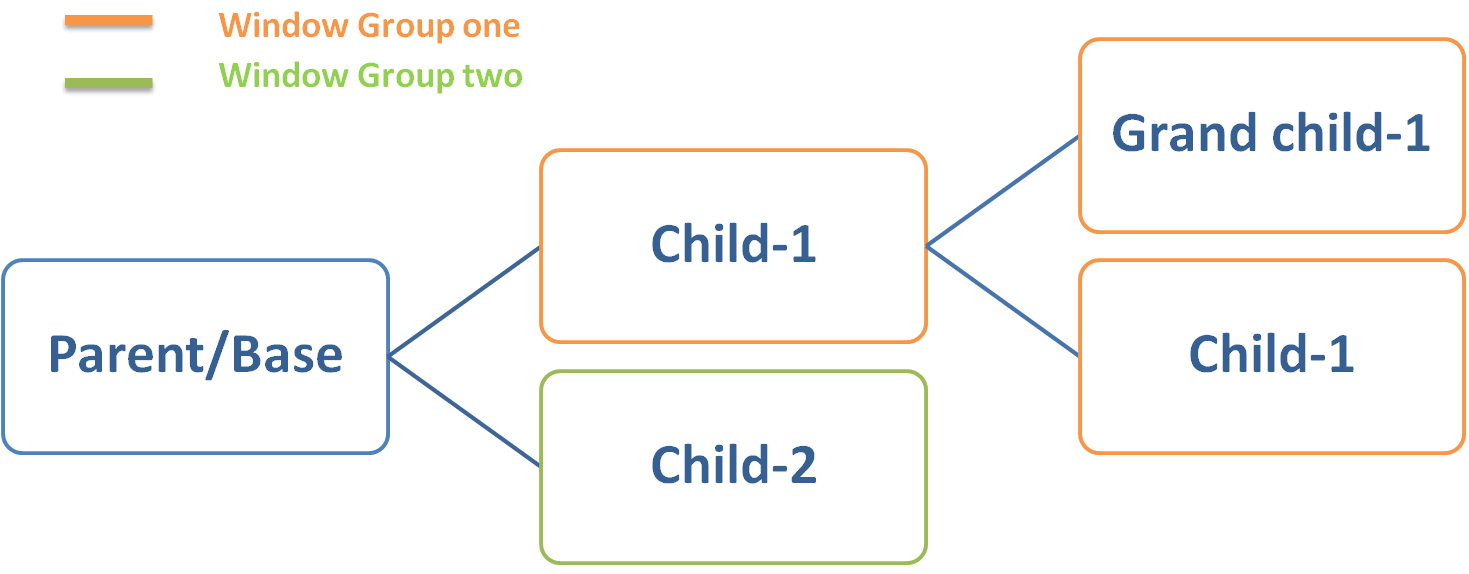
There will be times when you will require to perform some testing, where the testing operations open a new browser/tab; a test case may required you to perform some tasks on the newly opened browser window/tab and return back to the original window to perform the remaining tasks.  
  
Even if the window/tab is currently on focus, but still, it is not an active window, so to perform some tasks, you need to switch to a new browser window/tab in webdriver.

Commands will have an effect on the base window unless we switch our control to a new window/tab.  
  
Previously tab and Windows are two different items, as tabs don't have any GU iD. Now developers made tab and windows have GUIDs, so tabs and windows are treated as same in selenium.

**Multiple Windows**

Situations when we are likely to deal with multiple windows:

* Filling forms may require selecting the date from a separately opened window
* Clicking on some link/button can open another window
* Handling Advertisement windows



Above image depicts multiple browser windows

* Our Application is opened on Parent/Base window, the total number of windows is one
* When clicking on a link/button our application opens two new windows along with base window, now the total number of windows present is Three (base + child-1 + child-2)
* Now move to Child-1 window and perform [click](https://chercher.tech/java/sendkeys-click-clear-submit-selenium-webdriver#click)on a button/link which opens another one more window along with child-window, now the total number of windows is 4 (base + child-1 + child-2 + grand child-1)
* We can perform an operation on a grandchild -1 window, and we can close all windows or specific windows.

If we perform the above operation manually, we may not find difficulty, but handling the above scenario is tricky in selenium.

**Methods to Handle Multiple Windows:**

**getWindowHandle()** :  
getWindowHandle method in webdriver returns the current(active) browser's GU ID. It return GU ID as string value

GU ID abbreviation of Globally Unique Identifier, Every OS generates GU ID for application to identifying them uniquely. We will be using this GU ID to handle the multiple browsers; GU ID is a numeric string value.

|  |
| --- |
| driver.getWindowHandle(); |

**getWindowHandles()** :  
getWindowHandles method in selenium returns GU ID of all the browsers present at the moment, which are opened by the current driver. This method returns GU IDs as Set of String.

|  |
| --- |
| Set<String> s1=driver.getWindowHandles(); |

**switchTo().window()** : switchTo() method in webdriver helps user to switch between windows, frames, elements, alerts. switchTo().window(GU ID) method switches the control from the current browser window to the target browser window, which has the specified "GU ID"

|  |
| --- |
| driver.switchTo().window(guid); |

**Opening windows and performing actions:**

Below code which iteratively moves across all the open windows and navigates to a particular page in all the open windows one by one.

|  |
| --- |
| String parentWindowHandle = driver.getWindowHandle();  WebElement clickElement = driver.findElement(By.id("button1"));  for(int i = 0; i < 3; i++)  {  clickElement.click();  Thread.sleep(3000);  }  Set<String> allWindowHandles = driver.getWindowHandles();  for(String handle : allWindowHandles)  {  driver.switchTo().window(handle); //Switch to the desired window first and then execute commands using driver  driver.get("http://google.com");  }} |

**Force to open in new Window rather than Tab**

There will be situations where we have to open a link in a new window, but later browser does not support opening a URL in a new window as it is not recommended ( phishing is easy with new windows ).  
  
During such scenarios, we can use action class to open the link in a new window by pressing the Shift key, and to click the link; it opens the link in a new window in the webdriver.

|  |
| --- |
| String parentGUID = driver.getWindowHandle();  // store the element  WebElement ele = driver.findElement(By.id("force-new-window"));  // create object for Actions class  Actions act = new Actions(driver);  // press the shift key  act.keyDown(Keys.SHIFT)  //click the element  .click(ele)  // combine the actions  .build()  // perform the operations  .perform(); |

### Handling Frames

We can handle frames/iframes present on the webpage using the driver.switchTo() command in webdriver. Frame/iFrame is nothing but another [webelement in the HTML page](https://chercher.tech/java/find-elements-in-selenium" \t "_blank), which displays another part of the webpage.  
  
If you take a look at the DOM structure of a page that contains an iframe, you will never find the inner content of the iframe. And you won't be able to interact with it via the DOM. We have to switch into the frame to see the elements present in the frame.

Both Frame and iFrame are treated similar manner with Selenium; Selenium does not differentiate them so that we can handle both of them in the same way.  
  
A Frame is an HTML tag and used to divide the same web page or same domain into various frames/windows. Used as <frame> tag, it specifies each frame within a frameset tag. Frames are used along with frameset; frameset contains the multiple frames.

Iframe as <iframe> is also a tag used in HTML, but it specifies an in-line frame, which means it is used to embed not only the same domain but also other webpages within the current HTML document. It is like a Television; it displays a thing which is present somewhere else.

Frame requires a frameset, but iframes do not requires framesets.

Placement of Frames is a bit difficult compared with iFrames

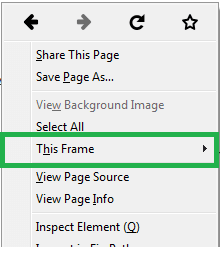
Adjusting the size and width of the Frame is difficult when compared with iFrame.

Frames cannot contain nested frames (frame inside the frame), but iFrames can contain nested iFrames.

HTML5 doesn't support Frames but supports iFrames.

Developers try to avoid using Iframes as these are external documents (webpages), so there is a chance of Phishing.

It is essential for the tester to check whether an element is inside an iframe/frame or just on the webpage. If the element is inside a frame, then we have to switch into the frame to access the element.  
  
1. Right-click on the page (not on the element) which part you want to check.  
  
2. On the right-click options, you can find This Frame Option; if this option is present then there is an iframe else there is no iframe



**Switch to Frame**

**Switch to Frame Using ID**

<iframe id="ifr" name="demo" src="demo.html" height="200" width="300"></iframe>

We can find the frame using Id attribute present in the iframe/frame

|  |
| --- |
| driver.switchTo().frame("ifr"); |

**Switch to Frame Using Name**

We can find the frame using name attribute present in the iframe/frame

<iframe id="ifr" name="demo" src="demo.html" height="200" width="300"></iframe>

|  |
| --- |
| driver.switchTo().frame(“demo”); |

***Switch to Frame* Using Element**

Most of the time, there will be multiple iframe/frame, and few of them may share the same id and name, so in such scenarios; we cannot use [id or name](https://chercher.tech/java/locators-find-elements-selenium-webdriver#id) for finding the frame. To find the iframe/frame uniquely, we have to find it as an element, the way we follow for [normal elements](https://chercher.tech/java/locators-find-elements-selenium-webdriver).  
  
Here we can use locators ( except id, name, as we used already) like classname, XPath, CSS. We cannot use link text and [partial link text](https://chercher.tech/java/locators-find-elements-selenium-webdriver#partial) as these two are only applicable for anchor tag <a>.We also should refrain using a tagname[locator](https://chercher.tech/java/locators-in-selenium" \t "_blank" \o "locators-in-selenium)as there are multiple frames, so we may not get the unique element.

<iframe id="ifr" name="demo" src="demo.html" height="200" width="300"></iframe>

<iframe id="ifr" name="demo" class='second' src="width.html" height="200" width="300"></iframe>

<iframe id="ifr" name="demo" src="width.html" height="200" width="300"></iframe>

|  |
| --- |
| // switch to 1st frame  driver.switchTo().frame("//iframe[@src='demo.html']") |

***Switch to Frame* Using Index**

Selenium assigns an index to every frame present in the page, using index is the least preferred way to find the frame because in future frame position may change when development introduce another frame in between

<iframe id="ifr" name="demo" src="demo.html" height="200" width="300"></iframe>

<iframe id="ifr" name="demo" class='second' src="width.html" height="200" width="300"></iframe>

<iframe id="ifr" name="demo" src="width.html" height="200" width="300"></iframe>

|  |
| --- |
| // switch to 1st frame  driver.switchTo().frame(1) |

### Handling Nested iFrames

Sometimes we will have multiple and nested iframes on a webpage if we have nested iframes, we have to switch to iframe inside the frame.  
  
We can access only the content of the frame when we are inside a frame; we cannot access outside the frame or inside any other frame.

|  |
| --- |
| //find the frame1 and store it in webelement  WebElement frame1 = driver.findElement(By.id("frame1"));  // switch to frame1  driver.switchTo().frame(frame1);  // find the frame 3  WebElement frame3 = driver.findElement(By.xpath("//iframe[@id='frame3']"));  // switch to frame 3  driver.switchTo().frame(frame3); |

**Switch to Parent Frame** : switchTo().parentFrame() method in selenium switches the control to outer position(one place) in the web page, the outer position could be a frame or page level.

Selenium provides driver.switchTo().parentFrame() method to move out of the current frame, once we move out of the frame, we can access the elements outside that frame, but we cannot access the elements inside that frame

|  |
| --- |
| driver.switchTo().parentFrame(); |

**Default Content** : switchTo().defaultContent() method exits all the iframes and the places the selenium control at the page level where as parentFrame() method exits the current iframe.  
  
Once we reach the page level, we cannot access any elements inside the iframe unless we switch to it.

Scenario

1. Switch to frame 1 and switch to frame 3  
2. Check the checkbox  
3. Now exit from all the frames using switchTo().defaultContent()

|  |
| --- |
| // navigate to page level  driver.switchTo().defaultContent(); |

### Handling Authentication Pop up

When we open password-protected pages, we tend to get Authentication pop up. Authentication pop up will have username and password fields, the UI look of the pop up may vary browser to browser

Properties of the Authentication Pop up :

* Pop up displayed on Page load
* We can move the pop up (except in chrome)
* We cannot inspect the pop up with [browser inspection tools](http://chercher.tech/java/developer-tool-chrome-firefox-selenium-webdriver) like (TryXpath or chrome dev tools)
* The look varies from browser to browser

The Solution to Authentication Pop Up : We have to pass the user name and password along with the URL to handle the authentication pop in [webdriver](http://chercher.tech/java/index-selenium-webdriver" \t "_blank" \o "index-selenium-webdriver).

|  |
| --- |
| driver.get(protocol://Usename:Password@URL); |

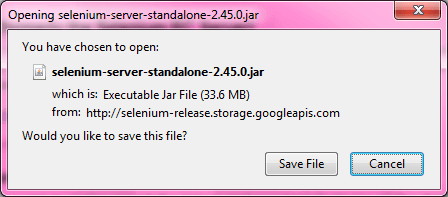
Limitations:

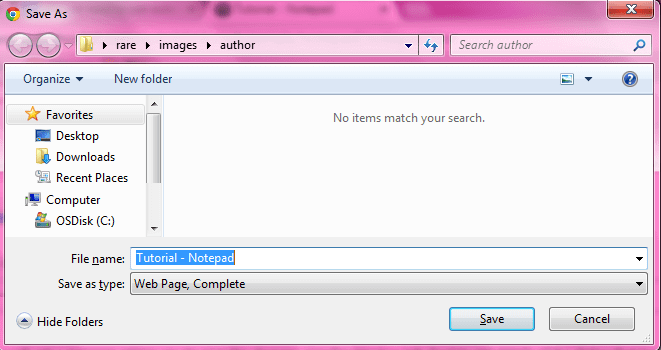
* Does not work for the https protocol
* It does not work when username or password contains special characters like : '@' , ':'

# Handling OS Pop ups

Web applications do not always confine themselves to working entirely on the web. Sometimes they need to interact with the desktop to do things like downloads & uploads. Automating these sorts of workflow is tricky in Selenium. Selenium is confined to automating browsers, so desktop windows are out of scope.  There are tools available for automating these sorts of workflow such as *AutoIt, Robot, Silk Test* etc.

 Some Window popups are

Download pop up 

Save as pop up 

File Upload pop up

### Auto IT

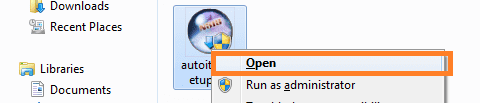
Auto It is an open-source tool used to automate the operating system based applications. It is basically used to simulate the keystrokes and mouse action, and also, we can create GUI. AutoIt has a built-in feature to compile the script and generate a .exe executable file. You do not need any extra add-on to do all these things.

**Auto IT Installation:**

* 1. Goto https://www.autoitscript.com/site/autoit/downloads/
  2. Click download AutoIt button

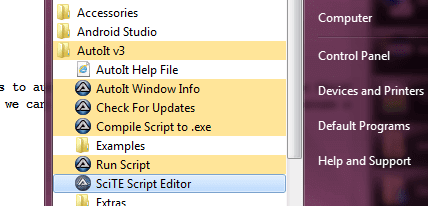


* 1. After downloading go to the downloaded folder and right-click on the AutoIt application and select Open and follow the download instructions. It may take 2 minutes to install

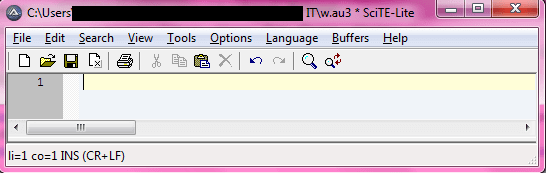


**Script using Auto IT**

1. Once the installation is complete , Goto Start ->All Programs->AutoIt v3->select sciTE Script Editor

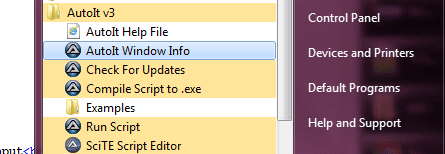


1. Script editor looks Like below one here we are going to write our script to perform an action on the window application



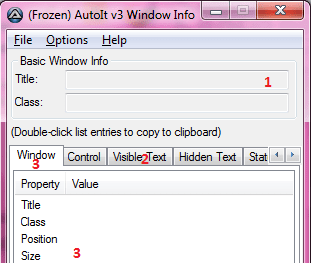
1. Before we perform any action with AutoIt, we need to find the object on which we are going to perform the action, like as in selenium.  
     
   So to inspect the element in AutoIt we are using AutoIt Window Info application, we do not need to download it separately as it comes with AutoIt application.

To open the AutoIt Window Info application, Goto Start->All Programs->AutoIt V3->AutoIt Window Info.

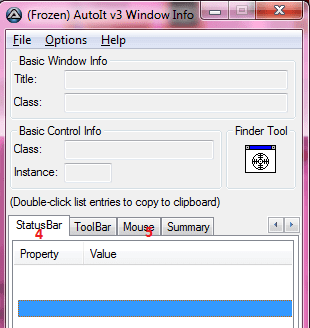


AutoIt v3 comes with a standalone tool called the [AutoIt Window Info Tool](https://chercher.tech/java/autoit-selenium" \l "info" \t "_blank" \o "AutoIt Window Info Tool) (Program FilesAutoIt3Au3Info.exe). Au3Info allows you to get information from a specified window that can be used to automate it effectively. Information that can be obtained includes:

1. Window titles
2. Text on the window (visible and hidden)
3. Window size and position



1. *Contents of the status bar*
2. *The Position of the mouse pointer. The Color of the pixels underneath the mouse pointer. Details of the Control underneath the mouse pointer*



### Robot Class

### Sikuli

# ****Page Loads and Time Outs****

Timeouts interface manages all the waits of the driver instances; This is the inner Interface of Webdriver Interface; in other words, the Timeouts interface is enclosed by the [Webdriver interface](https://chercher.tech/java/index-selenium-webdriver" \t "_blank" \o "index-selenium-webdriver).

Timeouts interface has three abstract methods, which are :

* implicitlyWait
* setScriptTimeout
* pageLoadTimeout

The browser classes(FirefoxDriver, ChromeDriver..) provides the implementations for these methods because the browser classes implement the Webdriver Interface.

### Page Load Time Out

Page load timeout in selenium requests/set the time limit for a page to load, If the page is not loaded within the given time frame selenium throws [TimeOutException](https://chercher.tech/java/exceptions-selenium-webdriver" \t "_blank" \o "exceptions-selenium-webdriver).  
  
We can set the page load timeout using the pageLoadTimeout method present in Browser classes(FirefoxDriver, ChromeDriver..)

Page Load timeout is applicable only to **[driver.get()](https://chercher.tech/java/open-webpage-selenium-webdriver" \t "_blank" \o "open-webpage-selenium-webdriver) and [driver.navigate().to()](https://chercher.tech/java/navigation-selenium-webdriver" \t "_blank" \o "navigation-selenium-webdriver)** methods in [selenium](https://chercher.tech/java/index-selenium-webdriver). Page load timeout is not applicable when the user clicks a link to open a page.

Setting Negative time limit makes the selenium to wait for the page load infinitely

|  |
| --- |
| driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS); |

### Set Script Load timeout

setScriptTimeout sets the time limit for the asynchronous script to finish execution, if the process is not completed before the time limit, [selenium](https://chercher.tech/java/index-selenium-webdriver) throws [TimeOutException](https://chercher.tech/java/exceptions-selenium-webdriver" \t "_blank" \o "exceptions-selenium-webdriver)

The setScriptTimeout method affects only [JavaScript code](https://chercher.tech/java/javascript-executor-selenium-webdriver) executed with **executeAsyncScript** and nothing else.

The default timeout for setScriptTimeout method is 0 (zero) if we do not set any time our executeAsyncScript method may fail because the [JavaScript code](https://chercher.tech/java/javascript-executor-selenium-webdriver) may take more than zero seconds. So to avoid unnecessary failures, we have to set the setScriptTimeout.

|  |
| --- |
| driver.manage().timeouts().setScriptTimeout(10, TimeUnit.SECONDS);  ((JavascriptExecutor) driver).executeAsyncScript("document.getElementById('dummy')"); |

### Implicit Wait

Nowadays, most of the applications are formed using ajax statements, ajax makes the DOM elements to load at different times but selenium tries to [find the element](https://chercher.tech/java/find-elements-in-selenium) after the moment, it completes the previous line execution.

Due following reasons application element may take a while to load.

* Poor Server Response time
* Ajax Loading
* Size of the page
* Internet Speed
* System hardware
* Browser slowness / less responsiveness
* Element created using javascript

[Selenium](https://chercher.tech/java/index-selenium-webdriver) tries to [find the element](https://chercher.tech/java/find-elements-in-selenium) without bothering about whether elements are loaded or not, and selenium throws [NoSuchElementException](https://chercher.tech/java/exceptions-selenium-webdriver" \t "_blank" \o "exceptions-selenium-webdriver) if the element is not present.  
Implicitly wait is one of the ways to request selenium not throw any [exception](https://chercher.tech/java/exceptions-selenium-webdriver) until provided time. The default wait time of the selenium is 500 milliseconds, implicitly wait overrides the default wait time of the selenium webdriver.

If the element is found before implicitly wait time, [selenium](https://chercher.tech/java/index-selenium-webdriver) moves to the next commands in the program without waiting to complete the implicitly wait time

The implicit wait is set for the entire duration of your webdriver and is set at the start of your program. Most of the automation tester writes the implicit wait after the creation of browser object.

Implicit wait tries to [find the element](https://chercher.tech/java/find-elements-in-selenium) in the first go if the element is not present implicit wait tries to [find the element](https://chercher.tech/java/find-elements-in-selenium) after 500ms of first polling, if the element is not available on the second time also then implicit wait tries the third time after 500 ms of the second try and it goes on till the time reaches the 30 seconds.

If the driver still does not find the element, then it throws [an exception](https://chercher.tech/java/exceptions-selenium-webdriver). Implicit wait does the same for all the elements in your program, so you just have to set it once.

|  |
| --- |
| WebDriver driver = new ChromeDriver();  // set implicit wait tme as 30 Seconds  driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS); |

Implicitly Wait is applicable only to **findElement** and **findElements** no other statement in selenium.

Implicitly wait in selenium webdriver supports all the time unit from nano-Seconds to Days, all the units are present in the TimeUnit abstract class.

* TimeUnit.NANOSECONDS : represents nano seconds (10pow-9 of a second is nanosecond)
* TimeUnit.MICROSECONDS: represents microseconds (10pow-6 of a second is microsecond)
* TimeUnit.MILLISECONDS : represents mill seconds (10pow-3 of a second is microsecond, i.e. 1000 milliseconds is a second)
* TimeUnit.SECONDS : represent a second
* TimeUnit.MINUTES : represent minutes, a minute is 60 seconds
* TimeUnit.HOURS : represent the hour, an hour is 60 minutes
* TimeUnit.DAYS : represents the day, a day is 24 hours

### Explicit Wait

The explicit wait is used to tell the Web Driver to wait for certain conditions or the maximum time limit before throwing an [Exception](https://chercher.tech/java/exceptions-selenium-webdriver).

We can reuse the WebdriverWait object once we create it. The explicit wait will be applicable for only one line (one condition); we have to use it with ExpectedConditions class.

ExplicitWait does not have any effect on [findElement and findElements](https://chercher.tech/java/locators-find-elements-selenium-webdriver" \l "element" \t "_blank). ExplicitWait also called WebdriverWait.  
  
WebDriverWait by default calls the ExpectedCondition every 500 milliseconds until it returns successfully.

###### Below are few Expected Conditions :

* alertIsPresent()
* elementSelectionStateToBe(locator, selected)
* elementToBeClickable(locator)
* elementToBeSelected(locator)
* frameToBeAvailableAndSwitchToIt(locator)
* presenceOfElementLocated(locator)
* textToBePresentInElement(locator, text)
* title()
* titleIs(title)
* visibilityOf(element)

**alertIsPresent()**

Selenium waits for an alert to be present when the user writes 'alertIsPresent()', when selenium finds the alert it moves to next line of code, in case if an alert is not present before the specified time, then selenium Throws TimoutException

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 30 /\*timeout in seconds\*/);  //throws TimeoutException if no alert is present  wait.until(ExpectedConditions.alertIsPresent());  driver.switchTo().alert().dismiss(); |

**elementToBeClickable()**

Selenium waits for an element to become clickable like disabled state to a normal state, selenium moves to next line of code if the element becomes clickable before the timeout otherwise selenium throws TimoutException

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 30 /\*timeout in seconds\*/);  //throws TimeoutException if element does not become as clickable in given time  wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//button[@id='btn1']")))); |

**elementToBeSelected()**

Selenium waits for an element to be selected when we use 'elementToBeSelected()', when selenium finds the element is selected it moves to next line of code, in case if the element is not selected before the specified time, then selenium Throws TimoutException

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 30 /\*timeout in seconds\*/);  //throws TimeoutException if element is not selected in given time  wait.until(ExpectedConditions.elementToBeSelected(By.xpath("//input[@id='hidden]")))); |

**textToBePresentInElement()**

Selenium waits for an element to have particular text when we use 'textToBePresentInElement()', when selenium finds the element has particular text it moves to next line of code, in case if an element does not have text before the specified time, then selenium Throws TimoutException

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 30 /\*timeout in seconds\*/);  //throws TimeoutException if some text is not present in he webpage  wait.until(ExpectedConditions.textToBePresentInElement(By.xpath("//input[@id='h2]")))); |

**titleIs()**

Selenium waits for a webpage to have a particular title when we use 'titleIs()', when selenium finds a webpage with a given title, it moves to next line of code, in case if the webpage does not have a title before the specified time, then selenium Throws TimoutException.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 30 /\*timeout in seconds\*/);  //throws TimeoutException if title does not contain give string  wait.until(ExpectedConditions.titleIs("selenium webdriver sample"))); |

**visibilityOfElementLocated()**

Selenium waits for visibility of element when we use 'visibilityOfElementLocated()', when the element is visible, it moves to next line of code, in case if the element is not visible before the specified time, then selenium Throws TimoutException

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 30 /\*timeout in seconds\*/);  //throws TimeoutException if the element is not displayed on the webpage  wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath("//input[@id='hidden]"))); |

**presenceOfElementLocated()**

Defines an expectation for checking that an element is present on the DOM of a page.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  WebElement element = wait.until(ExpectedConditions.presenceOfElementLocated(By.id(“FDFD”)); |

**presenceOfAllElementsLocatedBy()**

Defines an expectation for checking that there is at least one element present on a web page.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  List<WebElement> elements = wait.until(ExpectedConditions.presenceOfAllElementsLocatedBy(By.name(“d”)); |

**titleContains()**

An expectation for checking that the title contains a case-sensitive substring

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  assertTrue(wait.until(ExpectedConditions.titleContains(keyword))); |

**urlContains()**

Defines an expectation for the URL of the current page to contain specific text.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  wait.until(ExpectedConditions.urlContains(keyword)); |

**urlMatches()**

Expectation for the URL to match a specific regular expression

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  wait.until(ExpectedConditions.urlMatches(regularExpression)); |

**elementSelectionStateToBe()**

Defines an expectation for checking if the given element is selected.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  wait.until(ExpectedConditions.elementSelectionStateToBe(locator, true)); |

**invisibilityOfElementLocated()**

An expectation for checking that an element is either invisible or not present on the DOM.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  assertTrue(wait.until(ExpectedConditions.invisibilityOfElementLocated(locator))); |

**invisibilityOfElementWithText()**

Defines an expectation for checking that an element with text is either invisible or not present on the DOM.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  assertTrue(wait.until(ExpectedConditions.invisibilityOfElementWithText(locator, text))); |

**stalenessOf()**

Wait until an element is no longer attached to the DOM.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, 10);  assertTrue(wait.until(ExpectedConditions.stalenessOf(element))) |

**Complete List:**

ExpectedConditions.alertIsPresent()

ExpectedConditions.and(ExpectedCondition… conditions)

ExpectedConditions.attributeContains(org.openqa.selenium.By locator, String attribute, String value)

ExpectedConditions.attributeContains(org.openqa.selenium.WebElement element, String attribute, String value)

ExpectedConditions.attributeToBe(org.openqa.selenium.By locator, String attribute, String value)

ExpectedConditions.attributeToBe(org.openqa.selenium.WebElement element, String attribute, String value)

ExpectedConditions.attributeToBeNotEmpty(org.openqa.selenium.WebElement element, String attribute)

ExpectedConditions.elementSelectionStateToBe(org.openqa.selenium.By locator, boolean selected)

ExpectedConditions.elementSelectionStateToBe(org.openqa.selenium.WebElement element, boolean selected)

ExpectedConditions.elementToBeClickable(org.openqa.selenium.By locator)

ExpectedConditions.elementToBeClickable(org.openqa.selenium.WebElement element)

ExpectedConditions.elementToBeSelected(org.openqa.selenium.By locator)

ExpectedConditions.elementToBeSelected(org.openqa.selenium.WebElement element)

ExpectedConditions.frameToBeAvailableAndSwitchToIt(org.openqa.selenium.By locator)

ExpectedConditions.frameToBeAvailableAndSwitchToIt(int frameLocator)

ExpectedConditions.frameToBeAvailableAndSwitchToIt(String frameLocator)

ExpectedConditions.frameToBeAvailableAndSwitchToIt(org.openqa.selenium.WebElement frameLocator)

ExpectedConditions.invisibilityOfAllElements(List elements)

ExpectedConditions.invisibilityOfElementLocated(org.openqa.selenium.By locator)

ExpectedConditions.invisibilityOfElementWithText(org.openqa.selenium.By locator, String text)

ExpectedConditions.javaScriptThrowsNoExceptions(String javaScript)

ExpectedConditions.jsReturnsValue(String javaScript)

ExpectedConditions.not(ExpectedCondition condition)

ExpectedConditions.numberOfElementsToBe(org.openqa.selenium.By locator, Integer number)

ExpectedConditions.numberOfElementsToBeLessThan(org.openqa.selenium.By locator, Integer number)

ExpectedConditions.numberOfElementsToBeMoreThan(org.openqa.selenium.By locator, Integer number)

ExpectedConditions.numberOfWindowsToBe(int expectedNumberOfWindows)

ExpectedConditions.or(ExpectedCondition… conditions)

ExpectedConditions.presenceOfAllElementsLocatedBy(org.openqa.selenium.By locator)

ExpectedConditions.presenceOfElementLocated(org.openqa.selenium.By locator)

ExpectedConditions.presenceOfNestedElementLocatedBy(org.openqa.selenium.By locator, org.openqa.selenium.By sub\_locator)

ExpectedConditions.presenceOfNestedElementLocatedBy(org.openqa.selenium.WebElement element, org.openqa.selenium.By sub\_locator)

ExpectedConditions.presenceOfNestedElementsLocatedBy(org.openqa.selenium.By locator, org.openqa.selenium.By sub\_locator)

ExpectedConditions.refreshed(ExpectedCondition condition)

ExpectedConditions.stalenessOf(org.openqa.selenium.WebElement element)

ExpectedConditions.textMatches(org.openqa.selenium.By locator, Pattern pattern)

ExpectedConditions.textToBe(org.openqa.selenium.By locator, String value)

ExpectedConditions.textToBePresentInElementLocated(By, String)

ExpectedConditions.textToBePresentInElement(org.openqa.selenium.WebElement element, String text)

ExpectedConditions.textToBePresentInElementLocated(org.openqa.selenium.By locator, String text)

ExpectedConditions.textToBePresentInElementValue(org.openqa.selenium.By locator, String text)

ExpectedConditions.textToBePresentInElementValue(org.openqa.selenium.WebElement element, String text)

ExpectedConditions.titleContains(String title)

ExpectedConditions.titleIs(String title)

ExpectedConditions.urlContains(String fraction)

ExpectedConditions.urlMatches(String regex)

ExpectedConditions.urlToBe(String url)

ExpectedConditions.visibilityOf(org.openqa.selenium.WebElement element)

ExpectedConditions.visibilityOfAllElements(List elements)

ExpectedConditions.visibilityOfAllElementsLocatedBy(org.openqa.selenium.By locator)

ExpectedConditions.visibilityOfElementLocated(org.openqa.selenium.By locator)

ExpectedConditions.visibilityOfNestedElementsLocatedBy(org.openqa.selenium.By locator, org.openqa.selenium.By sub\_locator)

ExpectedConditions.visibilityOfNestedElementsLocatedBy(org.openqa.selenium.WebElement element, org.openqa.selenium.By sub\_locator)

### Fluent Wait

Wait is the generic interface, which makes selenium to wait for a particular time with the associated event. Events like elements to be click-able, element to present.  
  
Wait interface present under org.openqa.selenium.support.ui package, FluentWait, and [WebdriverWait](https://chercher.tech/java/explicit-wait-selenium" \t "_blank" \o "WebdriverWait in selenium)implements Wait interface.

until is an [abstract method](https://chercher.tech/java/abstraction-selenium-java)present in the Wait interface, whichever driver implements [Webdriver](https://chercher.tech/java/index-selenium-webdriver" \t "_blank" \o "Webdriver selenium) the also must implement the Wait interface.  
  
methods present in Wait and Webdriver interfaces.  
  
until method accepts ExpectedConditions values as a parameter, the class must wait till the given condition neither becomes null nor false when they implement until method. So until methods implementation must not have return type as void.  
  
FluentWait class implements Wait Interface and [WebdriverWait](https://chercher.tech/java/explicit-wait-selenium" \t "_blank" \o "WebdriverWait selenium) class extends FluentWait class, this is called as a [multi-level inheritance in java](https://chercher.tech/java/inheritance-selenium-java#multi-level).

* FluentWait class implements the Wait interface in selenium, FluentWait object defines the maximum amount of time to wait for a condition. Below is the example for a common way of using Fluent Wait,
* Users can configure the frequency with which to check the condition.
* Users may configure the FluentWait to ignore specific types of exceptions while waiting for the condition.
* Users can configure the error message to display in case of TimeoutException occurs.
* Can set the time limit for Fluent wait, i.e. how much time to wait for a condition when Fluent wait is used

|  |
| --- |
| Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)  .withTimeout(30, SECONDS)  .pollingEvery(5, SECONDS)  .ignoring(NoSuchElementException.class); |

# Take Screenshots

Test cases may fail while executing the test suite in selenium automation. When manual tester faces any discrepancy in the verification, i.e., when there is a difference in actual and expected values, manual testers tend to take screenshots as proof for failure.  
Failures may occur because of below reasons:

* Actual and [Expected values](https://chercher.tech/java/testng-assertions-selenium-webdriver) are not matching
* When there is no element
* When the page takes more time to load
* When an unexpected alert comes in to focus
* When there are Assertion issues

There is an interface called TakesScreenshot which provides the getScreenshotAs() method and which selenium uses to take a screenshot*.*

Our browser classes (Like ChromeDriver, FirefoxDriver) extends RemoteWebdriver, and RemoteWedriver implements TakesScreenshot Interface along with Webdriver Interface

Our browser classes (Like ChromeDriver, FirefoxDriver) extends RemoteWebdriver, and RemoteWedriver implements TakesScreenshot Interface along with Webdriver Interface

Formats available in the OutputType Interface:

1. OutputType.BASE64

2. OutputType.BYTES

3. OutputType.FILE

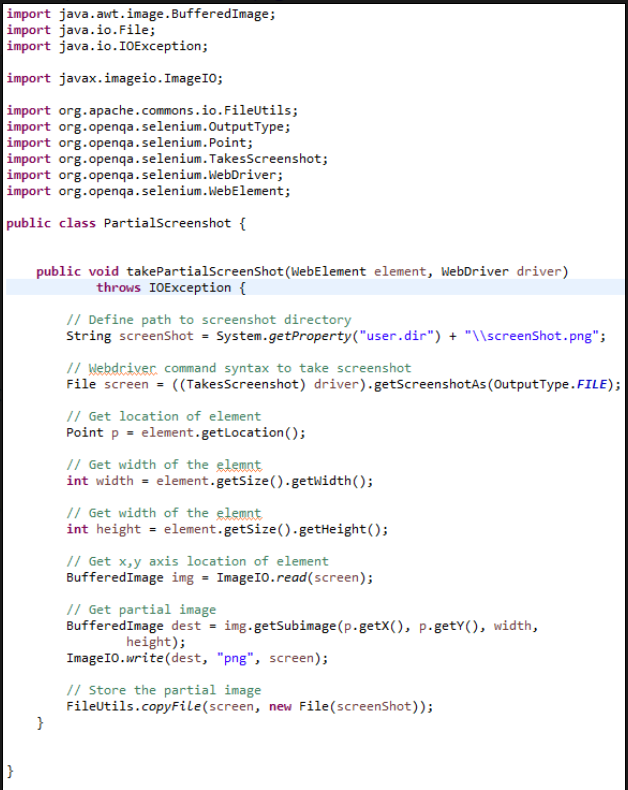
### Full Screenshot

We can take a screenshot of a webpage using the getScreenshotAs() method from the TakesScreenshot, but we cannot initialize [TakesScreenshot as it is an interface](https://chercher.tech/java/abstraction-selenium-java" \l "interface" \t "_blank)  
So to take a screenshot of the page, we have to cast our driver object to the TakesScreenshot interface type by adding the below code.

|  |
| --- |
| File scrFile = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);  // store the screenshot to your local machine  FileUtils.copyFile(scrFile, new File("D:pageScreenshot.png"), true); |

### Partial Screenshot

Sometimes we don't want to take screenshot of the full screen. Reasons might be, full size images would last in huge memory storage for image directory or sometimes we need to take element specific screenshot due to test case requirement which makes more sense.



# Java Script Executor

We should go for Javascript Executor only when we are not able to perform a particular task with our selenium; like some, we may not be able to [click an element](https://chercher.tech/java/sendkeys-click-clear-submit-selenium-webdriver#click), such cases we can go for Javascript Executor.

JavascriptExecutor Interface has two abstract methods which are :  
  
1. executeScript()  
2. executeAsyncScript()

We have to cast driver object into JavascriptExecutor type to use the methods present in the JavascriptExecutor interface

|  |
| --- |
| // cast the driver object to JavascriptExecutor  JavascriptExecutor js = (JavascriptExecutor) driver;  // access the methods  js.executeScript("javascript command");  js.executeAsyncScript("javascript command");  (or)  ((JavascriptExecutor) driver).executeScript("javascript command"); |

**Find an Element with JavascriptExecutor**

Call the executeScript("Command") method from the object; we have to use a return statement to return a value from the browser.  
  
We have to cast the object returned from JavascriptExecutor to webelement to access all the web element related methods.

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  Object searchTextbar = js.executeScript("**return document.getElementById('lst-ib')**");  // we have to cast the returned object into webelement to access  // all the webelement related methods  ((WebElement) searchTextbar).sendKeys("abc"); |

**Click an element with JavascriptExecutor**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("document.getElementsByName('btnI')[0].click()") |

**Sendkeys with JavascriptExecutor**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("document.getElementById('lst-ib').value='ppp'") |

**Scroll webpage in JavascriptExecutor**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("window.scrollBy(0,300)") |

scrollBy(horizontalDistance, verticalDistance) accepts two parameter  
  
1. horizontalDistance - distance to be scrolled horizontally  
2. verticalDistance - distance to be scrolled vertically

**Scroll Into View in JavascriptExecutor**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("document.getElementById('default').scrollIntoView(true)") |

**get the**[**title of the webpage**](https://chercher.tech/java/get-title-url-page-source-selenium-webdriver)**using JavascriptExecutor**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("return document.title"); |

**check the state of the webpage loaded or not**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("return document.readyState"); |

**Domain Name with JavascriptExecutor :**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("return document.domain"); |

**Zoom Levels with JavascriptExecutor :**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("document.body.style.zoom='90"); |

**Highlight an Element in JavascriptExecutor**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("document.getElementById('lst-ib').style.backgroundColor='red'") |

**get the height of the webpage**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("window.innerHeight"); |

**fetch the width of the webpage**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("window.innerWidth") |

**fetch the height and width of the browser;**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  js. executeScript("window.outerHeight");  js.executeScript("window.outerWidth"); |

**Scroll By Pages in JavascriptExecutor :**  
We can scroll the webpage based on the number of pages

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  // scrolls the page by three page downs  js.executeScript("window.scrollByPages(3)"); |

Similarly, we can perform page up as well in the above command; we have pass a negative number to simulate the pageup.

|  |
| --- |
| js.executeScript("window.scrollByPages(-5)"); |

**Navigate to Url in JavascriptExecutor :**

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  // Opens the Google page  js.executeScript("window.location="https://google.com";"); |

**Maximum Scroll distance in JavascriptExecutor :**  
With JavascriptExecutor in selenium, we can calculate the maximum scrollable distance using the window.scrollMaxY and window.scrollMaxX.

window.scrollMaxY gives detail about how much distance we can scroll Vertically, window.scrollMaxXgives maximum scroll detail about how much distance we can scroll Horizontally

|  |
| --- |
| JavascriptExecutor js = (JavascriptExecutor) driver;  // maximum scroll distance vertically  js.executeScript("return window.scrollMaxY  // maximum scroll distance Horizontally  js.executeScript("return window.scrollMaxX"); |

# Data Driven Testing

In every Development environment, data is a mandatory part of their development; we put all your data in your program files. But if any change in data results in the editing of the program file, which makes us recompile the code and retest the compiled code.

Development should happen in such that data don't have any effect on the program files. Placing data outside the program is the only way to do it, like placing the data on excel files, property files, config files, JSON Files, XML files.

### APACHI POI

Apache POI helps Java/Java related technologies to read and write Excel files on different platforms. Using apache poi, we can do read and write operation of both xls and xlsx file formats. Apache poi is an open-source tool developed by apache.

Apache POI excel library revolves around four key interfaces that actually represent the items in the excel file.

* Workbook: A workbook represents the excel file
* Sheet: A workbook may contain many sheets. We can access the sheets either with a name or with an index.
* Row: As the name suggests, It represents a row in the sheet.
* Cell: A cell represents a column in the sheet.

**Apache POI** is that it supports read and write operations on both .xls and .xslx files.

**Step #1)**

Firstly, we need to configure Eclipse with **Apache POI**.

[Download](https://archive.apache.org/dist/poi/release/bin/poi-bin-3.10-FINAL-20140208.zip) jar files for Apache POI.

**Step #2)**

Unzip the jar file, and add the following jars to your project and configure them.

* dom4j-1.6.1.jar
* poi-3.10-FINAL-20140208.jar
* poi-ooxml-3.10-FINAL-20140208.jar
* poi-ooxml-schemas-3.10-FINAL-20140208.jar
* xmlbeans-2.3.0.jar

**Step #3)**

After configuring the respective jars, create an excel sheet enter some data in it and save it as *TestData.xlsx* at your preferred location.

Apache POI is the most commonly used API for Selenium data driven tests. POI is a set of library files that gives an API to manipulate Microsoft documents like .xls and .xlsx. It lets you create, modify, read and write data into Excel. In POI, Workbook is a common interface for HSSF, XSSF and SXSSF; where HSSF, Horrible Spread sheet Format implements Excel ’97 (.xls) file format and XSSF, XML Spread sheet Format implements Excel 2007 (.xlsx) file format. SXSSF is the streaming extension for XSSF which was introduced later in Apache POI 3.8.

Below mentioned are some of the interfaces of POI.

* XSSFWorkbook: Represents workbook in xlsx file.
* HSSFWorkbook: Represents workbook in xls file.
* XSSFSheet: Represents a sheet in XLSX file.
* HSSFSheet: Represents a sheet in XLS file.
* XSSFRow: Represents a row in a sheet of XLSX file.
* HSSFRow: Represents a row in a sheet of XLS file.
* XSSFCell: Represents a cell in a row of XLSX file.
* HSSFCell: Represents a cell in a row of XLS file.

Fields available in a cell:

* CELL\_TYPE\_BLANK: Represents a blank cell.
* CELL\_TYPE\_BOOLEAN: Represents a Boolean cell (true or false).
* CELL\_TYPE\_ERROR: Represents an error value in a cell.
* CELL\_TYPE\_FORMULA: Represents a formula result on a cell.
* CELL\_TYPE\_NUMERIC: Represents numeric data in a cell.
* CELL\_TYPE\_STRING: Represents string in a cell.

Below are the methods will use to read the data, which are available in the above classes

1. getSheet(“sheetName”) – Get sheet with the given name
2. getLastCellNum() – Get the index of the last cell contained in the row Plus one as index starts from ZERO
3. getStringCellValue() – Get the value of the cell as a String
4. getRow(int) – Returns the row.
5. getCell(int) – Get the cell representing a given column
6. getNumericCellValue() – Get the value of the cell as a number.
7. getDateCellValue() – Get the value of the cell as a date.
8. getBooleanCellValue() – Get the value of the cell as a boolean.

**Points to Remember:**

1. Row index starts from ZERO
2. Column index starts from ZERO
3. Extension of excel file is ‘.xlsx’ from Excel 2007
4. If you want to read integer value then you have to use “‘”(i.e. apostrophe) value in front of that value.[Example : 10 → ‘10]
5. We need to use different kind of methods to read String, Integer and Date values from the excel.

#### Read Excel with Apache POI

**Workbook Factory** for creating the appropriate kind of Workbook (be it [HSSFWorkbook](https://poi.apache.org/apidocs/dev/org/apache/poi/hssf/usermodel/HSSFWorkbook.html" \o "class in org.apache.poi.hssf.usermodel) or XSSFWorkbook), by auto-detecting from the supplied input depending on the excel formats .xls or .xlsx.

|  |
| --- |
| package com.crm.qa.util;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.IOException;  import org.apache.commons.io.FileUtils;  import org.apache.poi.openxml4j.exceptions.InvalidFormatException;  import org.apache.poi.ss.usermodel.Sheet;  import org.apache.poi.ss.usermodel.Workbook;  import org.apache.poi.ss.usermodel.WorkbookFactory;  import org.openqa.selenium.OutputType;  import com.crm.qa.base.TestBase;  public class TestUtil extends TestBase{  public static long PAGE\_LOAD\_TIMEOUT = 20;  public static long IMPLICIT\_WAIT = 20;    public static String TESTDATA\_SHEET\_PATH = "D:\\SELENIUM\\eclipse-workspace\\POMTestNG\\src\\main\\java\\com\\crm\\qa\\testdata\\TestData.xlsx";    static Workbook book;  static Sheet sheet;    public static Object[][] getTestData(String sheetName) {  FileInputStream file = null;  try {  file = new FileInputStream(TESTDATA\_SHEET\_PATH);  } catch (FileNotFoundException e) {  e.printStackTrace();  }  try {  book = WorkbookFactory.create(file);  } catch (InvalidFormatException e) {  e.printStackTrace();  } catch (IOException e) {  e.printStackTrace();  }  sheet = book.getSheet(sheetName);  Object[][] data = new Object[sheet.getLastRowNum()][sheet.getRow(0).getLastCellNum()];    for (int i = 0; i < sheet.getLastRowNum(); i++) {  for (int k = 0; k < sheet.getRow(0).getLastCellNum(); k++) {  data[i][k] = sheet.getRow(i + 1).getCell(k).toString();    }  }  return data;  } |

#### Write Excel with Apache POI

### Reading CSV Files

# TESTNG

TestNG is a unit test framework designed for testing needs, TestNG is designed to cover all categories of tests: unit, functional, end-to-end, integration, etc.  
  
It is inspired from JUnit and NUnit by adding new functionalities, which made TestNG more powerful than other unit test frameworks.

**Features of TESTNG**

* Support for annotations
* Test that your code is multi-thread safe.
* Support for data-driven testing (with @DataProvider).
* Support for parameters.
* It is supported by a variety of tools and plug-ins (Eclipse, IDEA, Maven, etc...).
* Allows user to set execution priorities for the test methods
* Helps to generate reports
* To run test cases in the order which we provide
* To create Dependent Test cases

**Advantages of TESTNG**

There are advantages that make TestNG superior to JUnit. Some of them are

* Advance and easy annotations
* Execution patterns can be set
* Concurrent execution of test scripts
* Test case dependencies can be set
* Parallel Execution

### Installation of TESTNG

Follow the below steps to install TestNG to java project in eclipse.

1. Open Eclipse IDE, then goto help

2. Select install new software

3. Click on Add

4.Specify the name as TestNG

5.Specify the Location as <https://dl.bintray.com/testng-team/testng-eclipse-release/>

6. Click “Ok᾿

7.Select TestNG checkbox

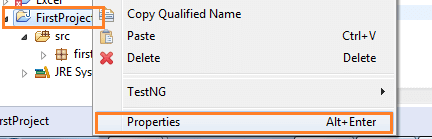
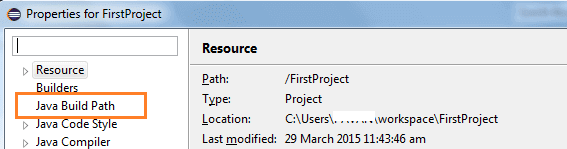
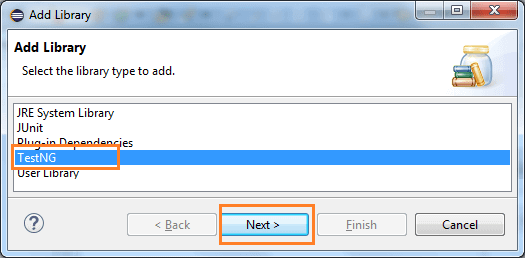
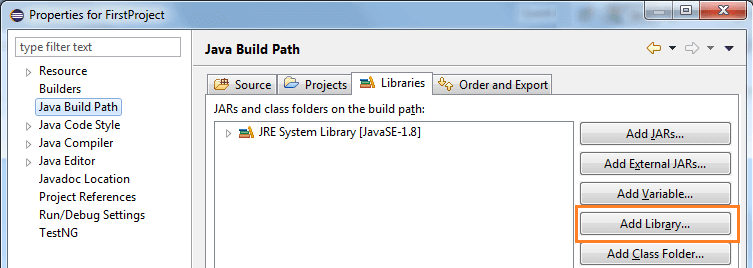
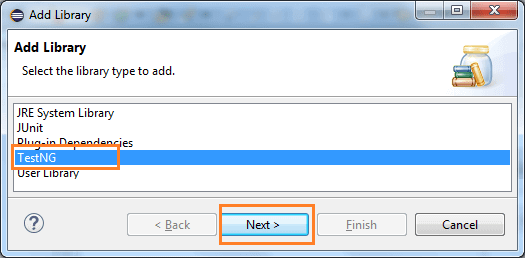
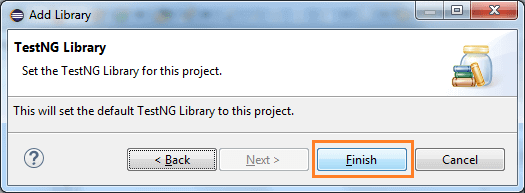
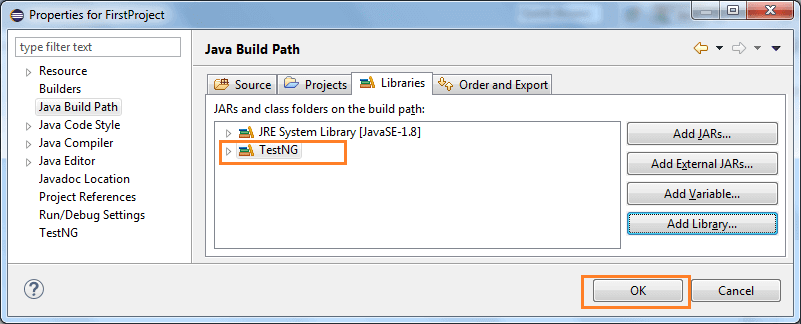
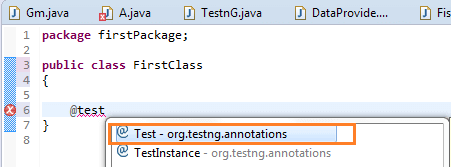
8.Click Next

9. Accept license

10. Click OK on the warning message

11. Click Yes Which will re-start the Eclipse IDE

**Associate TESTNG to Project**

Follow the below steps to associate TestNG to java project in eclipse.  
1. Right Click on the Java Project & Goto Properties>Java Build Path  
  
  
  
  
  
2. Goto Libraries>Add Library  
  
  
  
  
3.Select TestNG>Click Next>Finish  
  
  
  
  
  
  
  
4. After adding TestNG, write code like @Test, then you should get options like below.  
  


### TESNG XML

**Create TESTNG XML**

In TestNG framework, we need to create [Testng xml](https://www.qable.io/basic-of-testng-how-we-can-use-it-in-the-testing/) file to create and handle multiple test classes.

TestNG.xml file is an XML file which contains all the Test configuration and this XML file can be used to run and organize our test.

**The testng.xml file has the numerous uses as listed below**

* Test cases are executed in groups.
* Test methods can be included or excluded in the execution.
* The execution of multiple test cases from multiple java class files can be triggered.
* Comprises names of the folder, class, method.
* Capable of triggering parallel execution.
* Test methods belonging to groups can be included or excluded in the execution.

<suite>: TESTNG XML file can contain only one suite tag. We can give any name to the Suite

<test>: Can contain multiple test tags under suite. We can give any name to the test

<classes>Can contain only one classes tag for each test but can have multiple class tags

<class>: Can contain multiple class for <classes> under test. A Class is a Java class that contains TestNG annotations. we haveto provide the correct name to the classes tag which is a combination of Package name and the Java Class name

<packages> You can specify package names instead of class names. Can contain only one packages tag for each test but can have multiple package tags

<package> Can contain multiple package for <packages> under test.

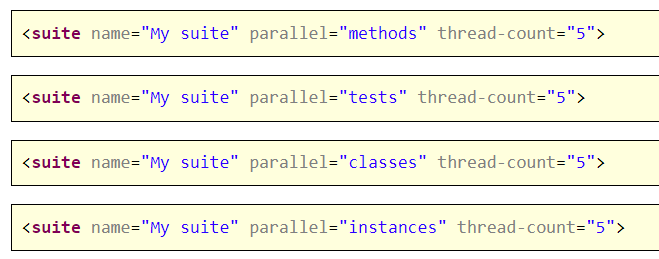
<groups> Groups are specified in the testng.xml file with <groups> tag. Groups can be specified either in the <suite> tag or <test> tag. If the <groups> tag is specified inside the <suite> tag, then it is applied to all the <test> tags of XML file. If the <groups> tag is specified within a particular <test> folder, then it is applied to that particular <test> tag only. Executes the tests having included groups and tests with no Group name. If the test contains both include and exclude Group name, the tests are executed.

<methods> methods can be specified either in the <class> tag. Can have include and exclude tags. If methods is specified we should specify either include or exclude tags or both else no methods in class will be executed

<parameter> pass parameters to test methods. Can be defined at Suite level or test level

<listeners> specified at suite level. Can contain only one listeners tag for each Suite but can have multiple listener tags

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">  <suite name="Suite1">  <listeners>  <listener class-name="com.example.MyListener" />  <listener class-name="com.example.MyMethodInterceptor" />  </listeners>  <parameter name="first-name" value="XXXX"/>  <test name="Regression suite 1" >  <groups>  <run>  <include name="SanityTest" />  <exclude name="SmokeTest" />  </run>  </groups>    <packages>  <package name="com.sample2" />  </packages>  <parameter name="second-name" value="Cedric"/>  <classes>  <class name="com.sample.Seltest2"/>  <class name="com.sample.Seltest1">  <methods>  <include name="testMethod1" />  <exclude name="testMethod2"/>  </methods>  </class>  </classes>    </test>  </suite> |



**CREATE MULTIPLE SUITES**

**Consolidation in TESTNG XML**

Prepare separate XML for each suite and have consolidated them in testng.xml.

For example suiteA.xml, suiteB.xml, suiteC.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >  <suite name="SuiteA" >  <!-- suite name="Suite Name" -->  <test name="TestA1" allow-return-values="true">  <classes>  <!-- packagename.Testcase class name -->  <class name ="com.qtpselenium.suiteA.TestCaseA1" />  </classes>  </test>  <test name="TestA2" allow-return-values="true">  <classes>  <!-- packagename.Testcase class name -->  <class name ="com.qtpselenium.suiteA.TestCaseA1" />  </classes>  </test>  </suite> |

Consolidate in TESTNG XML

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >  <suite name="TestNG Dadadriver suite" >  <!-- suite name="Suite Name" -->  <suite-files>  <suite-file path="./suiteA.xml" />  <suite-file path="./suiteB.xml" />  <suite-file path="./suiteC.xml" />  </suite-files>  </suite> |

**From MAVEN**

|  |
| --- |
| <plugin>  <artifactId>maven-surefire-plugin</artifactId>  <configuration>  <suiteXmlFiles>  <suiteXmlFile>src/test/resources/unit-testng.xml</suiteXmlFile>  <suiteXmlFile>src/test/resources/api-testng.xml</suiteXmlFile>  </suiteXmlFiles>  </configuration>  </plugin> |

### TESTNG Execution

TestNG can be invoked in different ways:

* Command line
* [Eclipse](https://testng.org/doc/eclipse.html)
* [MAVEN](https://testng.org/doc/idea.html)
* [ant](https://testng.org/doc/ant.html)

**Command Line**

|  |
| --- |
| java org.testng.TestNG testng1.xml [testng2.xml testng3.xml ...] |

**Eclipse**

Select the testng.xml file, right-click on it, and select **Run As** | **TestNG suite**.

**MAVEN**

Once Maven Plugin is installed. You need to create Maven Project. Add TestNG Dependency to pom.xml

|  |
| --- |
| <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  <modelVersion>4.0.0</modelVersion>  <groupId>com.javacodegeeks.testng.maven</groupId>  <artifactId>testngMaven</artifactId>  <version>0.0.1-SNAPSHOT</version>  <dependencies>  <dependency>  <groupId>org.testng</groupId>  <artifactId>testng</artifactId>  <version>6.8.8</version>  <scope>test</scope>  </dependency>  </dependencies>  </project> |

Right click on project, click on ‘Run As’ and then click on ‘Maven Test’



CMD

After adding dependencies, navigate to project directory and run maven command from command-line which downloads all dependencies and adds them to the project.

|  |
| --- |
| mvn clean install |

We have to include, maven-compiler-plugin and maven-surefire-plugin to the configuration file Pom.xml

Maven-compiler-plugin: Compiler-plugin is used to compile the sources of our project

|  |
| --- |
| <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <version>3.5.1</version>  <configuration>  <source>${jdk.level}</source>  <target>${jdk.level}</target>  </configuration>  </plugin> |

Maven-surefire-plugin: Surefire-plugin is responsible for running tests that are placed in test source directory /src/test/java.

|  |
| --- |
| <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>2.19.1</version>  <configuration>  <suiteXmlFiles>  <!-- TestNG suite XML files -->  <suiteXmlFile>testng.xml</suiteXmlFile>  </suiteXmlFiles>  </configuration>  </plugin> |

Execute tests using 'mvn test' from command prompt.

|  |
| --- |
| mvn test |

#### Parallel Execution

Quite often in automation testing, we want to reduce the test execution time to get the test results as fast as possible. When we run the selenium tests using as testNG suite, by default it runs the tests serially. But testNG provides an inherent support to run the tests in parallel.

In order to run the tests in parallel just add these two key value pairs in suite-

* parallel="{methods/tests/classes}"
* thread-count="{number of thread you want to run simultaneously}".

|  |
| --- |
| <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  <suite name="ArtOfTestingTestSuite" verbose="1" parallel="methods" thread-count="5">  <test name="parallelExecutionTestsA" >  <classes>  <class name="testsInParallelA1" />  <class name="testsInParallelA2" />  </classes>  </test>  <test name="parallelExecutionTestsB" >  <classes>  <class name="testsInParallelB1" />  <class name="testsInParallelB2" />  </classes>  </test>  </suite> |

Three options that can be set as value for the "parallel" parameter-

**methods** - run all the tests independently on separate threads(maximum available threads)

**tests** - run the methods specified in the test tag in the same thread

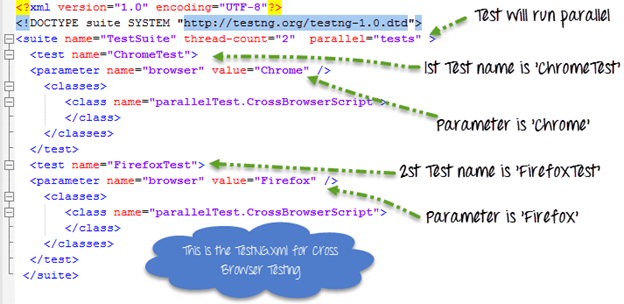
**classes** - run the all the test methods stated in a class in same thread

#### Cross Browser Testing

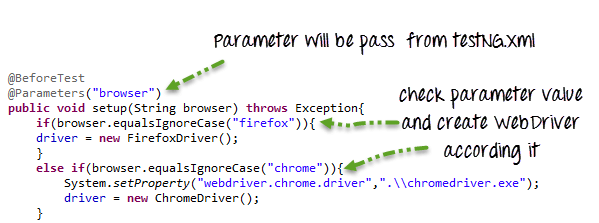
If we are using Selenium WebDriver, we can automate test cases using Internet Explorer, FireFox, Chrome, Safari browsers.

To execute test cases with different browsers in the same machine at the same time we can integrate[Testng](https://www.guru99.com/all-about-testng-and-selenium.html)framework with Selenium WebDriver.

Your testing.xml will look like that,



This testing.xml will map with the[Test Case](https://www.guru99.com/test-case.html)which will look like that



### ASSERTIONS

Assertions in TestNG are a way to verify that the expected result and the actual result matched or not.

There are two types of TestNg Assert:

* Hard Assert
* Soft Assert

Hard Asserts are those asserts that stop the test execution when an assert statement fails, and the subsequent assert statements are therefore not validated.  It plays a vital role in projects where we have an element without whose validation, asserting other elements is useless.

|  |
| --- |
| Assert.assertEqual(String actual, String expected):  Assert.assertEqual(String actual, String expected, String message)  Assert.assertEquals(boolean actual, boolean expected)  Assert.assertTrue(condition)  Assert.assertTrue(condition, message)  Assert.assertFalse(condition)  Assert.assertFalse(condition, message) |

Soft Assert  In soft asserts, the subsequent assertions keep on running even though one assert validation fails, i.e., the test execution does not stop. Soft assert does not include by default in TestNG. For this, you need to include the package org.testng.asserts.Softassert. We use soft asserts when we do not care about the failure of specific validations and want the test execution to proceed and also want to see the exception errors.A good example is multiple validations on an input form.

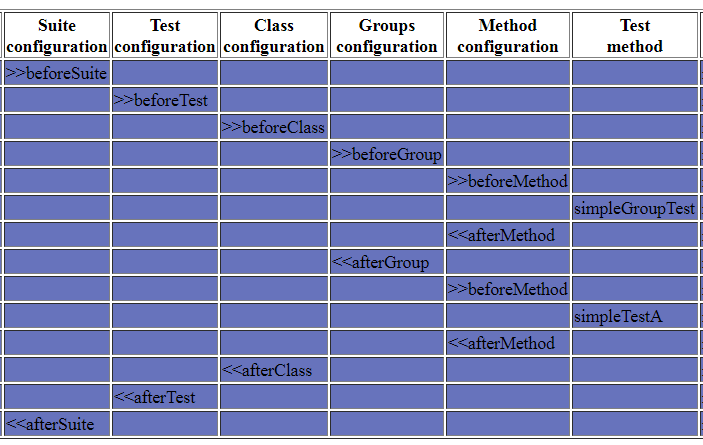
|  |
| --- |
| import org.testng.asserts.SoftAssert;  SoftAssert softassert = new SoftAssert();  softassert.assertEquals(originalTitle, expectedTitle);  softassert.assertEquals(originalTitle, "expected" );  softassert.assertAll(); |

object.assertAll() statement is required to see the exceptions; otherwise, the tester won't know what passed and what failed

### TESTNG Annotations

**@BeforeSuite:** The annotated method will be run before all tests in this suite have run.  
**@AfterSuite:** The annotated method will be run after all tests in this suite have run.  
**@BeforeTest:** The annotated method will be run before any test method belonging to the classes inside the tag is run.  
**@AfterTest:** The annotated method will be run after all the test methods belonging to the classes inside the tag have run.  
**@BeforeGroups:** The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.  
**@AfterGroups:** The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.  
**@BeforeClass:** The annotated method will be run before the first test method in the current class is invoked.  
**@AfterClass:** The annotated method will be run after all the test methods in the current class have been run.  
**@BeforeMethod:** The annotated method will be run before each test method.  
@AfterMethod: The annotated method will be run after each test method.  
**@BeforeMethod:** The annotated method will be run before each test method.  
**@AfterMethod:** The annotated method will be run after each test method.

**Execution Order**



**Points to Note**

* @BeforeMethod,@AfterMethod will not be executed if there are no @Test Methods available in the TESTNG class.
* @BeforeMethod Configuration fails the @Test,@AfterMethod will not be executed.
* @AfterMethod(alwaysRun=true) executes even if @BeforeMethod Configuration fails.
* @BeforeClass,@AfterClass will not be executed if there is no @Test Methods available in the TESTNG class.
* @BeforeClass Configuration fails the @Test methods will be skipped and Afterclass will not be executed.
* @AfterClass(alwaysRun=true) executes even if @Beforeclass Configuration fails.
* If there is no @Test, then no other annotation will get executed.
* @BeforeMethod will be executed for every @Test method present in the class, @BeforeMethod declared in one class will not affect @Test present in the Other classes
* @AfterMethod will have cleaning code for the method like deletion of details and logging out, or refreshing pages
* we will provide the group name to @AfterGroups , @BeforeGroups
* @BeforeTest solely depends on <test> tags present in testng.xml.
* Even there is no @Test method present in the class, the @BeforeTest method will be executed; also, you might see a message that there are not @Test methods.
* @BeforeSuite will executed before executing <suite> tag in testng.xml

#### Annotations along with Attributes/Properties

##### Enable /Disable

When you set enabled=false to any TESTNG method, then that method will not be part of the test suite, so test result will not have any information regarding these methods. This method will not be considered as skipped

It is an optional parameter. By default, TestNG considers enabled=true, if you don't provide one.

|  |
| --- |
| @Test(enabled=false)  @BeforeTest(enabled=true) |

Applicable for TESTNG Annotations - **@BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest, @BeforeGroups, @AfterGroups, @BeforeClass, @AfterClass, @BeforeMethod, @AfterMethod, @Test**

##### Always Run

This is used when we want to make sure a method always runs even if the parameters on which the method depends, fails. If set to true, this test method will always run.

|  |
| --- |
| *@Test(alwaysRun = true)* |

Applicable for TESTNG Annotations - **@BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest, @BeforeGroups, @AfterGroups, @BeforeClass, @AfterClass, @BeforeMethod, @AfterMethod, @Test**

##### Depends on Methods

Sometimes we want our test cases to run in specific order. One of the very common examples is when we want to run test cases for CRUD operations. So we want to make sure that test data is inserted first, then it’s retrieved, updated and finally deleted.

The better way is to use dependsOnMethods to tell TestNG which methods this test is dependent on, so those methods should be executed before this method.

dependsOnMethods takes String array as argument. We can provide method names

|  |
| --- |
| @Test(dependsOnMethods= {"insert","update"}) |

Applicable for TESTNG Annotations - **@BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest, @BeforeGroups, @AfterGroups, @BeforeClass, @AfterClass, @BeforeMethod, @AfterMethod, @Test**

##### Depends on Groups

We can also specify if our test depends on any other groups. This is helpful when we have multiple methods and they are grouped together. So we can simply specify the group this test depends on, rather than specifying the huge list of all the methods.

|  |
| --- |
| <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name="TestNG XML Dependency Suite" >  <test name="ToolsQA" >  <groups>  <dependencies>  <group depends-on= "openbrowser" name= "login"></group>  <group depends-on= "login" name= "viewaccount"></group>  <group depends-on= "viewaccount" name= "logout"></group>  </dependencies>  </groups>  <classes>  <class name="GroupDependency" />  </classes>  </test>  </suite> |

|  |
| --- |
| *@Test (groups = { “City” ,”State” })* |

Applicable for TESTNG Annotations - **@BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest, @BeforeGroups, @AfterGroups, @BeforeClass, @AfterClass, @BeforeMethod, @AfterMethod, @Test**

##### Groups

Grouping the Tests

|  |
| --- |
| @Test (groups = { "sanity”, "Regression" }) |

Applicable for TESTNG Annotations - **@BeforeSuite, @AfterSuite, @BeforeTest, @AfterTest, @BeforeGroups, @AfterGroups, @BeforeClass, @AfterClass, @BeforeMethod, @AfterMethod, @Test**

##### Inherit Groups

If true, this method will belong to groups specified in the @Test annotation at the class level.

|  |
| --- |
| @BeforeTest(inheritGroups = true) |

Applicable for TESTNG Annotations - **except @Test**

##### Description

We can provide a description of methods when we want to understand the purpose of the test.

|  |
| --- |
| @Test(description=”Text here”) |

Applicable for TESTNG Annotations - **@Test**

##### Priority

Priority is nothing but how much importance we give to a particular thing. Lower priorities will be scheduled first.

By default, TestNG provides zero priority to all @Tests and providing priority is not mandatory. When two @Test shares the same priority number then execution happens in alphabetical order only among them.

Priority is a numeric integer (negative also) constant value; we cannot use a variable in place of priority

|  |
| --- |
| @Test(priority=3) |

Applicable for TESTNG Annotations **- @Test**

##### Invocation Count

Sometimes you might need to invoke the same method, repeatedly, to make sure the method does not fail or to create data.

invocationCount property helps the user to invoke a particular method with a given number of times.

|  |
| --- |
| @Test(invocationCount =3) |

Applicable for TESTNG Annotations - **@Test**

##### Invocation Timeout

invocationTimeOut is the maximum number of milliseconds that the total number of invocations on this test method should take.  
  
For Example, if invocationTimeOut is set 30 seconds and invocationCount is set to 10 times, then this method should complete all 10 invocations within 30 seconds; otherwise, the test will be marked as a fail.

This annotation will be ignored if the attribute invocationCount is not specified on this method.

|  |
| --- |
| @Test(invocationCount = 10, invocationTimeOut = 30000) |

Applicable for TESTNG Annotations - **@Test**

##### Skip Failed Invocations

If true and invocationCount is specified with a value > 1, then all invocations after a failure will be marked as a SKIP instead of a FAIL.  
  
this method will be ignored if invocationTimeOut is specified in the @Test.

|  |
| --- |
| @Test(invocationCount = 7, skipFailedInvocations=true) |

Applicable for TESTNG Annotations - **@Test**

##### Thread Pool size

ThreadPoolSize is denotes how many threads should invoke a specific function.

The threadPoolSize method will be invoked from multiple threads as specified by the invocationCount.

This attribute is ignored if invocationCount is not specified

|  |
| --- |
| @Test(invocationCount = 7, threadPoolSize=7) |

Applicable for TESTNG Annotations - **@Test**

##### Time Out

timeOut is nothing but sets the time limit for the execution of the method; if the method execution is not completed within the given time, then the test method will be considered as fail and marks so

A timeout can be configured in two ways:

* @Test level : This will be applicable for the said test method and will override the time period if configured at the suite level
* testng.xml level : This will be applicable for all the tests in the said TestNG test suite

timeOut units are measured in terms of Milli-seconds

|  |
| --- |
| @Test(timeOut=5000) |

Applicable for TESTNG Annotations - **@Test**

All tests under suite whose execution time less than time-out will be executed and others marked as Failed

|  |
| --- |
| <suite name="Time test Suite" time-out="500" verbose="1" >    <test name="Timeout Test" >      <classes>        <class name="com.howtodoinjava.test.TimeoutSuite" />      </classes>    </test>  </suite> |

##### Expected Exceptions

Sometimes we might have scenarios where we expect an exception as a result; In such cases, we have to verify that an exception is being thrown by the program during execution.

|  |
| --- |
| @Test(expectedExceptions = ArithmeticException.class) |

Applicable for TESTNG Annotations - **@Test**

We can write an array of exceptions when we are expecting anyone of given exceptions.

|  |
| --- |
| @Test(expectedExceptions =  {NoSuchElementException.class,  ElementNotVisibleException.class}) |

### Annotations to Parameterize

There are two ways by which we can achieve parameterization in TestNG

1. With the help of Parameters annotation and TestNG XML file.
2. With the help of DataProvider annotation.

Parameters from Testng.xml can be suite or test level

Parameter from DataProvider can take Method and ITestContext as the parameter

#### @Parameter

In @Parameter+TestNG.xml parameters can be placed in suite level and test level. IfThe Same parameter name is declared in both places; test level parameter will get preference over suit level parameter.

using @Parameter+TestNG.xml only one value can be set at a time,

TESTNG.xml has below parameters

|  |
| --- |
| <parameter name="username" value="testuser"/>  <parameter name="password" value="testpassword"/> |

Class file

|  |
| --- |
| @Test      @Parameters({ "username", "password" })      public void test(String sUsername, String sPassword) {  } |

Set Optional values for Paramters:

|  |
| --- |
| @Parameters({ "param-one" })  @Test  **public** **void** optionTest(@Optional("param-one") String value) {  System.out.println("This is: " + value);  } |

An optional value for the said parameter is defined using the @Optional annotation against the said parameter.

TestNG passes the optional value to the test method if unable to find a parameter named param-one in the XML file. If found the parameter value in the XML and passes the said value to the test method during execution.

#### **@Data Providers** – Data Driven

@Parameters annotation is easy but to test with multiple sets of data we need to use Data Provider.

To fill thousand's of web forms using our testing framework we need a different methodology which can give us a very large dataset in a single execution flow.

This data driven concept is achieved by @DataProvider annotation in TestNG.

It has only one attribute 'name'. If you do not specify the name attribute then the DataProvider's name will be same as the corresponding method name.

Data provider returns a two-dimensional JAVA object to the test method and the test method, will invoke M times in a M\*N type of object array

|  |
| --- |
| @DataProvider(name = "Authentication")      public static Object[][] credentials() {            return new Object[][] { { "testuser\_1", "Test@123" }, { "testuser\_1", "Test@123" }};      }   @Test(dataProvider = "Authentication")      public void test(String sUsername, String sPassword) {  } |

**Invoke DataProvider from different class**

By default, DataProvider resides in the same class where test method is or its base class. To put it in some other class we need to make data provider method as static and in test method we need to add an attribute dataProviderClass in @Test annotation.

|  |
| --- |
| @Test(dataProvider="SearchProvider",dataProviderClass=DataproviderClass.class) |

Data provider class

|  |
| --- |
| package parameters;  import org.testng.annotations.DataProvider;  public class DataproviderClass {  @DataProvider(name="SearchProvider")  public static Object[][] getDataFromDataprovider(){  return new Object[][] {  { "Guru99", "India" },  { "Krishna", "UK" },  { "Bhupesh", "USA" }  };  }} |

A test method that uses DataProvider will be executed multiple numbers of times based on the data provided by the DataProvider. That means this annotation parametrizes the single test method and executes the test number of times based on the data provided by the DataProvider method.

#### @Factory

Factory can be used when you want to run each test method for n times. we can use the invocationCount, but if You want to login with ten different user credentials and do some operation after login, then you can’t do that with InvocationCount attribute.

The main advantage of @Factory is you can pass the parameters to test classes while you are initializing them, and you can use those parameters across all test methods.

TESTNGFactory Class

|  |
| --- |
| public class TestNGFactory {  @Factory(dataProvider ="GetItem")  public Object[] getTestClasses(String param, String url) {  Object[] tests = new Object[1];  tests[0] = new Test1(url, param);  return tests;  }  @DataProvider(name = "GetItem")  public Object[][] getItem() {  Object list[][] = new Object[][] { {"Param 1","URL 1"}, {"Param 2,"URL 2"},{"Param 3","URL 3"}};    return list;  }  } |

Test1.class

|  |
| --- |
| public class Test1 {  String url = "";  String param = "";  public Test1(String url, String param){  this.url = url;  this.param = param;  }  @Test  public void testMethod1(){  System.out.println("Param = "+param+", url = "+url);  }  @Test(dependsOnMethods = { "testMethod1" })  public void testMethod2(){  System.out.println("Param = "+param+", url = "+url);  }  } |

In the TestNG xml provide TESTNGfactory Class.

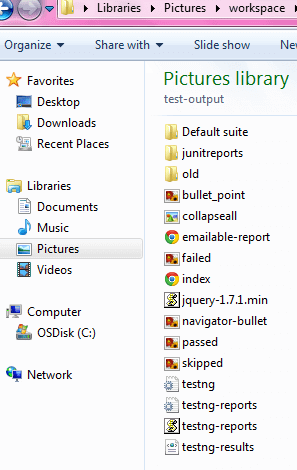
By using depends on, for first set of data testMethod1 is executed and then testMethod2. And for second set of data again testMethod1 is executed and then testMethod2 etc.

If depends on is not used, testMethod1 is executed first for all set of testdata and then second method for all set of test data etc..

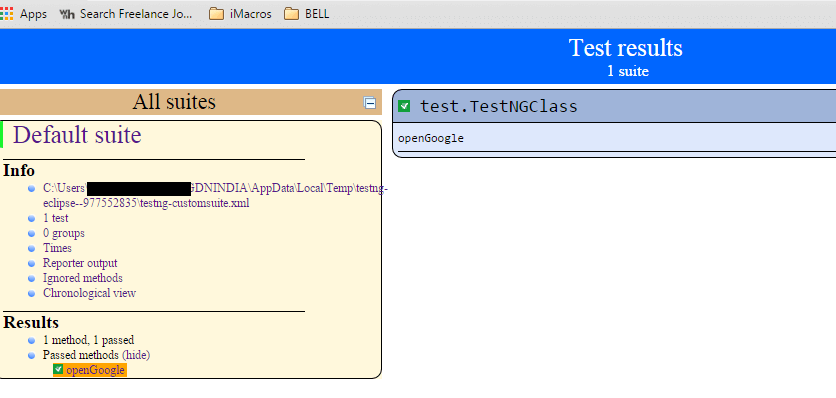
### Analyse the Result of TestNG

Selenium along with TestNG, allows the user to generate reports. Generating a Report for test automation execution is an essential part of the automation.

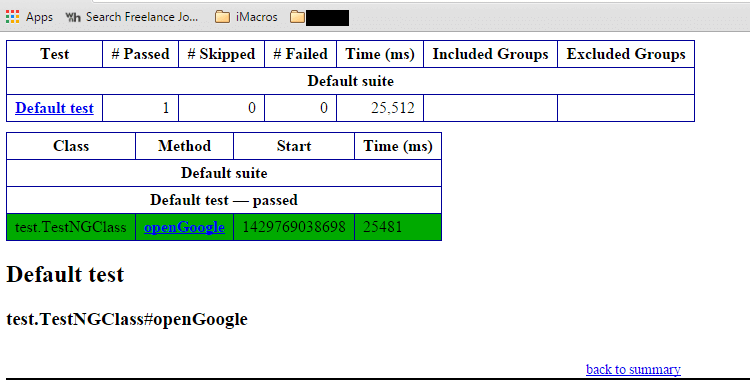
After Execeution Right Click on the project and Select Refresh option.  Now on you could see a test-output folder in the Project folder. After execution TestNG by default create this folder and updated if the folder already exists. Open the test-output folder with the system(not with eclipse). TestNG test-output folder contains below items



Open the index, which contains report and what test case it is, and passed tests and failed tests, time details.



Open emailable report, which we can email because it contains clear information about when the test started and what test it is and what class it is



### RE-RUN Failed Tests

You may have more than 200 test cases on your regression test suite; sometimes, a few test cases may fail. Below reasons may cause these failures.

1. Element not present

2. Element is not visible

3. Element not clickable

4. Browser slowness/responsiveness

Every time tests fail in a suite, TestNG creates a file called testng-failed.xml in the test-output directory.  
the testng-failed.xml file contains all the information to rerun only these methods that failed, allowing you to quickly reproduce the failures without having to run the entirety of your tests.

#### Retry Analyzer

When a test fails, the retry() method of @Test gets called, and the retry() method will result in either true or false.

If we have a test method that is annotated with retryAnalyzer along with @Test, so based on the result from the retry() method @Test method decides whether it should try or not.

If retry() results in true then @Test method will try to re-execute the test method. if results false then test method will not be re-executed

* Create a class called RetryFailedTest which should implement IRetryAnalyzer
* Initialize a two variables maxTries=5, and currentTry=0
* Override the method called retry() and write condition where it should enter to if block only if maxTries > currentTry
* Inside if block returns true; after if block return false.

|  |
| --- |
| Public class TestJenkins {  @Test(**retryAnalyzer = RetryFailedTest.class)**  public static void main() {  fail();  }  }  import org.testng.IRetryAnalyzer;  import org.testng.ITestResult;  public class **RetryFailedTest implements IRetryAnalyzer{**  int currentTry = 0;  int maxTries = 5;  @Override  public boolean retry(ITestResult result) {  if(currentTry < maxTries)  {  System.out.println(result.getName()+ " test will be re-executed");  // increase the currentTry  currentTry++;  return true;  }  System.out.println(result.getName()+ " test will not be re-executed");  return false;  }  } |

### TESTNG Interfaces

#### ****ITestContext**** This class defines a test context which contains all the information for a given test run. An instance of this context is passed to the test listeners so they can query information about their environment.

#### ITestContext is created once and remains active for the duration of your test run, this is the perfect way to implement object sharing in your test suite

Let’s Go through one scenario and try to understand how you can use with TestNG:

Suppose in a Class we have 10 Test Cases or @Test method, which are covering the end to end scenario. Now all 10 Test cases are sharing some data, for example, Customer\_id, which is unique, and the same value should be used for the end to end scenario.

We Can handle this in 2 ways:

* If all the test cases are present in a class, then we can create a Class level variable and share it. But it requires high maintenance.
* Use ITestContext

You can set the value for ITestContext TestNG Interface by using the setAttribute() method like below:

|  |
| --- |
| @Test  public void test1(ITestContext context)  {  String Customer\_id = "C11012034";  context.setAttribute("CustID", Customer\_id);  } |

Now we need to retrieve the value from the ITestContext by using the getAttribute() method

|  |
| --- |
| @Test  public void test2(ITestContext context)  {  String Customer\_id1 = (String) context.getAttribute("CustID");  System.out.println(Customer\_id1);  } |

# Listeners

Listeners possess the ability to “listen” to a certain event. The practical use of Listeners in Selenium Automation Testing could be to log the order of actions and to take the screenshot as and when an Exception is thrown. There are two types of Selenium Listeners:

* WebDriver Listeners
* TestNG Listeners

### WebDriver Listeners

### TestNG Listeners

One of the important features of the TestNG framework is listeners. It is an interface that listens to predefined events in test scripts and modifies the default behavior of the TestNG tool. Whether you want to customize reports or generate logs for specific tests, TestNG listeners help you to do so. Testers would also want to define prerequisites and clean-up configurations for each test project. Automate the process of setting up the prerequisites before starting your testing process. Performing clean-up operations after the testing finishes with a TestNG listener.

TestNG provides a bunch of listeners as a part of its testing environment. These listeners are as follows:

1. ITestListener
2. IReporter
3. ISuiteListener
4. IInvokedMethod
5. IHookable
6. IConfigurationListener
7. IConfigurableListener
8. IAnnotationTransformer
9. IExecution
10. IMethodInterceptor

popular TestNG listeners include ITestListerner, ISuiteListener, and Ireporter.

#### ****ITestListener****

*ITestListener* is the most used listener in TestNG with [Selenium webdriver](https://www.toolsqa.com/selenium-webdriver/selenium-tutorial/). The *ITestListener* implements since it is an interface, and the class in which we implement the listener overrides the *ITestListener* methods.

onFinish(): Invoked after all the tests have run and all their Configuration methods have been called.

* 1. **onStart():**Invoked after the test class is instantiated and before any configuration method is called.
  2. **onTestFailedButWithinSuccessPercentage(ITestResult result):**Invoked each time a method fails but has been annotated with successPercentage and this failure still keeps it within the success percentage requested.
  3. **onTestFailure(ITestResult result):** Invoked each time a test fails.
  4. **onTestSkipped(ITestResult result):**Invoked each time a test is skipped
  5. **onTestStart(ITestResult result):**Invoked each time before a test will be invoked.
  6. **onTestSuccess(ITestResult result):**Invoked each time a test succeeds.

|  |
| --- |
| public class TestNGListeners implements ITestListener {  public void onTestStart(ITestResult result) {  LogClass.logger.info("Test start: " + testName);  }  public void onTestSuccess(ITestResult result) {  LogClass.logger.info("Test ok: " + testName);  }  public void onTestFailure(ITestResult result) {  String ex = result.getThrowable().getCause().toString();  System.out.println(ex);  result.getThrowable().printStackTrace();  LogClass.logger.fatal("Test failed: " + testName);  }  public void onTestSkipped(ITestResult result) {  LogClass.logger.info("Iteration skipped: " + testName);  }    public void onFinish(ITestContext context) {  LogClass.logger.info("Suite finished: " + testName);  }  } |

IInvokedMethodListener  
The working of this listener is also exactly the same as ISuiteListerner & ITestListerner and the only difference is that it makes the call before and after every Method. It has only two methods in it.

* **afterInvocattion():**Invoke after each method
* **beforeInvocation():** Invoke before each method

#### Implementation

Create a ‘[**New Class’**](http://toolsqa.com/selenium-webdriver/configure-eclipse-with-selenium-webdriver/) file and Implement ISuiteListener, ITestListener and IInvokedMethodListener to this newly created class

|  |
| --- |
| public class Listener implements ITestListener, ISuiteListener, IInvokedMethodListener {  // This belongs to ISuiteListener and will execute before the Suite start  @Override  public void onStart(ISuite arg0) {  Reporter.log("About to begin executing Suite " + arg0.getName(), true);  }    // This belongs to ISuiteListener and will execute, once the Suite is finished  @Override  public void onFinish(ISuite arg0) {  Reporter.log("About to end executing Suite " + arg0.getName(), true);  }    // This belongs to ITestListener and will execute before starting of Test set/batch  public void onStart(ITestContext arg0) {  Reporter.log("About to begin executing Test " + arg0.getName(), true);  }    // This belongs to ITestListener and will execute, once the Test set/batch is finished  public void onFinish(ITestContext arg0) {  Reporter.log("Completed executing test " + arg0.getName(), true);  }    // This belongs to ITestListener and will execute only when the test is pass  public void onTestSuccess(ITestResult arg0) {  printTestResults(arg0);  }    // This belongs to ITestListener and will execute only on the event of fail test  public void onTestFailure(ITestResult arg0) {  printTestResults(arg0);  }    // This belongs to ITestListener and will execute before the main test start (@Test)  public void onTestStart(ITestResult arg0) {  System.out.println("The execution of the main test starts now");  }    // This belongs to ITestListener and will execute only if any of the main test(@Test) get skipped  public void onTestSkipped(ITestResult arg0) {  printTestResults(arg0);  }    public void onTestFailedButWithinSuccessPercentage(ITestResult arg0) {  }    // This is the method which will be executed in case of test pass or fail  // This will provide the information on the test  private void printTestResults(ITestResult result) {  Reporter.log("Test Method resides in " + result.getTestClass().getName(), true);  if (result.getParameters().length != 0) {  String params = null;  for (Object parameter : result.getParameters()) {  params += parameter.toString() + ",";  }  Reporter.log("Test Method had the following parameters : " + params, true);  }  String status = null;  switch (result.getStatus()) {  case ITestResult.SUCCESS:  status = "Pass";  break;    case ITestResult.FAILURE:  status = "Failed";  break;    case ITestResult.SKIP:  status = "Skipped";  }    Reporter.log("Test Status: " + status, true);  }    // This belongs to IInvokedMethodListener and will execute before every method including @Before @After @Test  public void beforeInvocation(IInvokedMethod arg0, ITestResult arg1) {  String textMsg = "About to begin executing following method : " + returnMethodName(arg0.getTestMethod());  Reporter.log(textMsg, true);  }    // This belongs to IInvokedMethodListener and will execute after every method including @Before @After @Test  public void afterInvocation(IInvokedMethod arg0, ITestResult arg1) {  String textMsg = "Completed executing following method : " + returnMethodName(arg0.getTestMethod());  Reporter.log(textMsg, true);  }    // This will return method names to the calling function  private String returnMethodName(ITestNGMethod method) {  return method.getRealClass().getSimpleName() + "." + method.getMethodName();  }  } |

There are essentially two ways of adding up a listener to a particular class.

* 1. Implement TestNG Listener to your test class

|  |
| --- |
| @Listeners(PackageName.ListenerClassName)  // For e.g. @Listeners(utility.Listener.class)  public class TestListener {  @Test  public void method() {  }  } |

* 1. Listener tag in TestNG xml:

Although approach 1 is more than enough to get you started, it’s not an “elegant” way of using Listeners, because you are forced to add this @Listeners section to each of your classes, which you perhaps won’t want. So what you do is, you create a TestNG Suite xml and then add up the listeners section to this suite xml file. That way, all of your tests would essentially leverage the listener that you wrote.

|  |
| --- |
| <suite name="Suite-Listeners" parallel="none">  <listeners>  <listener class-name="utility.Listener"></listener>  </listeners>    <test name="Batch-Listeners">  <classes>  <class name="automationFramework.TestListener" />  </classes>  </test>    </suite> |

#### ****ISuiteListener****

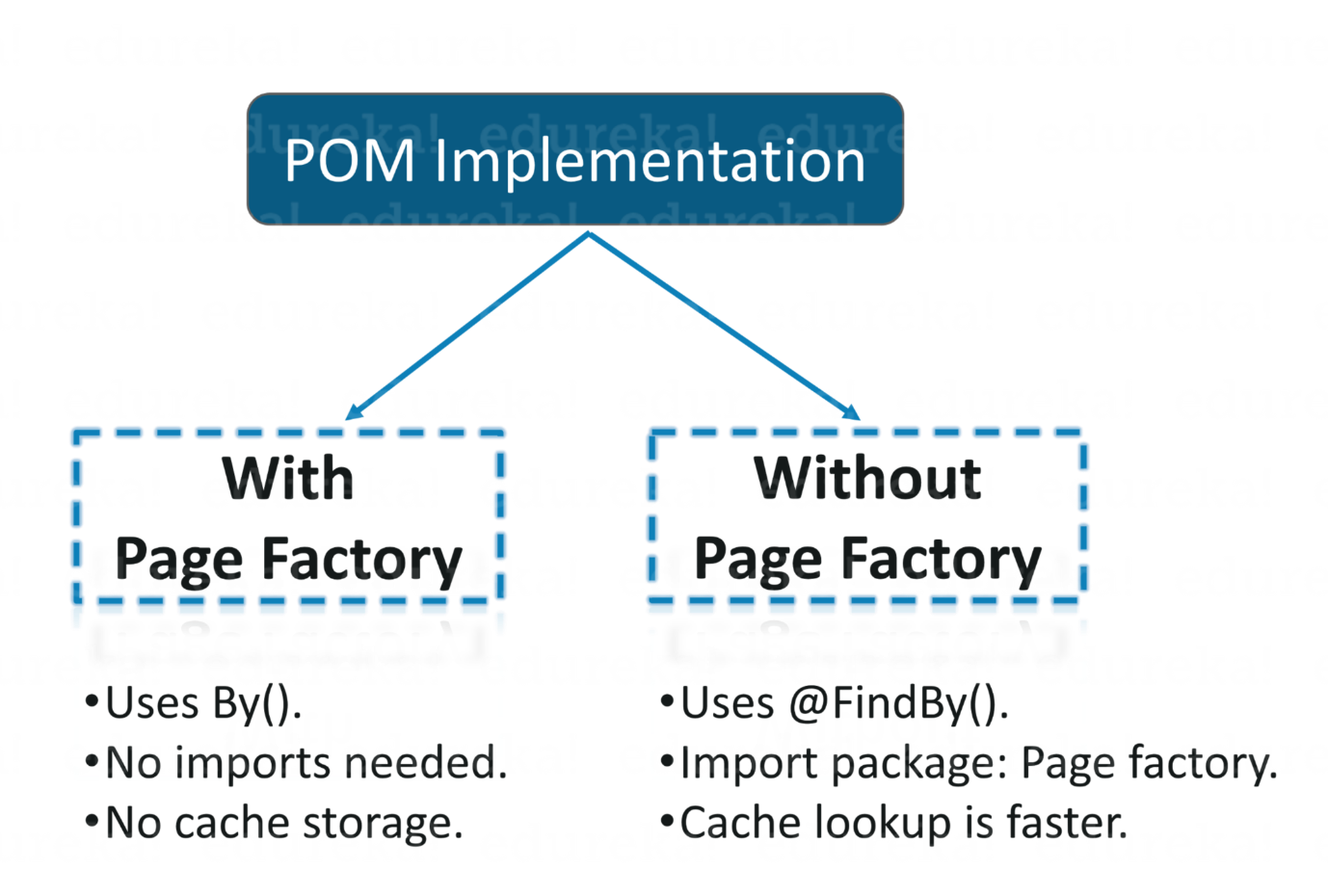
It has two method in it **onStart()** & **onFinish()**. Whenever a class implements this listener, TestNG guarantees the end-user that it will invoke the methods onStart() and onFinish() before and after running a TestNG Suite. So before TestNG picks up your suite for execution, it first makes a call to onStart() method and runs whatever has been scripted in this method. In a similar way, it again makes a call to onFinish() method after a suite has been run.

# Page Object Model

Creating Selenium test cases can result in an unmaintainable project. One of the reasons is that too many duplicated code is used. Duplicated code could be caused by duplicated functionality and this will result in duplicated usage of locators. The disadvantage of duplicated code is that the project is less maintainable. If some locator will change, you have to walk through the whole test code to adjust locators where necessary. By using the page object model we can make non-brittle test code and reduce or eliminate duplicate test code. Beside of that it improves the readability and allows us to create interactive documentation. Last but not least, we can create tests with less keystroke. An implementation of the page object model can be achieved by separating the abstraction of the test object and the test scripts

**Advantages of using Page Object Pattern:**

* Easy to Maintain
* Easy Readability of scripts
* Reduce or Eliminate duplicacy
* Re-usability of code
* Reliability



### ****Page Object Model without Page Factory****

* 1. Create a ‘[New Package‘](http://toolsqa.com/selenium-webdriver/configure-eclipse-with-selenium-webdriver/) file and name it as ‘pageObjects’,
  2. Create a ‘[New Class‘](http://toolsqa.com/selenium-webdriver/configure-eclipse-with-selenium-webdriver/) file and refer the name to the actual page
  3. Now create a Static Method for each Element (Object) in the Home Page. Each method will have an Argument (driver) and a Return value (element).

Driver is being passed as an Argument so that Selenium is able to locate the element on the browser (driver).

Element is returned, so that an Action can be performed on it.

Method is declared as Public Static, so that it can be called in any other method without instantiate the class.

|  |
| --- |
| public class Home\_Page {  private static WebElement element = null;  public static WebElement lnk\_MyAccount(WebDriver driver){  element = driver.findElement(By.id("account"));  return element;  } |

Test case class

|  |
| --- |
| Home\_Page.lnk\_MyAccount(driver).click(); |

### ****Page Object Model with Page Factory****

**PageFactory Class** is an extension to the Page Object design pattern.

Page Factory is a way to initialize the web elements you want to interact with within the page object when you create an instance of it.

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

Page Class

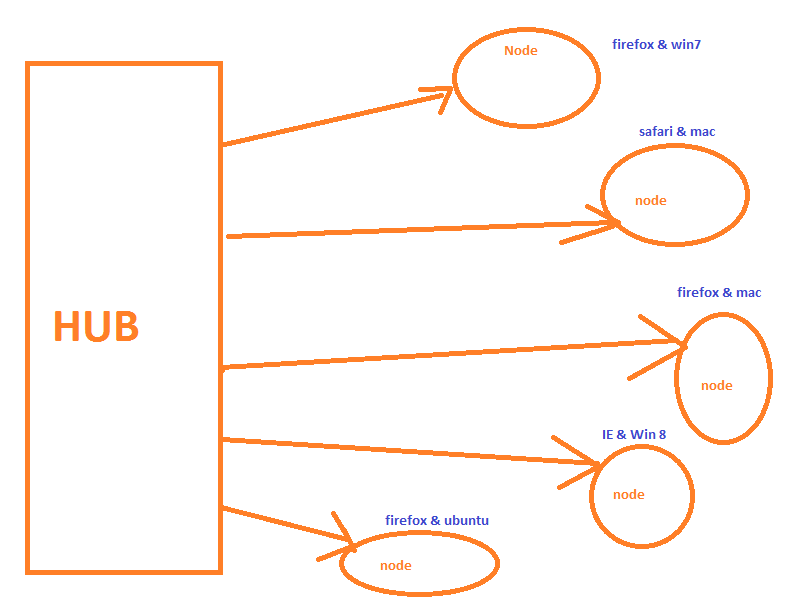
|  |
| --- |
| public class LogIn\_PG\_POF {    static WebDriver driver;    @FindBy(how = How.XPATH, using = ".//\*[@id='account']/a")  static WebElement lnk\_MyAccount;    @FindBy(name="btnLogin")  WebElement login;  public LogIn\_PG\_POF(WebDriver driver){  this.driver = driver;  }  public void clickLogin(){  login.click();  } |

Test case

|  |
| --- |
| public class TestCase\_POF {  static WebDriver driver;    @TestCase\_POF  public static void testMethod(){  driver = new FirefoxDriver();  driver.get("http://url");    LogIn\_PG\_POF LoginPage = PageFactory.initElements(driver, LogIn\_PG\_POF.class);  LoginPage.clickLogin();  } |

# Selenium Grid

It lets you distribute test execution across several machines. You can connect to it with Selenium Remote by specifying the browser, browser version, and operating system you want. To run tests in parallel in multiple browsers, multiple versions of a browser, and browsers running on different operating systems.



**Set Up**

1.Goto <https://seleniumhq.org/download/>

2. Download the Selenium server, yes, we use it for both purposes for writing selenium script and also a server.

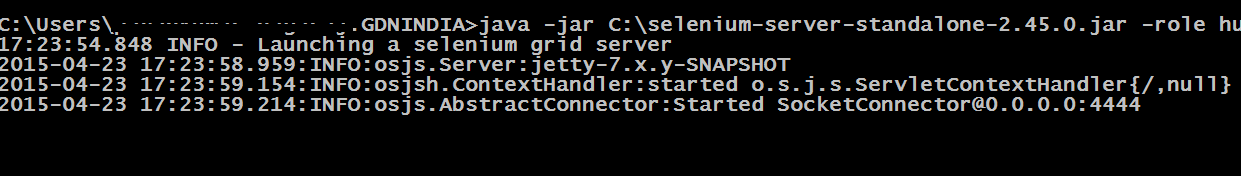
3. Save it in someplace in your system.

4. Start the Hub  
  
5. Start the node  
  
6.Execute the script from Hub machine  
  
7. Check results in Hub machine  
  
There is no such  thing called as Grid installation & Grid installation means installing HUB and NODE.

**HUB:**

Hub is a system where all the framework and all details available, follow below steps to install  
  
1. Go to the CMD  
2. Write the below command in the CMD and hit enter button

|  |
| --- |
| java -jar c:LocationOfServerSeleniumServerName.jar -role hub |

3. When Grid starts running, you can see like below  
  
  
Once you see this then hub has started  
4. Grid always starts in the port: 4444 by default  
5. You can also save this command in the .bat file (like running [TestNG script](https://chercher.tech/java/install-associate-testng-junit-drawbacks-selenium-webdriver)from a batch file)  
6. If port 4444 is occupied by some other process means you can give your own node like 8888,8899,1234...

|  |
| --- |
| java -jar c:LocationOfServerSeleniumServerName.jar -role hub -port 8888 |

**Node**

Node is system execution happens, and execution is controlled by the hub, follow below steps to install selenium node.

1. Go to the CMD

2. Write the below command in the CMD and hit enter button

|  |
| --- |
| java -jar c:LocationOfServerSeleniumServerName.jar -role node -hub <https://ipaddress:4444/grid/register> |

**Execution**

Node is a system where the execution happens, and the hub controls the execution.  
  
Run Grid:  
  
1.Goto program which you want to run in the grid  
2. In order execute a script in Grid we need to create a RemoteWebdriver Class

|  |
| --- |
| WebDriver driver=new RemoteWebDriver(hubUrl, dCap); |

3. We should supply the URL of the hub and the desired capabilities(browser and os) of the remote computer to the constructor of RemoteWebdriver class while creating an object

|  |
| --- |
| //hub URL  URL hubUrl=new URL("http://localhost:4444/grid/register");  //creating instance to Desired Capabilities class  DesiredCapabilities dCap=new DesiredCapabilities();  //set the browser name  dCap.setBrowserName("firefox");  //set the os  dCap.setPlatform(Platform.WINDOWS); |

# Headless browser Testing

# Exceptions in Selenium

An exception is known as an unusual or unprecedented event that occurs during the execution of a software program or application. It is a runtime error of an unexpected result or event which influence and disrupt usual program flow. An exception is also considered as a fault.

Selenium exceptions are divided into two types including Checked Exceptions and Unchecked Exceptions.

1. Checked Exceptions

Checked Exceptions are handled during the process of writing codes. These exceptions are handled before compiling the code, therefore, such exceptions are examined at the compile time.

2. Unchecked Exceptions

These exceptions are thrown at runtime. Unchecked exceptions are more catastrophic than the compile-time exception as it causes problems while running Automation pack in headless.

**The complete list of exceptions in Selenium**

* + 1. **NoSuchElementException**

The exception occurs when WebDriver is unable to find and locate elements.

**Reasons**

1) The page is still being rendered and you've already finished your element search and obtain no element exception.

2) AJAX has not returned yet and you've already obtain NoSuchElementException

3)The element is really not on the page. This happens when tester writes incorrect element locator in the findElement(By, by) method.

**Avoiding-And-Handling:** Try giving a wait command.

|  |
| --- |
| try {  WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions.presenceOfElementLocated(By.id("submit")));  driver.findElement(By.id("submit")).click();  } catch (TimeOutException e) {  System.out.println(“WebDriver couldn’t locate the element”);  } |

* + 1. **NoSuchWindowException**

This Exception occurs if the window target to be switch does not exist.

**Reasons**

1) One of the windows that previously existed is no longer exists; so, you can not switch to that window.

2) you may trying to switch to window before handling all the opened windows using driver.getWindowHandles() method.

**Avoiding-And-Handling:**

1. we need to make sure that you are not closing a window without refreshing the available list of windows using this method.
2. we should know which window we are trying to switch as getWindowHandles() holds all the windows which are opened. Once you have all the windows with you then you can loop through all the windows which are there in the list and we can switch to the desired window.Once you done with your work, then you need to come to the parent or default windows then only your execution will be smooth.
   * 1. **NoSuchFrameException**

This Exception occurs if the frame target to be switched to does not exist.

**Reasons**

1. Let’s assume one scenario where you have a webpage with more than one frame. We will call them frame A and frame B. We will assume that we first switched to frame B to check/verify something in the frame B. If we then try to find frame A, we will get NoSuchFrameException. This is because frame A does not exist in the context of frame B.
2. Searching for a fram and frame is not loaded yet

**Avoiding-And-Handling:**

1. To overcome this issue is to always go back to the parent frame before trying to switch to another frame
2. Try to give a wait command.

|  |
| --- |
| try {  WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions.frameToBeAvaliableAndSwitchToIt(frame\_11));  driver.switchTo().frame("frame\_11");  } catch (TimeOutException e) {  System.out.println(“WebDriver couldn’t locate the frame”);  } |

* + 1. **NoAlertPresentException**

This Exception occurs when you switch to no presented alert.

**Reasons**

1) Selenium is too quick and tries to accept an alert that has not yet been opened by the browser

**Avoiding-And-Handling:**

1. Try to give a wait command.

|  |
| --- |
| try {  WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions.alertIsPresent());  driver.switchTo().alert().accept();  } catch (TimeOutException e)  System.out.println(“WebDriver couldn’t locate the Alert”);  } |

* + 1. **InvalidSelectorException**

When a selector is incorrect or syntactically invalid.

**Reasons**

This exception occurs commonly when XPATH locator is used and XPATH syntax is incorrect.

**Avoiding-And-Handling:**

To avoid this, we should check the locator used because the locator is likely incorrect or the syntax is wrong. Using Firebug to find xpath can reduce this exception.

* + 1. **ElementNotVisibleException**

This type of Selenium exception occurs when an existing element in DOM has a feature set as hidden.

**Reasons**

1. If the element is hidden or page is not completely loaded

**Avoiding-And-Handling:**

use wait for the element to get completely.

|  |
| --- |
| try {  WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions.visibilityOfElementLocated(By.id(”submit”));  try {  driver.findElement(By.id("submit")).click();  } catch (WebDriverException e) {  System.out.println(“Exceptional case”);  }  } catch (TimeOutException e)  System.out.println(“WebDriver couldn’t find this element visible”);  } |

* + 1. **StaleElementReferenceException**

This exception says that a web element is no longer present in the web page.

**Reasons**

StaleElementReferenceException is thrown when an object for a particular web element was created in the program without any problem and however; this element is no longer present in the window. This can happen if there was a

* Navigation to another page
* DOM has refreshed
* A frame or window switch

**Avoiding-And-Handling:**

To avoid issues due to DOM refresh, we can use Dynamic Xpath

|  |
| --- |
| try {  driver.findElement(By.xpath(“//\*[contains(@id,firstname’)]”)).sendKeys(“Aaron”);  } catch (StaleElementReferenceException e) |

* + 1. **ElementNotSelectableException**

This Exception occurs if the web element is present in the web page but cannot be selected.

**Reasons**

Exception is thrown even if the element becomes enabled after a while.

**Avoiding-And-Handling:**

We can add a wait command to wait until the element becomes clickable. If there is still an exception, it is caught.

|  |
| --- |
| try {  WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions. elementToBeClickable(By.id(”swift”));  try {  Select dropdown = new Select(driver.findElement(By.id("swift")));  } catch (WebDriverException e) {  System.out.println(“Exceptional case”);  }  } catch (TimeOutException e)  System.out.println(“WebDriver found that this element was not selectable.”); |

* + 1. **TimeoutException**

This Exception occurs when there is not enough time for a command to be completed.

**Reasons**

If WebDriver tries to find an element in the webpage before the page completely loads, then exception ElementNotVisibleException is thrown. To avoid this exception, waits commands are added.However, if the components don’t load even after the wait, the exception org.openqa.selenium.TimeoutException will be thrown.

**Avoiding-And-Handling:**

To avoid this, we can manually check the average time for a page to load and adjust the wait (Or) we can add explicit wait using JavaScript executor until the page is loaded.

|  |
| --- |
| WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30));  wait.until(webDriver -> ((JavascriptExecutor)webDriver).executeScript("return document.readyState").equals("complete")); |

* + 1. **NoSuchSessionException**

This exception is thrown when a method is called after quitting the browser by WebDriver.quit().

**Reasons**

This can lead to issues when driver instance is null and upcoming test cases try to use it without initializing.

**Avoiding-And-Handling:**

Use TESTNG annotation to invoke and closing the browser

|  |
| --- |
| @BeforeSuite  public void setUp() throws MalformedURLException {  WebDriver driver = new FirefoxDriver();  }  @AfterSuite  public void testDown() {  driver.quit();  } |

# Cookies in Selenium

A HTTP cookie is comprised of information about the user and their preferences. It stores information using a key-value pair. It is a small piece of data sent from Web Application and stored in Web Browser, while the user is browsing that website.

|  |
| --- |
| driver.manage().getCookies(); // Return The List of all Cookies  driver.manage().getCookieNamed(arg0); //Return specific cookie according to name  driver.manage().addCookie(arg0); //Create and add the cookie  driver.manage().deleteCookie(arg0); // Delete specific cookie  driver.manage().deleteCookieNamed(arg0); // Delete specific cookie according Name  driver.manage().deleteAllCookies(); // Delete all cookies |

Each cookie is associated with a name, value, domain, path, expiry, and the status of whether it is secure or not. In order to validate a client, a server parses all of these values in a cookie.

When Testing a web application using selenium web driver, you may need to create, update or delete a cookie.

For example, when automating Online Shopping Application, you many need to automate test scenarios like place order, View Cart, Payment Information, order confirmation, etc.

If cookies are not stored, you will need to perform login action every time before you execute above listed test scenarios. This will increase your coding effort and execution time.

The solution is to store cookies in a File. Later, retrieve the values of cookie from this file and add to it your current browser session. As a result, you can skip the login steps in every Test Case because your driver session has this information in it.

The application server now treats your browser session as authenticated and directly takes you to your requested URL.

**Step 1) Storing cookie information.**

|  |
| --- |
| // create file named Cookies to store Login Information  File file = new File("Cookies.data"); try  {  // Delete old file if exists  file.delete();  file.createNewFile();  FileWriter fileWrite = new FileWriter(file); BufferedWriter Bwrite = new BufferedWriter(fileWrite);  // loop for getting the cookie information  for(Cookie ck : driver.manage().getCookies()) {  Bwrite.write((ck.getName()+";"+ck.getValue()+";"+ck.getDomain()+";"+ck.getPath()+";"+ck.getExpiry()+";"+ck.isSecure()));  Bwrite.newLine();  }  Bwrite.close();  fileWrite.close();  }  catch(Exception ex)  {  ex.printStackTrace();  }  } |

**Step 2) Using stored cookie to login into the application.**

Now, we will access the cookie generated in step 1 and use the cookie generated to authenticate our session in the application

|  |
| --- |
| try{    File file = new File("Cookies.data");  FileReader fileReader = new FileReader(file);  BufferedReader Buffreader = new BufferedReader(fileReader);  String strline;  while((strline=Buffreader.readLine())!=null){  StringTokenizer token = new StringTokenizer(strline,";");  while(token.hasMoreTokens()){  String name = token.nextToken();  String value = token.nextToken();  String domain = token.nextToken();  String path = token.nextToken();  Date expiry = null;    String val;  if(!(val=token.nextToken()).equals("null"))  {  expiry = new Date(val);  }  Boolean isSecure = new Boolean(token.nextToken()).  booleanValue();  Cookie ck = new Cookie(name,value,domain,path,expiry,isSecure);  System.out.println(ck);  driver.manage().addCookie(ck); // This will add the stored cookie to your current session  }  }  }catch(Exception ex){  ex.printStackTrace();  }  } |

# Extent Reports

ExtentReports is an open-source reporting library used in selenium test automation. Extent reports become the first choice of Selenium Automation Testers, even though Selenium comes with inbuilt reports using frameworks like JUnit and TestNG. With extent reports, you can offer a more extensive and insightful perspective on the execution of your automation scripts.

**Advantages of Extent Reports:**

Some of the advantages of Extent Reports are as follows

* Extent reports are more customizable than others
* Extent API can produce more interactive reports, a dashboard view, graphical view, capture screenshots at every test step, and emailable reports
* It can be easily integrated with frameworks like JUnit, NUnit, & TestNG
* It displays the time taken for test case execution

**Pre-requisites to Generate Extent Reports:**

1. Java should be installed
2. TestNG should be installed
3. Extent Report Jars
4. extent-config.xml (Not Mandatory) – It allows to Customize HTML Report

The package com.relevantcodes was used upto version 2 and com.aventstack is the package for version 3+.

ExtentHTMLReporter is deprecated from 4.1.3 and we need to use ExtentSparkReporter

|  |
| --- |
|  |

# MAVEN

A Selenium project for test automation requires you to have all the dependencies associated with it. These dependencies can be downloaded and upgraded manually. However, as the project gets bigger, it can be quite challenging to manage these dependencies.

Maven is a build automation tool from Apache Software Foundation and is commonly used to handle [Java projects](https://www.simplilearn.com/tutorials/java-tutorial/java-projects-for-beginner). It is widely used to manage project dependencies and the whole lifecycle of any project.

**Install Maven in Windows**

There are two ways to install Maven in Windows - through Command-Line or with Eclipse IDE.

1. **Eclipse**

Step 1:  Go to the top menu in Eclipse. Click on ‘Help’ and select ‘Install New Software’.

Step 2:  A new window will open. Click on the Add button.

Step 3: Type ‘Maven’ in the name text box. Type “<http://download.eclipse.org/technology/m2e/releases/>” in the location text box. This contains the link to the location from where Maven can be downloaded.

Step 4:  Click on the check-box and then click on the Next button.

Step 5:  Keep the settings as default and then click on the Next button.

Step 6:  Accept the Terms and Conditions and click on Finish.

Step 7:  You would need to restart Eclipse once the installation is finished.

1. **Command Line**

Step 1: Download and Install Java

Step 2: Set Up Java Environment Variable (JAVA\_HOME)

Step 3: Download Maven zip file and extract it to some location and Set up MAVEN Environment Variable (MAVEN\_HOME)

Step 4: Update the Path Variable with Maven’s installation ‘bin’ directory to run Maven from the command line.

Step 5: Testing the Maven Installation - Open your command prompt and type in ‘mvn -version’ and then hit Enter.

**Create Maven Project**

You can create a Maven project in two ways - through the Eclipse IDE or command line.

1. **Eclipse**

Step 1: Go to Eclipse IDE and create a new project.

Step 2: Select Maven Project.

Step 3: You can choose to create a simple project or just skip this option. For now, we’ll use a simple project.

Step 4: Fill in all the appropriate information for your project. If your project does not have any parent dependencies, you can skip the project dependencies. Click on Finish. Your Maven project is now created.

Step 5: You can open the pom.xml file to see the structure set up by Maven.

1. **Command Line**

Open the Command Prompt and go to the folder where you want to set up your project. Now, type the following command and click Enter:

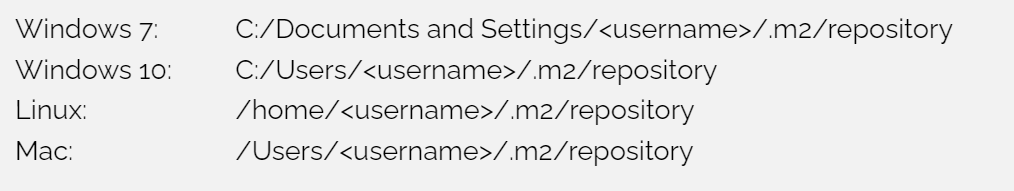
|  |
| --- |
| mvn archetype:generate -DgroupId=demoProject -DartifactId=DemoMavenProject -DarchetypeArtifactId=Maven-archetype-quickstart -DinteractiveMode=false |

Your Maven project is now created.

### MAVEN Repository

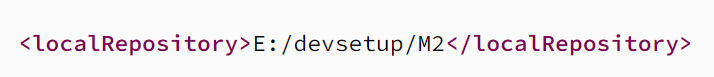
Maven saves all the dependencies of projects in the *.m2* folder. Maven downloads all dependencies under the *repository* folder. Therefore, the download into our local repository is necessary to access all needed code at runtime. We know that Maven works based on the *pom.xml* file configurations. When Maven executes this *pom.xml* file, the dependencies will be downloaded from a central Maven repository and put into your local Maven repository. If we already have the dependencies in our local repository, Maven will not download them.

In different operating systems, these paths are resolved to –



To change the maven repository path

* Navigate to path **{M2\_HOME}\conf\** where M2\_HOME is maven installation folder.
* Open file [settings.xml](https://howtodoinjava.com/maven/maven-settings-file/) in edit mode in some text editor.
* Fine the tag **<localRepository>**
* Update the desired path in value of this tag. Save the file.

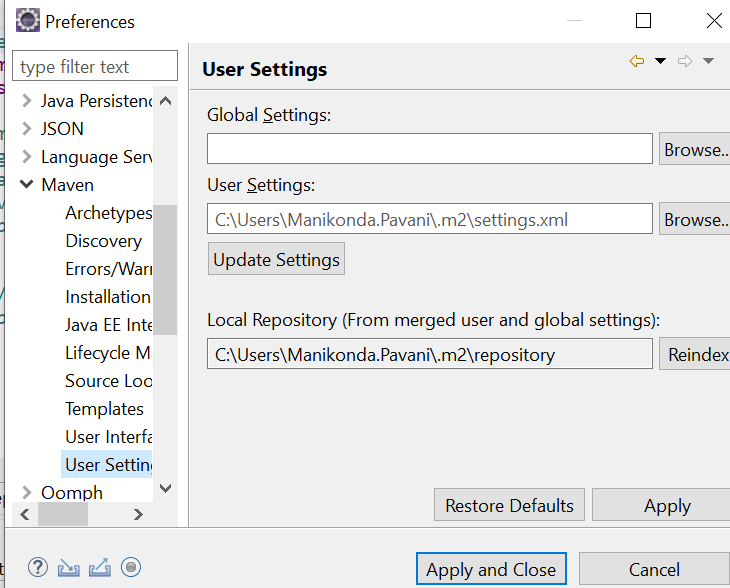


Note that if there are existing jar files and pom files stored in the previous local repository location, they will not be moved. We need to move them or remove them manually.

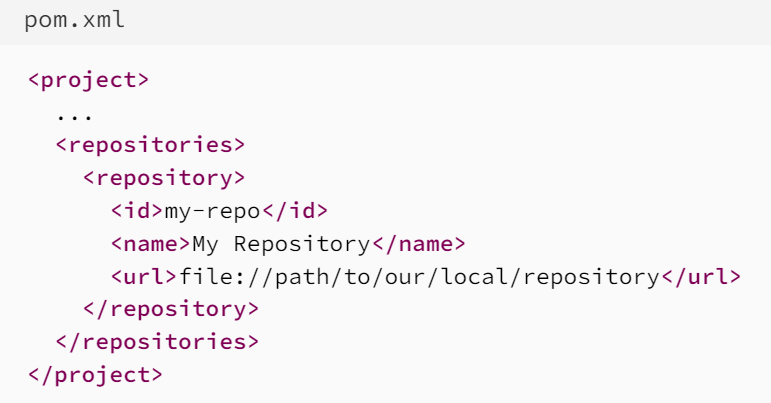
**In Eclipse:** These days, IDEs also come with inbuilt Maven support and we only need to [install Maven](https://howtodoinjava.com/maven/how-to-install-maven-on-windows/) if we are making the project builds from Console.Each IDE has a separate process to change the local repository path and you can read its official documentation.

For example in Eclipse and STS (Spring Tools Suite), we can change the local repository path in the following location: Windows -> Pfreferences -> Maven -> User Settings

Here you can give the path of settings.xml file where we have configured the folder location of the local repository.



**From pom.xml:** Maven allows configuring the repository locations in the pom.xml file also. We need to use the file:// protocol to specify the location.

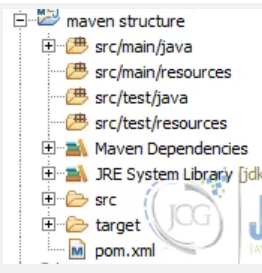


### Proxy Settings

To configure you proxy, add a file named .m2/settings.xml to your user home with the proxy settings, as the following:



**MAVEN Project Structure**



**src/main/java** Inside this folder you can put all the application source files. Classes and packages for the main (real) artifact should be put in this folder. All the content inside of this directory will be put in the classpath of the generated artifact. If the artifact is a jar file, all the classes and packages will be in the root folder of the generated jar, so it will be available by default on the runtime classpath.

If the artifact is a war, all the classes and packages will be put inside the WEB-INF/classes directory, so it will be available on the runtime classpath by default.

When the project is build or packaged all those classes and packages will be put in the target folder.

**src/test/java** Inside this folder you can put all the application test source files. Classes and packages for the test artifact should be put in this folder.

All the content inside of this directory will NOT be put in the classpath of the generated artifact.

When the project is build or packaged all those classes and packages will be put in the target folder.

When you run your test you must be awared that maven surefire plugin will run the classes from the target directory.

**src/main/resources** Inside this folder you can put all the application resource files. Resources for the main (real) artifact should be put in this folder.

All the content inside of this directory will be put in the classpath of the generated artifact. If the artifact is a jar file, all the resources will be in the root folder of the generated jar, so it will be available by default on the runtime classpath.

If the artifact is a war, all resources will be put inside the WEB-INF/classes directory, so it will be available on the runtime classpath by default.

When the project is build or packaged all those resources will be put in the target folder.

**src/test/resources** Inside this folder you can put all the application test resource files. Resources for the test artifact should be put in this folder.

All the content inside of this directory will NOT be put in the classpath of the generated artifact.

When the project is build or packaged all those test resources will be put in the target folder.

When you run your test you must be awared that maven surefire plugin will use resources from the target directory.

**Pom.xml** The pom (Project Object Model) file is a maven special file that describe how the project have to be build and from when maven should download artifacts/dependencies, what are those dependecies and so many more things.

This file is placed on the root project folder.

**Maven Dependency for Automation Project**

Maven allows you to automatically download all the required libraries from its central repository. Add dependencies in pom.xml

Maven central repository is the default location '*http://mvnrepository.com/*' for Maven to download all the project dependency libraries. For any library required in the project, Maven first looks into the .m2 folder of Local Repository, if it does not find the required library then it looks in Central Repository and downloads the library into the local repository.

### POM.xml

POM is a Project Object Model XML file that contains information about the project and configuration details used by Maven to build the project. By default, the POM is generated in the project folder.

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>SeleniumJavaFramework</groupId>  <artifactId>SeleniumJavaFramework</artifactId>  <version>0.0.1-SNAPSHOT</version>  <properties>  <maven.compiler.source>1.8</maven.compiler.source>  <maven.compiler.target>1.8</maven.compiler.target>  </properties>  <build>  <plugins>  <plugin>  <artifactId>maven-assembly-plugin</artifactId>  <version>3.2.0</version>  <configuration>  <descriptorRefs>  <descriptorRef>jar-with-dependencies</descriptorRef>  </descriptorRefs>  </configuration>  <executions>  <execution>  <id>make-assembly</id>  <phase>package</phase>  <goals>  <goal>single</goal>  </goals>  </execution>  </executions>  </plugin>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-surefire-plugin</artifactId>  <version>3.0.0-M5</version>  <configuration>  <suiteXmlFiles>  <suiteXmlFile>Testng.xml</suiteXmlFile>  </suiteXmlFiles>  </configuration>  </plugin>  </plugins>  </build>  <dependencies>  <!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->  <dependency>  <groupId>org.seleniumhq.selenium</groupId>  <artifactId>selenium-java</artifactId>  <version>4.1.1</version>  </dependency>  </dependencies>  </project> |

| **Name** | **Description** |
| --- | --- |
| groupId | Defines a unique base name of the organization or group that created the project. This is normally a reverse domain name or the name of an open source project. For the generation of new projects, the groupId also defines the package of the main class. |
| artifactId | Defines the unique name of the project within the groupId. If you generate a new project via Maven this is also used as root folder for the project. |
| version | This defines the version of the project. If a new version of the project is build, this version should change so that consumers can see that a different version is used. |
| packaging | Defines the packaging method. This could be e.g. a jar, war or ear file. If the packaging type is pom, Maven does not create anything for this project, it is just meta-data. |

**Maven Dependency:** All the external libraries that are used in a project are called dependencies. Maven has an excellent feature that automatically downloads required libraries from its central repository to local Maven repository, which makes it easy as you don’t have to store them locally.

**Surefire Plugin**: We can run the tests of a project using the surefire plugin  and can generate the reports using HTML format. The surefire plugin can work with the test frameworks JUnit and TestNG. As TestNG has the capability to select and execute test scripts or suites, we are integrating this with Maven using the Maven Surefire plugin.

We can have any number of test suites in the <suiteXmlFile> tag. So that the Scripts we have in each suite will be executed. After execution test suite Report can be seen in the target/surefire-reports folder of the Project Explorer window.

### Maven Lifecycle

The Maven build follows a specific lifecycle to deploy and distribute the target project.

There are three built-in lifecycles:

**default (cannot be executed):** the main lifecycle, as it's responsible for project deployment

**clean**: to clean the project and remove all files generated by the previous build

**site**: to create the project's site documentation

You can execute a full life cycle via command like clean or site. You can also execute build phases like verify or single goals like dependency:copy-dependencies.

**MAVEN PHASE & GOAL**

Each lifecycle consists of a sequence of phases. The default build lifecycle consists of 23 phases, as it's the main build lifecycle.

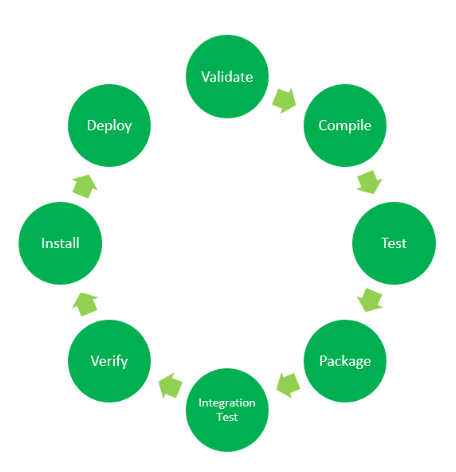
Here are some of the most important phases in the default build lifecycle.

Phases are executed in a specific order. This means that if we run a specific phase using the command:

mvn <PHASE>

It won't only execute the specified phase, but all the preceding phases as well.

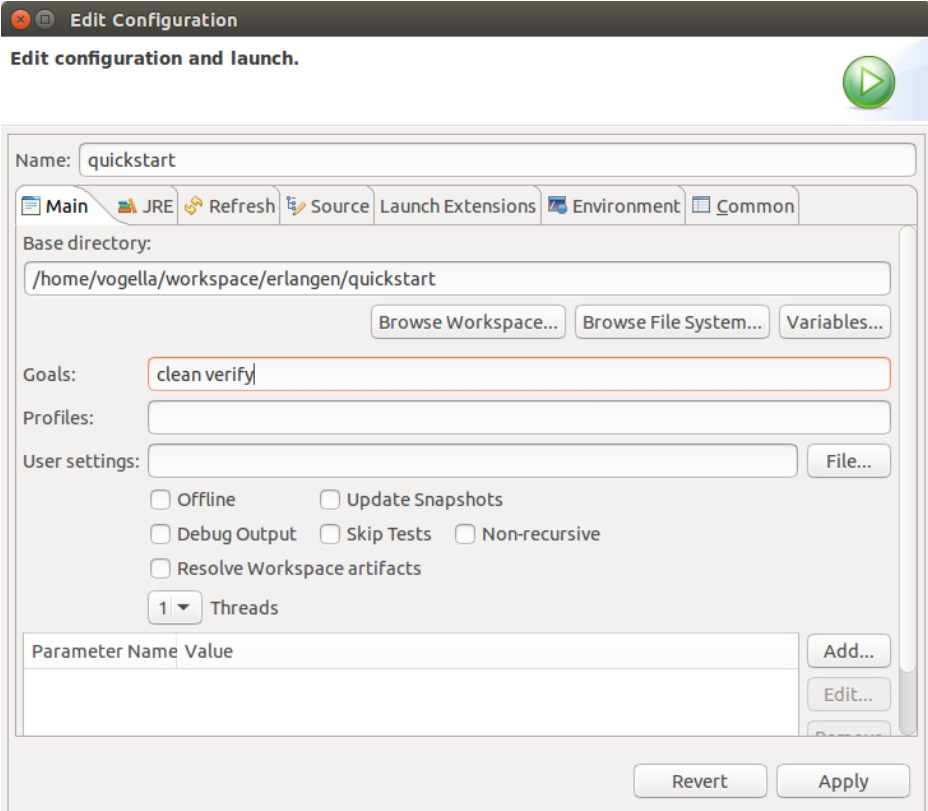
Each phase is a sequence of goals, and each goal is responsible for a specific task. When we run a phase, all goals bound to this phase are executed in order.



1. **validate**: This step validates if the project structure is correct and all the required information available. For example – It checks if all the dependencies have been downloaded and are available in the local repository.

From eclipse IDE:right-click the pom.xml file and select Run As-> Maven build.

This opens a dialog which allows to define the parameters for the start. Enter **clean verify** in the *Goals:* field and press the Run button.



1. **compile**:  It compiles the source code, converts the .java files to .class and stores the classes in target/classes folder.

From command Line

|  |
| --- |
| mvn compile |

This will run Maven, telling it to execute the *compile* goal. When it’s finished, you should find the compiled *.class* files in the *target/classes* directory.

1. **test**: It will simply run the unit tests without creating any deployment file.

from command Line

|  |
| --- |
| mvn test |

From Eclipse IDE: select the project and go to “Run As -> Maven test”.

1. **package**: This step packages the compiled code in distributable format like JAR or WAR file within the target directory. The name of the JAR/WAR file will be based on the project’s <artifactId> and <version>.

|  |
| --- |
| mvn package |

From Eclipse IDE: select the project and go to “Run As -> Maven Build”. The “Edit Configuration” popup window will open. Enter the “Goals” as “**package**” to build the project and click on the Run button.

1. **integration-test**: It runs the integration tests for the project.
2. **verify**: This step runs checks to verify that the project is valid and meets the quality standards.
3. **install**: Maven also maintains a repository of dependencies on your local machine (usually in a *.m2/repository* directory in your home directory) for quick access to project dependencies. If you’d like to install your project’s JAR file to that local repository, then you should invoke the install goal:

from command Line

|  |
| --- |
| mvn install |

From Eclipse IDE: select the project and go to “Run As -> Maven Install”.

The *install* goal will compile, test, and package your project’s code and then copy it into the local dependency repository, ready for another project to reference it as a dependency.

1. **deploy**: It copies the packaged code to the remote repository for sharing it with other developers.
2. **clean**: Maven Clean is used to clean the target folder. This is where the previous build’s files, libraries, reports, output files, etc., are saved.

from command line

|  |
| --- |
| mvn -clean |

From eclipse IDE: select the project and go to “Run As -> Maven Clean”.

1. site: creates site documentation for the project

Maven Commands**:**

* **mvn clean**: Cleans the project and removes all files generated by the previous build.
* **mvn compile**: Compiles source code of the project.
* **mvn test-compile**: Compiles the test source code.
* **mvn test:** Runs tests for the project.
* **mvn package**: Creates JAR or WAR file for the project to convert it into a distributable format.
* **mvn install:** Deploys the packaged JAR/ WAR file to the local repository.
* **mvn deploy:** Copies the packaged JAR/ WAR file to the remote repository after compiling, running tests and building the project.

# Jenkins Integration

# Properties File

# Log Files

# GIT Integration

# Selenium Cucumber

Cucumber is a test automation framework which leverages Behavior Driven Development for collaboration in between Business and IT teams. It empowers a user to define an application’s behavior in plain English language which makes it easier for non-programmers to understand the acceptance criteria.  
The core of cucumber has been developed in Ruby programming language however it supports all the majorly used programming languages for testing such as Java, C#, Python. In this article, I will be demonstrating automation testing with Selenium, Cucumber and TestNG.

Both the story and the feature file are written from a high-level point of view, and because we want to automate the scenarios, it only seems natural to start writing step definitions in Cucumber which call Selenium to drive the application, do the actions and verify the outcome.

## Behavior Driven Development

Behavior Driven Development is derived from Test Driven Development as a software development methodology. As a part of the Test Driven Development approach, the developer creates tests as a part of the acceptance criteria first. The developer will make sure the test is passed and will commit changes in the code. if required. Test Driven Development is a way to ensure the system meets its requirements by ensuring a 100% test coverage. It provides an edge over other methodologies in terms of finding defects early in the cycle which reduces the cost of finding bugs and refactoring improves the code.

Cons of Test Driven Development are the maintenance of tests created, it is a test-centric methodology, non-programmers might find it hard to understand for performing selenium automation testing. The test or acceptance criteria being targeted makes it hard for team collaboration. To overcome the above trouble heads, Behavior Driven Development was introduced to reduce the time to test, less code with more collaboration.

**Collaborative Approach:** Bridges the communication gap between stakeholder or business teams who can’t easily understand or read the code. This problem is addressed by defining the behavior of application under test in a ubiquitous language i.e. easily understandable.

Focuses more in terms of end-user experience or inclined more towards behavior specifications while compared to that of traditional Test Driven methodology. Tests are written more from end user’s point of view, making them easier to understand for performing Selenium automation testing.

## Selenium & Cucumber Automation Challenges

## Set up

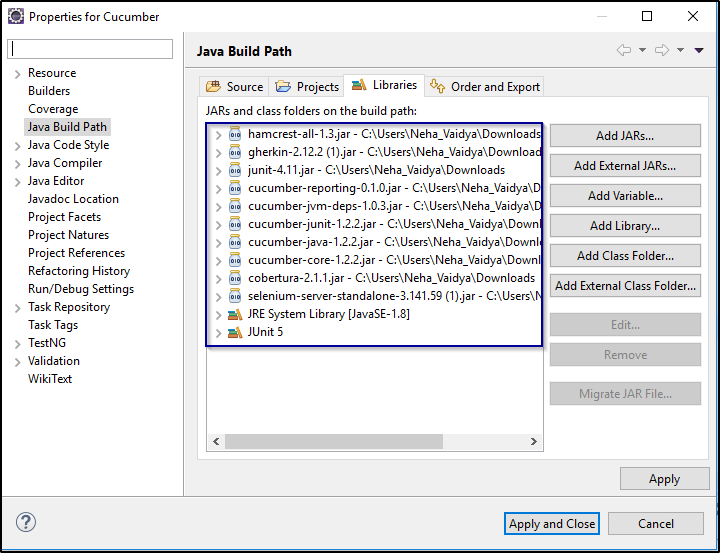
**Step1:** Download Cucumber and Selenium Jar files

Cucumber doesn’t actually interact with the browser and performs actions on the website under test. Interacting with the website is the action that is performed by Selenium WebDriver. Due to this reason, you will need to download Cucumber as well as Selenium Webdriver jar files.

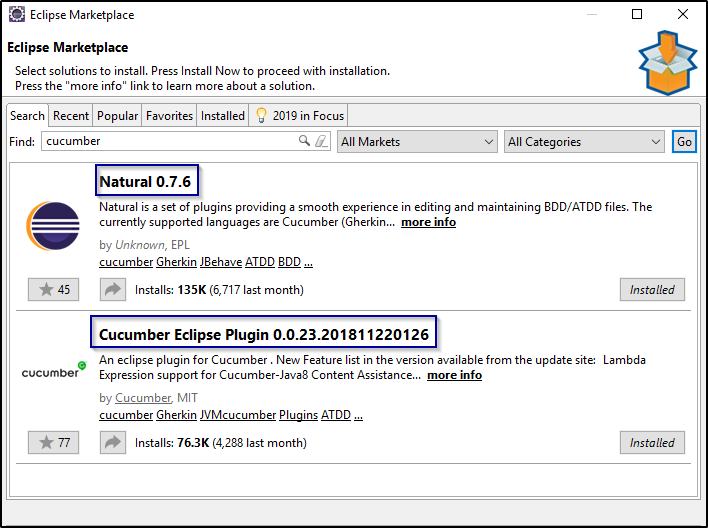
Below is the list of jar files for Cucumber.

* Cucumber-core
* Cucumber-html
* cobertura code coverage
* Cucumber-java
* Cucumber-junit
* Cucumber-jvm-deps
* Cucumber-reporting
* Hamcrest-core
* Gherkin
* Junit

Once you download these jar files you can configure the build path of your project and add all the *.jar* files and libraries as shown in the below snapshot.



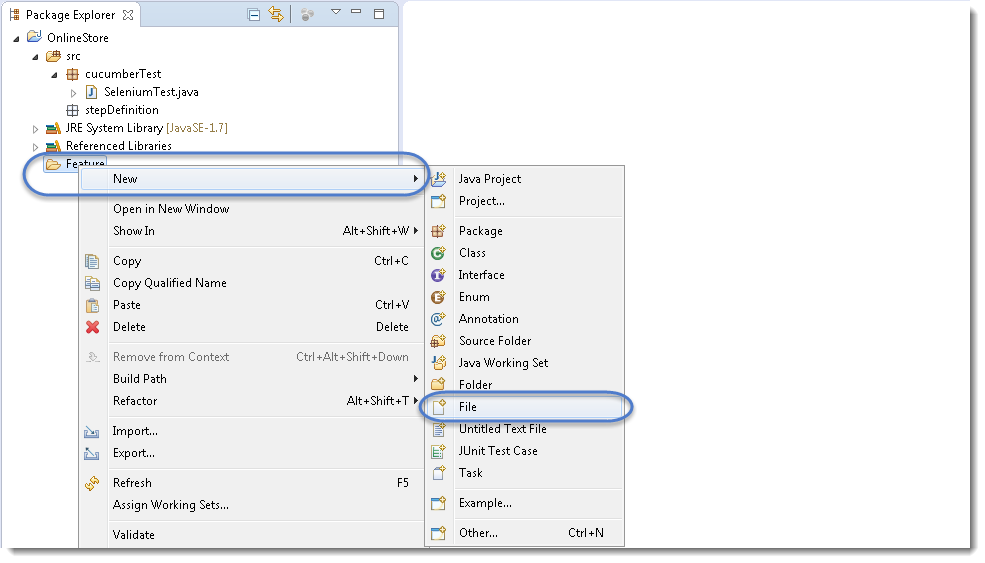
**Step 2:** The next step is to install Cucumber on your Eclipse. For that, you need to go to Help -> Eclipse Marketplace -> Search Cucumber and install **Cucumber and Naturals** on your eclipse. Below snapshot depicts the same.



## ****Feature File****

Once you configure Cucumber, the next step is to create a feature file.

A ***Feature File*** is an entry point to the Cucumber tests. This is a file where you will describe your tests in Descriptive language (Like English - Gherkin). It is an essential part of Cucumber, as it serves as an automation test script as well as live documents. A feature file can contain a scenario or can contain many scenarios in a single feature file but it usually contains a list of scenarios. Let’s create one such file.

On the ***Feature*** folder Right click and select ***New > File*** 

In order for Cucumber to automatically detect the stories (or ***features***, as they’re known in Cucumber), you need to make sure that they carry the ‘***.feature***‘ file extension.



Sample Feature File

|  |
| --- |
| **Feature**: LogIn Action Test Description: This feature will test a LogIn and LogOut functionality  **Scenario**: Successful Login with Valid Credentials **Given**User is on Home Page **When**User Navigate to LogIn Page **And**User enters UserName and Password **Then**Message displayed Login Successfully |

### Gherkin Keywords

Here is the list of keywords that *Gherkin* supports:

* ***Feature***
* ***Background***
* ***Scenario***
* ***Given***
* ***When***
* ***Then***
* ***And***
* ***But***
* ***\****

#### Feature

Each Gherkin file begins with a **Feature** keyword. Feature defines the logical test functionality you will test in this feature file. For e.g if you are testing a payment gateway your Feature will become Payment Gateway or if you are testing the LogIn functionality then the Feature will become Login. The idea of having a feature file is to put down a summary of what you will be testing. This will serve as the documentation for your tests as well as a good point to start for a new team member. Note that a feature keyword is present at the starting of the feature file.

|  |
| --- |
| ***Feature***: LogIn Action TestThis feature will test a LogIn and LogOut functionality |

Everything after Feature: till the next Keyword is encountered is considered as feature description.

#### Background

***Background***keyword is used to define steps which are common to all the tests in the feature file.

|  |
| --- |
| **Feature**: Add to Cart This feature will test functionality of adding different products to the User basket from different flow  **Background:** User is Logged In |

#### Scenario

Each Feature will contain some number of tests to test the feature. Each test is called a ***Scenario*** and is described using the Scenario: keyword.

A scenario is equivalent to a test in our regular development process. Each scenario/test can be basically broken down into three parts:

* Precondition to the test, which represent with (Given) keyword
* Test step execution, which represent with (When) keyword
* Verification of the output with expected result, which represent with (Then)

|  |
| --- |
| **Scenario**: Search a product and add the first result/product to the User basket |

#### Given

***Given*** defines a precondition to the test. For e.g. In shopping website, assume that the LogIn page link is only present on the Home Page, so the precondition for clicking the LogIn link is that the user is at the Home Page. If user is not at the Home Page, user would not be able to enter Username & Password. This precondition can be expressed in Gherkin like this:

|  |
| --- |
| **Scenario**: Successful LogIn with Valid Credentials  **Given**User is on Home Page **When**User Navigate to LogIn Page |

#### When

***When*** keyword defines the test action that will be executed. By test action we mean the user input action.

Here user is performing some action using When keyword, clicking on the LogIn link. We can see that when defines the action taken by the user. It’s the event that will cause the actual change in state of the application.

|  |
| --- |
| **Scenario**: Successful LogIn with Valid Credentials  **Given**User is on Home Page **When**User Navigate to LogIn Page |

#### Then

***Then***keyword defines the Outcome of previous steps. We can understand it best by looking at the test above and adding a Then step there.

Here we can see that ***Then*** is the outcome of the steps above. The reader of this test would easily be able to relate to Then step and would understand that when the above conditions are fulfilled then the Then step will be executed.

|  |
| --- |
| **Feature**: LogIn Action Test Description: This feature will test a LogIn and LogOut functionality  **Scenario**: Successful Login with Valid Credentials **Given**User is on Home Page **When**User Navigate to LogIn Page **And**User enters UserName and Password **Then**Message displayed LogIn Successfully |

#### And

**And** keyword is used to add conditions to your steps

#### But

**But**keyword is used to add negative type comments. It is not a hard & fast rule to use but only for negative conditions. It makes sense to use But when you will try to add a condition which is opposite to the premise your test is trying to set

|  |
| --- |
| **Feature**: LogIn Action Test Description: This feature will test a LogIn and LogOut functionality  **Scenario**: Unsuccessful Login with InValid Credentials **Given**User is on Home Page **When**User Navigate to LogIn Page **And**User enters UserName and Password **But**The user credentials are wrong **Then**Message displayed Wrong UserName & Password |

#### \*

his keyword defies the whole purpose of having Given, When, Then and all the other keywords. Basically Cucumber doesn’t care about what Keyword you use to define test steps, all it cares about what code it needs to execute for each step. That code is called a **step definition.** All the keywords can be replaced by the***\** keyword**

|  |
| --- |
| **Feature**: LogIn Action Test Description: This feature will test a LogIn and LogOut functionality  **Scenario**: Successful Login with Valid Credentials **\***User is on Home Page **\***User Navigate to LogIn Page **\***User enters UserName and Password **\***Message displayed Login Successfully |

## Step definition

In order to test the feature file, we need to write the implementation or step definition for each step in the feature file in java. When Cucumber executes a Step in a Scenario it will look for a matching Step Definition to execute.

A Step Definition is a small piece of code with a pattern attached to it or in other words a Step Definition is a java method in a class with an annotation above it. An annotation followed by the pattern is used to link the Step Definition to all the matching Steps, and the code is what Cucumber will execute when it sees a Gherkin Step. Cucumber finds the Step Definition file with the help of Glue code in **Cucumber Options.**

|  |
| --- |
| import cucumber.api.java.en.Given;  import cucumber.api.java.en.Then;  import cucumber.api.java.en.When;  public class Test\_Steps {  public static WebDriver driver;  @Given("^User is on Home Page$")  public void user\_is\_on\_Home\_Page() throws Throwable {  driver = new FirefoxDriver();  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  driver.get("http://www.store.demoqa.com");  }  @When("^User Navigate to LogIn Page$")  public void user\_Navigate\_to\_LogIn\_Page() throws Throwable {  driver.findElement(By.xpath(".//\*[@id='account']/a")).click();  }    @Then("^Message displayed Login Successfully$")  public void message\_displayed\_Login\_Successfully() throws Throwable {  System.out.println("Login Successfully");  }  } |

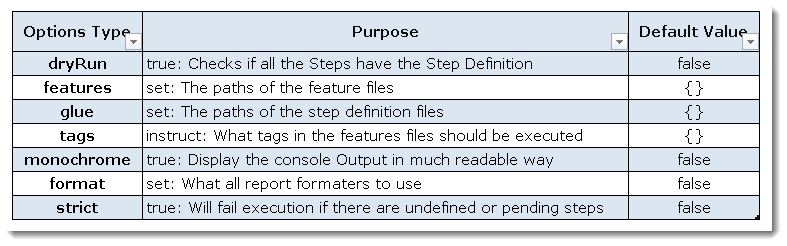
## Test Runner

Now that we have defined the test its time to run our test. But before we do that we have to add a class for running our tests. It more like a starting point for Junit to start executing your tests. In the src folder create a class called TestRunner**.**

This class just need annotations to understand that cucumber features would be run through it and you can specify feature files to be picked up plus the steps package location

### ****Cucumber Options****

Following Main Options are available in Cucumber:



**Dry Run**

**dryRun** option can either set as **true** or **false**. If it is set as true, it means that Cucumber will only checks that every Step mentioned in the Feature File have corresponding code written in Step Definition file or not. So in case any of the function is missed in the Step Definition for any Step in Feature File, it will give us the message

|  |
| --- |
| package cucumberTest;    import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  ,dryRun = true  )    public class TestRunner {    } |

**Monochrome**

This option can either set as **true**or **false**. If it is set as *true*, it means that the *console output* for the *Cucumber test*are much more readable. And if it is set as *false*, then the *console output* is not as readable as it should be.

|  |
| --- |
| **package cucumberTest;**    **import org.junit.runner.RunWith;**  **import cucumber.api.CucumberOptions;**  **import cucumber.api.junit.Cucumber;**    **@RunWith(Cucumber.class)**  **@CucumberOptions(**  **features = "Feature"**  **,glue={"stepDefinition"}**  **,monochrome = false**  **)**    **public class TestRunner {**    **}** |

***Features***

***Features Options***helps Cucumber to locate the Feature file in the project folder structure. All we need to do is to specify the folder path and Cucumber will automatically find all the ‘***.features***‘ extension files in the folder. It can be specified like:

***features = “Feature“***

Or if the Feature file is in the deep folder structure

***features = “src/test/features“***

**Glue**

**It helps Cucumber to locate the Step Definition file. Whenever Cucumber encounters a Step, it looks for a Step Definition inside all the files present in the folder mentioned in Glue Option. It can be specified like:**

***glue = “stepDefinition“***

Or if the Step Definition file is in the deep folder structure

***glue = “src/test/stepDeinition“***

**Format**

Format Option is used to specify different formatting options for the output reports. Various options that can be used as for-matters are:

***Pretty:***Prints the Gherkin source with additional colours and stack traces for errors. Use below code:

***format = {“pretty“}***

***HTML:***This will generate a HTML report at the location mentioned in the for-matter itself. Use below code:

***format = {“html:Folder\_Name“}***

***JSON:***This report contains all the information from the gherkin source in JSON Format. This report is meant to be post-processed into another visual format by 3rd party tools such as Cucumber Jenkins. Use the below code:

***format = {“json:Folder\_Name/cucumber.json“}***

***JUnit:*** This report generates XML files just like Apache Ant’s JUnit report task. This XML format is understood by most Continuous Integration servers, who will use it to generate visual reports. use the below code:

***format = { “junit:Folder\_Name/cucumber.xml“}***

### ****JUNIT Test Runner****

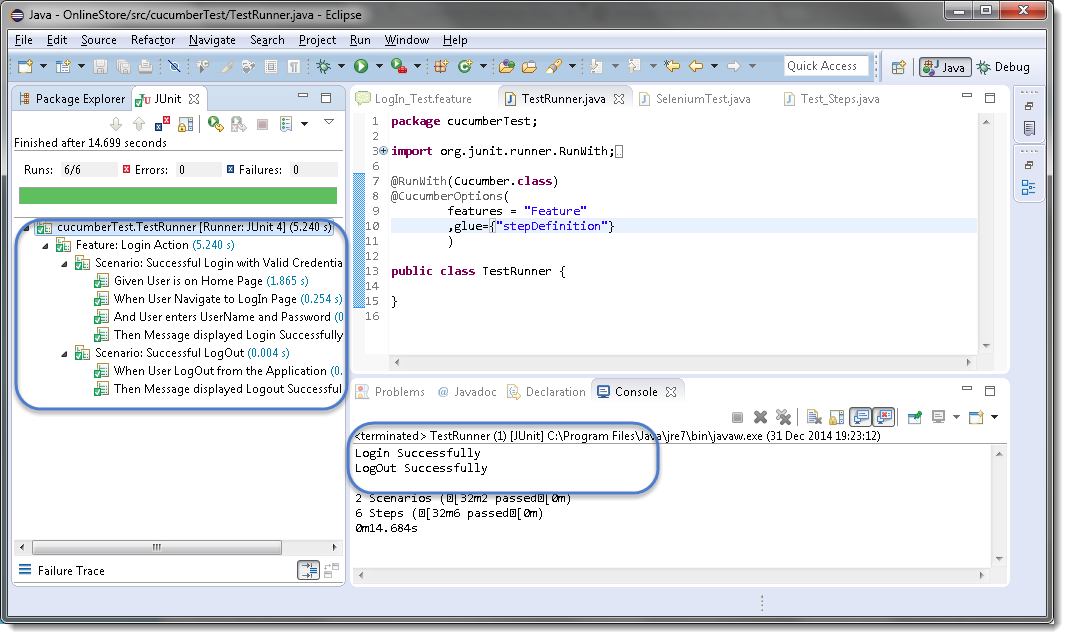
* + 1. import statement ‘org.junit.runner.RunWith’ imports @RunWith annotation from the Junit class. @RunWith annotation tells JUnit that tests should run using Cucumber class present in ‘Cucumber.api.junit‘ package.
    2. import statement ‘cucumber.api.CucumberOptions‘ imports the @CucumberOptions annotation. This annotation tells Cucumber a lot of things like where to look for feature files, what reporting system to use and some other things also

|  |
| --- |
| import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  )    public class TestRunner {    } |

Right Click on ***TestRunner*** class and Click ***Run As***  > ***JUnit Test.***

Cucumber will run the script the same way it runs in Selenium WebDriver and the result will be shown in the left hand side project explorer window in JUnit tab.

Cucumber starts it’s execution by reading the feature file steps. As soon as Cucumber reaches to the first step for e.g. Given statement of Scenario, it looks for the same statement in the Step Definition file, the moment it find the statement, it executes the piece of code written inside the function.



### TESTNG Test Runner

TestRunner.class

|  |
| --- |
| package MyRunner;  import org.testng.annotations.AfterClass;  import org.testng.annotations.BeforeClass;  import org.testng.annotations.DataProvider;  import org.testng.annotations.Test;  import cucumber.api.CucumberOptions;  import cucumber.api.testng.CucumberFeatureWrapper;  ***import cucumber.api.testng.TestNGCucumberRunner;***  @CucumberOptions(  features = "src/main/java/Features",  glue = {"stepDefinitions"},  tags = {"~@Ignore"},  format = {  "pretty",  "html:target/cucumber-reports/cucumber-pretty",  "json:target/cucumber-reports/CucumberTestReport.json",  "rerun:target/cucumber-reports/rerun.txt"  },plugin = "json:target/cucumber-reports/CucumberTestReport.json")  public class TestRunner {  private ***TestNGCucumberRunner testNGCucumberRunner;***    @BeforeClass(alwaysRun = true)  public void setUpClass() throws Exception {  testNGCucumberRunner = new TestNGCucumberRunner(this.getClass());  }    @Test(groups = "cucumber", description = "Runs Cucumber Feature", dataProvider = "features")  public void feature(CucumberFeatureWrapper cucumberFeature) {  testNGCucumberRunner.runCucumber(cucumberFeature.getCucumberFeature());  }    @DataProvider  public Object[][] features() {  return testNGCucumberRunner.provideFeatures();  }    @AfterClass(alwaysRun = true)  public void tearDownClass() throws Exception {  testNGCucumberRunner.finish();  }  } |

Testing.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">  <suite name="BDD Test Suite" verbose="1" parallel="tests"  thread-count="1" configfailurepolicy="continue">  <test name="FREE CRM Test" annotations="JDK"  preserve-order="true">  <classes>  <class name="MyRunner.TestRunner" />  </classes>  </test>  </suite |

# SELENIUM FRAMEWORK

Java as the programming language

TestNG as the assertion framework

Maven as the build tool

WebDriver as the browser automation tool

Eclipse as the IDE