Automation Testing Using Selenium

Contents

- How to handle tooltip in Selenium
- Multi Select Actions in Selenium
- Mouse hover and Auto Suggestions in Selenium

4. ALERT, WINDOWS and MANY MORE

- Alert Interface and Handle JavaScript Alerts in Selenium
- AutoIT tool and two ways to handle windows Authentication
- Different ways to upload and Download file in Selenium
- Different ways of Scrolling in Selenium WebDriver
- Handle multiple windows in Selenium WebDriver
- Handling iFrames in Selenium Webdriver
- Waits in Selenium WebDriver
- How To Find Broken Links Using Selenium
- Headless Browser Testing Using Selenium
- How To Run Selenium Tests In Headless Google Chrome
- Handle Dynamic WebTable in Selenium
- Handle Bootstrap Date-picker in Selenium

5. SCREEN Shot, REPORTS, LOGS

- How to capture Screen Shot in Selenium
- How to capture Screen Shot for failed test cases
- How to take Full Page ScreenShot and ScreenShot of WebElement
- Generate Logs in Selenium Web Driver using log4j
- Generate Extent report and attach screen shots in extent report
- Extent Report implementation using ITestListener

6. TestNG Tutorials

- Introduction to TestNG - Features and Installation Guide
- TestNG Annotations - Different Annotations and their execution
- Different Attributes in TestNG

- Working with testng.xml
- Parameterization using TestNG
- Parallel Testing in TestNG
- Cross Browser Testing in TestNG
- Run failed test cases in Selenium
- Listeners in TestNG
- Assertions in TestNG

7. Maven Tutorials

- ✓ Maven Introduction and Setup

8. POM, DataDriven FrameWork

- ❖ Introduction to Page Object Model(POM)
- ❖ Page Object Model(POM) Using PageFactory
- ❖ Read excel file in Selenium using Apache POI
- ❖ Write data into excel file in Selenium using Apache POI
- ❖ Complete Excel Library
- ❖ Data Driven Framework/Testing Part 1
- ❖ Data Driven Framework/Testing Part 2

9. Git and GitHub

- ✓ Selenium Integration with Git and GitHub

10. Continuous Integration with Jenkins

- Selenium Integration with Jenkins

11. Selenium Framework:

- Selenium Data Driven Framework with POM

Selenium Basics:

- **Selenium and it's History**

What is Selenium?

Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms.

Selenium is not just a single tool but a suite of software's, each catering to different testing needs of an organization. It has four components.

Selenium Integrated Development Environment (IDE)
Selenium Remote Control (RC)
WebDriver
Selenium Grid

Selenium Core

The story starts in 2004 at ThoughtWorks in Chicago, with Jason Huggins building the Core mode as "JavaScriptTestRunner". Its JavaScript program that would automatically control the browser's actions.

Jason Huggins

"JavaScriptTestRunner" was later named as "Selenium Core" and released into the market as an Open Source tool.

This Open Source tool started gaining demand in the market and people started using it for automating the repeated tasks in their Web Applications.

Selenium Remote Control

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. To resolve this another ThoughtWork's engineer, Paul Hammant created system (in 2007) known as the Selenium Remote Control or Selenium 1.

Selenium 1 = Selenium IDE + Selenium RC + Selenium Grid

Paul Hammant

Selenium Grid

Patrick Lightbody to address the need of minimizing test execution times as much as possible, So he created Selenium Grid. Basically grid is for parallel execution and execute your test scripts on multiple environments.

Patrick Lightbody

Using “Selenium Grid”, testers were able to distribute the tests across multiple machines and get them executed them on different machines over their network to reduce or minimize the time taken for overall execution of tests.

Selenium IDE

"Shinya Kasatani", who developed a Firefox extension named as "Selenium IDE".

"Selenium IDE" using its record and playback feature, records the automation tests like recording a video and executes the recorded tests like playing the recorded videos.

Shinya Kasatani

Selenium WebDriver

Earlier Selenium 1 used to be the major project of Selenium.

Selenium 1 = Selenium IDE + Selenium RC + Selenium Grid

Later Selenium Team has decided to merge both Selenium WebDriver and Selenium RC to form a more powerful Selenium tool.

They both got merged to form "Selenium 2"

"Selenium WebDriver" was the core of "Selenium 2" and "Selenium RC" used to run in maintenance mode.

Hence **Selenium 2** = Selenium IDE + Selenium WebDriver 2.x + Selenium Grid.

"Selenium 2" released on July 8, 2011.

Selenium team has decided to completely remove the dependency for Selenium RC.

After 5 years, "Selenium 3" was released on October 13, 2016 with a major change, which is the original Selenium Core implementation and replacing it with one backed by WebDriver and lot more improvements.

Hence **Selenium 3** = Selenium IDE + Selenium WebDriver 3.x + Selenium Grid.

After 3 years from it's a major release, now Selenium has put out its first alpha version of Selenium 4 on Apr 24, 2019. Still, there is no official announcement about the release date of Selenium 4, but we are expecting it around October 2019. Till that there can be several alpha or beta versions released time to time with stabilization.

- **Download and Install Java and Eclipse**

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented. It is intended to let application developers "write once, run anywhere.

Let's understand what is JDK, JRE and JVN in brief.

JDK
Java Development Kit is the core component of Java Environment and provides all the tools, executables and binaries required to compile, debug and execute a Java Program. JDK is a platform-specific software and that's why we have separate installers for Windows, Mac, and Unix systems. We can say that JDK is the superset of JRE since it contains JRE with Java compiler, debugger, and core classes.

JRE
JRE is the implementation of JVM, it provides a platform to execute java programs. JRE consists of JVM and java binaries and other classes to execute any program successfully. JRE doesn't contain any development tools like java compiler, debugger etc.

JVM
JVM is the heart of Java programming language. When we run a program, JVM is responsible for converting Byte code to the machine specific code. JVM is also platform dependent and provides core java functions like memory management, garbage collection, security etc.

- **First Selenium Program**

Prerequisites to run first simple WebDriver Program are:

1. Java IDE (Eclipse/ IntelliJ) - Environment that you will use to write your code in.
2. Selenium Webdriver libraries - Libraries which allows you use all the Selenium functions and classes.
3. Browser drivers (Chrome/ Firefox/IE) - The driver with with Selenium WebDriver communicates.

- **Selenium WebDriver Architecture**

There are four components of **Selenium Architecture**:

Selenium Client Library
JSON Wire Protocol over HTTP
Browser Drivers
Browsers

Selenium Client Library:

Selenium supports multiple libraries such as Java, Ruby, Python, etc., Selenium Developers have developed language bindings to allow Selenium to support multiple languages.
Below is the link to download Selenium Client Language Bindings:

https://www.seleniumhq.org/download/#client-drivers

JSON Wire Protocol over HTTP:

JSON stands for JavaScript Object Notation. It is used to transfer data between a server and a client on the web. JSON Wire Protocol is a REST API that transfers the information between HTTP server. Each BrowserDriver (such as FirefoxDriver, ChromeDriver etc.,) has its own HTTP server.

Browser Drivers:

Each browser contains separate browser driver. Browser drivers communicate with respective browser without revealing the internal logic of browser's functionality. When a browser driver is received any command then that command will be executed on the respective browser and the response will go back in the form of HTTP response.

Real Browsers:

Selenium supports multipe browsers such as Firefox, Chrome, IE, Safari etc.

How Selenium Works Internally:

Lets say you have written below couple of lines of code:

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

driver.get(https://www.google.com)

Based on the above statements, Chrome browser will be launched and it will navigates to automationtestinginsider website.

Once you Run the program, every statement in your script will be converted as a URL with the help of JSON Wire Protocol over HTTP. The URL's will be passed to the Browser Drivers. (In this case, ChromeDriver). Here in our case the client library (java) will convert the statements of the script to JSON format and communicates with the ChromeDriver. URL looks as shown below.
http://localhost:8080/{"url":"https://www.google.com"}

Every Browser Driver uses a HTTP server to receive HTTP requests. Once the URL reaches the Browser Driver, then the Browser Driver will pass that request to the real browser over HTTP. Then the commands in your selenium script will be executed on the browser.

If the request is POST request then there will be an action on browser

If the request is a GET request then the corresponding response will be generated at the browser end and it will be sent over HTTP to the browser driver and the Browser Driver over JSON Wire Protocol and sends it to the UI (Eclipse IDE).

This is how Selenium Commands works internally.

- **Launch Chrome, FireFox and IE browsers with Selenium WebDriver**

Prerequisites to run Selenium Scripts on Different Browsers:

Programming Language – Java
https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

IDE (Code Editor) - Eclipse
https://www.eclipse.org/downloads/

Browser Drivers – In order to communicate with actual browsers, selenium need something called driver. Every browser has its own driver.
https://www.seleniumhq.org/download/

Selenium WebDriver Libraries – Libraries which allow you to use all the selenium classes and functions

https://www.seleniumhq.org/download/

Browser – Chrome. Firefox, IE
Application Under Test – Orange HRM (https://opensource-demo.orangehrmlive.com/)

Basic Script:

```
package launchBrowser;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class FirstSeleniumScriptChrome {

 WebDriver driver;

 public void launchBrowser() {
  System.setProperty("webdriver.chrome.driver", "D://chromedriver.exe");
  driver=new ChromeDriver();
  System.out.println("Browser is Launched");
  driver.manage().window().maximize();
  driver.get("https://opensource-demo.orangehrmlive.com/");
  System.out.println("Orange HRM url is opened");
 }

 public void login() {
  driver.findElement(By.name("txtUsername")).sendKeys("Admin");
  driver.findElement(By.id("txtPassword")).sendKeys("admin123");
  driver.findElement(By.id("btnLogin")).click();
  System.out.println("User is Logged in");
 }

 public void title() {
  String title=driver.getTitle();
  System.out.println("Title of the Page is:"+title);
 }

 public void logout() throws InterruptedException {
  driver.findElement(By.id("welcome")).click();
  Thread.sleep(2000);
```

```
 driver.findElement(By.xpath("//*[@id='welcome-menu']//*[text()='Logout']")).click();
 System.out.println("Logged out from Oranage HRM Website");
}

public void closeBrowser() {
 driver.close();
 System.out.println("Browser is Closed");
}

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

 FirstSeleniumScriptChrome obj= new FirstSeleniumScriptChrome();
 obj.launchBrowser();
 obj.login();
 obj.title();
 obj.logout();
 obj.closeBrowser();

 }

}
```

HTML, DOM and Locators:

- ✓ **Basics of HTML for Selenium WebDriver**

What is HTML?

- HTML stands for Hyper Text Mark-up Language
- HTML describes the structure of a Web page
- HTML consists of a different elements
- HTML elements tell the browser how to display the content
- HTML elements are represented by tags
- HTML provides the basic structure of sites, which is enhanced and modified by other technologies like
- CSS is used to control presentation, formatting, and layout.
- JavaScript is most well-known as the scripting language for Web pages and it is used to control the behaviour of different elements. is most well-known as the scripting language for Web pages.

```
!DOCTYPE html>
   <html>
     <head>
      <title>Page Title</title>
     </head>
       <body>
         <h1>Heading</h1>
         <p>Paragraph</p>
       </body>
   </html>
```

Sample HTML

In Above Image:

<The <!DOCTYPE html> declaration defines this document to be HTML5
The <html> element is the root element of an HTML page
The <head> element contains meta information about the document
The <title> element specifies a title for the document
The <body> element contains the visible page content
The <h1> element defines a large heading
The element defines a paragraph

HTML Tags:

--**HTML tags** are element names surrounded by angle brackets:
<tagname>content here...</tagname>
The <!DOCTYPE> declaration represents the document type, and helps browsers to display web pages correctly.
--**HTML Headings** - HTML headings are defined with the <h1> to <h6> tags.
<h1>This is a Heading</h1>
--**HTML Paragraphs** - HTML paragraphs are defined with the tag.
This is a paragraph.
--**HTML Links** - HTML links are defined with the <a> tag:
This is a link

Attributes allow you to customise a tag, and are defined within the opening tag. Attributes are used to provide additional information about HTML element.

--**HTML Images** - HTML images are defined with the <img> tag.
The source file (src), alternative text (alt), width, and height are provided as an attributes
<img src=“ATI.jpg" alt=“automationtestinginsider.com" width="105" height="144">
--**HTML Buttons**- HTML buttons are defined with the <button> tag.

<button>Click me</button>
--**HTML Lists** - HTML lists are defined with the <ul> (unordered/bullet list) or the <ol> (ordered/numbered list) tag, followed by <li> tags (list items):

```
<ul>
  <li>C</li>
  <li>C++</li>
  <li>Java</li>
</ul>

<ol>
  <li>C</li>
  <li>C++</li>
  <li>Java</li>
</ol>
```

The <form> Element

--The HTML <form> element defines a form that is used to collect user input:
Form elements are different types of input elements, like text fields, checkboxes, radio buttons, submit buttons, and more.

The <input> Element

--The <input> element is the most important form element.
--The <input> element can be displayed in several ways, depending on the type attribute.

```
<input type="text">
<input type="radio">
<input type="submit">
```

--The <select> Element - The <select> element defines a drop-down list:

```
<select name="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="fiat">Fiat</option>
  <option value="audi">Audi</option>
</select>
```

div tag

--The tag defines a division or a section in an HTML document.
--The element is often used as a container for other HTML elements to style them with CSS or to perform certain tasks with JavaScript.

- ✓ **Different WebElements on a WebPage**

Basic WebElements:

- Link
- Button
- Image – with link, without link, with button.
- Edit Box/Text Box
- Text Area
- Check Box
- Radio Button
- Drop Down Box
- List Box
- Combo Box
- WebTable/HTML table
- Frame

Advance web techniques:

- JQuery
- Ajax
- Bootstrap

- ✓ **Document Object Model (DOM)**

What is DOM?

- Is the HTML you write the DOM?

Answer is No, But the HTML you write is parsed by the browser and turned into the DOM.

- Is 'View Page Source' the DOM?

Answer is No, View Source just shows you the HTML that makes up that page. It's probably the exact HTML that you wrote.

- Is the Code in Dev Tools the DOM?

Answer is Yes, We can say that that is a visual representation of the DOM!

When is the DOM different than the HTML?

- If there are mistakes in your HTML that will be fixed by the browser for you.
- Let's say you have a <table> element in your HTML and leave out the required <tbody> element. The browser will just insert that <tbody> for you. It will be there in the DOM, so you'll be able to find it with JavaScript and style it with CSS, even though it's not in your HTML.

DOM Definition:

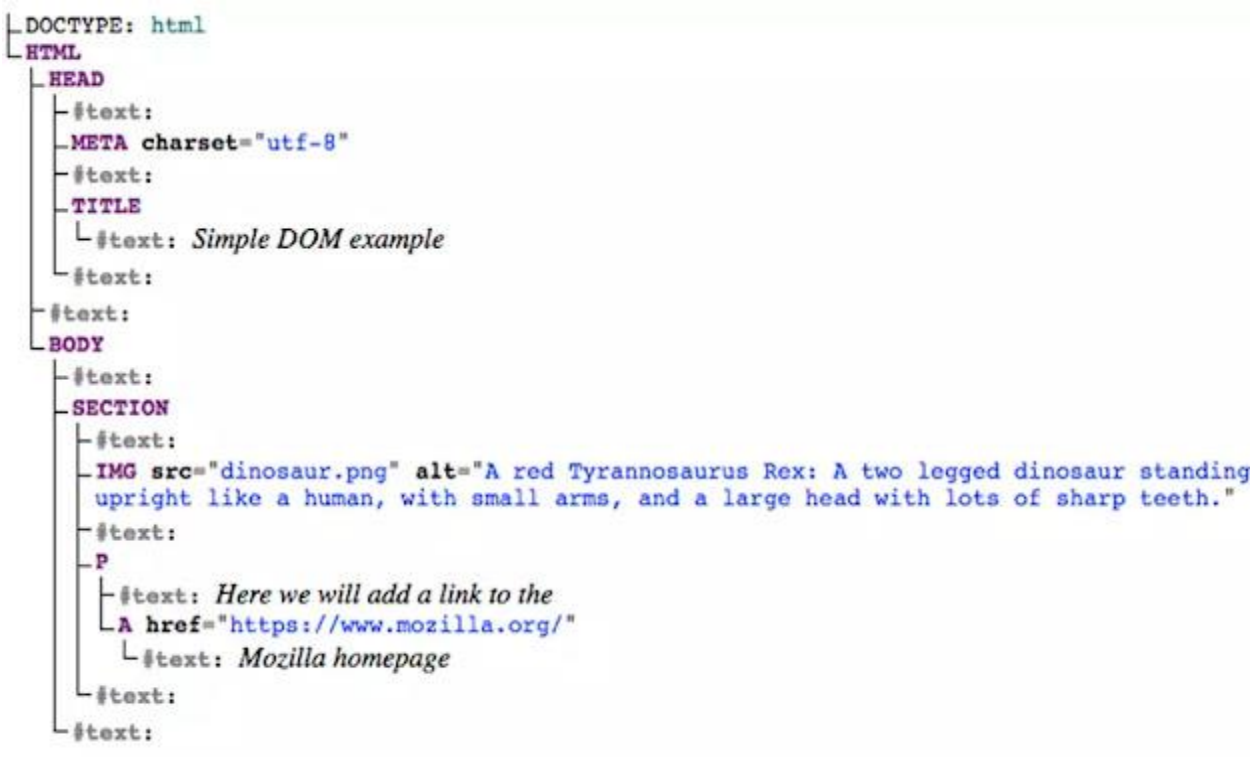
- When a web page is loaded, browser creates a tree of objects on web page is called DOcument Object Model (DOM).
- The DOM (Document Object Model) is an interface (API) that represents how your HTML and XML documents are read by the browser. It allows a language (JavaScript) to manipulate, structure, and style your website. After the browser reads your HTML document, it creates a representational tree called the Document Object Model and defines how that tree can be accessed.
- The Document Object Model (DOM) is a programming API for HTML and XML documents. It defines the logical structure of documents and the way a document is accessed and manipulated.
- With the Document Object Model, programmers can create and build documents, navigate their structure, and add, modify, or delete elements and content. Anything found in an HTML or XML document can be accessed, changed, deleted, or added using the Document Object Model.

In the beginning, JavaScript and the DOM were tightly intertwined, but eventually, they evolved into separate entities. The page content is stored in the DOM and may be accessed and manipulated via JavaScript, so that we may write this equation:

API (HTML or XML page) = DOM + JS (scripting language)

--The DOM was designed to be independent of any particular programming language
--The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript

Simple DOM Tree:



Simple DOM Tree

In Above Image:

Element node: An element, as it exists in the DOM.

Root node: The top node in the tree, which in the case of HTML is always the HTML node (other markup vocabularies like SVG and custom XML will have different root elements).

Child node: A node directly inside another node. For example, IMG is a child of SECTION in the above example.

Descendant node: A node anywhere inside another node. For example, IMG is a child of SECTION in the above example, and it is also a descendant. IMG is not a child of BODY, as it is two levels below it in the tree, but it is a descendant of BODY.

Parent node: A node which has another node inside it. For example, BODY is the parent node of SECTION in the above example.

Sibling nodes: Nodes that sit on the same level in the DOM tree. For example, IMG and P are siblings in the above example.

Text node: A node containing a text string.

HTML DOM Methods:

getElementById()
getElementsByClassName()
getElementsByName()

getElementsByTagName()
querySelector()
querySelectorAll()

Advantage for Programmers:

By manipulating the DOM, you have infinite possibilities. You can create applications that update the data of the page without needing a refresh. Also, you can create applications that are customizable by the user and then change the layout of the page without a refresh. You can drag, move, and delete elements.

- ✓ **Locators in Selenium and Their Priorities**

What are Locators?

Locators provide a way to access the HTML elements from a web page. In Selenium, we can use locators to perform actions on the text boxes, links, checkboxes and other web elements. They are the basic building blocks of a web page.

Locator Types:

- Selenium doesn't have any inbuilt capability or mechanism to locate any UI elements on the Web Pages.
- In order to find UI elements on the Web Page, Selenium has to take the help of Locators.
- We have 8 types of locators in Selenium:

1. id
2. name
3. className
4. linkText
5. partialLinkText
6. tagName
7. cssSelector
8. xapth

Locators can be classified into two categories:

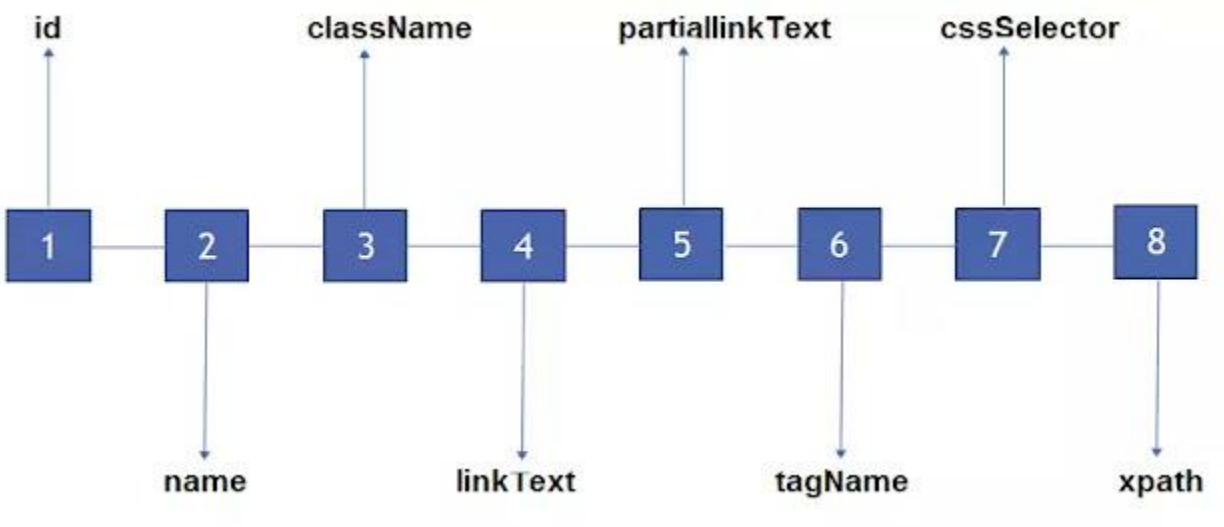
- Structure-based locators: locators that rely on the structure of the page to find elements.

--XPath
--CSS

- Attributes-based locators: locators that relies on the attributes of the elements to locate them

--Id
--Name
--Link
--CSS

Locator Priorities



Locator Priorities
id Locator

- Where, driver is an web driver objectID locator is faster among all locators. So, if any web element has ID as an attribute we must use ID locator. ID should be unique in a web page. ID locator is faster because at its roots, it calls “document.getElementById()”(A javaScript syntax) which is very much optimized by many browsers. Even ID helps in finding web element uniquely.
- Syntax - By.id("username")
- Example - driver.findElement(By.id("username"));

findElement is webdriver method
By - is a class

id - locator
username – value

- Pros - Each id is supposed to be unique so no chance of matching several elements
- Cons - Works well only on elements with fixed ids and not generated ones

name Locator

- “Name” locator comes after ID. If any web element has not ID attribute, we can use name attribute if available.
- Syntax - By.name(“log")
- Example - driver.findElement(By.name(“log"));
- Pros - Works well with fixed list of similar elements.
- Cons - But it may not identify web element uniquely because it has multiple web elements can share same name.

className Locator

- ClassName locator comes after name. If any web element has not ID and name attribute, we can use className attribute if available.
- Syntax - By.className("username")
- Example - driver.findElement(By.className("username"));
- Pros - Works well with fixed list of similar elements.
- Cons - But it may not identify web element uniquely because it has multiple web elements can share same name.

linktext and partiallinktext Locator

- This Link Text locator works only on links (hyperlinks) so it is called as Link Text locator.
- Partial Link Text: In some situations, we may need to find links by a portion of the text in a Link Text .
- Syntax and example: By.linkText("Forgot password"); By.partiallinkText("Forgot");
- Pros - Will only select anchor elements. Useful when testing navigation.
- Cons- You have to know the text of the link before.

tagName Locator

- A tagName is a part of a DOM structure where every element on a page is been defined via tag like input tag, button tag, anchor tag, etc. Each tag has multiple attributes like ID, name, value class, etc.
- Syntax and example: By.tagName("img");

- Pros - You can also use the tagName locator in combination with attribute value using XPath or CSS selectors.
- Cons - In a simple basic scenario where an element is located just via tag, it may lead to a lot of values being identified and may cause issues.

cssSelector Locator

- CSS stands for Cascading Style Sheets. By using CSS selectors, we can find or select HTML elements on the basis of their id, class or other attributes. CSS is faster and simpler than Xpath particularly in case of IE browser where Xpath works very slowly.

--Tag and ID
--Tag and class
--Tag and attribute
--Tag, class, and attribute
--Inner text - CSS in Selenium has an interesting feature of allowing partial string matches using ^, $, and *.

- Examples:

By.cssSelector("input#Email")
By.cssSelector("input.inputtext")
By.cssSelector("input[name=Email]")
By.cssSelector("input.inputtext[name=email]")

- Pros - It is faster. It also improves the performance. It is very compatible across browsers.CSS is best for IE as XPath does not work in IE always.
- Cons - Writing CSS is not simpler than XPath

xpath Locator

- It is slowest among all locators. But it provides you reliable ways to locate web elements.
- Syntax - Xpath=//tagname[@attribute='value']
- Example - //input[@type='text']
- Pros - Allows very precise locators
- Cons - XPath engines are different in each browser, hence make them inconsistent across browsers. That means if you write XPath for your application in Chrome browser, it may not work on IE.

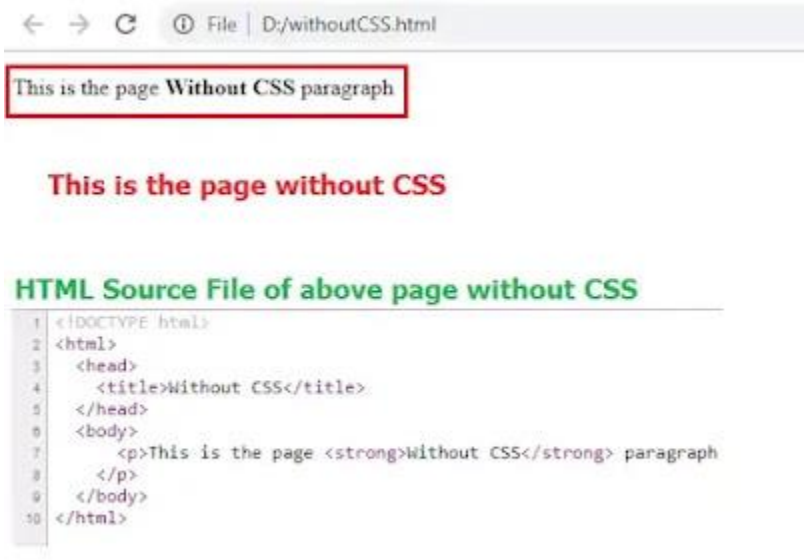
✓ **How to Locate Element Using CSS**

What is CSS?

- CSS stands for Cascading Style Sheets.
- CSS describes how HTML elements are to be displayed on screen.
- Why CSS is for locating web elements? --> To identify Dynamic web elements on webpage (Those elements whose attribute values are changing frequently by performing operation like page refresh so it is difficult to handle such elements so we use CSS and XPath to locate such elements).

Below are the examples shown in the images with and without CSS.

Simple Web Page without CSS:



Simple Web Page without CSS

Simple Web Page with CSS:

Simple Web Page with CSS

What is CSS Selector:
CSS selectors are used to select the content you want to style. CSS selectors select HTML elements according to its id, class, type, attribute etc.

XPath Vs cssSelector

- Xpath engines are different in each browser, hence make them inconsistent. Particularly in IE. IE does not have a native xpath engine.
- CSS is faster. It also improves the performance. It is very compatible across browsers.
- Writing CSS is not simpler than XPath.

Different ways to write CSS selector in Selenium Web driver: Please look into below table:

S. No.	Attribute	Syntax	Symbol
1	Locate by id	tag#id or #id	Using # symbol
2	Locate by className	tag.className	Using . symbol
3	Locate by Name or Attribute or multiple attributes	1. tagName[attribute='value'] 2. [attributeName='value'] 3.[attribute1='val'][attribute='val'] 4. [attribute1='val'],[attribute='val']	tagName[attributeName='value']
4	Match with Prefix or start with	[Attribute^='Prefix of Attribue value']	Using ^ symbol
5	Match with suffix or ends with	[Attribute^='Suffix of Attribue value']	Using $ symbol
6	Match with subString or contains	[Attribute*='subString of Attribue value']	Using * symbol

Different Ways to locate element using cssSelector

1. **Locate by id** (using symbol # hash)
   Syntax: tag#id or #id
   Example: input#user_login or #user_login

2. **Locate by className** (using symbol . dot)
   Syntax: tag.className
   Example: input.input

3. **Locate by Name or Attribute or using multiple attributes**:
   Syntax:

   - tagName[attributeName='value']
   - [attributeName='value']
   - tagName[attribute1='value'][attribute2='value']
   - tagName[attribute1='value'],[attribute2='value']

Example: few examples given below
input[name=‘username']
input[name='log'],[id='input'],[type='text']

4. **Match with Prefix or start with** (using symbol ^ Exponential operator)

Syntax: tagname[Attribute^= ‘Prefix of Attribute value']
Example: input[name^='txtUser']

5. **Match with suffix or ends with** (using symbol $ dollar sign)

Syntax: tagname[Attribute$= ‘suffix of Attribute value']
Example: input[name$='name']

6. **Match with substring or contains** (using symbol * asterisk)

Syntax: tagname[Attribute*='subString of Attribue value']
Example: input[name*='User']

- ✓ **How to Locate Element Using XPath- Part A**

What is XPath?

- XPath is XML Path Language which is used to traverse through the XML file and finding the desired information web element on the web page using XML path expression.
- Since the structure of HTML is similar to XML, we can use XPath language even to traverse through HTML tags and get the desired information.
- XPath expressions are the one of the locator types that can be used for locating the UI elements on the web pages

Syntax: XPath=//tagname[@attribute='value']

Where:
// : Select current node.
Tagname: Tagname of the particular node.
@: Select an attribute.
Attribute: Attribute name of the node like name, id etc.
Value: Value of the attribute.

Types of XPath:

- Absolute
- Relative XPath

Absolute XPath - Absolute XPath tries to locate the element from the root. i.e. complete path The disadvantage of the absolute XPath is that if there are any changes made in the path of the element then that XPath gets failed. it begins with the single forward slash(/).

Example: /html/body/p[@id='user123′]

Relative XPath - Unlike Absolute XPath, Relative XPath tries to locate the element directly, instead of locating from root. It webpage. starts with the double forward slash (//), which means it can search the element anywhere at the

Example: //p[@id='user123′]

Basic XPath:

1. Using Basic Attributes-- Basic XPath select nodes or list of nodes on the basis of attributes like ID , Name, Classname, etc.
Example: XPath=//input[@name='uid']

2. Contains() - Contains() is a method used in XPath expression. It is used when the value of any attribute changes dynamically, for example, login information.
Example: XPath=//*[contains(@type,'user')]

3. Using OR & AND Operator:
XPath=//*[@type='submit' or @name='btnReset']
XPath=//input[@type='submit' and @name='btnLogin']

4. Start-with function: Start-with function finds the element whose attribute value changes on refresh or any operation on the webpage.
XPath=//label[starts-with(@id,'message')]

5. Text(): In this expression, with text function, we find the element with exact text match
XPath=//input[text()='username']

- ✓ **How to Locate Element Using XPath- Part B (XPath Axess)**

What is XPath Axess?

- XPath Axes are used to find dynamic elements
- XPath AXES is the concept which makes the XPath Expressions powerful out of all the locators. i.e. By using XPath AXES, we can traverse both forward and backward in the HTML code of the web pages.
- Axes methods are used to find those elements, which dynamically change on refresh or any other operations. There are few axes methods commonly used in Selenium Webdriver like child, parent, ancestor, sibling, preceding etc.

Following are the axes techniques:

- Ancestor:
- Child:
- Descendant
- Following:
- Following-sibling:
- Parent
- Preceding:
- Preceding-sibling:

Ancestor: ancestor selects any Parent and Grandparent of the current node.
Example: //input[@id='user123']/ancestor::*
Child: Selects all children elements of the current node.
Example: //div[@id='user123']/child::*
Descendant: Descendant lets you select Children and Grandchildren of the current node.
Following: Following returns all in the document after the closing tag of the current node.
Following-sibling: Following-sibling returns all the sibling after the closing tag of the current node.

Parent: Parent returns the parent of the current node.
Preceding: Preceding returns all in the document before the current node.
Preceding-sibling: returns all the sibling before the current node.
Self: returns the current node.

- ✓ **What is ChroPath?**

ChroPath

For automation testers, identifying elements and performing actions is an important task. To make it easier, the concept called Locators was introduced in Selenium. But manually locating elements is a time-consuming task due to the complexity of the DOM structure of a webpage. This is where ChroPath comes into the picture and eases the task of locating elements in Selenium. In this article we will talk about chropath and its features.

What is ChroPath?

ChroPath is a development tool to edit, inspect and generate XPath and CSS selectors. ChroPath makes it easy to write, edit, extract, and evaluate XPath and CSS queries on any webpage and saves atleast 40–50% manual effort in automation script writing.

Browser Support: Chrome/Mozilla

Features:

1. **Generate locators in seconds:** Inspect the element or click on dom node to generate unique locators
2. **iframe Support:** Generate and verify locators inside iframe, Dynamic ID and Dark Theme Support.
3. **Editor:** Verify locators and get all matching node extracted in one place.
4. Verify and Modify Locators.
5. You can record multiple selectors and store locators and export data in xls file.

How to use ChroPath:

It mainly involves four steps:

1. Add ChroPath extension to Google Chrome/Mozilla Firefox.
2. Inspect element and choose ChroPath.
3. Copy the required XPath and CSS Selector from ChroPath menu.
4. Use the copied XPath/CSS in your selenium script and execute your program.

- ✓ **What is MRI?**

MRI is a free cross browser tool that lets you test selectors with any web page. Selectors, particularly complex ones can be difficult to get exactly right - MRI lets you experiment with them on any web page (local or online, static or dynamic).

Click Here to get MRI

Click Detailed Instructions how to use MRI.

Click to know More about MRI

- ✓ **What is SelectorsHub?**

This article is shared by Mr. Sanjay Kumar. In this post we will cover the following points about SelectorsHub:

- What is SelectorsHub?
- How to install & use SelectorsHub?
- Why to use SelectorsHub
- ShadowDOM Support
- Proper error message
- Iframe support
- SVG element support

What is SelectorsHub?

SelectorsHub helps to write and verify the XPath and cssSelector. SelectorsHub is the complete new way to write and verify the XPath and cssSelectors.

SelectorsHub auto suggests all attributes, text and everything along with their occurrences to complete Selectors quickly. Now you need not to copy and paste attribute values from DOM anymore to build XPath and cssSelector. It also supports shadowDOM, iframe and SVG elements. It gives the proper error message like what is wrong in your xpath and cssSelector.

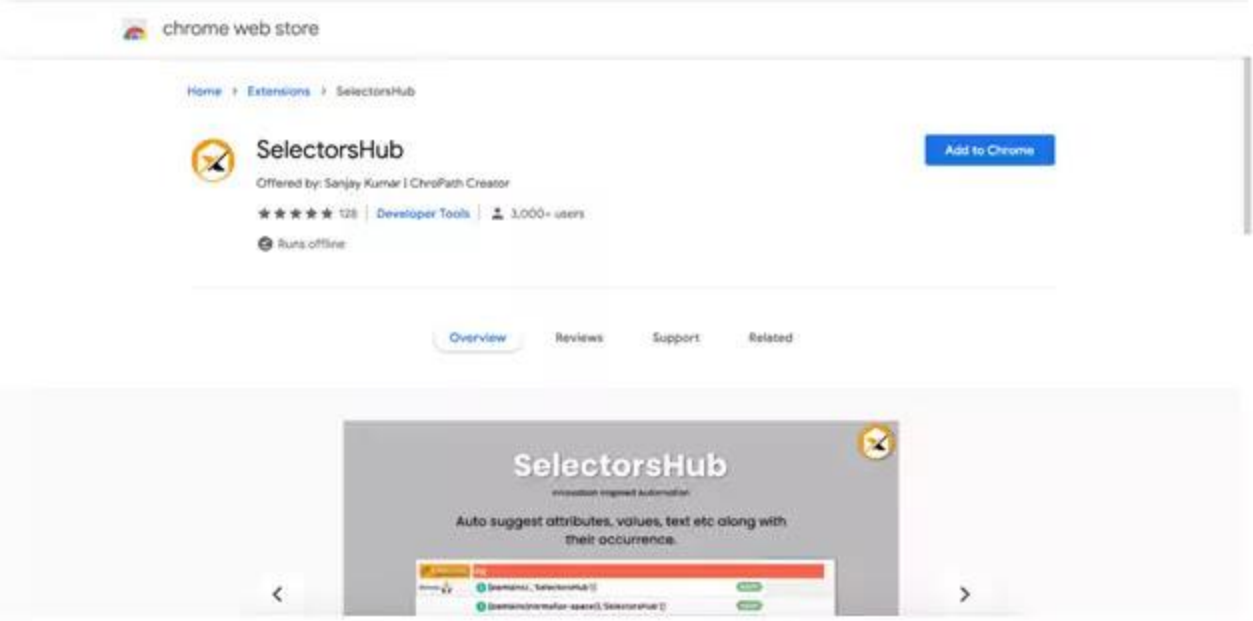
How to install & use SelectorsHub?

Find the download link here. Try it now it's absolutely FREE.

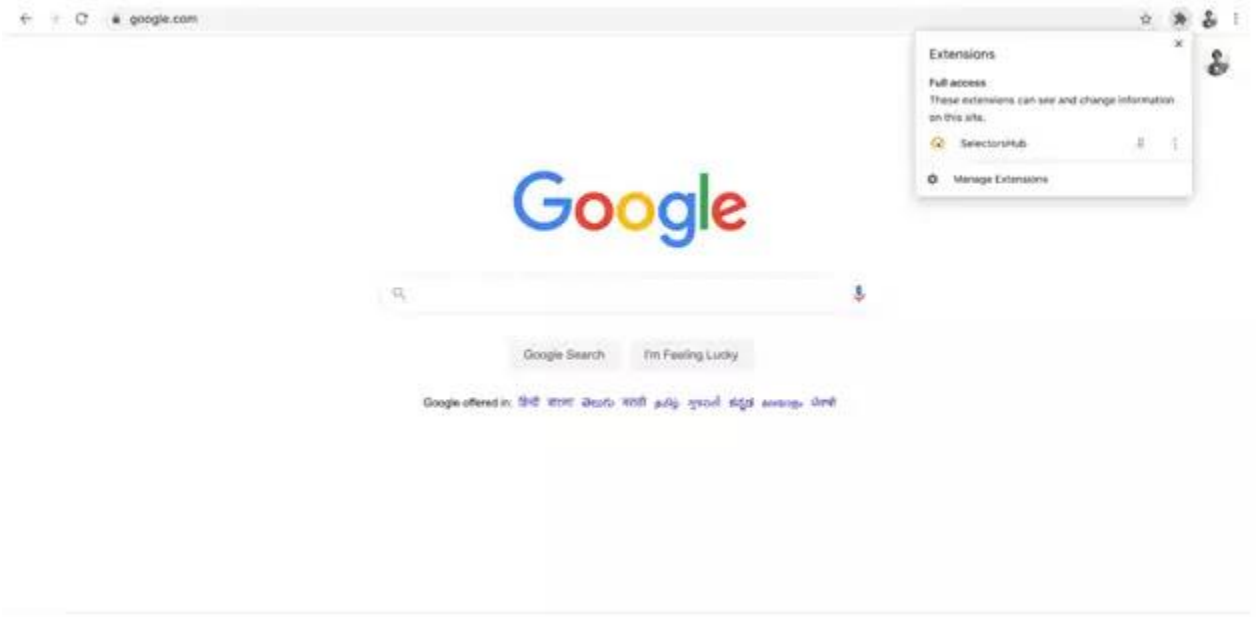
https://www.selectorshub.com/

1. Click on the download link for whichever browser you want.

2) Click on Add to Chrome.

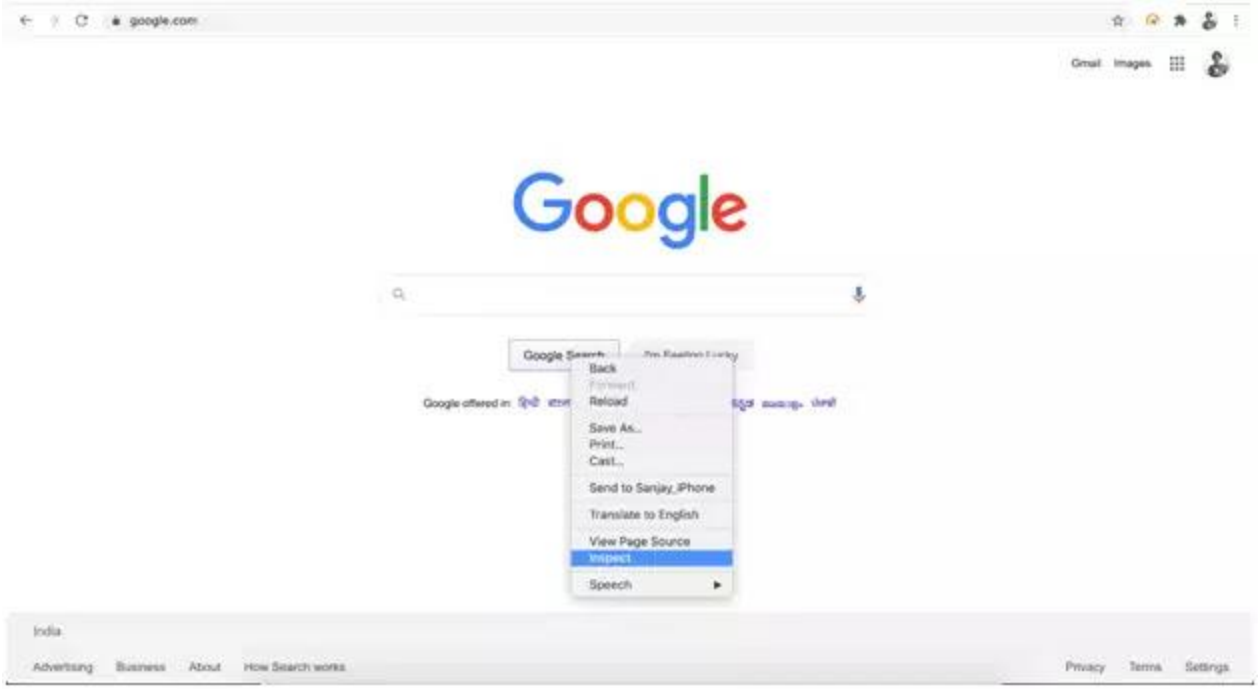


3) After adding the extension, it will show in the browser toolbar like this. You can pin to the toolbar by clicking on the pin icon.



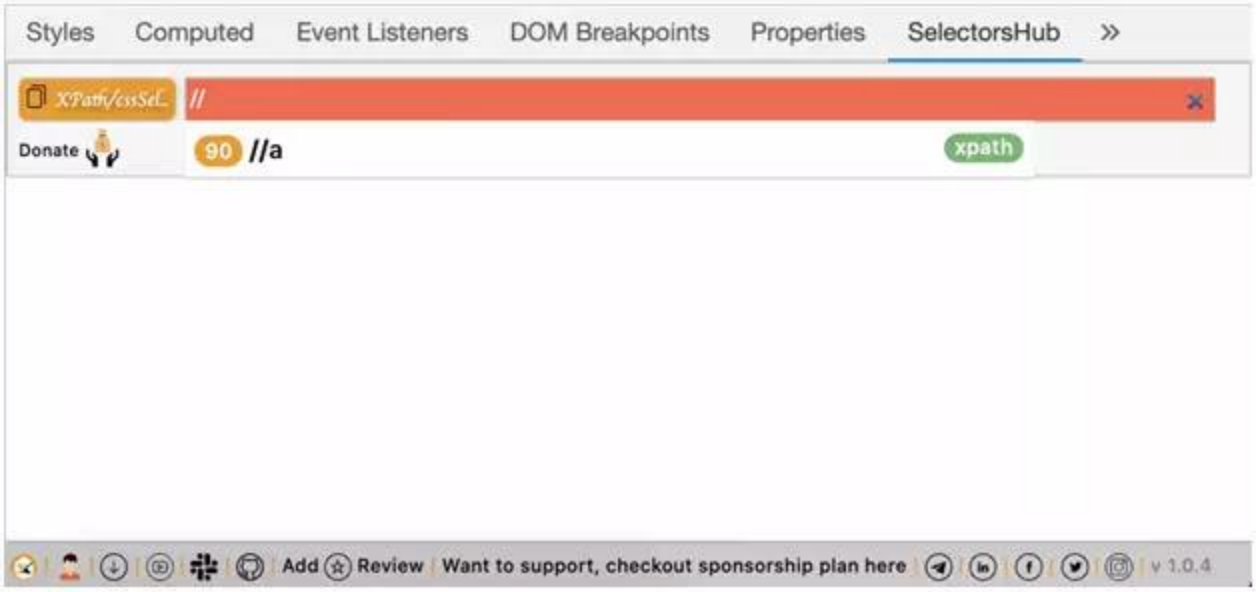
4) After adding the extension, restart the browser.

5) Now open DevTools by right clicking on any element and clicking on inspect.



6) On the right side of the Elements tab, SelectorsHub will be the last tab as shown in below image. If not visible, expand the sidebar or click on the two arrow icons as shown in below gif.

7) Now here you start typing your xpath or cssSelector. You will get auto suggest for inspected element.



Why to use SelectorsHub while there are so many other good XPath tools & selectors tools?

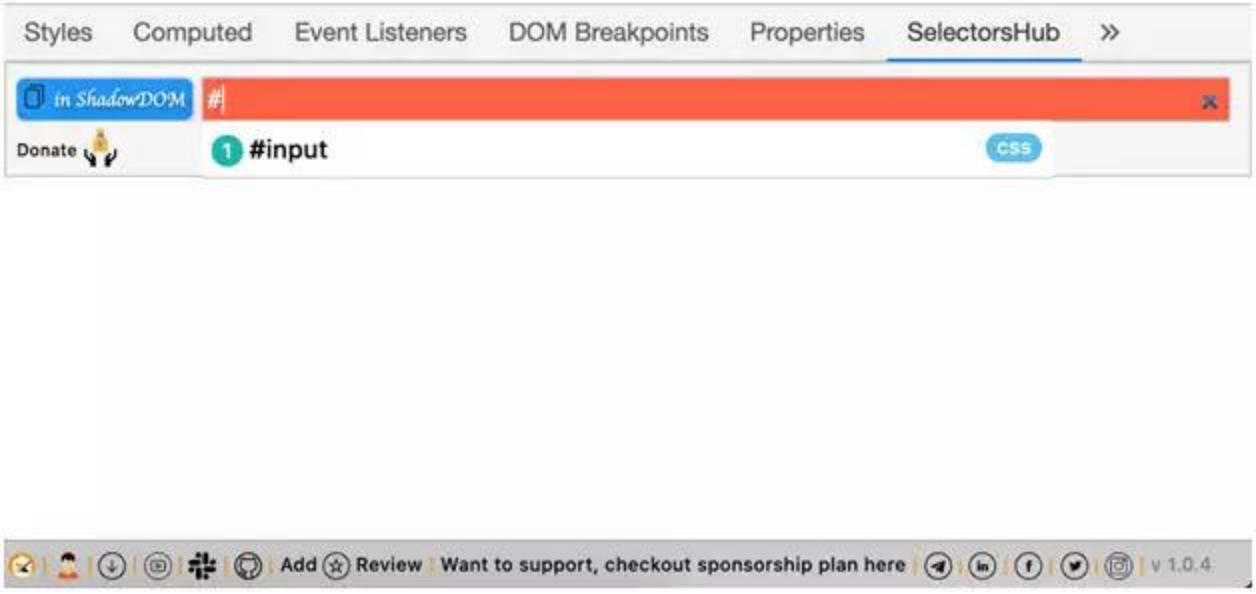
Biggest reason is, SelectorsHub helps to improve XPath and cssSelector writing skills.

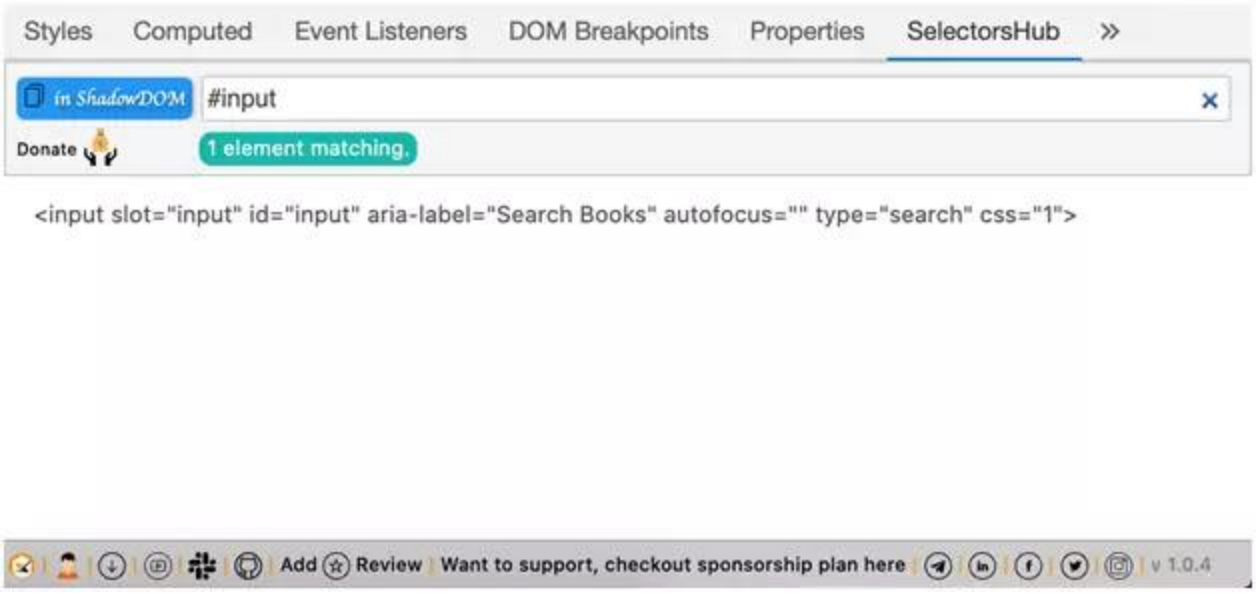
Not one reason, there are many reasons which makes SelectorsHub the unique and best xpath tool.

1. SelectorsHub is the only tool which made it possible to write own selectors in less than 5sec with its auto suggest feature without compromising learning skills.
2. SelectorsHub is the only tool which supports #shadowDOM, in fact even Chrome DevTools doesn't support shadowDOM.
3. SelectorsHub is the only tool which gives the proper error message for the missing elements in your selectors.
4. It helps you to improve your xpath and cssSelectors writing skills.
5. It has iframe support.
6. It supports svg elements.
7. It supports dark theme.

ShadowDOM Support

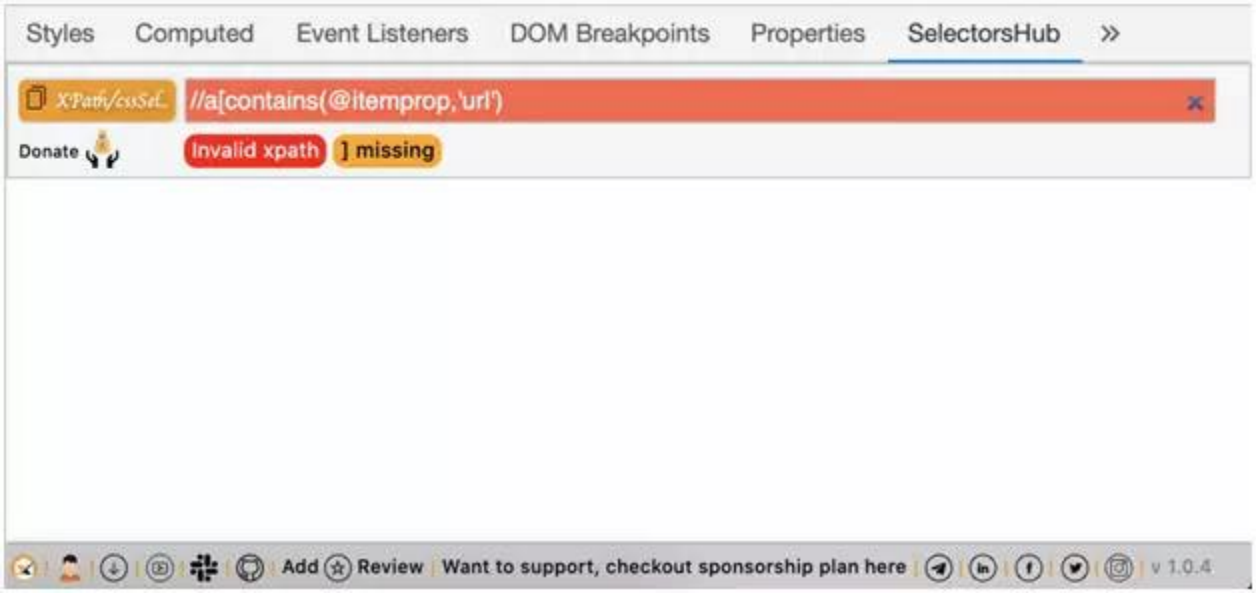
It was never possible to verify and write the cssSelectors for shadowDOM elements but this amazing innovation made it possible.





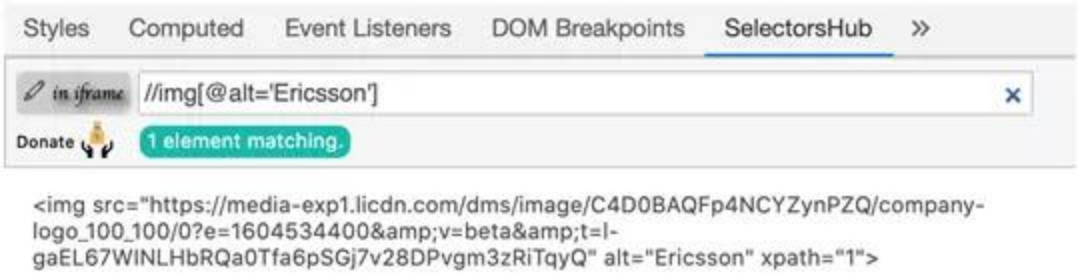
Proper error message for missing elements in selectors

Earlier there was nothing which can tell us what's wrong in our selector. We were never able to understand what's wrong in our selector until we are not expert in it. Devtools suggests 0 matching node for wrong selector pattern. But now we have the Selector's guide which helps us with the correct error message and what is wrong or missing in our selector.



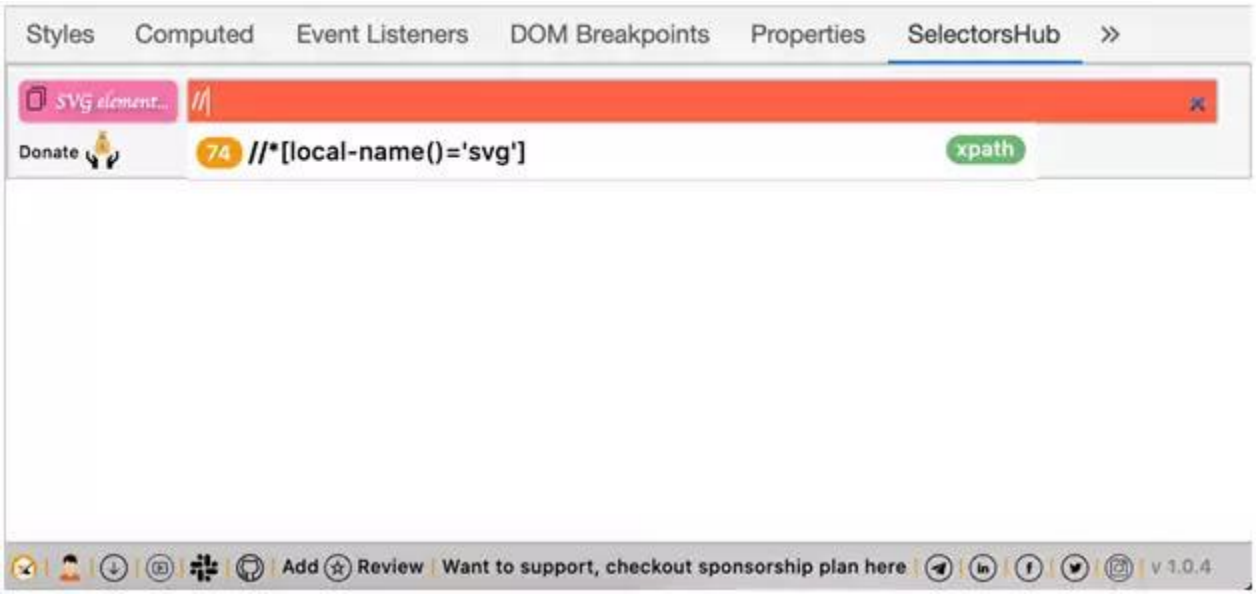
Iframe support

It supports iframes as well. It will let you know if the inspected element is inside an iframe or not. Now we can easily write and verify selectors for elements inside an iframe without wasting any time.



SVG element support

Many of us are not aware that svg elements don't support the standard xpath format. SVG elements support different xpath patterns. SelectorsHub has made it easy and let you know the correct format and helps you to learn how to write selectors for svg elements.



For more details and to make the best use of SelectorsHub please follow the video tutorials available here.
https://www.selectorshub.com/videotutorials/

WEB DRIVER Commands and CONTROLS:

- **Different WebDriver Commands**

We can divide WebDriver Commands into three categories:

- Browser Specific Commands
- Browser Navigation Commands and,

- Element Specific Commands

In this post we will discuss about 25 WebDriver commands which are used frequently in Selenium Automation:

Browser Specific Commands:

1. Maximize window:
To maximize a browser window, we need to call the maximize() method of the Window interface of the driver class. Add the second line right below where you define an instance of FirefoxDriver.
Example:
driver = new FirefoxDriver();
driver.manage().window().maximize();

2. Delete cookies
Delete all the cookies for the current domain using deleteAllCookies() method.
Example:
driver.manage().deleteAllCookies();

Deleting the specific cookie with cookie name "--abcd"
driver.manage().deleteCookieNamed("__abcd");

3. Get command:
The command launches a new browser and opens the specified URL in the browser instance
The command takes a single string type parameter that is usually a URL of application under test.
Example: driver.get("https://google.com");

4. GetTitle:
The command is used to retrieve the title of the webpage the user is currently working on. A null string is returned if the webpage has no title.
The command doesn't require any parameter and returns a trimmed string value.
Example: String title = driver.getTitle();

5. GetCurrentUrl:
The command is used to retrieve the URL of the webpage the user is currently accessing
The command doesn't require any parameter and returns a string value.
Example: driver.getCurrentUrl();

6. GetPageCource:
The command is used to retrieve the page source of the webpage the user is currently accessing. The command doesn't require any parameter and returns a string value. The command can be used with various string operations like contains() to ascertain the presence of the specified string value.
Example:
boolean result = driver.getPageSource().contains("String to find");

7. Close:
WebDriver's close() method closes the web browser window that the user is currently working on or we can also say the window that is being currently accessed by the WebDriver. The command neither requires any parameter nor does it return any value.
Example:
driver.close();

8. Quit
Unlike close() method, quit() method closes down all the windows that the program has opened. Same as close() method, the command neither requires any parameter nor does it return any value.
Example:
driver.quit();

Please refer below youtube video on Browser Specific commands:

Browser Navigation Commands:

WebDriver provides some basic Browser Navigation Commands that allows the browser to move backwards or forwards in the browser's history.

1. Navigate To:
Method - String to(String arg0)
In WebDriver, this method loads a new web page in the existing browser window. It accepts String as parameter and returns void. The respective command to load/navigate a new web page can be written as:
Example:
driver.navigate().to("www.automationtestinginsider.com");

2. Backward:
Method - void back()

This method enables the web browser to click on the back button in the existing browser window. It neither accepts anything nor returns anything. The respective command that takes you back by one page on the browser's history can be written as:

Example:

driver.navigate().back();

3. Forward:

Method - void forward()

This method enables the web browser to click on the forward button in the existing browser window. It neither accepts anything nor returns anything. The respective command that takes you forward by one page on the browser's history can be written as:

Example:

driver.navigate().forward();

4. Refresh:

Method - void refresh()

In WebDriver, this method refresh/reloads the current web page in the existing browser window. It neither accepts anything nor returns anything. The respective command that takes you back by one page on the browser's history can be written as:

Example:

driver.navigate().refresh();

Please refer below youtube video on Browser Navigation commands:

Element Specific Commands:

1. Clear:

clear( ) predefined method of Selenium 'WebDriver' Class is used to clear the text entered or displayed in the text fields

Example: driver.findElement(By.id("userName")).sendKeys("Admin").clear();

2. Click:

The click command emulates a click operation for a link, button, checkbox or radio button. In Selenium Webdriver, execute click after finding an element.

Example: driver.findElement(By.id("button")).click();

3. getText:

The command is used to retrieve the inner text of the specified web element. The command doesn't require any parameter and returns a string value:

Example: String elementText = driver.findElement(By.id("Text")).getText();

4. IsSelected:
isSelected() is the method used to verify if the web element is selected or not.
Example: boolean ele= driver.findElement(By.id("button")).isSelected();

5. IsEnabled:
isEnabled() is the method used to verify if the web element is enabled or disabled within the webpage. isEnabled() is primarily used with buttons.
Example: boolean ele= driver.findElement(By.id("button")).isEnabled();

6. IsDisplayed:
isDisplayed() is the method used to verify a presence of a web element within the webpage. The method returns a “true” value if the specified web element is present on the web page and a “false” value if the web element is not present on the web page.
Example: boolean ele= driver.findElement(By.id("button")).IsDisplayed();

7. getAttribute:
The command is used to retrieve the value of the specified attribute. The command requires a single string parameter that refers to an attribute whose value we aspire to know and returns a string value. as a result.
Example: driver.findElement(By.id("findID")).getAttribute("value");

8. sendkeys:
This is a method for sending one or more keystrokes to the active window
Example: driver.findElement(By.id("userName")).sendKeys("Admin")

9. Submit:
submit() is used to click Button in Web page. Selenium Webdriver has one special method to submit any form and that method name Is submit(). You can use .click() method to click on any button.There is no restriction for click buttons. We can use .submit() method for only submit form after click on button. That means element's type = "submit" and button should be inside <form> tag, then only submit() will work.

10. getSize:
It will returns the "Dimension" object. If you want to get the width and Height of the specific element on the webpage then use "getsize()" method.

11. getCssValue: - getCssValue method in selenium fetches the value of a CSS property of an web element.

12. getLocation:
In Selenium WebDriver APIs, there is a method getLocation() which returns point, containing location of top left hand corner of the element. Let's say, it returns (x, y).

13. getTagname:
This method gets the tag name of this element. This accepts nothing as a parameter and returns a String value. Command – element.getTagName();

- **Handle Radio Buttons and Check-boxes in Selenium**

The main difference between Radio button and Checkbox is that,
using radio button we will be able to select only one option from the options available. whereas using checkbox, we can select multiple options.

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

Before selecting the Radio buttons or check boxes 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 mentioned verification can be done using predefined methods in Selenium:

- isDisplayed()
- isEnabled()
- isSelected()

isDisplayed()

WebElement maleRadioBtn = driver.findElement (By.xpath("//input[@type='radio' and @value='Male']"));

maleRadioBtn.isDisplayed // this returns a Boolean value, if it returns true then said radio button is present on the webpage or it returns False if not present.

isEnabled()

maleRadioBtn.isEnabled() // this returns a Boolean value, if it returns true then said radio button is enabled on the webpage or it returns False if not enabled.

isSelected()

maleRadioBtn.isSelected() // this returns a Boolean value, if it returns true then said radio button is selected or it returns False if not selected.

Keep a note that Above methods can be used while working with Check boxes.

Please refer the below code:

```
public class CheckBoxRadioDemo {

	public WebDriver driver;

	public void launch() throws InterruptedException {

		System.setProperty("webdriver.chrome.driver",
				"chromedriver.exe path");
		driver = new ChromeDriver();
		driver.manage().window().maximize();
		driver.get("https://www.automationtestinginsider.com/2019/08/student-
				registration-form.html");
		Thread.sleep(2000);
	}

	public void radioBtnDemo() throws InterruptedException {

		WebElement radioBtn = driver.findElement(By.xpath("//input[@type='radio' and
				@value='Male']"));
		radioBtn.click();
		System.out.println("Male radio btn is Selected: "+radioBtn.isSelected());
		Thread.sleep(2000);
	}

	public void checkBoxDemo(String hobby) throws InterruptedException {
```

```
            //WebElement chekBox=driver.findElement(By.xpath("//input[@type='checkbox'
                                                    and @value='Drawing']"));
            //chekBox.click();
            //Thread.sleep(2000);

            List<WebElement> list= driver.findElements(By.xpath
                        ("//input[@type='checkbox' and @name='Hobby']"));

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

                WebElement ele=list.get(i);
                String hobbies=ele.getAttribute("value");
                //System.out.println(hobbies);
                if(hobbies.contains(hobby)) {
                    ele.click();
                    Thread.sleep(2000);
                    break;
                }
            }

    }


    public void closeBrowser() {
            driver.close();
    }

    public static void main(String[] args) throws InterruptedException {
            CheckBoxRadioDemo obj = new CheckBoxRadioDemo();
            obj.launch();
            obj.radioBtnDemo();
  obj.checkBoxDemo("Others");
  //obj.closeBrowser();
    }

}
```

- **Handle Dropdown in Selenium (Single and multi dropdown list)**

In this article, you will learn how to handle drop-downs in Selenium WebDriver.

Select Class in Selenium WebDriver:

The 'Select' class in Selenium WebDriver is used for selecting and deselecting option in a dropdown. The objects of Select type can be initialized by passing the dropdown webElement as parameter to its constructor.

```
WebElement dropDownTest = driver.findElement(By.id("dropdown"));
Select dropDown = new Select(dropDownTest);
```

How to select an option from drop-down menu?

WebDriver provides three ways to select an option from the drop-down menu.

1. selectByIndex - It is used to select an option based on its index, beginning with 0.

2. selectByValue - It is used to select an option based on its 'value' attribute.

3. selectByVisibleText - It is used to select an option based on the text over the option.

Below is the complete example given, **Lets first look single dropdown list.**
With below example, we will understand:
--Different methods to select values from drop down
--How to get first selected item
--Print all values from dropdown

```
public class DropDownDemo {

    public WebDriver driver;

    public void launch() throws InterruptedException {

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

        driver = new ChromeDriver();

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

```
        driver.get("https://www.automationtestinginsider.com/2019/08/student-registration-
                                        form.html");

        Thread.sleep(2000);

    }

    public void dropDownDemo() throws InterruptedException {


        WebElement bDay = driver.findElement(By.id("Birthday_Day"));

        WebElement bMonth = driver.findElement(By.id("Birthday_Month"));

        WebElement bYear = driver.findElement(By.id("Birthday_Year"));

        Select selectDay = new Select(bDay);

        Select selectMonth = new Select(bMonth);

        Select selectYear = new Select(bYear);

        selectDay.selectByIndex(4); // 4

        Thread.sleep(1000);

        selectMonth.selectByValue("February");

        selectMonth.selectByValue("January");

        Thread.sleep(1000);

        selectYear.selectByVisibleText("1990");

        Thread.sleep(1000);

        WebElement firstSelectedItem = selectMonth.getFirstSelectedOption();

        System.out.println(firstSelectedItem.getText());
```

```
        List<WebElement> monthList = selectMonth.getOptions();

        System.out.println("total Months: " + monthList.size()); // 13

        for (WebElement ele : monthList) {
            System.out.println(ele.getText());
        }
    }

    public void tearDown() {
        driver.close();
    }
    public static void main(String[] args) throws InterruptedException {
        DropDownDemo obj = new DropDownDemo();
        obj.launch();
        obj.dropDownDemo();
        obj.tearDown();
    }
}
```

Multi select dropdown list:

With below example, we will understand:
--Deselect method
--Method getAllSelectedOptions Print all selected options
--Compare with actual List with expected list

```
public class DropDownDemo1 {

    public WebDriver driver;

    public void launch() throws InterruptedException {

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

                " chromedriver.exe path");

        driver = new ChromeDriver();

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

        driver.get("https://www.seleniumeasy.com/test/basic-select-dropdown-demo.html");

        Thread.sleep(2000);

    }


    public void dropDownDemo() throws InterruptedException {

        WebElement multiSelectList = driver.findElement(By.id("multi-select"));

        Select selectCity = new Select(multiSelectList);

        selectCity.selectByIndex(0); // 4

        Thread.sleep(1000);

        selectCity.selectByValue("Florida");

        // selectCity.deselectByValue("Florida");
```

```
    // selectCity.deselectByValue("Florida");

    // selectMonth.selectByValue("January");

    Thread.sleep(1000);

    selectCity.selectByVisibleText("New Jersey");

    Thread.sleep(1000);

    List<String> expectedList = new ArrayList<>();

    expectedList.add("California");

    expectedList.add("New Jersey");

    expectedList.add("Florida");

    List<String> actualList = new ArrayList<>();

    List<WebElement> allSelectedoptions = selectCity.getAllSelectedOptions();

    for (WebElement ele : allSelectedoptions) {

       actualList.add(ele.getText());

       System.out.println(ele.getText());

    }

    Collections.sort(expectedList);

    Collections.sort(actualList);

    boolean result = actualList.equals(expectedList);

    System.out.println("The result of Comparison is:" + result);

}
```

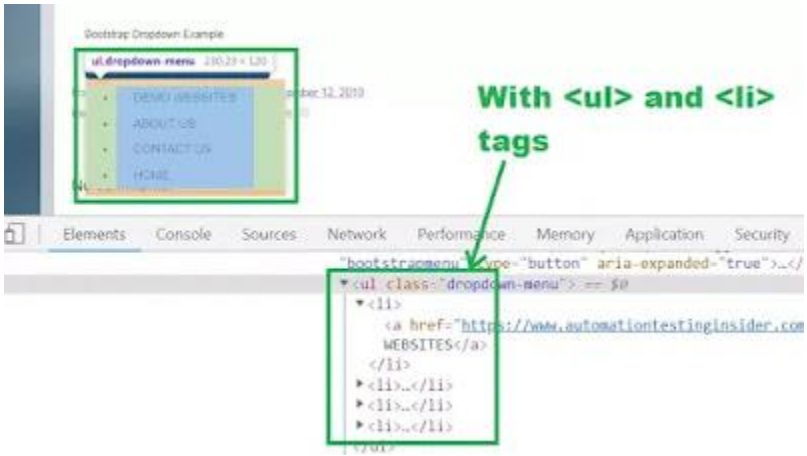
```
    public void tearDown() {
        driver.close();
    }
    public static void main(String[] args) throws InterruptedException {
        DropDownDemo1 obj = new DropDownDemo1();
        obj.launch();
        obj.dropDownDemo();
        obj.tearDown();
    }
}
```

- **Handle Bootstrap Dropdown in Selenium**

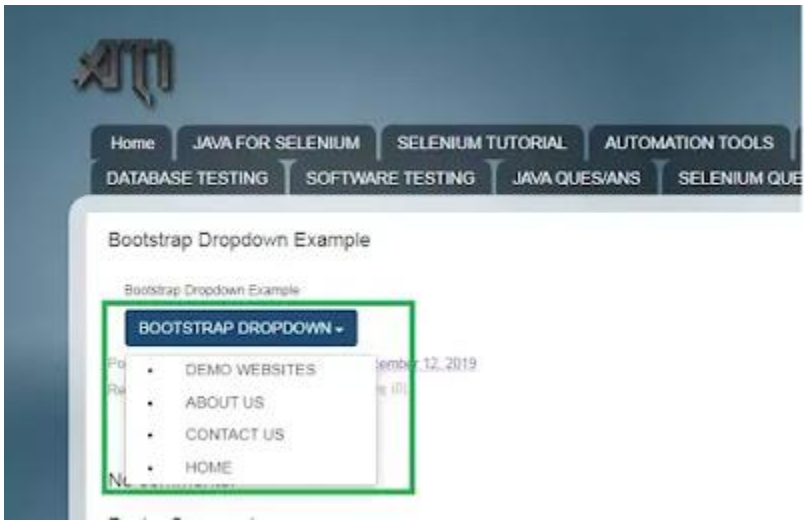
What is Bootstrap?
Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing responsive, mobile-first websites.

Refer below links to know more about bootstrap.
https://www.w3schools.com/bootstrap/default.asp
https://getbootstrap.com/docs/4.0/components/dropdowns/

Bootstrap dropdown
Bootstrap dropdown is not like traditional dropdowns as it doesn't use <select> tags, rather it uses <ul> and <li> tags to make a dropdown.



Bootstrap Dropdown UI



Inspecting Bootstrap Dropdown

So to handle bootstrap dropdown, you need to use findElements() method which will return all the options of the dropdown.
Following code shows how to handle a bootstrap dropdown in Selenium WebDriver:

```
public class BootstrapDropDown {

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

        WebDriver driver;
        System.setProperty("webdriver.chrome.driver"," chromedriver.exe path");

        driver = new ChromeDriver();

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

```
        driver.get("https://www.automationtestinginsider.com/2019/12/bootstrap-dropdown-example_12.html");

        Thread.sleep(2000);

        driver.findElement(By.xpath("//button[@id='bootstrapmenu']")).click();

        List <WebElement> options=driver.findElements(By.xpath("//ul[@class='dropdown-menu']//li/a"));


        for(WebElement ele:options) {

            String value=ele.getText();

            System.out.println(value);

            if(value.equalsIgnoreCase("contact us")) {

                ele.click();

                break;

            }

        }



        Thread.sleep(2000);

        driver.close();

    }

}
```

- **Actions Class in Selenium**

Actions Class in Selenium

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.

Some Examples: Mouse double click, Drag and Drop, Handling tooltip, Mouse hover, entering uppercase letters in the textbook using Shift+Letters .

Actions Class: Actions class is an API for performing complex user web interactions like double-click, right-click, etc. and it is the only choice for emulating Keyboard and Mouse interactions.

Actions class implements the builder pattern. Builder Pattern is one of the design patterns. The intent of the Builder design pattern is to separate the construction of a complex object from its representation.

What is the difference between Actions Class and Action Interface in Selenium?

Actions is a class that is based on a builder design pattern. This is a user-facing API for emulating complex user gestures.

Whereas Action is an Interface which represents a single user-interaction action. It contains one of the most widely used methods perform().

1. clickAndHold() -- Clicks (without releasing) at the current mouse location.
2. contextClick() - Performs a context-click at the current mouse location. (Right Click Mouse Action)
3. doubleClick() - Performs a double-click at the current mouse location.
4. dragAndDrop(source, target) - Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.
5. dragAndDropBy(source, x-offset, y-offset) - Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.
6. moveToElement(toElement) - Moves the mouse to the middle of the element.
7. release() - Releases the depressed left mouse button at the current mouse location
8. sendKeys(onElement, charsequence) - Sends a series of keystrokes onto the element.
9. keyDown(modifier_key) - Performs a modifier key press. Does not release the modifier key - subsequent interactions may assume it's kept pressed.

10. keyUp(modifier _key) - Performs a key release.

Selenium has the built-in ability to handle various types of keyboard and mouse events. In order to do action events, you need to use org.openqa.selenium.interactions Actions class. The user-facing API for emulating complex user gestures. Use the selenium actions class rather than using the Keyboard or Mouse directly. This API includes actions such as drag and drop, clicking multiple elements.

To create an object 'action' of Selenium Actions class, we use below command:
Actions action=new Actions(driver);

To focus on element using WebDriver:
action.moveToElement(element).perform();
element is the webelement which we capture

perform() method is used here to execute the action.

To click on the element:
action.moveToElement(element).click().perform();
click() method is used here to click the element.

- **How to Perform double click and right click**

Double click action in Selenium web driver can be done using Actions class.

Right click operation is mostly used when performing right click on an element opens a new menu. Right click operation in Selenium web driver can be done using the pre-defined command Context Click

```
Actions action = new Actions(driver);
WebElement link = driver.findElement(By.ID ("Element ID"));
action.contextClick(link).perform();
```

Double click operation is used when the state of web element changes after double click operation. Double Click operation in Selenium web driver can be done using the pre defined command Double Click as mentioned below

```
Actions action = new Actions(driver);
WebElement link = driver.findElement(By.ID ("Element ID"));
action. doubleClick (link).perform();
```

Please refer below Program to understand how to perform double click and right click:

```
import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;


public class DoubleAndRightClick {

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


  WebDriver driver;

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

  driver = new ChromeDriver();

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

  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-defines-multi.html");

  Thread.sleep(1000);

  WebElement doubleClickBtn= driver.findElement(By.id("doubleClickBtn"));

  WebElement rightClickBtn= driver.findElement(By.id("rightClickBtn"));

  Actions act= new Actions(driver);

  act.doubleClick(doubleClickBtn).perform();
```

```
 System.out.println("Double clicked");

 Thread.sleep(2000);

 driver.switchTo().alert().accept();

 Thread.sleep(2000);

 act.contextClick(rightClickBtn).perform();

 Thread.sleep(2000);

 driver.close();

 }

}
```

- **How to handle drag and drop in Selenium**

he basic sequence involved in drag and drop is:
Move the pointer to the object.
Press, and hold down, the button on the mouse or other pointing device, to "grab" the object.
"Drag" the object to the desired location by moving the pointer to this one.
"Drop" the object by releasing the button.

Please refer below Program to understand how to perform drag and drop:

```
import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

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

```
public class DragAndDropDemo {

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

  WebDriver driver;

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

  driver = new ChromeDriver();

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

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

  Thread.sleep(1000);


  WebElement from = driver.findElement(By.id("draggable"));

  WebElement to = driver.findElement(By.id("droppable"));

  Actions act = new Actions(driver);

  // act.dragAndDrop(from, to).perform();

  // act.dragAndDropBy(from, 133, 22).perform();

  act.clickAndHold(from).moveToElement(to).release().build().perform();


  WebElement droppedMsg = driver.findElement(By.xpath("//div/p[text()='Dropped!']"));

  if (droppedMsg.isDisplayed()) {

   System.out.println("Success");

   System.out.println("Message is: "+droppedMsg.getText());

  } else {
```

```
 System.out.println("Failed");
 }
 Thread.sleep(1000);
 // driver.close();
 }
}
```

- **How to handle tooltip in Selenium**

In many web pages, it's very common that when hovered over some link, text, some text or sometimes image gets displayed. For example, in Gmail inbox, if hovered over right-hand side menu options, small "hover box" with text gets displayed. This is called a tooltip of web element.

This text usually is a brief description of the functionality of the object or in some cases, it displays a detailed description of the object. In some cases, it may display only the full name of the object. So basically, the purpose of a tooltip is to provide some hint to the user about the object. In many instances, it is required to verify if this text description being displayed as expected.

Please refer below Program to understand how to handle tooltip in Selenium:

Program1 (without actions class when title attribute present for the element):

```
import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;


public class ToolTipDemo {
```

```
 public static void main(String[] args) throws InterruptedException {
  WebDriver driver;
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/");
  Thread.sleep(1000);
  WebElement searchBox=driver.findElement(By.xpath("//input[@class='gsc-input']"));
  String tooltipText=searchBox.getAttribute("title");
  System.out.println("Tooltip text is: "+tooltipText);
  driver.close();
 }
}
```

Program2(using Actions Class):

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;

public class ToolTipDemo1 {
```

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

 WebDriver driver;

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

  " chromedriver.exe path");

 driver = new ChromeDriver();

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

 driver.get("http://demoqa.com/tooltip-and-double-click/");

 Thread.sleep(1000);

 Actions act= new Actions(driver);

 WebElement mouseHover=driver.findElement(By.id("tooltipDemo"));

 act.moveToElement(mouseHover).perform();

 WebElement tooltipMsg=driver.findElement(By.xpath("//span[@class='tooltiptext']"));


 System.out.println("Tooltip Message is: "+tooltipMsg.getText());


 driver.close();

 }

}
```

- **Multi Select Actions in Selenium**

Please refer below Program to understand how to handle Multi Select Actions in Selenium:

```
import java.util.List;
```

```
import org.openqa.selenium.By;

import org.openqa.selenium.Keys;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Action;

import org.openqa.selenium.interactions.Actions;

public class MultiSelectActions {

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


  WebDriver driver;

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

  driver = new ChromeDriver();

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

  driver.get("https://www.seleniumeasy.com/test/jquery-dual-list-box-demo.html");

  Thread.sleep(1000);

  Actions act = new Actions(driver);


  List<WebElement>list=driver.findElements(By.xpath

   ("//select[@class='form-control pickListSelect pickData']/option"));

  Action actions=act.keyDown(Keys.CONTROL)
```

```
  .click(list.get(0))

  .click(list.get(2))

  .click(list.get(4))

  .click(list.get(5))

  .keyUp(Keys.CONTROL)

  .build();

  actions.perform();

  Thread.sleep(2000);

  driver.findElement(By.xpath("//div[@class='col-sm-2 pickListButtons']/button[1]"))

  .click();

  Thread.sleep(2000);

  driver.close();

 }
}
```

- **Mouse hover and Auto Suggestions in Selenium**

Mouse Hover Actions in Selenium Webdriver. In order to perform a 'mouse hover' action, we need to chain all of the actions that we want to achieve in one go. So move to the element that which has sub elements and click on the child item. It should the same way what we do normally to click on a sub menu item.

Please refer below Program to understand how to perform mouse hover:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;
```

```
public class MousehoverDemo {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://s1.demo.opensourcecms.com/wordpress/wp-login.php");
  Thread.sleep(1000);

  Actions act = new Actions(driver);

  driver.findElement(By.id("user_login")).sendKeys("opensourcecms");
  driver.findElement(By.id("user_pass")).sendKeys("opensourcecms");
  driver.findElement(By.id("wp-submit")).click();
  Thread.sleep(1000);

  WebElement logoutOption = driver.findElement(By.xpath("//a[text()='Howdy, ']"));

  act.moveToElement(logoutOption).perform();

  driver.findElement(By.xpath("//a[@class='ab-item'][text()='Log Out']")).click();

  Thread.sleep(2000);
  driver.close();

 }

}
```

Please refer below Program to understand how to handle Auto Suggestions in Selenium:

```
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Autosuggestion {
```

```
public static void main(String[] args) throws InterruptedException {
 WebDriver driver;
 System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
 driver = new ChromeDriver();
 driver.manage().window().maximize();
 driver.get("https://www.google.com/");
 Thread.sleep(2000);

 driver.findElement(By.name("q")).sendKeys("Selenium");
 Thread.sleep(3000);

 List<WebElement> list = driver.findElements(By.xpath("//ul/li[@class='sbct']"));

 for(int i = 0 ;i< list.size();i++)
 {
  System.out.println(list.get(i).getText());

  String searchText=list.get(i).getText();

  if(searchText.equals("selenium testing"))
  {
   list.get(i).click();
   break;
  }
 }

 driver.close();
}

}
```

ALERT, WINDOWS and MANY MORE:

- **Alert Interface and Handle JavaScript Alerts in Selenium**

What is an Alert?

- Alert is a small message box which displays on-screen notification to give the user some kind of information or ask for permission to perform certain kind of operation. It may be also used for warning purpose.

- There are many user actions that can result in an alert on screen. For e.g. user clicked on a button that displayed a message or may be when you entered a form, HTML page asked you for some extra information. So in this video we will learn Handling of Alerts, JavaScript Alerts and PopUp Boxes.

There are two types of alerts that we would be focusing on majorly:

- Web-based alert pop-ups
- Windows-based alert pop-ups

Handling windows based pop-ups is beyond Web Driver's capabilities, thus we would exercise some third-party utilities to handle window pop-ups.

Types of Web Based/Java Script Alerts
Java scrip provides mainly following three types of alerts:

1. Simple alert("This is a simple alert");
2. confirmation alert- confirm("Confirm pop up with OK and Cancel button");
3. prompt- prompt("Do you like ATI?", "Yes/No");

There are the four methods that we would be using along with the Alert interface.

- void dismiss() – The dismiss() method clicks on the “Cancel” button as soon as the pop up window appears.
- void accept() – The accept() method clicks on the “Ok” button as soon as the pop up window appears.
- String getText() – The getText() method returns the text displayed on the alert box.
- void sendKeys(String stringToSend) – The sendKeys() method enters the specified string pattern into the alert box.

Simple Alert Program:

```
import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class SimpleAlertDemo {

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

```
  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-
defines-multi.html");
  Thread.sleep(2000);
  driver.findElement(By.id("simpleAlert")).click();
  Alert alert=driver.switchTo().alert();
  String alertText=alert.getText();
  System.out.println("Alert Text is: "+alertText);
  Thread.sleep(2000);
  alert.accept();
  driver.close();
 }
}
```

Output:

```
Alert Text is: Simple Alert

This is a Simple Alert
```

Confirmation Alert Demo:

```
import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.NoAlertPresentException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class ConfirmationAlertDemo {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
```

```
 driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-defines-multi.html");
  // Thread.sleep(2000);

  driver.findElement(By.id("confirmationAlert")).click();

  if (isAlertPresent(driver)) {
   System.out.println("Alert is present");
  } else {
   System.out.println("Alert is not present");
  }
  driver.close();
 }

 public static boolean isAlertPresent(WebDriver ldriver) {

  try {
   Alert alert = ldriver.switchTo().alert();
   String alerteTxt = alert.getText();
   alert.accept();
   //alert.dismiss();
  } catch (NoAlertPresentException e) {
   // TODO Auto-generated catch block
  }
  return false;
 }
}
```

Output:
Alert is present

Prompt Alert Demo:

```
import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class PromptAlertDemo {

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

```
  WebDriver driver;
  System.setProperty("webdriver.gecko.driver",
    " geckodriver.exe pth");
  driver = new FirefoxDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-
defines-multi.html");
  Thread.sleep(3000);
  driver.findElement(By.id("promptAlert")).click();
  Alert alert = driver.switchTo().alert();
  Thread.sleep(2000);
  String alertText = alert.getText();
  System.out.println("Alert Text is: " + alertText);
  alert.sendKeys("Yes");
  Thread.sleep(2000);
  alert.accept();
  driver.close();
 }
}
```

Output:

Alert Text **is:** Do you like ATI?

- **AutoIT tool and two ways to handle windows Authentication**

What is AutoIt tool?
AutoIt V3 is a freeware tool which is used for automating anything in Windows environment. AutoIt script is written in a BASIC language. It can simulate any combination of keystrokes, mouse movement and window/control manipulation.

Selenium Usage

- Handle window GUI and non HTML popups
- Upload files

Download AutoIT and AutoIT editor
https://www.autoitscript.com/site/autoit/downloads/

Handle Windows Authentication using AutoIT tool.

```
import java.io.IOException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class HandleWindwowAuthentication {
 public static void main(String[] args) throws IOException, InterruptedException {
  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("http://the-internet.herokuapp.com/basic_auth");
  Thread.sleep(2000);
  Runtime.getRuntime().exec("Authen.exe (file path)");
  Thread.sleep(2000);
       //driver.close();
 }
}
```

2nd way to Handle Windows Authentication:

```
import java.io.IOException;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
public class HandleWindwowAuthentication1 {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("http://admin:admin@the-internet.herokuapp.com/basic_auth");
       //driver.close();

 }

}
```

Different ways to upload and Download file in Selenium

There are different ways to upload files in Selenium like:

- Using sendkeys() method
- Using AutoIT tool
- Using Robot Class

Using sendKeys() method:
Uploading files in WebDriver is done by simply using the sendKeys() method on the file-select input field to enter the path to the file to be uploaded.
Let's say we wish to upload the file "C:\\Users\\Hitendra\\Desktop\\TestFile.txt". Our WebDriver code should be like the one shown below.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;

public class UploadFileUsingSendkeys {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-defines-multi.html");
  Thread.sleep(2000);
  Actions act= new Actions(driver);
  WebElement chooseFile=driver.findElement(By.id("fileupload1"));
  act.moveToElement(chooseFile).perform();
  Thread.sleep(1000);
  chooseFile.sendKeys("TestFile.txt (file path)");
 }
}
```

Using AutoIT tool:
(refer YouTube video on Introduction to AutoIT tool -- https://www.youtube.com/watch?v=be05PN2XW3k&list=PLsGOlyTzNH6f_pZuGZDFjI5IeCWGQqiwF&index=32)

We need three tools in order to upload file using selenium.

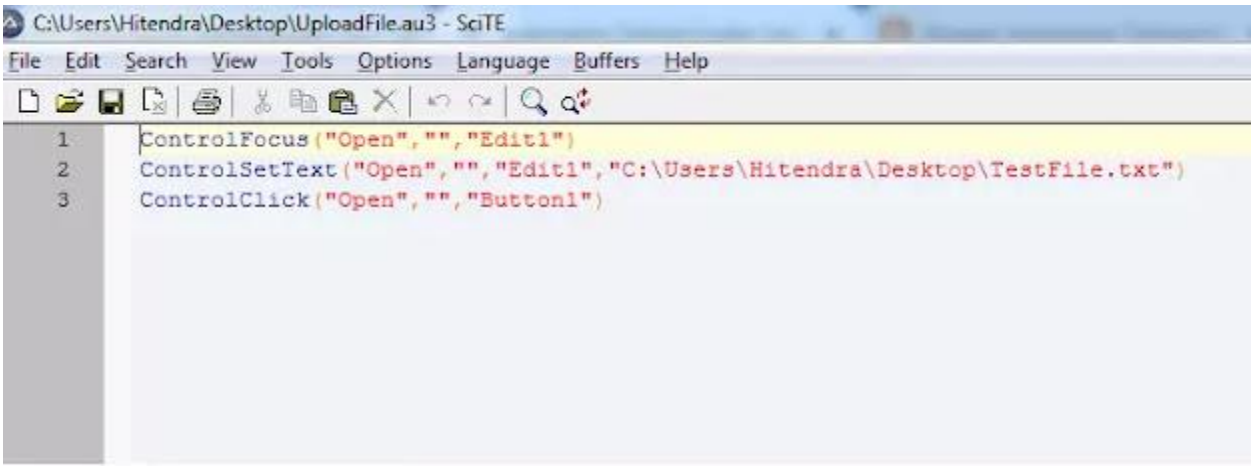
- Selenium Webdriver
- AutoIT editor and element identifier
- The window that you want to automate

Open AutoIT script editor,and start writing a script for selecting a file to upload.

There are lots of method available which we can use in a script according to the requirement, but right now we will focus on the below methods as these methods are required for writing file upload script:

ControlFocus(" title "," text ",controlID) //Sets input focus to a given control on a window.
ControlSetText(" title "," text ",controlID ," File path which need to upload ") // Sets text of a control.
ControlClick(" title "," text ",controlID) //Sends a mouse click command to a given control.

Refer below Image:



Sample AutoIT Script
Please refer below YouTube video (at the end of this article) on how to upload file using AutoIT tool.

Complete program to upload file using AutoIT tool in Selenium:

```
import java.io.IOException;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;

public class UploadUsingAutoIT {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-
defines-multi.html");
  Thread.sleep(2000);
  Actions act= new Actions(driver);
  WebElement chooseFile=driver.findElement(By.id("fileupload1"));
  act.moveToElement(chooseFile).click().perform();
  Thread.sleep(2000);
  Runtime.getRuntime().exec("UploadFile.exe (file path)");
 }
}
```

Using Robot Class:

We have discussed uploading a file using using Webdriver Sendkeys method and Using AutoIT Tool. Now here we will look into an other way of doing file upload using Robot class and StringSelection class in java.

In certain Selenium Automation Tests, there is a need to control keyboard or mouse to interact with OS windows like Download pop-up, Alerts, Print Pop-ups, etc. or native Operation System applications like Notepad, Skype, Calculator, etc.

Selenium Webdriver cannot handle these OS pop-ups/applications. In Java version 1.3 Robot Class was introduced. Robot Class can handle OS pop-ups/applications.

Robot class is a separate class in Java which will allow us to perform multiple tasks based on our requirement. It generally will throw AWT exception so we need to deal with it accordingly.
Using Robot class, we can simulate keyboard event in Selenium. To use keyboard events you have to use to a method of Robot class.
Robot class is used to (generate native system input events) take the control of mouse and keyboard. Once you get the control, you can do any type of operation related to mouse and keyboard through with java code.

There are different methods which robot class uses. Here in the below example we have used 'keyPress' and 'keyRelease' methods.

keyPress - takes keyCode as Parameter and Presses here a given key.
keyrelease - takes keyCode as Parameterand Releases a given key

We will follow below steps:
Step 1- We have to copy the file location in system clipboard.
Step 2- We have to click on upload button and use CTR+V and ENTER.
Note- Robot class each key has to press and release respectively

Complete program given below

```
import java.awt.AWTException;
import java.awt.Robot;
import java.awt.Toolkit;
import java.awt.datatransfer.StringSelection;
import java.awt.event.KeyEvent;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.interactions.Actions;

public class UploadFileUsingRobot {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
```

```
 driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-defines-multi.html");
 Thread.sleep(2000);
 Actions act= new Actions(driver);
 WebElement chooseFile=driver.findElement(By.id("fileupload1"));
 act.moveToElement(chooseFile).click().perform();
 Thread.sleep(2000);
 uploadFile("TestFile.txt (file path)");
 }

 public static void setClipBoard(String file) {
 StringSelection obj= new StringSelection(file);
 Toolkit.getDefaultToolkit().getSystemClipboard().setContents(obj, null);
 }

 public static void uploadFile(String filePath) throws AWTException {
 setClipBoard(filePath);
 Robot rb= new Robot();
 rb.keyPress(KeyEvent.VK_CONTROL);
 rb.keyPress(KeyEvent.VK_V);
 rb.keyRelease(KeyEvent.VK_CONTROL);
 rb.keyRelease(KeyEvent.VK_V);
 rb.keyPress(KeyEvent.VK_ENTER);
 rb.keyRelease(KeyEvent.VK_ENTER);
 }
}
```

DownLoadFile file using ChromeOptions Class: Please refer below program to download file on chrome browser using ChromeOptions Class.

```
import java.util.HashMap;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.remote.CapabilityType;

public class DownLoadFile {
 static WebDriver driver;
 public static void main(String[] args) {
 System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
```

```
        String downloadFilepath = "C:\\Users\\Hitendra\\Downloads";
        HashMap<String, Object> chromePrefs = new HashMap<String, Object>();
        chromePrefs.put("profile.default_content_settings.popups", 0);
        chromePrefs.put("download.default_directory", downloadFilepath);
        ChromeOptions options = new ChromeOptions();
        options.setExperimentalOption("prefs", chromePrefs);
        options.addArguments("--test-type");
        options.addArguments("--disable-extensions"); //to disable browser extension popup
        options.setCapability(CapabilityType.ACCEPT_SSL_CERTS, true);
        driver = new ChromeDriver(options);
                driver.get("http://www.seleniumhq.org/download/");
                driver.findElement(By.linkText("32 bit Windows IE")).click();
 }
}
```

- **Different ways of Scrolling in Selenium WebDriver**

What is Scroll and Scroll bar?
A vertical or horizontal bar commonly located on the far right or bottom of a window that allows you to move the window viewing area up, down, left, or right
Selenium Webdriver does not require scroll to perform actions as it manipulates DOM. But in certain web pages, elements only become visible once the user have scrolled to them. In such cases scrolling may be necessary.

To scroll using Selenium, you can use JavaScriptExecutor interface that helps to execute JavaScript methods through Selenium Webdriver.

JavascriptExecutor js = (JavascriptExecutor) driver; js.executeScript(Script,Arguments);

- Scroll up or down by pixel

```
js.executeScript("window.scrollBy(1000,4000)")
```

Please refer below program:

```
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class ScrollUpDown {
```

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.get("https://www.automationtestinginsider.com/");
  Thread.sleep(2000);
  JavascriptExecutor js= (JavascriptExecutor) driver;
  js.executeScript("window.scrollBy(1000,0)");
 }
}
```

- Scroll down by Visibility of Element

```
js.executeScript("arguments[0].scrollIntoView();", ele);
```

Please refer below program:

```
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class ScrollDownByVisibilityOfEle {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver"," chromedriver.exe path");
  driver = new ChromeDriver();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-
defines-multi.html");
  Thread.sleep(2000);
  JavascriptExecutor js= (JavascriptExecutor) driver;
  WebElement ele=driver.findElement(By.id("simpleAlert"));
  js.executeScript("arguments[0].scrollIntoView()",ele );
 }
}
```

- Scroll down by end of the page

```
js.executeScript("window.scrollTo(0, document.body.scrollHeight)");
```

Please refer below program:

```
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class ScrollDownByEndOfthePage {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver"," chromedriver.exe path");
  driver = new ChromeDriver();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-defines-multi.html");
  Thread.sleep(2000);
  JavascriptExecutor js= (JavascriptExecutor) driver;
  js.executeScript("window.scrollTo(0, document.body.scrollHeight)");
 }
}
```

- **Handle multiple windows in Selenium WebDriver**

In this article, we will see how to handle multiple windows using Selenium WebDriver. In real time, we face many scenarios, where an application throws multiple popups. We can easily achieve this using Windows Handles in Selenium WebDriver. We use ‘Switch To’ method which allows us to switch control from one window to other.

What is a window handle?

A window handle is a unique identifier that holds the address of all the windows. This is basically a pointer to a window, which returns the string value. This window handle function helps in getting the handles of all the windows. It is guaranteed that each browser will have a unique window handle.

Methods and Commands:

get.windowhandle(): helps in getting the window handle of the current window
get.windowhandles(): helps in getting the handles of all the windows opened

set: helps to set the window handles which is in the form of a string.

set<string> set=driver.get.windowhandles();
switch to: helps in switching between the windows
action: helps to perform certain actions on the windows.

Please refer complete program below:

```
import java.util.Iterator;
import java.util.Set;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class WindowHandle1 {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  // To open Naukri.com with multiple windows
  driver.get("https://www.naukri.com/");
  Thread.sleep(2000);
  // It will return the parent window name as a String
  String parentWinID = driver.getWindowHandle();
  System.out.println("Parent Win ID is: " + parentWinID);
  // It returns no. of windows opened by WebDriver and will return Set of Strings
  Set<String> allWinID = driver.getWindowHandles();
  System.out.println("Total Window size:" +allWinID.size());
  System.out.println("All win IDs are:");
  for (String allIDs : allWinID) {
   System.out.println(allIDs);
  }
  // Using Iterator to iterate with in windows
  Iterator<String> itr = allWinID.iterator();
  while (itr.hasNext()) {
   String childWinID = itr.next();
   // Compare whether the main window is not equal to child window. If not equal, we will close
child windows.
            if(!parentWinID.equalsIgnoreCase(childWinID)) {
             driver.switchTo().window(childWinID);
             System.out.println("Child URL is:"+driver.getCurrentUrl());
```

```
            System.out.println("Child Win Title is:"+driver.getTitle());
            Thread.sleep(2000);
            driver.close();
           }
  }
  // This is to switch to the main window
  driver.switchTo().window(parentWinID);
  driver.quit();
 }
}
```

Please refer below YouTube video to get more details:
Below scenarios covered in this video:

- How to get total windows and print them.
- How to switch to child window and perform action.
- Come back (switch) to parent window.
- Create a Program (Library) which will handle multiple windows.
- Create a Program (Library) which will handle different tabs.
- How to switch through index.

Handling iFrames in Selenium Webdriver

What is an Iframe or frame?

An iframe is also known as the inline frame. It is a tag used in HTML5 to embed an HTML document within a parent HTML document. An iframe tag is defined using <iframe></iframe> tags.
IFrame is a web page which is embedded in another web page or an HTML document embedded inside another HTML document.

The frame enables a developer to split the screen horizontally or vertically by using the frameset tag. Frame and frameset tags are deprecated as they are no longer supported by HTML 5.

How to Identify the Iframe?

- Right click on the element, If you find the option like View Frame Source' then it is an iframe.
- From DOM tree. Search with tag ‘iframe’
- Right click on the page and click 'View Page Source' and Search with the 'iframe‘

- Through selenium locator tagName in the script -- By.tagName("iframe")

To work with different frames, we need to use the SwitchTo().frame method. This method enables the browser to switch between multiple frames. It can be implemented in the following ways:

switchTo.frame(int frame number): Defining the frame index number, the Driver will switch to that specific frame

switchTo.frame(string frameNameOrId): Defining the frame element or Id, the Driver will switch to that specific frame
switchTo.frame(WebElement frameElement): Defining the frame web element, the Driver will switch to that specific frame.

Selenium Script on iframe:

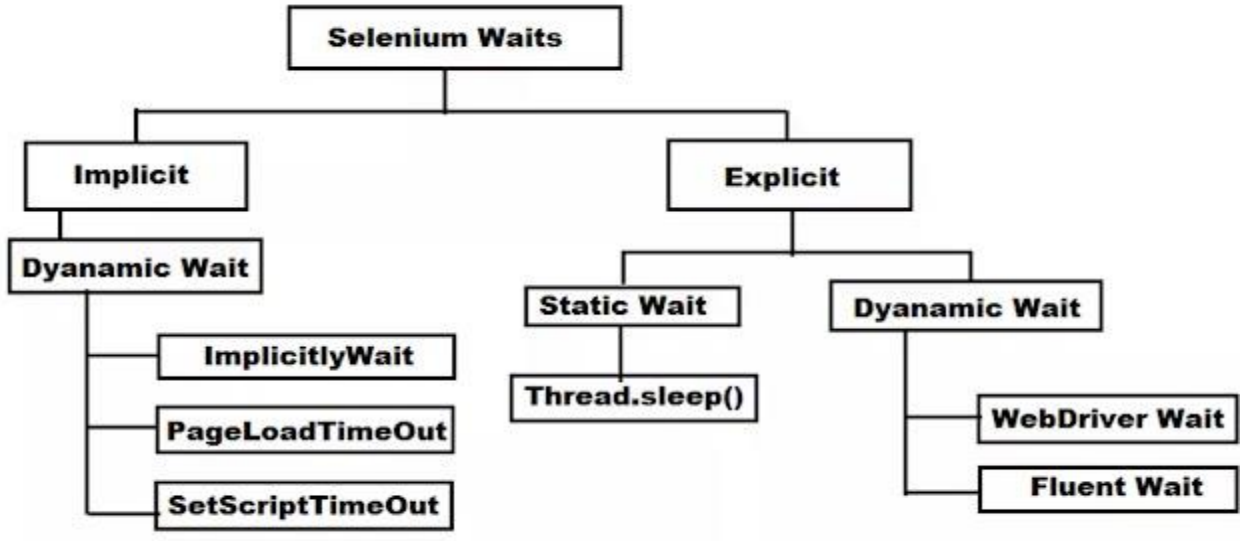
```
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class IframeDemo {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-
defines-multi.html");
  Thread.sleep(2000);
  //First finding the elements using any of locator strategy
  List<WebElement> iframList = driver.findElements(By.tagName("iframe"));
  int totalFrames = iframList.size();
  System.out.println("No of Frames:" + totalFrames);
  WebElement ele=driver.findElement(By.name("iframe_b"));
  System.out.println("Frame Names are:");
  for (WebElement iframe : iframList) {
   System.out.println(iframe.getAttribute("name"));
```

```
 if (iframe.getAttribute("name").equals("iframe_b")) {
  //switch to frame by element
  driver.switchTo().frame(ele);
  //Perform all the required tasks
  driver.findElement(By.id("searchInput")).sendKeys("iframe Testing");
  Thread.sleep(2000);
  //Switching back to the main window
          driver.switchTo().defaultContent();
 }
 }
 boolean btnDisplayed=driver.findElement(By.id("simpleAlert")).isDisplayed();
 System.out.println(btnDisplayed);
 driver.close();
}
}
```

- **Waits in Selenium WebDriver**

Different wait commands in selenium:

In selenium "Waits" play an important role in executing tests. Selenium waits can be classified as below, we will discuss each wait in detail:



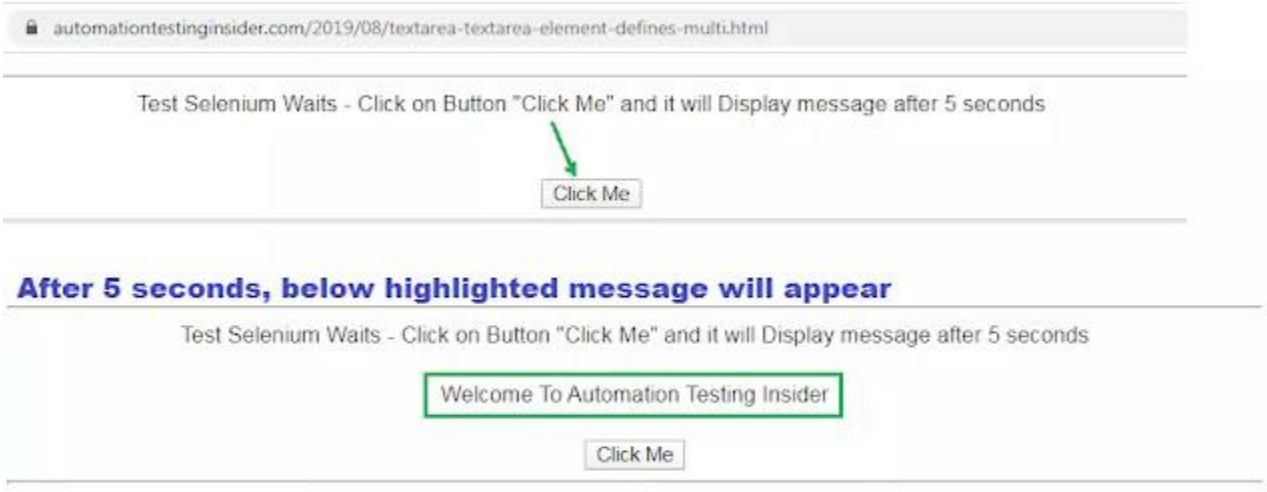
Classification of Selenium Waits

Why do we need waits in Selenium?

Most web applications are developed with Ajax and Javascript. When a page loads on a browser, the various web elements that someone wants to interact with may load at various time intervals.

This obviously creates difficulty in identifying any element. On top of that, if an element is not located then the “ElementNotVisibleException/NoSuchElementException” appears.

Please have a look at below image:



In above web page we have a button '**Click Me**', after clicking on that button, a message appears after waiting for 5 seconds.
Page Link: https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-defines-multi.html
Let's consider this scenario and write a script without using any waiting commands in selenium.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Demo1 {

 public static void main(String[] args) {

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-defines-multi.html");
  driver.findElement(By.id("testWait123")).click();
  WebElement ele=driver.findElement(By.xpath("//div[text()='Welcome To Automation Testing Insider']"));
  String text=ele.getText();
```

```
  System.out.println("The msg is: "+text);
  driver.close();
 }
}
```

Output:

```
Starting ChromeDriver 79.0.3945.36 (3582db32b33893869b8c1339e8f4d9ed1816f143-refs/branch-heads/3945@{#614}) on port 16094
Only local connections are allowed.
Please protect ports used by ChromeDriver and related test frameworks to prevent access by malicious code.
Feb 21, 2020 3:22:49 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Exception in thread "main" org.openqa.selenium.NoSuchElementException: no such element: Unable to locate element: {"method":"xpath","selector":"//div[text()='Welcome To Automation Testing Insider']"}
  (Session info: chrome=79.0.3945.130)
For documentation on this error, please visit: https://www.seleniumhq.org/exceptions/no_such_element.html
Build info: version: '3.141.59', revision: 'e82be7d358', time: '2018-11-14T08:25:48'
System info: host: 'HITENDRA-PC', ip: '169.254.161.182', os.name: 'Windows 7', os.arch: 'amd64', os.version: '6.1', java.version: '1.8.0_162'
Driver info: org.openqa.selenium.chrome.ChromeDriver
Capabilities {acceptInsecureCerts: false, browserName: chrome, browserVersion: 79.0.3945.130, chrome: {chromedriverVersion: 79.0.3945.36 (3582db32b3389..., userDataDir: C:\Users\Hitendra\AppData\L...}, goog:chromeOptions: {debuggerAddress: localhost:59460}, javascriptEnabled: true, networkConnectionEnabled: false, pageLoadStrategy: normal, platform: WINDOWS, platformName: WINDOWS, proxy: Proxy(), setWindowRect: true, strictFileInteractability: false, timeouts: {implicit: 0, pageLoad: 300000, script: 30000}, unhandledPromptBehavior: dismiss and notify}
Session ID: b49a3b9faf99bfbcc99834650e0182ff
*** Element info: {Using=xpath, value=//div[text()='Welcome To Automation Testing Insider']}
 at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
 at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
 at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
 at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
```

```
 at
org.openqa.selenium.remote.http.W3CHttpResponseCodec.createException(W3CHttpResponse
Codec.java:187)
 at
org.openqa.selenium.remote.http.W3CHttpResponseCodec.decode(W3CHttpResponseCodec.jav
a:122)
 at
org.openqa.selenium.remote.http.W3CHttpResponseCodec.decode(W3CHttpResponseCodec.jav
a:49)
 at
org.openqa.selenium.remote.HttpCommandExecutor.execute(HttpCommandExecutor.java:158)
 at
org.openqa.selenium.remote.service.DriverCommandExecutor.execute(DriverCommandExecuto
r.java:83)
 at org.openqa.selenium.remote.RemoteWebDriver.execute(RemoteWebDriver.java:552)
 at org.openqa.selenium.remote.RemoteWebDriver.findElement(RemoteWebDriver.java:323)
 at
org.openqa.selenium.remote.RemoteWebDriver.findElementByXPath(RemoteWebDriver.java:4
28)
 at org.openqa.selenium.By$ByXPath.findElement(By.java:353)
 at org.openqa.selenium.remote.RemoteWebDriver.findElement(RemoteWebDriver.java:315)
 at WaitsDemo.Demo1.main(Demo1.java:20)
```

So you can see in the above program we got 'NoSuchElementException' if we do not use wait commands in such scenarios. Now lets talk about Synchronization and later on we will handle the above scenario using different wait commands.

Basically in Test Automation, we have two components:

- Application Under Test
- Test Automation Tool

Both these components will have their own speed. We should write our scripts in such a way that both the components should move with same and desired speed, so that we will not encounter "Element Not Found" errors.

So Synchronization is a mechanism which involves more than one components to work parallel with Each other.

Synchronization can be classified into two categories:

- Unconditional – Thread.sleep() wait
- Conditional Synchronization – Implicit, Explicit and Fluent wait

Why use wait commands?
Wait commands direct test scripts to pause for a certain time before throwing exceptions. Selenium WebDriver provides three commands to implement wait in tests.

Implicit Wait - Implicit waits are used to provide a default waiting time (say 30 seconds) between each consecutive test step/command across the entire test script. Implicitly wait is applied globally, which means it is always available for all the web elements throughout the driver instance. It implies that if the driver is interacting with 100 elements, then implicitly wait is applicable for all the 100 elements.

Syntax: driver.manage().timeouts().implicitlyWait(4,TimeUnit.SECONDS);

Please consider the below program code and usage of implicitly code.

```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Demo3 {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-
defines-multi.html");
  driver.findElement(By.id("testWait123")).click();
  WebElement ele = driver.findElement(By.xpath("//div[text()='Welcome To Automation
Testing Insider']"));
  String text = ele.getText();
  System.out.println("The msg is: " + text);
  driver.close();
 }
}
```

Output
The msg **is:** Welcome To Automation Testing Insider

In the above code, I have given an implicit wait at 30 seconds, which implies that the maximum wait time is 30 seconds for the particular element to load or to arrive at the output. Implicit wait is a dynamic wait.

What are dynamic waits?

Implicit, Explicit, and Fluent waits are dynamic waits. Consider a situation where you have given a TimeOut value of 30 seconds. If the element is loaded in 5 seconds, then rest 25 seconds will be ignored. It won't wait until the TimeOut value is completed, i.e 30 seconds. That's why all waits are considered as dynamic waits.

Explicit Wait - Explicit waits are used to halt the execution until the time a particular condition is met or the maximum time has elapsed. Explicit waits are a concept from the dynamic wait, which waits dynamically for specific conditions. It can be implemented by the WebDriverWait class. To understand the explicit wait in Selenium WebDriver, you should know the requirements and why we use wait statements in programs.

In order to declare explicit wait, one has to use "ExpectedConditions". The following Expected Conditions can be used in Explicit Wait.

alertIsPresent()
elementSelectionStateToBe()
elementToBeClickable()
elementToBeSelected()
frameToBeAvaliableAndSwitchToIt()
invisibilityOfTheElementLocated()
invisibilityOfElementWithText()
presenceOfAllElementsLocatedBy()
presenceOfElementLocated()
textToBePresentInElement()
textToBePresentInElementLocated()
textToBePresentInElementValue()
titleIs()
titleContains()
visibilityOf()
visibilityOfAllElements()
visibilityOfAllElementsLocatedBy()
visibilityOfElementLocated()

Syntax:

```
WebDriverWait wait= new WebDriverWait(driver, 5);
      WebElement
ele=wait.until(ExpectedConditions.visibilityOfElementLocated(By.xpath(“xpath")));
```

Example – Suppose I have a web page that has some login form, and after login, it takes a lot of time to load the account or home page. This page is dynamic — it means that sometimes it takes 10 seconds to load the homepage, and sometimes, it’s 15 seconds and so on. In such situations, explicit wait helps us to wait until a specific page is not present.

Please refer below example code, here we have created an object of WebDriver wait and passed the driver reference and timeout as parameters. However in below example i have created user defined method findElement which we can use for any web elements.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class Demo4 {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-
defines-multi.html");
  driver.findElement(By.id("testWait123")).click();
  String xpath="//div[text()='Welcome To Automation Testing Insider']";
  WebElement ele=findElement(driver,xpath,10);
  String text = ele.getText();
  System.out.println("The msg is: " + text);
  driver.close();
 }
```

```
 public static WebElement findElement(WebDriver driver1, String xpath1, int timeout) {
  WebElement ele=new WebDriverWait(driver1,
timeout).until(ExpectedConditions.visibilityOfElementLocated(By.xpath(xpath1)));
  return ele;
 }
}
```

Output:
The msg **is:** Welcome To Automation Testing Insider

Fluent Wait - The Fluent Wait command defines the maximum amount of time for Selenium WebDriver to wait for a certain condition to appear. It also defines the frequency with which WebDriver will check if the condition appears before throwing the "ElementNotVisibleException".

Syntax:

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
.withTimeout(Duration.ofSeconds(40))
.pollingEvery(Duration.ofSeconds(8))
.ignoring(Exception.class);

  WebElement ele = wait.until(new Function<WebDriver, WebElement>() {
    public WebElement apply(WebDriver driver) {
      return driver.findElement(By.xpath("xpath"));
    }
  });
```

Example: Let's consider a scenario where an element is loaded at different intervals of time. The element might load within 10 seconds, 20 seconds or even more then that if we declare an explicit wait of 20 seconds. It will wait till the specified time before throwing an exception. In such scenarios, the fluent wait is the ideal wait to use as this will try to find the element at different frequency until it finds it or the final timer runs out.

Please refer below example code:

```
import java.time.Duration;
import java.util.function.Function;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
```

```
import org.openqa.selenium.support.ui.FluentWait;
import org.openqa.selenium.support.ui.Wait;

public class Demo5 {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-
defines-multi.html");
  driver.findElement(By.id("testWait123")).click();
  String xpath="//div[text()='Welcome To Automation Testing Insider']";
  Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
    .withTimeout(Duration.ofSeconds(20))
    .pollingEvery(Duration.ofSeconds(2))
    .ignoring(Exception.class);

      WebElement ele = wait.until(new Function<WebDriver, WebElement>() {
        public WebElement apply(WebDriver driver) {
          return driver.findElement(By.xpath(xpath));
        }
      });
  String text = ele.getText();
  System.out.println("The msg is: " + text);
  driver.close();
 }
}
```

Output

The msg **is:** Welcome To Automation Testing Insider

Some other waits are:

PageLoadTimeOut - One of the timeouts is focused on the time a webpage needs to be loaded – the pageLoadTimeout limits the time that the script allots for a web page to be displayed. Page load timeout is useful when we perform a performance test. Page Load timeout is applicable only to driver.get() and driver.navigate().to() methods in selenium.

Syntax: driver.manage().timeouts().pageLoadTimeout(5, TimeUnit.**SECONDS**);

Consider below example where we used **PageLoadTimeOut** to limit the page to be loaded in 5 seconds but page did not load in 5 seconds and throws TimeoutException exception.

```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Demo2 {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.manage().timeouts().pageLoadTimeout(5, TimeUnit.SECONDS);
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-defines-multi.html");
  driver.findElement(By.id("testWait123")).click();
  Thread.sleep(5000);
  WebElement ele = driver.findElement(By.xpath("//div[text()='Welcome To Automation Testing Insider']"));
  String text = ele.getText();
  System.out.println("The msg is: " + text);
  driver.close();
 }
}
```

Output

```
Starting ChromeDriver 79.0.3945.36 (3582db32b33893869b8c1339e8f4d9ed1816f143-refs/branch-heads/3945@{#614}) on port 31941
Only local connections are allowed.
Please protect ports used by ChromeDriver and related test frameworks to prevent access by malicious code.
Feb 21, 2020 3:51:50 PM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
[1582280516.761][SEVERE]: Timed out receiving message from renderer: 0.015
```

```
[1582280516.791][SEVERE]: Timed out receiving message from renderer: -0.031
Exception in thread "main" org.openqa.selenium.TimeoutException: timeout
  (Session info: chrome=79.0.3945.130)
Build info: version: '3.141.59', revision: 'e82be7d358', time: '2018-11-14T08:25:48'
System info: host: 'HITENDRA-PC', ip: '169.254.161.182', os.name: 'Windows 7', os.arch:
'amd64', os.version: '6.1', java.version: '1.8.0_162'
Driver info: org.openqa.selenium.chrome.ChromeDriver
Capabilities {acceptInsecureCerts: false, browserName: chrome, browserVersion:
79.0.3945.130, chrome: {chromedriverVersion: 79.0.3945.36 (3582db32b3389..., userDataDir:
C:\Users\Hitendra\AppData\L...}, goog:chromeOptions: {debuggerAddress: localhost:59888},
javascriptEnabled: true, networkConnectionEnabled: false, pageLoadStrategy: normal, platform:
WINDOWS, platformName: WINDOWS, proxy: Proxy(), setWindowRect: true,
strictFileInteractability: false, timeouts: {implicit: 0, pageLoad: 300000, script: 30000},
unhandledPromptBehavior: dismiss and notify}
Session ID: 2465cdde761b595d659c73ff38c7aa58
 at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
 at
sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:6
2)
 at
sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorIm
pl.java:45)
 at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
 at
org.openqa.selenium.remote.http.W3CHttpResponseCodec.createException(W3CHttpResponse
Codec.java:187)
 at
org.openqa.selenium.remote.http.W3CHttpResponseCodec.decode(W3CHttpResponseCodec.jav
a:122)
 at
org.openqa.selenium.remote.http.W3CHttpResponseCodec.decode(W3CHttpResponseCodec.jav
a:49)
 at
org.openqa.selenium.remote.HttpCommandExecutor.execute(HttpCommandExecutor.java:158)
 at
org.openqa.selenium.remote.service.DriverCommandExecutor.execute(DriverCommandExecuto
r.java:83)
 at org.openqa.selenium.remote.RemoteWebDriver.execute(RemoteWebDriver.java:552)
 at org.openqa.selenium.remote.RemoteWebDriver.get(RemoteWebDriver.java:277)
 at WaitsDemo.Demo2.main(Demo2.java:20)
```

Thread.sleep() - Sleep is a static method that belongs to the thread class. This method can be called using the reference of the class name i.e Thread. If you use Thread.sleep while performing automation testing with Selenium, then this method will stop the execution of the script for the specified duration of time, irrespective of whether the element is found or not on the web page.

Syntax: Thread.sleep(5000);

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class Demo2 {

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

  WebDriver driver;
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/2019/08/textarea-textarea-element-defines-multi.html");
  driver.findElement(By.id("testWait123")).click();
  Thread.sleep(10000);
  WebElement ele = driver.findElement(By.xpath("//div[text()='Welcome To Automation Testing Insider']"));
  String text = ele.getText();
  System.out.println("The msg is: " + text);
  driver.close();
 }
}
```

Output

The msg **is:** Welcome To Automation Testing Insider

Consider the above example, the element will be visible in 5 seconds but WebDriver will still wait for another 5 seconds. It will increase test execution time.

Difference between Implicit wait and Explicit wait:

Implicit Wait	Explicit Wait
Implicit Wait time is applied to all the elements in the script	Explicit Wait time is applied only to those elements based on the conditions
In Implicit Wait, we need not specify "ExpectedConditions" on the element to be located	In Explicit Wait, we need to specify "ExpectedConditions" on the element to be located
It is recommended to use when the elements are located with the time frame specified in implicit wait. E.g. For Small Application	It is recommended to use when the elements are taking long time to load and also for verifying the property of the element like(visibilityOfElementLocated, elementToBeClickable,elementToBeSelected)

What happens when we use Implicit wait and Explicit wait together:

Implicit wait destroys meaning of using explicit wait when using together. So it is advised not to use implicit wait and explicit wait together. Actually when we use both waits together, both waits will be applied at the same time and it get messed up.

Now we will see reason behind these using below scenarios:

1. Explicit Wait= Implicit Wait (Say 10 seconds)

Both waits get activated at same time to locate element. Explicit wait keeps searching for an element till it is found and implicit wait allows webdriver to search till timeout. When explicit wait starts and looks for element, because of implicit wait it needs to wait for 10 seconds because element is not found. So both waits completes 10 seconds wait time.

2. Implicit wait(20) > Explicit Wait(10)

When explicit wait stars looking for element, it needs to wait for 20 seconds because of implicit wait time.

3. Implicit wait(10) < Explicit Wait(20)

When explicit wait starts looking for element, it needs to wait for 10 seconds because of implicit wait. After that implicit wait throws exception because of not able to locate element. Exception stops explicit wait to search further and does not allow to reach its timeout.

Few more points to remember:

1. The most widely used waits are implicit and explicit waits. Fluent waits are not preferable for real-time projects.
2. We use FluentWait commands mainly when we have web elements which sometimes visible in few seconds and some times take more time than usual. Mostly in Ajax applications.
3. We use mostly explicit wait because it is for specific condition/element and we have more flexibility in using explicit rather than implicit.

- **How To Find Broken Links Using Selenium**

In this post we will cover the following points:

- What are broken links and images?
- Examples of a broken link error code
- HTTP status codes.
- Reasons for broken links
- Why should you check Broken links?
- Steps to check broken links and images
- Program

What are broken links?
Broken links are links or URLs that are not reachable.
A broken link is a web-page that can't be found or accessed by a user,

Examples of a broken link error code
Here are some examples of errors that a web server may present for a broken link:

- 404 Page Not Found: the page/resource doesn't exist on the server
- 400 Bad Request: the host server cannot understand the URL on your page
- Bad host: Invalid host name: the server with that name doesn't exist or is unreachable
- Bad URL: Malformed URL (e.g. a missing bracket, extra slashes, wrong protocol, etc.)
- Bad Code: Invalid HTTP response code: the server response violates HTTP spec
- Empty: the host server returns "empty" responses with no content and no response code
- Timeout: Timeout: HTTP requests constantly timed out during the link check
- Reset: the host server drops connections. It is either misconfigured or too busy.

Some of the HTTP status codes.
Below are the some of the status codes:
200 – Valid Link
404 – Link not found
400 – Bad request

401 – Unauthorized
500 – Internal Error

Reasons for broken links
There are various reasons that broken links can occur, for example:

- The website owner entered the incorrect URL (misspelled, mistyped, etc.).
- The external site is no longer available, is offline, or has been permanently moved.
- Links to content (PDF, Google Doc, video, etc.) that has been moved or deleted.
- Broken elements within the page (HTML, CSS, Javascript, or CMS plugins interference).
- Firewall or geolocation restriction does not allow outside access.

Why should you check Broken links?

- You should always make sure that there are no broken links on the site because the user should not land into an error page.
- Manual checking of links is a tedious task, because each webpage may have a large number of links & manual process has to be repeated for all pages.
- An Automation script using Selenium that will automate the process is a great solution.

Steps to check broken links and images:

- Collect all the links in the web page based on <a> and <img> tags.
- Send HTTP request for the link and read HTTP response code.
- Find out whether the link is valid or broken based on HTTP response code.
- Repeat this for all the links captured.

Program:
Utility Class for Broken Links and Images:

```
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;

public class VerifyBrokenLink {

 int validLink=0;
 int invalidLink=0;

 public void verifyBrokenLinks(String linkURL) {
```

```
  try {
   URL url = new URL(linkURL);
   HttpURLConnection httpURLConnect = (HttpURLConnection) url.openConnection();
   httpURLConnect.connect();

   int respCode = httpURLConnect.getResponseCode();

   if (respCode >= 400) {
    System.out.println(linkURL + ": is a broken link" + "---" +
httpURLConnect.getResponseMessage()+ "---" + httpURLConnect.getResponseCode());
    invalidLink=invalidLink+1;
   } else {
    System.out.println(url + ": is a valid link" + "---" + httpURLConnect.getResponseMessage()+
"---" + httpURLConnect.getResponseCode());
    validLink=validLink+1;
   }
   httpURLConnect.disconnect();
  } catch (MalformedURLException e) {
   e.printStackTrace();
  } catch (IOException e) {
   e.printStackTrace();
  }
 }
}
```

Test Class from where verifying broken links:

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class BrokenLinkTest {

 WebDriver driver;
```

```
List<WebElement> activeLinkImage = new ArrayList<WebElement>();

@BeforeTest
public void launchApp() {
 System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
 driver = new ChromeDriver();
 driver.manage().window().maximize();
 driver.get("https://www.google.com");
}
@Test(priority = 1)
public void getLinks() {

 // List of all links and images
 List<WebElement> linkImgList = driver.findElements(By.tagName("a"));
 linkImgList.addAll(driver.findElements(By.tagName("img")));

 // Total Links and Images Before
 int total = linkImgList.size();
 System.out.println("Total Links and Images are: " + total);

 for (int i = 0; i < linkImgList.size(); i++) {
  if (linkImgList.get(i).getAttribute("href") != null
&&(!linkImgList.get(i).getAttribute("href").contains("javascript"))) {
    activeLinkImage.add(linkImgList.get(i));
  }
 }
 // Total Links and Images After
 int total1 = activeLinkImage.size();
 System.out.println("Total Active Links and Images: " + total1);
}
@Test(priority = 2)
public void VerifyBrokenLinks() throws IOException {
 VerifyBrokenLink obj= new VerifyBrokenLink();
 for(int j=0;j<activeLinkImage.size();j++)
 {
  WebElement ele= activeLinkImage.get(j);
  String url=ele.getAttribute("href");
  obj.verifyBrokenLinks(url);
 }
 System.err.println("Total Valid Links: "+obj.validLink);
```

```
 System.err.println("Total Invalid Links: "+obj.invalidLink);
 }
 @AfterTest
 public void tearDown() {
  driver.close();
 }
}
```

Output: output will be look like below

```
Total Links and Images are: 32
Total Active Links and Images: 30
https://mail.google.com/mail/?tab=wm&ogbl: is a valid link---OK---200
https://www.google.co.in/imghp?hl=en&tab=wi&ogbl: is a valid link---OK---200
https://www.google.co.in/intl/en/about/products?tab=wh: is a valid link---OK---200
https://www.google.com/search?q=coronavirus+tips&oi=ddle&ct=153021071&hl=en&sa=X&v
ed=0ahUKEwjm8I-KgNHoAhW0yTgGHbqHACUQPQgN: is a broken link---Forbidden---403
.
.
.
.
Total Valid Links: 28
Total Invalid Links: 2
PASSED: getLinks
PASSED: VerifyBrokenLinks

===============================================
   Default test
   Tests run: 2, Failures: 0, Skips: 0
```

- **Headless Browser Testing Using Selenium**

What Is Headless Browser?

A headless browser is a web-browser without a graphical user interface. This program will behave just like a browser but will not show any GUI.

Benefits Of Headless Browser Testing

1. Since this type of testing does not actually open a browser, the system saves the processing power that would otherwise be used in a real browser test. Consequently, the tests are executed faster.
2. Allows testing browser less setups.

There may be setups where installing a browser is not possible, such as servers. In these cases, headless browsers help run automation tests easily.
3. Helps you multitask
You can use your browser or your machine to do anything else while the tests run in the background. Save hours of time that is otherwise spent staring at the screen.

Some of the examples of Headless Drivers include

- HtmlUnit
- Ghost
- PhantomJS
- ZombieJS
- Watir-webdriver

In this tutorial we will focus on HtmlUnit and PhatomJS

Headless Browser Testing Using HTMLUnitDriver
HTML UnitDriver is the most light weight and fastest implementation headless browser for of WebDriver. It is based on HtmlUnit. It is known as Headless Browser Driver. It is same as Chrome, IE, or FireFox driver, but it does not have GUI so one cannot see the test execution on screen.
Steps:
1. You need Eclipse with Selenium installed
2. Download the HTMLUnit Driver. Add the jar to your project
https://github.com/SeleniumHQ/htmlunit-driver/releases

Program:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;
import org.testng.annotations.Test;

public class HTMLUnitTest {

 @Test
 public void headLessBrowserTest() {
  WebDriver driver = new HtmlUnitDriver();
  driver.get("https://www.google.com/");
  driver.findElement(By.name("q")).sendKeys("Test");
  String titleString=driver.getTitle();
  System.out.println("Page title is: "+titleString);
```

```
  String urlString=driver.getCurrentUrl();
  System.out.println("Page URL is: "+urlString);
  driver.quit();
 }
}
```

Output:

```
Page title is: Google
Page URL is: https://www.google.com/
PASSED: headLessBrowserTest

===============================================
   Default test
   Tests run: 1, Failures: 0, Skips: 0
===============================================
```

Headless Browser Testing Using PhantomJS

PhantomJS is a headless browser with JavaScript API. It is an optimal solution for Headless Website Testing, access and manipulate webpages & comes with the standard DOM API.

In order to use PhantomJS with Seleniun, one has to use GhostDriver. GhostDriver is a implementation of Webdriver Wire protocol in simple JS for PhantomJS.

1. You need Eclipse with Selenium installed
2. Download PhantomJS

https://phantomjs.org/download.html

3. Extract the downloaded folder to Program Files
4. Download the PhantomJS Driver. Add the jar to your project

Program:

```
import java.io.File;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.phantomjs.PhantomJSDriver;
import org.testng.annotations.Test;

public class PhantomJS {

 @Test
 public void headLessBrowserTest() {
```

```
 File file = new File(
  " hantomjs.exe (file path)");
 System.setProperty("phantomjs.binary.path", file.getAbsolutePath());
 WebDriver driver = new PhantomJSDriver();
 driver.get("https://www.google.com/");
 driver.findElement(By.name("q")).sendKeys("Test");
 String titleString = driver.getTitle();
 System.out.println("Page title is: " + titleString);
 String urlString = driver.getCurrentUrl();
 System.out.println("Page URL is: " + urlString);
 driver.quit();
 }
}
```

Output:

```
Page title is: Google
Page URL is: https://www.google.com/#spf=1586097652826
[INFO  - 2020-04-05T14:40:53.977Z] ShutdownReqHand - _handle - About to shutdown
PASSED: headLessBrowserTest

===============================================
   Default test
   Tests run: 1, Failures: 0, Skips: 0
===============================================
```

- **How To Run Selenium Tests In Headless Google Chrome**

We are going to use ChromeOptions class to run out test case on Chrome Browser in HeadLess mode.
ChromeOptions is a Class to manage options specific to ChromeDriver.

Please refer below Program to run tests in headless mode using Chrome Driver.

```
import java.io.File;
import java.io.IOException;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
```

```
import org.openqa.selenium.chrome.ChromeOptions;
import org.testng.annotations.Test;

public class HeadlessChrome {
 @Test
 public void headlessTest() throws IOException {
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
  ChromeOptions chromeOptions= new ChromeOptions();
  chromeOptions.addArguments("headless");
  chromeOptions.addArguments("window-size=1200x600");
  WebDriver driver= new ChromeDriver(chromeOptions);
  driver.get("https://www.google.com/");
  System.out.println("Test Start");
  System.out.println("Google title is: "+driver.getTitle());
  System.out.println("Test End");
  TakesScreenshot takesScreenshot = (TakesScreenshot) driver;
  File source = takesScreenshot.getScreenshotAs(OutputType.FILE);
  FileUtils.copyFile(source, new
File("D:\\Workspace_Eclipse\\WebDriverConcept3\\ScreenShot\\Headless.png"));
  driver.close();
 }
}
```

Output:

```
Test Start
Google title is: Google
Test End
PASSED: headlessTest

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================
```

- **Handle Dynamic WebTable in Selenium**

What is 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.

Basic Structure of Web Table: Below given the basic structure of a webtable which is having 2 rows and 3 columns.

```
<table>
 <tbody>
  <tr>
    <th>Employee Name</th>
    <th>Designation</th>
    <th>Salary</th>
   </tr>
  <tr>
    <td>John</td>
    <td>Software Tester</td>
    <td>50000</td>
  </tr>
  <tr>
    <td>Tony</td>
    <td>Sr. Software Tester</td>
    <td>100000</td>
  </tr>
 </tbody>
</table>
```

Web Table Types?

There are two types of HTML tables published on the web-

- Static tables: Data is static i.e. Number of rows and columns are fixed.
- Dynamic tables: Data is dynamic i.e. Number of rows and columns are NOT fixed.

We are going to handle following web table questions in this post.

- How to get the table headers?
- Count no of rows and column in a web table
- Get value from particular column wrt given data

- Conditional (get all employees > 40 yrs)
- How to Print last row in a web table?
- Get particular cell value
- Get particular cell value using custom method
- How to get all table data
- How to get Particular Column
- How to retrieve More Than One Column

Please refer the below programs to handle above questions:

Base Class: In this class, we are having setup method to launch the browser and some variables

```
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class WebTableBaseClass {
 WebDriver driver;
 String xpathString2="//table";

 // test case#2,8
 List<WebElement> c;
 List<WebElement> r;

 // test case#3
 String empRole = "Software Engineer";
 int empPosColumn = 2;
 int empNameColumn = 1;

 // test case#4
 int empAge = 40;
 int empAgeColumn = 4;

 // test case#6
 int rowN = 2;
 int colN = 3;

 // test case#7
 public String getColValue(int row, int col) {
  WebElement colValue = driver
```

```
   .findElement(By.xpath(""+xpathString+"/tbody/tr[" + row + "]/td[" + col + "]"));
  return colValue.getText();
 }
 // test case#9
 int getcolNo = 2;

 // test case#10
 int noOfColumns = 3;

 // test case#11
 int firstColumnNo = 2;
 int secondColumnNo = 5;

 public void setup() {
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.seleniumeasy.com/test/table-sort-search-demo.html");
  c = driver.findElements(By.xpath(""+xpathString+"/thead/tr/th"));
  r = driver.findElements(By.xpath(""+xpathString+"/tbody/tr"));
 }
}
```

TestClass: This is our test class and we have all our test cases in this class. This class extending the above BaseClass.

```
import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.Test;

public class WebtableDemo extends WebTableBaseClass {

 @Test(priority = 1, description = "Get the Table Headers")
 public void getTableHeaders() {
  setup();
  System.out.println("=========Get Table Headers=====================");
  List<WebElement> allHeadersOfTable =
driver.findElements(By.xpath(""+xpathString+"/thead/tr/th"));
  System.out.println("Headers in table are below:");
```

```
 System.out.println("Total headers found: " + allHeadersOfTable.size());
 for (WebElement header : allHeadersOfTable) {
  System.out.println(header.getText());
 }
 System.out.println("================================================");
}

@Test(priority = 2, description = "Count total Rows And Coulmns")
public void countRowsAndCoulmns() {
 setup();
 System.out.println("=========countRowsAndCoulmns==================");
 // Total Columns are
 System.out.println("Total Columns: " + c.size());
 // Total Rows are
 System.out.println("Total Rows: " + r.size());
 System.out.println("================================================");
}

@Test(priority = 3, description = "Get the Employee Position whose designation is SW Engg.")
public void getEmpPosition() {
 setup();
 System.out.println("=======Get All Emp Names of "+empRole+" position=======");
 for (int i = 1; i <= r.size(); i++) {
  WebElement posColumn = driver.findElement(By.xpath(""+xpathString+"/tbody/tr[" + i +
"]/td["+empPosColumn+"]"));
  if (posColumn.getText().toLowerCase().equalsIgnoreCase(empRole)) {
   WebElement empNameColumn1 = driver
     .findElement(By.xpath(""+xpathString+"/tbody/tr[" + i + "]/td["+empNameColumn+"]"));
   System.out.println(empNameColumn1.getText());
  }
 }
 System.out.println("================================================");
}

@Test(priority = 4, description = "Get the name of the employees Age>40")
public void getAge() {
 setup();
 System.out.println("=======Get All Emp Names whose age>40==========");
 for (int i = 1; i <= r.size(); i++) {
```

```
   WebElement ageColumn = driver.findElement(By.xpath(""+xpathString+"/tbody/tr[" + i +
"]/td["+empAgeColumn+"]"));
   if (Integer.parseInt(ageColumn.getText()) >= empAge) {
    WebElement empName = driver.findElement(By.xpath(""+xpathString+"/tbody/tr[" + i +
"]/td["+empNameColumn+"]"));
    System.out.println(empName.getText());
   }
  }
  System.out.println("==================================================");
 }

 @Test(priority = 5, description = "Print the Last Row")
 public void printLastRow() {
  setup();
  System.out.println("================Get Last Row of table================");
  List<WebElement> columnOfLastRow =
driver.findElements(By.xpath(""+xpathString+"/tbody/tr[last()]/td"));
  for (WebElement e : columnOfLastRow) {
   System.out.print(e.getText() + "   ");
  }
  System.out.println();
  System.out.println("==================================================");
 }

 @Test(priority = 6, description = "Get particular Cell Value")
 public void getCellValue() {
  setup();
  System.out.println("====Retrive cell value by providing row and column number========");
  WebElement colValue = driver
    .findElement(By.xpath(""+xpathString+"/tbody/tr[" + rowN + "]/td[" + colN + "]"));
  System.out.println("Cell Value : " + colValue.getText());
  System.out.println("==================================================");
 }

 @Test(priority = 7, description = "Get Cell Value By CustomMethod")
 public void getCellValueByCustomMethod() {
  setup();
  System.out.println("====Retrive cell value by providing row and column number========");
  System.out.println(getColValue(2, 3));
  System.out.println("==================================================");
```

```
}

@Test(priority = 8, description = "Get All TableData")
public void getAllTableData() {
 setup();
 System.out.println("====================Retrive All table data============");
 for (int i = 1; i <= r.size(); i++) {

  for (int j = 1; j <= c.size(); j++) {

   System.out.print(
    driver.findElement(By.xpath(""+xpathString+"/tbody/tr[" + i + "]/td[" + j + "]")).getText()
     + "   ");
  }
  System.out.println();
  System.out.println();
 }
 System.out.println("=================================================");
}

@Test(priority = 9, description = "Get Particular Column")
public void getParticularColumn() {
 setup();
 System.out.println("============================get particular
Column=======================");
 for (int i = 1; i <= r.size(); i++) {
  WebElement element = driver
    .findElement(By.xpath(""+xpathString+"/tbody/tr[" + i + "]/td[" + getcolNo + "]"));
  System.out.println(element.getText());
 }
 System.out.println("=================================================");
}

@Test(priority = 10, description = "Retrive More Than One Column")
public void retriveMoreThanOneColumn() {
 setup();
 System.out.println("===========retriveMoreThanOneColumns==========");
 for (int i = 1; i <= r.size(); i++) {

  for (int j = 1; j <= noOfColumns; j++) {
```

```
   System.out.print(
     driver.findElement(By.xpath(""+xpathString+"/tbody/tr[" + i + "]/td[" + j + "]")).getText()
       + "    ");
   }
   System.out.println();
  }
  System.out.println("================================================");
 }
 @AfterMethod
 public void tearDown() {
  driver.close();
 }
}
```

Output: The output looks like below

```
=========Get Table Headers=====================
Headers in table are below:
Total headers found: 6
Name
Position
Office
Age
Start date
Salary
================================================

=========countRowsAndCoulmns===================
Total Columns: 6
Total Rows: 10
================================================

=======Get All Emp Names of Software Engineer position=======
B. Greer
B. Wagner
================================================

=======Get All Emp Names whose age>40===========
A. Cox
A. Ramos
```

B. Greer
B. Williamson
==========

==========Get Last Row of table==========
C. Vance Pre-Sales Support New York 21 Mon 12th Dec 11 $106,450/y
==========

====Retrive cell value by providing row **and** column number========
Cell Value : London
==========

====Retrive cell value by providing row **and** column number========
London
==========

==========Retrive All table data==========
A. Cox Junior Technical Author San Francisco 66 Mon 12th Jan 09 $86,000/y

A. Ramos Chief Executive Officer (CEO) London 47 Fri 9th Oct 09 $1,200,000/y

A. Satou Accountant Tokyo 33 Fri 28th Nov 08 $162,700/y

B. Greer Software Engineer London 41 Sat 13th Oct 12 $132,000/y

B. Wagner Software Engineer San Francisco 28 Tue 7th Jun 11 $206,850/y

B. Williamson Integration Specialist New York 61 Sun 2nd Dec 12 $372,000/y

C. Hurst Javascript Developer San Francisco 39 Tue 15th Sep 09 $205,500/y

C. Kelly Senior Javascript Developer Edinburgh 22 Thu 29th Mar 12 $433,060/y

C. Marshall Regional Director San Francisco 36 Thu 16th Oct 08 $470,600/y

C. Vance Pre-Sales Support New York 21 Mon 12th Dec 11 $106,450/y

==========

==========get particular Column==========

Junior Technical Author
Chief Executive Officer (CEO)
Accountant
Software Engineer
Software Engineer
Integration Specialist
Javascript Developer
Senior Javascript Developer
Regional Director
Pre-Sales Support
==========

==========retriveMoreThanOneColumns==========
A. Cox Junior Technical Author San Francisco
A. Ramos Chief Executive Officer (CEO) London
A. Satou Accountant Tokyo
B. Greer Software Engineer London
B. Wagner Software Engineer San Francisco
B. Williamson Integration Specialist New York
C. Hurst Javascript Developer San Francisco
C. Kelly Senior Javascript Developer Edinburgh
C. Marshall Regional Director San Francisco
C. Vance Pre-Sales Support New York
==========

==========
Default test
Tests run: **10**, Failures: **0**, Skips: **0**
==========

- **Handle Bootstrap Date-picker in Selenium**

Below are the two programs given to handle date picker in selenium.

1. Using loop:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
```

```
public class HandleDatePciker {
 WebDriver driver;

 @BeforeMethod
 public void setup() {
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.seleniumeasy.com/test/bootstrap-date-picker-demo.html");
 }

 @Test
 public void datePicker() {

  String month = "May 2019";
  String day = "13";

  driver.findElement(By.xpath("//*[@id=\"sandbox-container1\"]//input")).click();

  while (true) {

   String textString =
driver.findElement(By.xpath("/html/body/div[3]/div[1]/table/thead/tr[2]/th[2]"))
     .getText();

   if (textString.equals(month)) {
    break;
   } else {
    driver.findElement(By.xpath("/html/body/div[3]/div[1]/table/thead/tr[2]/th[1]")).click();
   }
  }
  driver.findElement(By.xpath("//table/tbody/tr/td[contains(text()," + day + ")]")).click();
 }
}
```

2. Using JavaScript:

```
import org.openqa.selenium.By;
import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.WebDriver;
```

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;

public class DatePickerByJS {
 WebDriver driver;

 @BeforeMethod
 public void setup() {
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.seleniumeasy.com/test/bootstrap-date-picker-demo.html");
 }

 @Test
 public void datePcikerTestbyJS() {
  WebElement element=driver.findElement(By.xpath("//*[@id=\"sandbox-
container1\"]//input"));
  String dateString="04/08/1996";
  datePcikerByJS(driver, dateString, element);
 }

 public void datePcikerByJS(WebDriver driver, String date,WebElement element) {
  JavascriptExecutor js=(JavascriptExecutor) driver;
  js.executeScript("arguments[0].setAttribute('value','"+date+"')", element);
 }

}
```

SCREEN Shot, REPORTS, LOGS etc:

- **How to capture Screen Shot in Selenium**

Why Screen shots are required?

- When application issues occur
- When an assertion failure occurs

- It is helpful to track the execution when working with headless browser.
- It helps us to understand the flow of the application.
- For cross browser testing.
- When there is some difficulty in finding web elements on a page
- Where there is a Timeout in finding web elements on a web page

Screenshots are desirable for bug analysis. Selenium can automatically take screenshots during execution. You need to type cast WebDriver instance to TakesScreenshot.

Taking Screenshot in Selenium is just 3 Step process:

Step 1: Convert web driver object to TakesScreenshot
TakesScreenshot takesScreenshot = (TakesScreenshot) driver;
Step 2: Call getScreenshotAs method to create image file
File source = takesScreenshot.getScreenshotAs(OutputType.FILE);
Step 3: Copy file to Desired Location
Example: In this example we will take screen capture of https://google.com & save it as D:\Workspace_Eclipse\ScreehShotDemo\ScreenShot

Please refer below example

Utility Class where we kept screenShot method

```
import java.io.File;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;

public class TakeScreenShot {
 public static void screenShot(WebDriver driver, String filename) {
  TakesScreenshot takesScreenshot = (TakesScreenshot) driver;
  File source = takesScreenshot.getScreenshotAs(OutputType.FILE);
  try {
   FileUtils.copyFile(source, new
File(System.getProperty("user.dir")+"\\ScreenShot\\"+filename+".png"));
  } catch (Exception e) {
   // TODO Auto-generated catch block
   e.getMessage();
  }
 }
}
```

Test Class where we have created few test cases and called screenShot method to take the screen shots :

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.ITestResult;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
import com.utility.TakeScreenShot;

public class ScreenShotTest {
 WebDriver driver;
 @BeforeMethod
 public void setup() {
  System.setProperty("webdriver.chrome.driver",
    " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.google.com/");
 }
 @Test
 public void testCase1() {
  driver.findElement(By.name("q")).sendKeys("ScreenShot Demo");
  Assert.assertTrue(false);
            TakeScreenShot.screenShot(driver, "testCase1");
}
 @Test
 public void testCase2() {
  Assert.assertTrue(true);
            TakeScreenShot.screenShot(driver, "testCase2");
 }
 @AfterMethod
 public void tearDown() {
  driver.close();
 }
}
```

POM.xml file given below:

```
<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>ScreehShotDemo</groupId>
 <artifactId>ScreehShotDemo</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <dependencies>
  <dependency>
   <groupId>org.testng</groupId>
   <artifactId>testng</artifactId>
   <version>6.14.3</version>
   <scope>test</scope>
  </dependency>
  <dependency>
   <groupId>org.seleniumhq.selenium</groupId>
   <artifactId>selenium-java</artifactId>
   <version>3.141.59</version>
  </dependency>
  <dependency>
   <groupId>commons-io</groupId>
   <artifactId>commons-io</artifactId>
   <version>2.6</version>
  </dependency>
 </dependencies>
</project>
```

- **How to capture Screen Shot for failed test cases**

In this post we will capture the screen shots in two ways:
1. By using @AfterMethod annotation method in test class.
2. By using TestNG listener ITestListener.

Let's have a look at the first method:
1. By using @AfterMethod annotation method in test class.

Utility Class:

```
import java.io.File;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;

public class TakeScreenShot {

 public static void screenShot(WebDriver driver, String filename) {
  TakesScreenshot takesScreenshot = (TakesScreenshot) driver;
  File source = takesScreenshot.getScreenshotAs(OutputType.FILE);
  try {
   FileUtils.copyFile(source, new
File(System.getProperty("user.dir")+"\\ScreenShot\\"+filename+".png"));
  } catch (Exception e) {
   // TODO Auto-generated catch block
   e.getMessage();
  }
 }
}
```

TestClass:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.ITestResult;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
import com.utility.TakeScreenShot;

public class ScreenShotTest {
 WebDriver driver;

 @BeforeMethod
 public void setup() {
  System.setProperty("webdriver.chrome.driver"," chromedriver.exe path");
  driver = new ChromeDriver();
```

```
 driver.manage().window().maximize();
 driver.get("https://www.google.com/");
}
@Test
public void testCase1() {
 driver.findElement(By.name("q")).sendKeys("ScreenShot Demo");
 Assert.assertTrue(false);
}
@Test
public void testCase2() {
 Assert.assertTrue(true);
}
@AfterMethod
public void tearDown(ITestResult result) {
 if(result.FAILURE==result.getStatus())
 {
  TakeScreenShot.screenShot(driver, result.getName());
 }
 driver.close();
}
}
```

2. By using TestNG listener ITestListener.

BaseClass: We have created baseclass where we kept setup method to launch the browser and application and screenShot method to capture the screenshots

```
import java.io.File;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class BaseClass {

 public static WebDriver driver;

 public void setup() {
```

```
 System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
 driver = new ChromeDriver();
 driver.manage().window().maximize();
 driver.get("https://www.google.com/");
}
public void screenShot(String filename) {
 String dateName = new SimpleDateFormat("yyyyMMddhhmmss").format(new Date());
 TakesScreenshot takesScreenshot = (TakesScreenshot) driver;
 File source = takesScreenshot.getScreenshotAs(OutputType.FILE);
 try {
  FileUtils.copyFile(source, new
File(System.getProperty("user.dir")+"\\ScreenShot\\"+filename+"_"+dateName+".png"));
 } catch (Exception e) {
  // TODO Auto-generated catch block
  e.getMessage();
 }
}
}
```

Utility Class: In ListenerClass we have classed screenShot method from base class. This class implements ITestListener

```
import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;
import com.base.BaseClass;

public class ListenerClass extends BaseClass implements ITestListener {

public void onTestStart(ITestResult result) {
 // TODO Auto-generated method stub
}

public void onTestSuccess(ITestResult result) {
 // TODO Auto-generated method stub
}
public void onTestFailure(ITestResult result) {
 screenShot(result.getName());
}
public void onTestSkipped(ITestResult result) {
```

```
 // TODO Auto-generated method stub
 }
 public void onTestFailedButWithinSuccessPercentage(ITestResult result) {
  // TODO Auto-generated method stub
 }
 public void onStart(ITestContext context) {
  // TODO Auto-generated method stub
 }
 public void onFinish(ITestContext context) {
  // TODO Auto-generated method stub

 }
}
```

We have two test classes here:

ScreenShotTest:

```
import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.Test;
import com.base.BaseClass;

public class ScreenShotTest extends BaseClass {

 @Test
 public void testCase1() {
  setup();
  driver.findElement(By.name("q")).sendKeys("ScreenShot Demo");
  Assert.assertTrue(false);
 }
 @Test
 public void testCase2() {
  setup();
  Assert.assertTrue(false);
 }
 @AfterMethod
 public void tearDown() {
  driver.close();
 }
}
```

2nd TestClass:

```
import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.Test;
import com.base.BaseClass;

public class TestClass extends BaseClass {
 @Test
 public void testCase3() {
  setup();
  driver.findElement(By.name("q")).sendKeys("ScreenShot Demo");
  Assert.assertTrue(true);
 }
 @Test
 public void testCase4() {
  setup();
  Assert.assertTrue(false);
 }
 @AfterMethod
 public void tearDown() {
  driver.close();
 }
}
```

TestNG.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
<listeners>
<listener class-name="com.utility.ListenerClass"></listener>
</listeners>
  <test name="Test">
    <classes>
      <class name="com.testPackage.ScreenShotTest"/>
      <class name="com.testPackage.TestClass"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

- **How to take Full Page ScreenShot and ScreenShot of WebElement**

In this post, we are going to learn, how we can capture Full Page ScreenShot and ScreenShot of a WebElement using Third-party utility "Ashot" API.

We are going to cover the following points:
1. Full page screen shot using Ashot API.
2. ScreenShot of WebElement by Ashot API.
3. ScreenShot of WebElement by croping the image.

What is Ashot API?

Ashot is a third party utility by Yandex supported by Selenium WebDriver to capture the Screenshots.

How to download and configure Ashot API in your project?
1.Using Maven: get the latest maven Dependency from
https://mvnrepository.com/artifact/ru.yandex.qatools.ashot/ashot
and Copy the Dependency code and add to your pom.xml file
Example:

```
<dependency>
  <groupId>ru.yandex.qatools.ashot</groupId>
  <artifactId>ashot</artifactId>
  <version>1.5.3</version>
</dependency>
```

2.Manually without using any tool:
Download jar file from below location:

https://mvnrepository.com/artifact/ru.yandex.qatools.ashot/ashot
In Eclipse, right-click on the project -> go to properties -> Build Path -> Libraries -> Add External jars--> Select the jar file-->Apply and Close

1. Full page screen shot using Ashot API.
Step 1: Create an Ashot object. If you want a screenshot of the page bigger then the screen size, call the shootingStrategy() method before calling takeScreenshot() method to set up the policy. Then call a method takeScreenshot() passing the webdriver, for example,

```
Screenshot Screenshot = new AShot()
   .shootingStrategy(ShootingStrategies.viewportPasting(500))
   .takeScreenshot(driver);
```

Here 500 is scrolled out time in milliseconds, so for taking a screenshot, the program will scroll for each 500 msec.

Step 2: Now, get the image from the screenshot and write it to the file. You can provide the file type as jpg, png, etc.

```
ImageIO.write(Screenshot.getImage(), "png",
  new File("D:\\Workspace_Eclipse\\WebDriverConcept3\\ScreenShot\\Ashot.png"));
```

Complete code given below:

```
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;
import ru.yandex.qatools.ashot.AShot;
import ru.yandex.qatools.ashot.Screenshot;
import ru.yandex.qatools.ashot.shooting.ShootingStrategies;

public class AshotTest1 {
 WebDriver driver;

 @Test
 public void setup() throws InterruptedException {
  System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://www.automationtestinginsider.com/p/selenium-vi.html");

  //Create the object of AShot() class and set image strategy by shootingStrategy method
    //and viewportPasting method and take screenshot using takeScreenshot method
  Screenshot Screenshot = new AShot()
    .shootingStrategy(ShootingStrategies.viewportPasting(500))
    .takeScreenshot(driver);

  //Copy the element screenshot to desired location
  try {
   ImageIO.write(Screenshot.getImage(), "png",
     new File("D:\\Workspace_Eclipse\\WebDriverConcept3\\ScreenShot\\Ashot.png"));
  } catch (IOException e) {
```

```
   e.getMessage();
  }

  driver.close();
 }
}
```

Output: You will get full-page screenshot of a page which is bigger than screen size.

2. ScreenShot of WebElement by Ashot API: Below code given to take the screen shot of a webelement (Login Button) from OrangeHRM application using Ashot API.

```
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;
import ru.yandex.qatools.ashot.AShot;
import ru.yandex.qatools.ashot.Screenshot;
import ru.yandex.qatools.ashot.coordinates.WebDriverCoordsProvider;

public class AshotTest2 {
 WebDriver driver;

 @Test
 public void setup() throws InterruptedException {
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://opensource-demo.orangehrmlive.com/index.php/auth/validateCredentials");
  WebElement element = driver.findElement(By.id("btnLogin"));

  //Create the object of AShot() class and get the image co-ordinates
  //by coordsProvider method and take screenshot using takeScreenshot method
  Screenshot screenshot = new AShot()
    .coordsProvider(new WebDriverCoordsProvider())
    .takeScreenshot(driver,element);

  //Copy the element screenshot to desired location
  try {
   ImageIO.write(screenshot.getImage(), "png",
     new File("D:\\Workspace_Eclipse\\WebDriverConcept3\\ScreenShot\\Ashot2.png"));
  } catch (IOException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
```

```
 driver.close();
 }
}
```

Output: Looks like below

3. ScreenShot of WebElement by croping the image: Below code given to take the screen shot of a webelement (Login Button) from WordPress application using Cropping technique.

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.By;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.Point;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;

public class Test1 {

 WebDriver driver;

 @Test
 public void setup() throws IOException  {
  System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://s1.demo.opensourcecms.com/wordpress/wp-login.php?");
  WebElement element = driver.findElement(By.id("wp-submit"));

  // Get entire page screenshot
  File screenshot = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
      BufferedImage  fullImg = ImageIO.read(screenshot);
```

```
 // Get the location,height,width of element on the page
 Point point = element.getLocation();
    int eleWidth = element.getSize().getWidth();
 int eleHeight = element.getSize().getHeight();
 // Crop the entire page screenshot to get only element screenshot
 try {
  BufferedImage eleScreenshot= fullImg.getSubimage(point.getX(), point.getY(),
      eleWidth, eleHeight);
  ImageIO.write(eleScreenshot, "png", screenshot);
 } catch (Exception e) {
  // TODO Auto-generated catch block
  e.getMessage();
 }
 // Copy the element screenshot to desired location
 File screenshotLocation = new
File("D:\\Workspace_Eclipse\\WebDriverConcept3\\ScreenShot\\Ashot3.png");
 FileUtils.copyFile(screenshot, screenshotLocation);
 // Close the browser
 driver.close();
 }

}
```

Output: Looks like below

- **Generate Logs in Selenium Web Driver using log4j**

What is log and why it is required?

During the execution of test case, user wants some information to be logged in the console. Information could be any detail depends upon the purpose. Keeping this in mind that we are using Selenium for testing, we need the information which helps the user to understand the test steps or any failure during the test case execution. With the help of Log4j it is possible to enable loggings during the Selenium test case execution

What is log4j?

- Log4j is a fast, flexible and reliable logging framework written in Java developed in 1996. It is distributed under the Apache Software License.
- This is used for logging mechanism for small to large scale selenium automation projects.

Why log4j?

- It is an open source
- With Log4j, it is possible to store the flow details of our Selenium Automation in a file, console or database.
- It can be used for large as well as small projects.

Log 4j Components:

- **Loggers:** It is responsible for logging information.

Create the instance of logger class
Log4j level: Primarily there are five kinds of log levels

1. All - This level of logging will log everything (it turns all the logs on)
2. DEBUG – print the debugging information and is helpful in development stage
3. INFO – print informational message that highlights the progress of the application
4. WARN – print information regarding faulty and unexpected system behavior.
5. ERROR – print error message that might allow system to continue
6. FATAL – print system critical information which are causing the application to crash
7. OFF – No logging

- **Appenders:** It is used to deliver LogEvents to their destination.

1. ConsoleAppender logs to standard output
2. File appender prints logs to some file
3. Rolling file appender to a file with maximum size

- **Layouts:** It is responsible for formatting logging information in different styles.

Steps:

1. Create a Project and add required dependencies (like selenium-java and TestNG etc.) if you are using Maven Project.
2. Download log4j jar files or get the dependency if you are using maven project.
3. Create an log4j.xml configuration file. Keep it in your project directory.

4. Create a utility class (Log) to Initialize Log4j logs.
5. Create Base Class and configure your log4j.xml using DOMConfigurator.
6. Write your test cases using test class.

Please refer below complete setup and project.

log4j.xml file

```
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration debug="true" xmlns:log4j='http://jakarta.apache.org/log4j/'>

  <appender name="console" class="org.apache.log4j.ConsoleAppender">
    <param name="Target" value="System.out"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L -
%m%n" />
    </layout>
  </appender>

  <appender name="fileAppender" class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="log4j.log"/>
    <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L -
%m%n" />
    </layout>
  </appender>

  <root>
    <priority value ="debug"></priority>
    <appender-ref ref="console"></appender-ref>
    <appender-ref ref="fileAppender"></appender-ref>
  </root>

</log4j:configuration>
```

log4j Utility Class:

```
package com.utility;

import org.apache.log4j.Logger;

public class Log {
```

```
 // Initialize Log4j logs
 public static Logger Log = Logger.getLogger(Log.class.getName());//

 public static void startTestCase(String sTestCaseName){
   Log.info("====================================="+sTestCaseName+" TEST
START=========================================");
   }

 public static void endTestCase(String sTestCaseName){

  Log.info("====================================="+sTestCaseName+" TEST
END=========================================");
   }

 // Need to create below methods, so that they can be called

  public static void info(String message) {

   Log.info(message);

   }

  public static void warn(String message) {

     Log.warn(message);

  }

  public static void error(String message) {

     Log.error(message);

  }

  public static void fatal(String message) {

     Log.fatal(message);

  }
```

```
 public static void debug(String message) {

    Log.debug(message);

 }

}
```

BaseClass: To configure log4j.xml file and initialize web driver.

```
import org.apache.log4j.xml.DOMConfigurator;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;

import com.utility.Log;

public class BaseClass {
 public WebDriver driver;

 @BeforeSuite
 public void beforeSuite() {
  DOMConfigurator.configure("log4j.xml");
  Log.info("This is beforeSuite Method");
 }

 @BeforeMethod
 public void setup() {
  System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://s1.demo.opensourcecms.com/wordpress/wp-login.php");
 }

 @AfterMethod
```

```
 public void tearDown() {
  driver.close();
 }

 @AfterSuite
 public void afterSuite() {
  Log.info("This is afterSuite Method");
 }
}
```

TestClass1:

```
import org.testng.annotations.Test;
import com.utility.Log;

public class TestClass1 {

 @Test
 public void testCase1() {
  Log.startTestCase("testCase1");
  Log.info("This is testCase1");
  Log.endTestCase("testCase1");
 }
 @Test
 public void testCase2() {
  Log.startTestCase("testCase2");
  Log.info("This is testCase2");
  Log.endTestCase("testCase2");
 }
 @Test
 public void testCase3() {
  Log.startTestCase("testCase3");
  Log.info("This is testCase3");
  Log.endTestCase("testCase3");
 }

}
```

TestClass2:

```
import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.Test;
import com.baseClass.BaseClass;
import com.utility.Log;

public class TestClass2 extends BaseClass {

 @Test
 public void wordPressTest() {
  Log.startTestCase("wordPressTest");
  Log.info("Entering User Name");
  driver.findElement(By.id("user_login")).sendKeys("opensourcecms");
  Log.info("Entering User Password");
  driver.findElement(By.id("user_pass")).sendKeys("opensourcecms");
  Log.info("Click on Login Button");
  driver.findElement(By.id("wp-submit")).click();
  Log.info("Home Page");
  Log.info("Get Current URL");
  String urlString=driver.getCurrentUrl();
  Log.info("Validate the URL");
  String expectedURL="https://s1.demo1.opensourcecms.com/wordpress/wp-admin/";
  Assert.assertEquals(urlString, expectedURL);
  Log.info("URL Matches");
  Log.endTestCase("wordPressTest");
 }

}
```

Output: Following logs will be captured in file as well as on console.

```
2020-05-04 00:21:50 INFO  Log:23 - This is beforeSuite Method
2020-05-04 00:21:50 INFO  Log:11 -
==================================testCase1 TEST
START=======================================
2020-05-04 00:21:50 INFO  Log:23 - This is testCase1
```

2020-05-04 00:21:50 INFO Log:16 - ==========testCase1 TEST END==========
2020-05-04 00:21:50 INFO Log:11 - ==========testCase2 TEST START==========
2020-05-04 00:21:50 INFO Log:23 - This **is** testCase2
2020-05-04 00:21:50 INFO Log:16 - ==========testCase2 TEST END==========
2020-05-04 00:21:50 INFO Log:11 - ==========testCase3 TEST START==========
2020-05-04 00:21:50 INFO Log:23 - This **is** testCase3
2020-05-04 00:21:50 INFO Log:16 - ==========testCase3 TEST END==========
2020-05-04 00:21:58 INFO Log:11 - ==========wordPressTest TEST START==========
2020-05-04 00:21:58 INFO Log:23 - Entering User Name
2020-05-04 00:21:58 INFO Log:23 - Entering User Password
2020-05-04 00:21:59 INFO Log:23 - Click on Login Button
2020-05-04 00:22:02 INFO Log:23 - Home Page
2020-05-04 00:22:02 INFO Log:23 - Get Current URL
2020-05-04 00:22:02 INFO Log:23 - Validate the URL
2020-05-04 00:22:03 INFO Log:23 - This **is** afterSuite Method

- **Generate Extent report and attach screen shots in extent report**

What is Extent Report

ExtentReports is an open-source reporting library useful for test automation. It can also be easily integrated with major testing frameworks like JUnit, NUnit, TestNG, etc. These reports are HTML rich documents that depict results as pie charts.

Extent Reports offer several advantages when compared to the built-in reports that are generated through JUnit and TestNG such as test stepwise report generation ,pie chart representation, adding screenshots etc.

What are the main classes in Extent Report

ExtentHtmlReporter - To create/generate report and for look and feel of the reports.
ExtentReports - Add your test cases in extent report.
ExtentTest - update the status of the test cases like pass/fail/skip etc.

Please refer below Program to generate extent report.

BaseClass:

```
import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.ITestResult;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import com.aventstack.extentreports.ExtentReports;
import com.aventstack.extentreports.ExtentTest;
import com.aventstack.extentreports.Status;
import com.aventstack.extentreports.markuputils.ExtentColor;
import com.aventstack.extentreports.markuputils.MarkupHelper;
import com.aventstack.extentreports.reporter.ExtentHtmlReporter;
import com.aventstack.extentreports.reporter.configuration.Theme;

public class BaseClass {
 public WebDriver driver;

 public ExtentHtmlReporter htmlReporter;
 public ExtentReports extent;
 public ExtentTest test;

 @BeforeSuite
 public void setExtent() {
  htmlReporter= new ExtentHtmlReporter(System.getProperty("user.dir")+"/test-
output/ExtentReport/MyReport.html");
```

```
 htmlReporter.config().setDocumentTitle("Automation Test Report");
 htmlReporter.config().setReportName("OrangeHRM Test Automation Report");
 htmlReporter.config().setTheme(Theme.DARK);

 extent = new ExtentReports();
 extent.attachReporter(htmlReporter);

 extent.setSystemInfo("HostName", "MyHost");
 extent.setSystemInfo("ProjectName", "OrangeHRM");
 extent.setSystemInfo("Tester", "Hitendra");
 extent.setSystemInfo("OS", "Win10");
 extent.setSystemInfo("Browser", "Chrome");

 }

 @AfterSuite
 public void endReport() {
  extent.flush();
 }

 @BeforeMethod
 public void setup() {
   System.setProperty("webdriver.chrome.driver",
     "C:\\Users\\Hitendra\\Downloads\\chromedriver_win32 (1)\\chromedriver.exe");
   driver = new ChromeDriver();
   driver.manage().window().maximize();
   driver.get("https://opensource-demo.orangehrmlive.com/index.php/auth/validateCredentials");
 }

 @AfterMethod
 public void tearDown(ITestResult result) throws IOException {
  if(result.getStatus()==ITestResult.FAILURE) {
   test.log(Status.FAIL, MarkupHelper.createLabel(result.getName() + " - Test Case Failed",
ExtentColor.RED));
   test.log(Status.FAIL, MarkupHelper.createLabel(result.getThrowable() + " - Test Case Failed",
ExtentColor.RED));

   String pathString=BaseClass.screenShot(driver, result.getName());
   test.addScreenCaptureFromPath(pathString);
```

```
  } else if(result.getStatus()==ITestResult.SKIP) {
   test.log(Status.SKIP, "Skipped Test case is: "+result.getName());
  } else if(result.getStatus()==ITestResult.SUCCESS) {
   test.log(Status.PASS, "Pass Test case is: "+result.getName());
  }
  driver.close();
 }

 public static String screenShot(WebDriver driver,String filename) {
  String dateName = new SimpleDateFormat("yyyyMMddhhmmss").format(new Date());
  TakesScreenshot takesScreenshot = (TakesScreenshot) driver;
  File source = takesScreenshot.getScreenshotAs(OutputType.FILE);
  String destination =
System.getProperty("user.dir")+"\\ScreenShot\\"+filename+"_"+dateName+".png";
  File finalDestination= new File(destination);
  try {
   FileUtils.copyFile(source, finalDestination);
  } catch (Exception e) {
   // TODO Auto-generated catch block
   e.getMessage();
  }
  return destination;
 }

}
```

TestClass:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.testng.Assert;
import org.testng.annotations.Test;
import com.basePackage.BaseClass;

public class OrangeHRMTest extends BaseClass {

 @Test
 public void loginPageTest() {
  test=extent.createTest("loginPageTest");
```

```
 WebElement imgElement=driver.findElement(By.xpath("//*[@id=\"divLogo\"]/img"));
 Assert.assertTrue(imgElement.isDisplayed());
}

@Test
public void loginTest() {
 test=extent.createTest("loginTest");
 driver.findElement(By.id("txtUsername")).sendKeys("admin");
 driver.findElement(By.id("txtPassword")).sendKeys("admin123");
 driver.findElement(By.id("btnLogin")).click();
      String actualTitle=driver.getTitle();
      String expectedTitle="OrangeHRM";
      Assert.assertEquals(actualTitle, expectedTitle);
}

@Test
public void sampleCase() {
 test=extent.createTest("sampleCase");
 test.createNode("Validation1");
 Assert.assertTrue(true);
 test.createNode("Validation2");
 Assert.assertTrue(true);
 test.createNode("Validation3");
 Assert.assertTrue(true);
 test.createNode("Validation4");
 Assert.assertTrue(true);
 }
}
```

- **Extent Report implementation using ITestListener**

In this post we will implement the extent report using listeners. Below complete implementation code given in the form of different classes/components.

POM.xml: to setup selenium-java, testng, extent report etc dependencies.

```
<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>ExtentDemo</groupId>
<artifactId>ExtentDemo</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>
 <dependency>
  <groupId>org.testng</groupId>
  <artifactId>testng</artifactId>
  <version>6.14.3</version>
  <scope>test</scope>
 </dependency>
 <dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>3.141.59</version>
 </dependency>
 <dependency>
  <groupId>com.aventstack</groupId>
  <artifactId>extentreports</artifactId>
  <version>4.0.9</version>
 </dependency>
 <dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.6</version>
 </dependency>
</dependencies>
</project>
```

BaseClass: webdriver initialization, setup method and common methods.

```
import java.io.File;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterMethod;
```

```
import org.testng.annotations.AfterSuite;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import com.extentManager.ExtentManager;

public class BaseClass {
 public static WebDriver driver;

 @BeforeSuite
 public void BeforeSuite() {
  ExtentManager.setExtent();
 }

 @AfterSuite
 public void AfterSuite() {
  ExtentManager.endReport();
 }

 @BeforeMethod
 public void setup() {
   System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
   driver = new ChromeDriver();
   driver.manage().window().maximize();
   driver.get("https://opensource-demo.orangehrmlive.com/index.php/auth/validateCredentials");
 }

 @AfterMethod
 public void tearDown() throws IOException {
  driver.close();
 }

 public static String screenShot(WebDriver driver,String filename) {
  String dateName = new SimpleDateFormat("yyyyMMddhhmmss").format(new Date());
  TakesScreenshot takesScreenshot = (TakesScreenshot) driver;
  File source = takesScreenshot.getScreenshotAs(OutputType.FILE);
  String destination =
System.getProperty("user.dir")+"\\ScreenShot\\"+filename+"_"+dateName+".png";
  File finalDestination= new File(destination);
  try {
   FileUtils.copyFile(source, finalDestination);
```

```
 } catch (Exception e) {
  // TODO Auto-generated catch block
  e.getMessage();
 }
 return destination;
}

public static String getCurrentTime() {
    String currentDate = new SimpleDateFormat("yyyy-MM-dd-hhmmss").format(new Date());
    return currentDate;
}
}
```

extent-config.xml to configure Extent Report

```
<?xml version="1.0" encoding="UTF-8"?>
<extentreports>
   <configuration>
      <!-- report theme -->
      <!-- standard, dark -->
      <theme>standard</theme>

      <!-- document encoding -->
      <!-- defaults to UTF-8 -->
      <encoding>UTF-8</encoding>

      <!-- protocol for script and stylesheets -->
      <!-- defaults to https -->
      <protocol>http</protocol>

      <!-- title of the document -->
      <documentTitle>OrangeHRMProject</documentTitle>

      <!-- report name - displayed at top-nav -->
      <reportName>OrangeHRMProject Report
       <![CDATA[
            <img src='D:/Workspace_Eclipse/ExtentDemo/Logo/ATI.png'/>
        ]]>
      </reportName>
```

```
    <!-- location of charts in the test view -->
    <!-- top, bottom -->
    <testViewChartLocation>bottom</testViewChartLocation>

    <!-- custom javascript -->
    <scripts>
      <![CDATA[
        $(document).ready(function() {

        });
      ]]>
    </scripts>

    <!-- custom styles -->
    <styles>
      <![CDATA[
      .report-name { padding-left: 18px; } .report-name > img { float:
  left;height: 90%;margin-left: 30px;margin-top: 3px;width: auto; }
    ]]>
    </styles>
  </configuration>
</extentreports>
```

ExtentManger Class: to setup extent report

```
import com.aventstack.extentreports.ExtentReports;
import com.aventstack.extentreports.ExtentTest;
import com.aventstack.extentreports.reporter.ExtentHtmlReporter;
import com.basePackage.BaseClass;

public class ExtentManager {

 public static ExtentHtmlReporter htmlReporter;
 public static ExtentReports extent;
 public static ExtentTest test;

 public static void setExtent() {
  htmlReporter= new ExtentHtmlReporter(System.getProperty("user.dir")+"/test-
output/ExtentReport/"+"MyReport_"+BaseClass.getCurrentTime()+".html");
  htmlReporter.loadXMLConfig(System.getProperty("user.dir")+"/extent-config.xml");
  //htmlReporter.config().setDocumentTitle("Automation Test Report");
```

```
 //htmlReporter.config().setReportName("OrangeHRM Test Automation Report");
 //htmlReporter.config().setTheme(Theme.DARK);

 extent = new ExtentReports();
 extent.attachReporter(htmlReporter);

 extent.setSystemInfo("HostName", "MyHost");
 extent.setSystemInfo("ProjectName", "OrangeHRM");
 extent.setSystemInfo("Tester", "Hitendra");
 extent.setSystemInfo("OS", "Win10");
 extent.setSystemInfo("Browser", "Chrome");
 }
 public static void endReport() {
  extent.flush();
 }
}
```

Listener Class: configure listener class

```
import java.io.IOException;
import org.testng.ITestContext;
import org.testng.ITestListener;
import org.testng.ITestResult;
import com.aventstack.extentreports.Status;
import com.aventstack.extentreports.markuputils.ExtentColor;
import com.aventstack.extentreports.markuputils.MarkupHelper;
import com.basePackage.BaseClass;
import com.extentManager.ExtentManager;

public class ListenerClass extends ExtentManager implements ITestListener {

 public void onTestStart(ITestResult result) {
  test = extent.createTest(result.getName());
 }
 public void onTestSuccess(ITestResult result) {
  if (result.getStatus() == ITestResult.SUCCESS) {
   test.log(Status.PASS, "Pass Test case is: " + result.getName());
  }
 }
 public void onTestFailure(ITestResult result) {
  if (result.getStatus() == ITestResult.FAILURE) {
```

```
  test.log(Status.FAIL, MarkupHelper.createLabel(result.getName() + " - Test Case Failed",
ExtentColor.RED));
  test.log(Status.FAIL,
    MarkupHelper.createLabel(result.getThrowable() + " - Test Case Failed ",
ExtentColor.RED));

  String pathString = BaseClass.screenShot(BaseClass.driver, result.getName());
  try {
   test.addScreenCaptureFromPath(pathString);
  } catch (IOException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
 }
}

public void onTestSkipped(ITestResult result) {
 if (result.getStatus() == ITestResult.SKIP) {
  test.log(Status.SKIP, "Skipped Test case is: " + result.getName());
 }
}
public void onTestFailedButWithinSuccessPercentage(ITestResult result) {
 // TODO Auto-generated method stub
}
public void onStart(ITestContext context) {
 // TODO Auto-generated method stub

}
public void onFinish(ITestContext context) {
 // TODO Auto-generated method stub
}
}
```

TestClass: create test cases

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.testng.Assert;
import org.testng.annotations.Test;
import com.basePackage.BaseClass;
import com.extentManager.ExtentManager;
```

```
public class OrangeHRMTest extends BaseClass {

 @Test
 public void loginPageTest() {

  WebElement imgElement=driver.findElement(By.xpath("//*[@id=\"divLogo\"]/img"));
  Assert.assertTrue(imgElement.isDisplayed());
 }
 @Test
 public void loginTest() {
  driver.findElement(By.id("txtUsername")).sendKeys("admin");
  driver.findElement(By.id("txtPassword")).sendKeys("admin123");
  driver.findElement(By.id("btnLogin")).click();
       String actualTitle=driver.getTitle();
       String expectedTitle="OrangeHRM1";
       Assert.assertEquals(actualTitle, expectedTitle);
 }
 @Test
 public void sampleCase() {
  ExtentManager.test.createNode("Validation1");
  Assert.assertTrue(true);
  ExtentManager.test.createNode("Validation2");
  Assert.assertTrue(true);
  ExtentManager.test.createNode("Validation3");
  Assert.assertTrue(true);
  ExtentManager.test.createNode("Validation4");
  Assert.assertTrue(true);
 }
}
```

testng.xml - to run test suites

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
<listeners>
<listener class-name="com.listener.ListenerClass"></listener>
</listeners>
  <test thread-count="5" name="Test">
    <classes>
```

```
    <class name="com.test.OrangeHRMTest"/>
   </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

TestNG Tutorials:

- **Introduction to TestNG - Features and Installation Guide**

In this article we will cover the following points:

- What is TestNG?
- Different Features of TestNG
- TestNG vs Junit
- Install TestNG Plugin
- Write First TestNG Program

What is TestNG?

- TestNG is an automation testing framework.
- NG stands for "Next Generation".
- Java unit testing framework.
- TestNG is an advance framework designed in a way to leverage the benefits by both the developers and testers.
- TestNG is a testing framework inspired from JUnit and NUnit but introducing some new functionalities that make it more powerful and easier to use.

TestNG Features:

- TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests.
- Support for annotations (@).
- Using TestNG you can generate a proper report and can see passed, failed, skipped test results.
- Test case Prioritization.
- Support for Data Driven Testing using Data providers.
- Multiple test cases can be grouped more easily by converting them into testng.xml file.
- The same test case can be executed multiple times without loops just by using keyword called 'invocation count.'.
- Cross browser testing.
- Parallel Testing is possible using TestNG.

Advantages of TestNG over Junit:

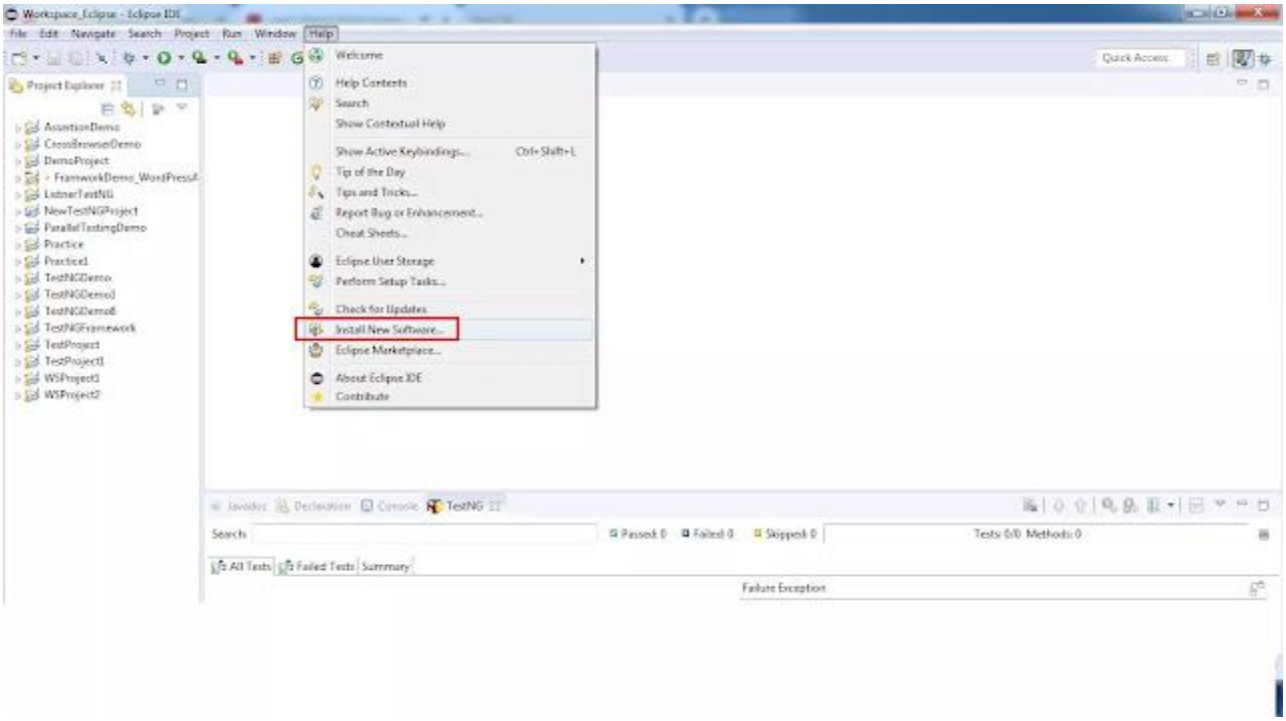
- There are three major advantages of TestNG over Junit.
- More Annotations than JUnit and Annotations are easier to understand
- Test cases can be grouped more easily.
- Parallel testing is possible.
- The testing framework can be easily integrated with tools like Maven, Jenkins, etc.

Install TestNG step by step:

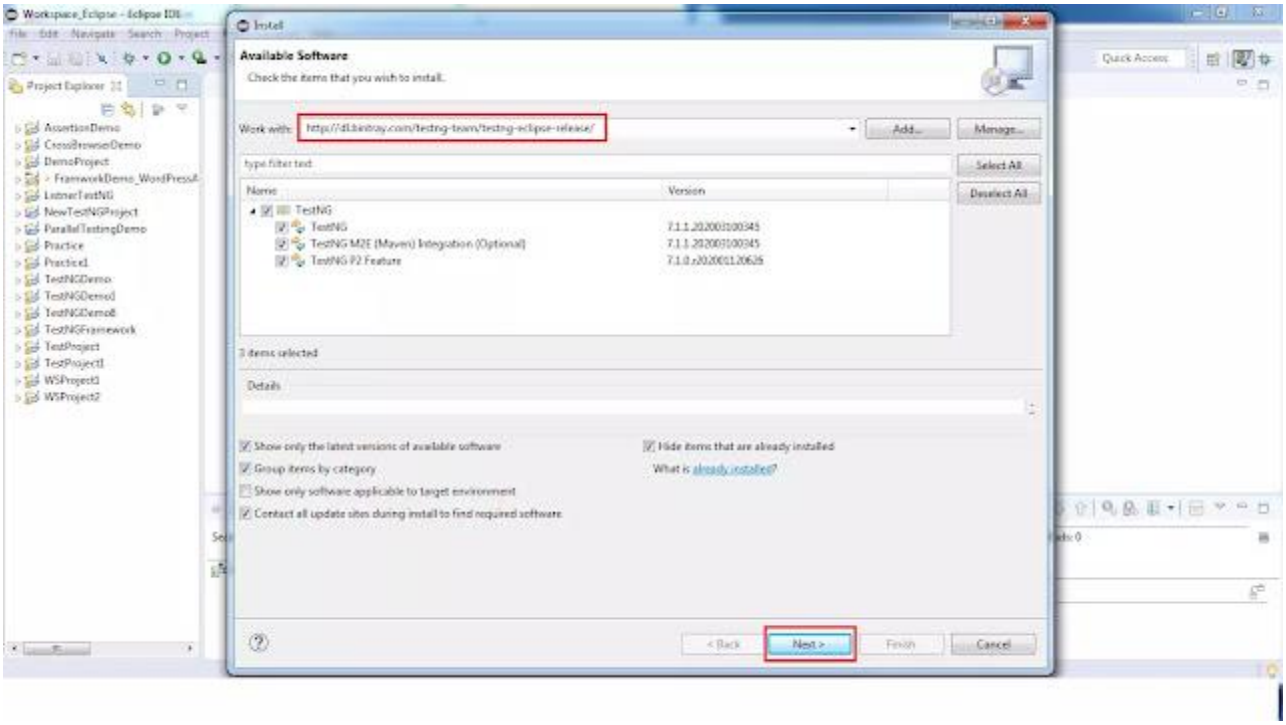
There are two ways you can install TestNG in Eclipse.
First Way on installing Eclipse is using "Install new software" option.
Second way is using "Eclipse Market Place". - This option will be available in new versions of eclipse.

Lets have a look at the first version:

Step 1:
In Eclipse, on top menu bar, Under Help Menu, Click on "Install new Software" in help window.



Step 2:
Enter the URL (http://dl.bintray.com/testng-team/testng-eclipse-release/) at Work With field and click on "Add" button. Add the URL and later click on next.

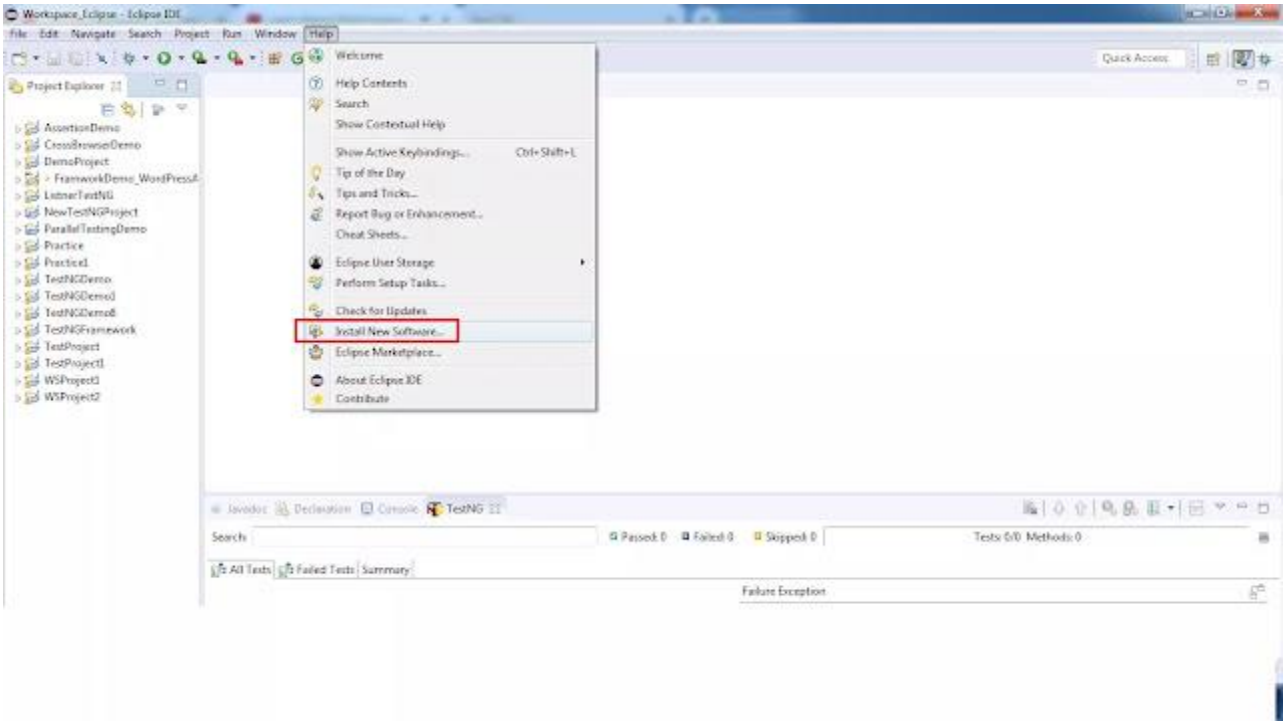


Step 3:
It will check for the requirement and dependencies before starting the installation. Once the above step is done, it will ask you to review the installation details. If your are ready or OK to install TestNG, click on "Next" to continue..Later Accept the Terms of the license agreement and Click on "Finish" button. If there is any problem with the requirements/dependencies, it will ask you to install them first before continuing with TestNG. Most of the cases it will successfully get installed nothing to worry about it.

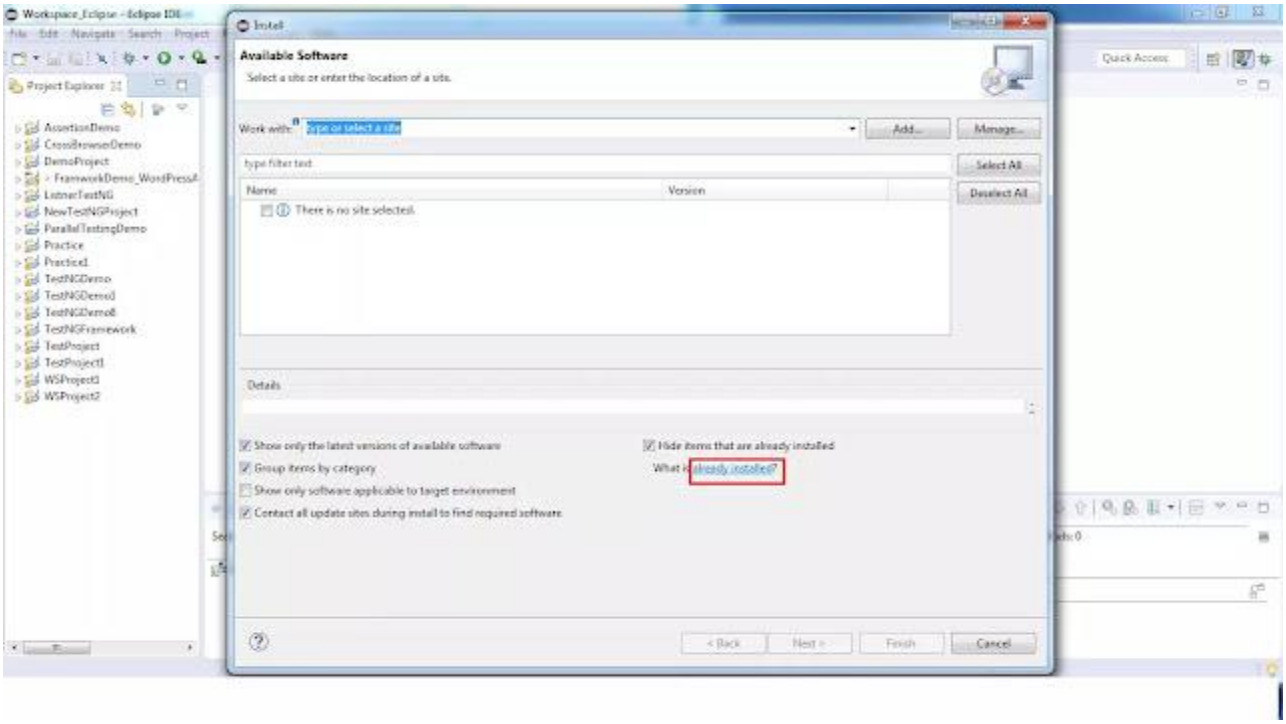
It will take few minutes to get installed.

Verify TestNG installed or not?

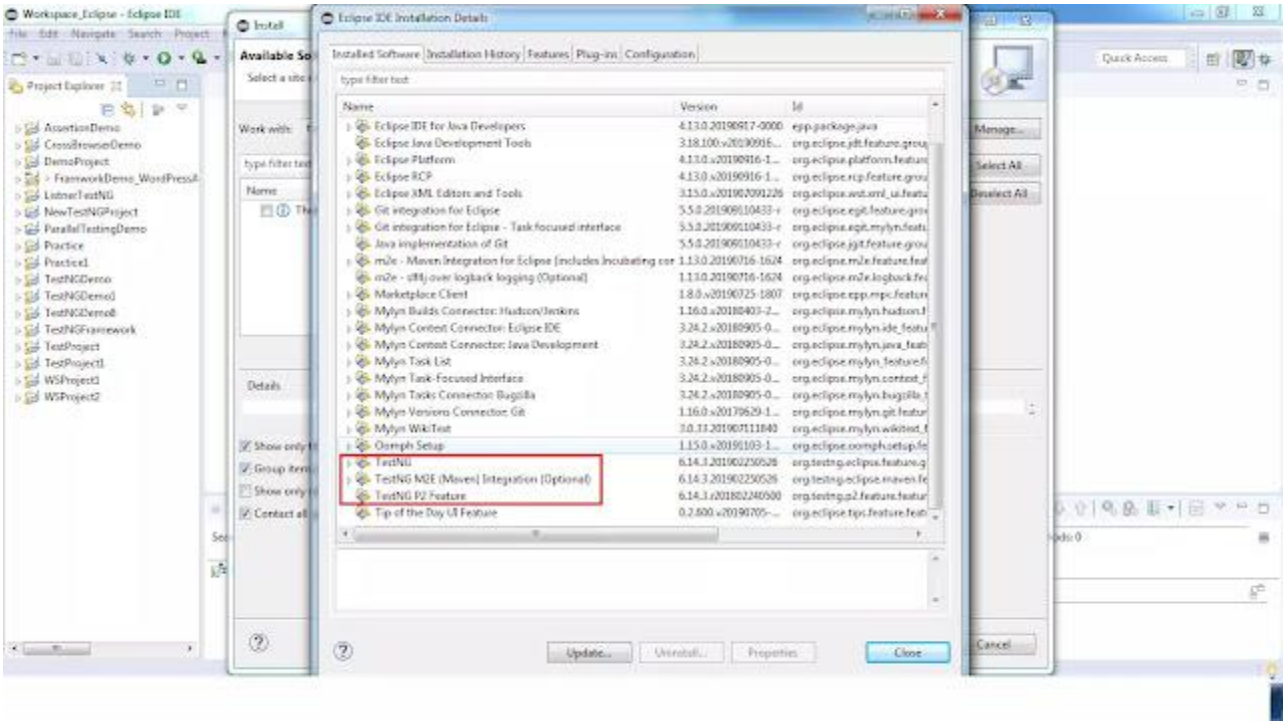
Step 4: In Eclipse, on top menu bar, Under Help Menu, Click on "Install new Software" in help window.



Step 5: Click on already installed?



Step 6: You can verify if the TestNG is available as per the below screen shot.



- **TestNG Annotations - Different Annotations and their execution**

What is TestNG Annotation?

TestNG Annotation is a piece of code which is inserted inside a program or business logic used to control the flow of execution of test methods.

List of TestNG Annotations

Below are the different annotations used in TestNG

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

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

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

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

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

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

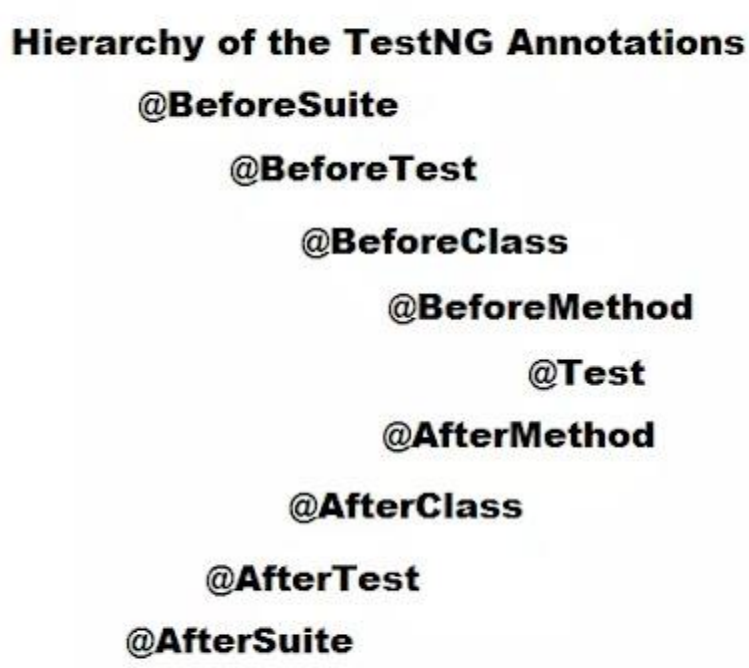
@BeforeMethod: The annotated method will be run before each test method.

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

@BeforeGroups - The @BeforeGroups annotated method run only once for a group before the execution of all test cases belonging to that group.

@AfterGroups - The @AfterGroups annotated method run only once for a group after the execution of all test cases belonging to that group.

Hierarchy of the TestNG Annotations:



TestNG Annotations
Benefits of using TestNG Annotations:

- TestNG Annotations made the life of testers very easy. Based on your requirements, you can access the test methods, i.e., it has no predefined pattern or format.

- You can pass the additional parameters to TestNG annotations.
- In the case of TestNG annotations, you do not need to extend any test classes.
- TestNG Annotations are strongly typed, i.e., errors are detected at the compile time.

Different Attributes in TestNG

1. priority attribute :

In TestNG "Priority" is used to schedule the test cases. When there are multiple test cases, we want to execute test cases in order. Like First we need to execute a test case "Registration" before login.
In order to achive, we use need to add annotation as @Test(priority=??). The default value will be zero for priority.
If you don't mention the priority, it will take all the test cases as "priority=0" and execute alphabetically.
Lower priorities will be scheduled first.
For example if you give the priority like @Test(priority=0), @Test(priority=-1), @Test(priority=1), @Test(priority=2)
Then test will be executed in following order: @Test(priority=-1), @Test(priority=0), @Test(priority=1), @Test(priority=2)

2. description attribute: gives information about the test it is attached to.
Example:

```
@Test(description='Title Test')
public void titleTest(){
  String actualTitle = driver.getTitle();
  String expectedTitle = "OrangeHRM";
  Assert.assertEquals(actualTitle, expectedTitle);
}
```

3. timeOut: maximum number of milliseconds for a test run.
Example: 5000 milliseconds (5 seconds) time allotted to run below test method.

```
@Test(timeOut=5000)
public void testCase(){

}
```

4. dependsOnMethods: specify when you want to run a test, only after another test has run successfully, making the second test's run dependent on the first test's successful outcome.
Example:

```
@Test(priority = 1)
 public void testcase3() {
  String actualUrl = driver.getCurrentUrl();
  String expectedURL = "https://opensource-demo.orangehrmlive.com/index.php/dashboard";
  Assert.assertEquals(actualUrl, expectedURL);
 }
@Test(priority = 2, dependsOnMethods = "testcase3")
  public void testcase4() {
  String actualTitle = driver.getTitle();
  String expectedTitle = "OrangeHRM";
  Assert.assertEquals(actualTitle, expectedTitle);
 }
```

5. enabled: this attribute has boolean values, and by default is 'true'. Only worth specifying explicitly when you don't want a certain test method or class to be run, by setting the attribute to false.

Example: Below test case will not be executed as specified enabled = false.

```
@Test(enabled = false)
  public void testcase4() {
  String actualTitle = driver.getTitle();
  String expectedTitle = "OrangeHRM";
  Assert.assertEquals(actualTitle, expectedTitle);
 }
```

6. groups: useful for grouping together tests that relate to the same functionality, are of the same importance or are of the same type.

7. alwaysRun- If set to true, this test method will always be run even if it depends on a method that failed.

Example: Below testcase4 will always execute even it depends on a method that fails (testcase3)

```
@Test(priority = 1)
 public void testcase3() {
  String actualUrl = driver.getCurrentUrl();
  String expectedURL = "https://opensource-demo.orangehrmlive.com/index.php/dashboard";
  Assert.assertEquals(actualUrl, expectedURL);
 }

@Test(priority = 2, dependsOnMethods = "testcase3", alwaysRun = true)
  public void testcase4() {
  String actualTitle = driver.getTitle();
```

```
 String expectedTitle = "OrangeHRM";
 Assert.assertEquals(actualTitle, expectedTitle);
}
```

8. invocationcount - The number of times this method should be invoked.

Example: Below testcase3 will execute 3 times.

```
@Test(priority = 4, description = "Verify URL", invocationCount = 3)
 public void testcase3() {
 String actualUrl = driver.getCurrentUrl();
 String expectedURL = "https://opensource-demo.orangehrmlive.com/index.php/dashboard";
 Assert.assertEquals(actualUrl, expectedURL);
}
```

9. How to ignore test case - We can ignore test at test, method and package levels

Example:

1. Test Level - Below test method will be ignored as @Ignore specified at Test Level

```
@Ignore
@Test(priority = 4, description = "Verify URL", invocationCount = 3)
 public void testcase3() {
 String actualUrl = driver.getCurrentUrl();
 String expectedURL = "https://opensource-demo.orangehrmlive.com/index.php/dashboard";
 Assert.assertEquals(actualUrl, expectedURL);
}
```

2. Class Level - All test methods will be ignored for this classe as @Ignore specified at class Level

```
@Ignore
public class OrangeHRMTest1 {
 WebDriver driver;

 @Test(priority = 1)
 public void testcase1() {
 driver.get("https://opensource-demo.orangehrmlive.com/index.php/auth/login");
 boolean img = driver.findElement(By.xpath("//*[@id=\"divLogo\"]/img")).isDisplayed();
 Assert.assertTrue(img);
}

 @Test(priority = 2)
 public void testcase2() {
 String actualTitle = driver.getTitle();
```

```
  String expectedTitle = "OrangeHRM";
  Assert.assertEquals(actualTitle, expectedTitle);
 }

}
```

3. Package Level - All test classes will be ignored under this package
Example

```
@org.testng.annotations.Ignore
package testNGPackage;
```

10. @Test annotation at Class Level - TestNG has great feature to define annotations on a class instead of a each test method.
If say suppose there are 10 test methods, where adding @Test on a class level is simpler than adding @Test for each method.
When we make class level @Test annotation, all the public methods of this class will become test methods even if they are not annotated.
We can still define @Test annotation on of the method if we want to add any attributes to particular test method.

Example:

```
@Test
public class OrangeHRMTest {
 WebDriver driver;

 @BeforeClass
 public void setup() {
  System.setProperty("webdriver.chrome.driver",
    "C:\\Users\\Hitendra\\Downloads\\chromedriver_win32\\chromedriver.exe");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
 }
public void testCase1(priority=1) {
 }
public void testCase2(priority=2) {
 }
}
```

- **Working with testng.xml**

Working with testng.xml

In this post we will discuss about following points:

- What is testng.xml file
- Importance of testng.xml
- How to generate?
- Different ways to setup testng.xml
- How to create testng.xml file manually
- How to run multiple testng.xml in one go

What is testng.xml?

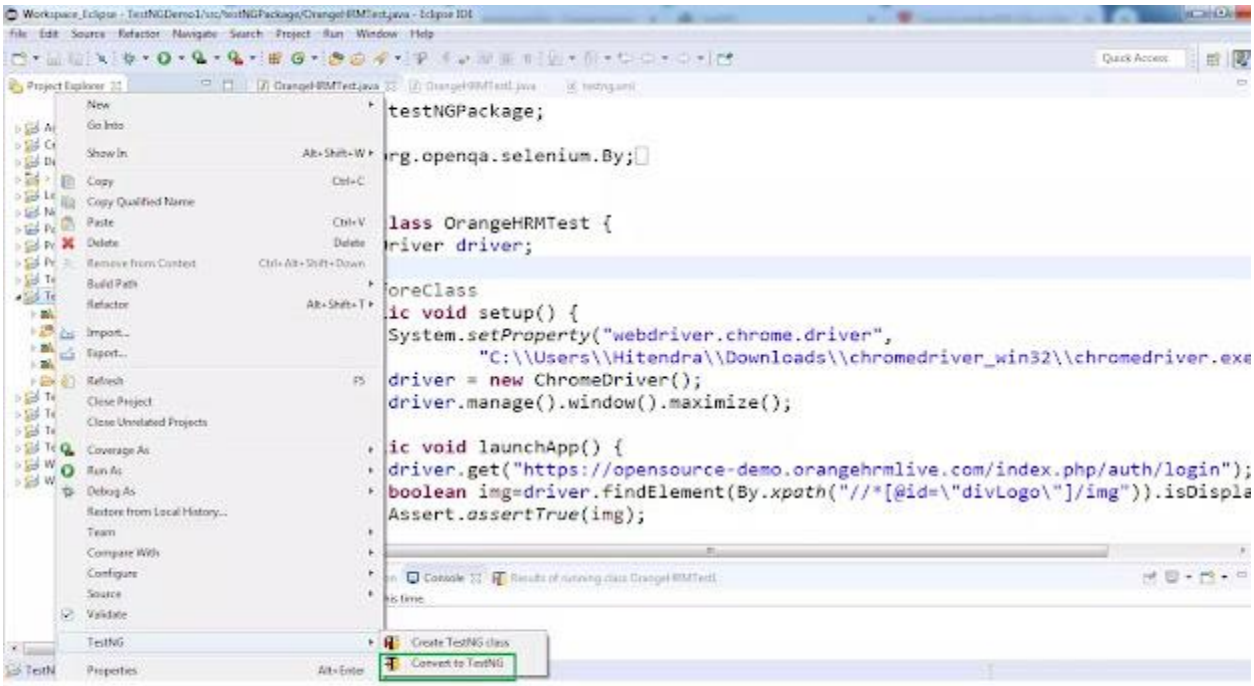
- testng. xml file is a configuration file in TestNG.
- It is used to define test suites and tests. It provides different options to include packages, classes and independent test methods in our test suite.
- It also allows us to configure multiple tests in a single test suite and run them in multi threaded environment.

Importance of testng.xml

In TestNG, you can define multiple test cases in a single class whereas, in Java, you can define only one test in a single class in the main() method. In Java, if you want to create one more test, then you need to create another java file and define the test in the main() method.
Instead of creating test cases in different classes, we recommend you to use TestNG framework that allows you to create multiple test cases in a single class.

You can create multiple test cases with the help of @Test annotation.
You can execute all the test cases from a single file known as xml file. Xml file is the heart of TestNG framework.

How to generate testng.xml?
Right click on the project. Move your cursor down, and you will see TestNG and then click on the Convert to TestNG.



Create testng.xml

Different ways to setup testng.xml:

Below xml describes the different format of testng.xml

Example1:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test name="Test">
    <classes>
      <class name="com.packageName.Class1"/>
      <class name=" com.packageName.Class2"/>
      <class name=" com.packageName.Class3"/>
</classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

You can specify package names instead of class names:

Example2:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test name="Test">
    <packages>
```

```
    <package name="com.pack1"></package>
    <package name="com.pack2"></package>
    <package name="com.pack3"></package>
    </packages>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

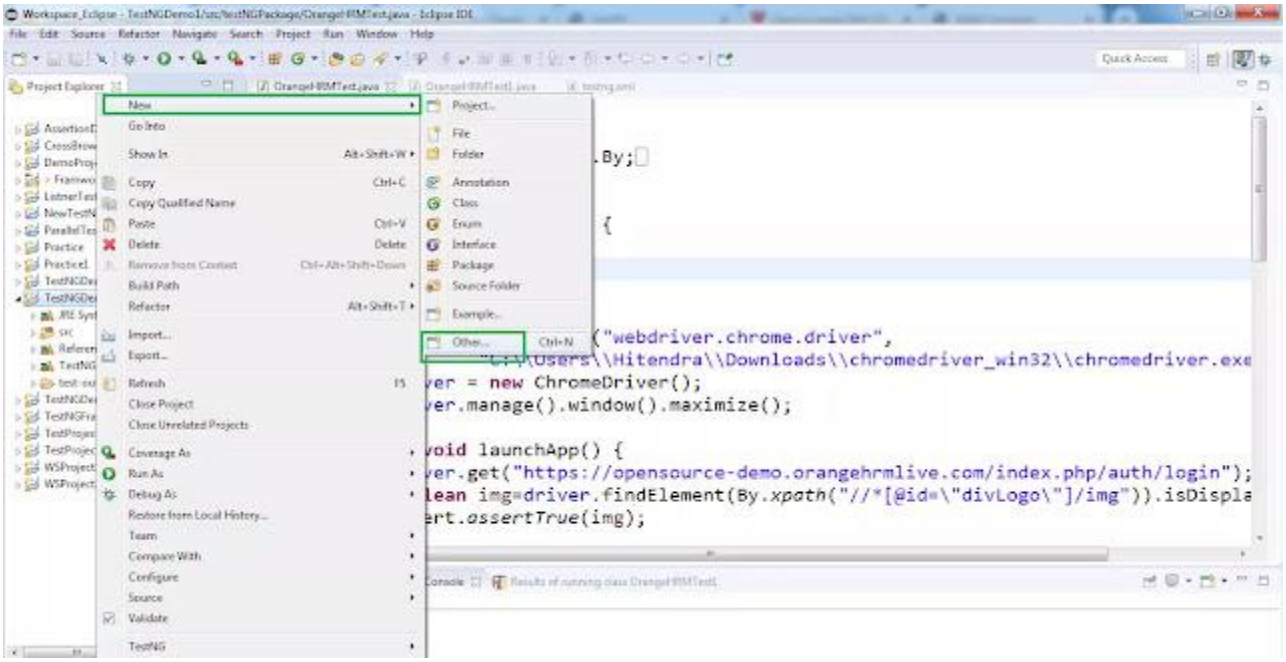
You can also specify groups and methods to be included and excluded:
Example3:

```
<test name="Regression1">
  <groups>
    <run>
      <exclude name="brokenTests"  />
      <include name="checkinTests"  />
    </run>
  </groups>
  <classes>
    <class name="test.IndividualMethodsTest">
      <methods>
        <include name="testMethod" />
      </methods>
    </class>
  </classes>
</test>
```

You can also specify methods to be included and excluded:
Example4:

```
<test name="Regression1" preserve-order="false">
  <classes>
    <class name="test.Test1">
      <methods>
        <include name="m1" />
        <exclude name="m2" />
      </methods>
    </class>
    <class name="test.Test2" />
  </classes>
</test>
```

How to create testng.xml file manually:
Right click on Project--New--Other--



In Select a wizard window select General--file and click on next

Create new file window -- provide file name with .xml extension and click on finish. XML file will be generated

How to run multiple testng.xml in one go
Run multiple testng suites in one file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <suite-files>
  <suite-file path="testng1.xml"></suite-file>
  <suite-file path="testng2.xml"></suite-file>
  </suite-files>
</suite> <!-- Suite -->
```

- **Parameterization using TestNG**

Parameterization[Data driven test] is an execution strategy, which allows us to run a test case automatically, multiple times with different input values.
To pass multiple data to the application at runtime, we need to parameterize our test scripts.

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 annotation and TestNG XML file

When to use: Select parameterization using annotations when you do want to deal with complexity & the number of input combinations are less.

Example: Like we can pass user name and password through testng.xml instead of hard coding it in testmethods. or we can pass browser name as parameter to execute in specific browser.

Parameters from Testng.xml can be suite or test level
The @Parameters annotation can be placed on any method that has a @Test, @Before/After or @Factory annotation.

Limitations:
Parameter value in testng.xml and values in @Parameters cannot be different (data type).
Your @Parameters do not have a corresponding name in testing.xml. Use @Optional
You cannot test multiple values of the same parameter using Testng.xml

Syntax:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
  <parameter name="username" value="admin"></parameter>
  <parameter name="password1" value="admin123"></parameter>
    <classes>
      <class name="OrangeHRMBaseTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Parameters annotation should be configured as below:

```
@Parameters({"username","password"})
```

Example:

```
public class OrangeHRMBaseTest {

 WebDriver driver;

  @BeforeMethod()

public void setup() {

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

  driver = new ChromeDriver();

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

  System.out.println("This is Before Test method");

  driver.get("https://opensource-demo.orangehrmlive.com/index.php/auth/login");

  boolean img = driver.findElement(By.xpath("//*[@id=\"divLogo\"]/img")).isDisplayed();

  Assert.assertTrue(img);

 }
  @Parameters({"username","password"})

 @Test()

 public void loginTest(String uname, @Optional("admin123")String pswd) throws
InterruptedException {

  driver.findElement(By.id("txtUsername")).sendKeys(uname);

  driver.findElement(By.name("txtPassword")).sendKeys(pswd);

driver.findElement(By.id("btnLogin")).click();
  boolean img1 =
driver.findElement(By.xpath("//*[@id=\"branding\"]/a[1]/img")).isDisplayed();
```

```
 Assert.assertTrue(img1);

 }
 @AfterMethod()

 public void tearDown() {

 System.out.println("This is After Test method");
 driver.close();
 }
}
```

testng.xml for above program

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
 <test name="Test">
 <parameter name="username" value="admin"></parameter>
 <parameter name="password1" value="admin123"></parameter>
  <classes>
   <class name="com.OrangeHRMPackage.OrangeHRMBaseTest"/>
  </classes>
 </test> <!-- Test -->
</suite> <!-- Suite -->
```

Passing Parameters with Data providers

When you need to pass complex parameters or parameters that need to be created from (objects read from a property file or a database, Excel sheet etc.)
Parameters can be passed using Dataproviders.

A Data Provider is a method annotated with @DataProvider.

- @DataProvider annotation has only one string attribute: its name. If the name is not supplied, the data provider's name automatically defaults to the method's name.
- A data provider returns an array of objects.

Syntax:

```
@Test(dataProvider="test1")
```

DataProvider method

```
@DataProvider(name = "test1")
      public static Object[][] loginCredentails() {
           return new Object[][] {{“username1", “password1"},{“uname2", “password2"},{123,
23424}};
        }
```

Example: Passing username and password as data providers in loginTest method

```
public class OrangeHRMBaseTest2 {

WebDriver driver;
 @BeforeMethod()
 public void setup() { System.setProperty("webdriver.chrome.driver"," chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  System.out.println("This is Before Test method");

  driver.get("https://opensource-demo.orangehrmlive.com/index.php/auth/login");
  boolean img = driver.findElement(By.xpath("//*[@id=\"divLogo\"]/img")).isDisplayed();
  Assert.assertTrue(img);
 }
 @Test(dataProvider = "loginCredentails")
 public void loginTest(String uname,String pswd) throws InterruptedException {
driver.findElement(By.id("txtUsername")).sendKeys(uname);
driver.findElement(By.name("txtPassword")).sendKeys(pswd);
driver.findElement(By.id("btnLogin")).click();
  boolean img1 =
driver.findElement(By.xpath("//*[@id=\"branding\"]/a[1]/img")).isDisplayed();
  Assert.assertTrue(img1);
 }
 @AfterMethod()
 public void tearDown() {
  System.out.println("This is After Test method");
  driver.close();
 }
 @DataProvider()
   public static Object[][] loginCredentails() {
        return new Object[][] {{"admin", "admin123"},{"admin",
"admin123"},{"admin","admin12345"}};
     }
}
```

Parallel Testing in TestNG

What is a Thread?

Java provides built-in support for multi threaded programming. A multi-threaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.

Parallel Testing

Parallelism or multi-threading in software terms is defined as the ability of the software, operating system, or program to execute multiple parts or sub-components of another program simultaneously.

Benefits of having parallel testing

- Reduces execution time – As tests are executed in parallel, multiple tests get executed simultaneously, hence reducing the overall time taken to execute the tests.
- Allows multi-threaded tests – Using this feature, we can write tests to verify certain multi-threaded code in the applications.

There are situations where you want to run multiple tests at the same time.
Situation 1 : If you have 100 test cases and Running these 100 tests sequentially will take you magnitudes of times longer because you can only execute one test at a time.
Situation 2: Cross browser testing - If there are more number of scripts to be executed and executing them on each and every browsers sequentially is time consuming.
The above situations can be avoided using a concept in Selenium called Parallel Execution.
We can reduce the 'execution time' as tests are executed simultaneously in different threads.

In testNG we can achieve parallel execution by two ways.

1. With testng.xml file using parallel attribute of suite tag.

2. We can configure an independent test method to run in multiple threads.

1st Method:

1. With testng.xml file using parallel attribute of suite tag.
Syntax: parellel =“methods”
The parallel attribute of suite tag can accept four values: methods ,classes, tests, instances
Sample XML <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel=“methods">

```
  <test name="Test">
    <classes>
      <class name="com.TestPackage.ClassB"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Example: I am giving an example here of parallel testing by methods, rest you can watch below youtube video. In below example 4 methods from two classes will be executed in two threads (thread-count="2").

```
public class ClassA {

 @Test

 public void testcase1() {

  System.out.println("This is testcase1: "+Thread.currentThread().getId());

 }
 @Test

 public void testcase2() {

  System.out.println("This is testcase2: "+Thread.currentThread().getId());
 }
}

public class ClassB {

 @Test
 public void testCase3() {

  System.out.println("This is testcase3: "+Thread.currentThread().getId());

 }
 @Test

 public void testCase4() {

  System.out.println("This is testcase4: "+Thread.currentThread().getId());
```

```
 }
}
```

TestNG.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="Suite" parallel="methods" thread-count="2">

  <test name="Test">

    <classes>

      <class name="com.parallelDemo.ClassA"/>

      <class name="com.parallelDemo.ClassB"/>

    </classes>

  </test> <!-- Test -->

</suite> <!-- Suite -->
```

Output:

```
This is testcase2: 14
This is testcase1: 13
This is testcase3: 14
This is testcase4: 13
```

2nd Method: 2. We can configure an independent test method to run in multiple threads.

We can configure test method like below and in classC test method testMethod1 will be executed in
two threads (threadPoolSize = 2) and will be executed 3 times(invocationCount = 3)

```
public class ClassC {

 @Test(threadPoolSize = 2, invocationCount = 3, timeOut = 1000)

 public void testMethod1() {
```

```
  System.out.println("This is testcase1: "+Thread.currentThread().getId());

 }
}
```

Output:
This is **testcase1: 11**
This is **testcase1: 12**
This is **testcase1: 11**

- **Cross Browser Testing in TestNG**

Cross Browser Testing in TestNG

What is Cross Browser Testing?
Cross Browser Testing is a type of functional test to check that your web application works as expected in different browsers..

Why do we need?
A web application can be opened in any browser by the end user. For example, some people prefer to open in Firefox, some in chrome, ie etc. We need to ensure that the web application will work as expected in all popular browsers so that more people can access it and use it.

This motive can be fulfilled with Cross Browser Testing of the product.

Some Cross Browser Issues

- Font size mismatch in different browsers.
- JavaScript implementation can be different.
- CSS,HTML validation difference can be there.
- Some browser still not supporting HTML5.
- Page alignment and div size.
- Image orientation.
- Browser incompatibility with OS.

How to Achieve Cross Browser in Selenium?
To execute test cases with different browsers in the same machine at the same time we can integrate TestNG framework with Selenium WebDriver.

Example: We have created a class CrossBrowserDemo and we have a test method launchApp in it. Using @Parameters annotation we are passing browser name(as string) to test method. And based on the browser parameter our test method launchApp will be executed on particular browser.

Our xml looks like below: Inside <suite> tag we have <parameter name="browser" value="Firefox"></parameter>

Running test on single browser

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="tests">
 <test name="FireFoxTest">
 <parameter name="browser" value="Firefox"></parameter>
  <classes>
   <class name="com.CrossBrowserTest.CrossBrowserDemo"></class>
  </classes>
 </test> <!-- Test -->
</suite> <!-- Suite -->
```

Running test on multiple browser

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="tests">
    <test name="FireFoxTest">
        <parameter name="browser" value="Firefox"></parameter>
          <classes>
   <class name="com.CrossBrowserTest.CrossBrowserDemo"></class>
         </classes>
</test> <!-- Test -->
 <test name="ChromeTest">
 <parameter name="browser" value="Chrome"></parameter>
  <classes>
   <class name="com.CrossBrowserTest.CrossBrowserDemo"></class>
  </classes>
 </test> <!-- Test -->
 <test name="IETest">
 <parameter name="browser" value="IE"></parameter>
  <classes>
   <class name="com.CrossBrowserTest.CrossBrowserDemo"></class>
```

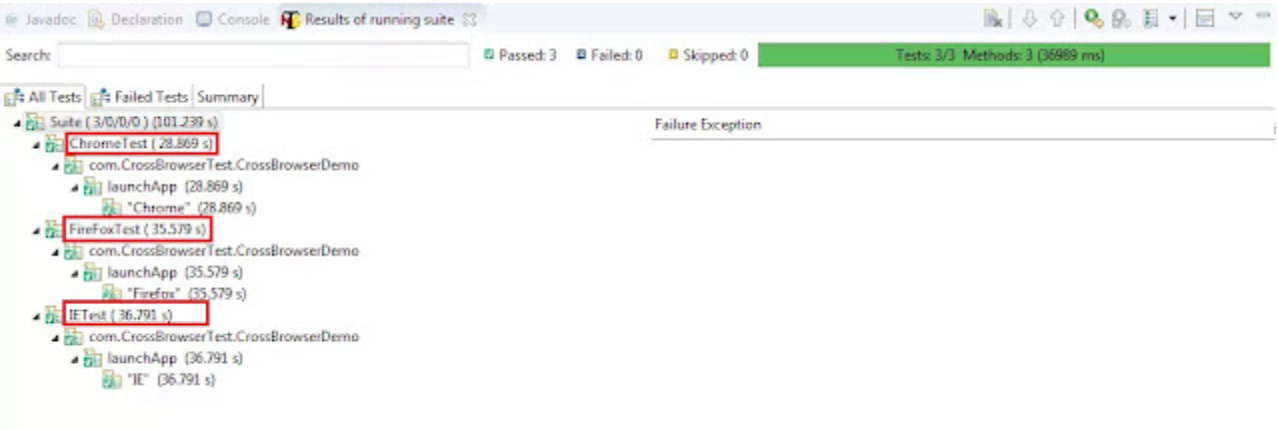
```
 </classes>
 </test> <!-- Test -->
</suite> <!-- Suite -->
```

Test Class

```
public class CrossBrowserDemo {

 WebDriver driver;

 @Parameters("browser")

 @Test

 public void launchApp(String browser) throws InterruptedException {
  if (browser.equalsIgnoreCase("Chrome")) {

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

   driver = new ChromeDriver();

  } else if (browser.equalsIgnoreCase("Firefox")) {

   System.setProperty("webdriver.gecko.driver",

     " geckodriver.exe path");

   driver = new FirefoxDriver();

  } else if (browser.equalsIgnoreCase("IE")) {

   System.setProperty("webdriver.ie.driver",

     " IEDriverServer.exe path");

   driver = new InternetExplorerDriver();

  }

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

```
 driver.get("https://www.automationtestinginsider.com/");

 Thread.sleep(2000);

 driver.close();

 }

}
```

Output:



- [Run failed test cases in Selenium]

Run failed test cases in selenium

At times, test cases may fail while running automated test scripts. The reason may be anything (say, Network issue, System issue or browser issue) but as an automation tester, you need to analyze the result and need to execute the test scripts again. Here is a solution to run failed test cases using TestNG in Selenium.

Why Test fails:
The reason for the failures can be anything

- Application Failure – not getting expected output
- Network issue – wifi, LAN issue etc.
- Server is not responding
- Scripting issue – Locator change due to new functionality
- Application is down
- Browser/browser driver issue

How to execute failed test cases?
We can execute failed test cases using two methods

- By running “testng-failed.xml”
- By Implementing TestNG IRetryAnalyzer interface

Lets have a look the first method:

1. By running “testng-failed.xml"

We have created two classes ClassA and ClassB. Each class has two test methods of which we are deliberately failing one of the tests.

```
public class ClassA {

 @Test

 public void testCase1() {

  System.out.println("this is testcase1");

  Assert.assertTrue(false);

 }
 @Test

 public void testCase2() {
  System.out.println("this is testcase2");

  Assert.assertTrue(true);
 }

}
public class ClassB {

 @Test

 public void testCase3() {

  System.out.println("this is testcase3");

  Assert.assertTrue(true);
```

```
 }
 @Test

 public void testCase4() {

  System.out.println("this is testcase4");

  Assert.assertTrue(false);
 }
}
```

testng.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Smoke Suite">

  <test name="Test">

    <classes>

      <class name="Com.runFailedTestCases1.ClassA"/>

      <class name="Com.runFailedTestCases1.ClassB"/>

    </classes>

  </test> <!-- Test -->

</suite> <!-- Suite -->
```

Output

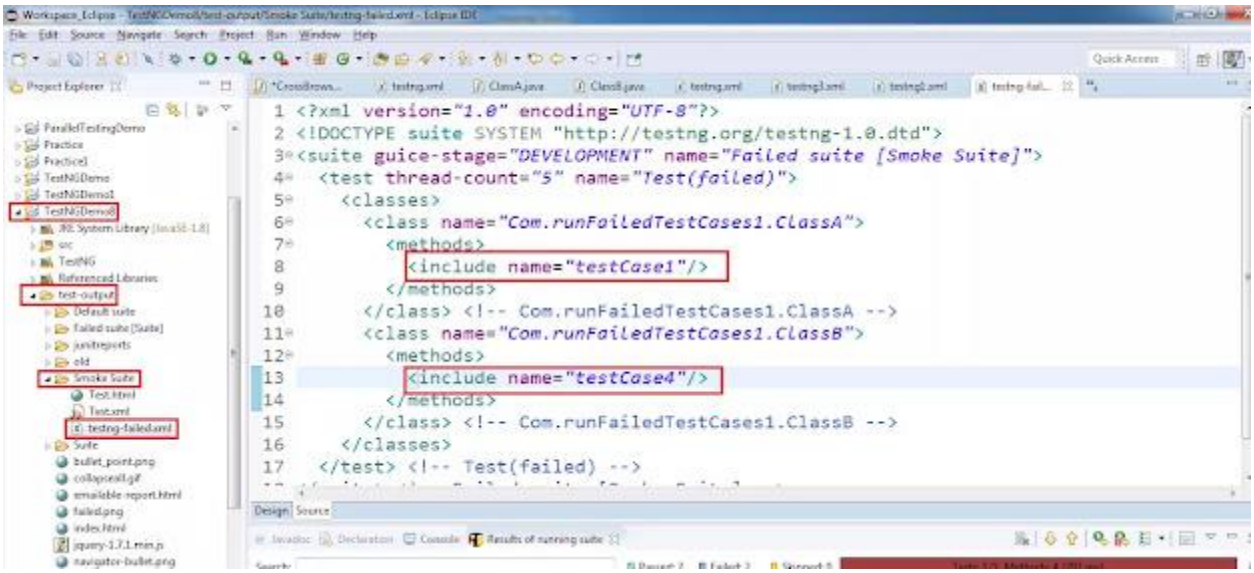
```
this is testcase1
this is testcase2
this is testcase3
this is testcase4
===============================================
Suite
```

Total tests **run: 4**, **Failures: 2**, **Skips: 0**

==========

If you want to execute only failed test cases through the Eclipse, then first refresh the project. Now navigate to test-output folder and then navigate to Suite folder (in our case suite name is Smoke Suite) Expand that, you will be able to see testng-failed.xml

Open the xml and you will see failed test cases in xml file.
Right-click on this file and click on run as and select the option called "testNG suite".



testng-failed.xml

Suppose if you have three test cases if all the test cases are executed successfully means you are not able to see this folder under the test-output folder. This folder will appear only when one of the test case is failed. Then run this file, it will going to run only failed test cases.

You can create Runner file to execute your failed test cases through script.Please refer below program.

```
public class FailTestRunner {

 @AfterTest

 public void runFailTestCases() {

  TestNG obj= new TestNG();

  List<String> list= new ArrayList<String>();
```

```
 list.add("D:\\Workspace_Eclipse\\TestNGDemo8\\test-output\\Suite\\testng-failed.xml");

 obj.setTestSuites(list);

 obj.run();
 }
}
```

2. By Implementing TestNG IRetryAnalyzer interface

Create a class to implement IRetryAnalyzer. Here I am creating a class (say, RetryAnalyzerTest) and implementing IRetryAnalyzer.

```
import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

public class RetryAnalyzerTest implements IRetryAnalyzer {

 int count=0;

 int maxCount=2;

 public boolean retry(ITestResult result) {

 if(count<maxCount) {

  System.out.println("Retrying " +result.getName()

  +" again and count is " +(count+1));

  count++;

  return true;
 }
 return false;
 }

}
```

Now create another class ‘RetryListnerClass’ by Implementing ‘IAnnotationTransaformer’ interface. transform method is called for every test during test run. A simple implementation of this ‘IAnnotationTransformer’ interface can help us set the ‘setRetryAnalyzer’ for ‘ITestAnnotation’. Add the above class name (RetryAnalyzerTest .class) in the below program. This interface does its work in run time by adding annotation to the test methods.

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import org.testng.IAnnotationTransformer;
import org.testng.annotations.ITestAnnotation;

public class RetryListnerClass implements IAnnotationTransformer {

 public void transform(ITestAnnotation annotation, Class tesClass,

   Constructor testConstructor, Method testMethod) {

  annotation.setRetryAnalyzer(RetryAnalyzerTest.class);
 }
}
```

Let us see the example by executing simple tests below

```
public class ClassA {

 @Test()

 public void testCase1() {
  System.out.println("this is testcase1");
  Assert.assertTrue(true);

 }
 @Test
 public void testCase2() {
  System.out.println("this is testcase2");
  Assert.assertTrue(false);

 }
}
```

We have two test cases in above program and one of the test case (testCase2) is getting failed. First lets include below mentioned Listener to testng.xml file. Below mentioned syntax is to add Listener for RetryListnereClass.

```
<listeners>
<listener class-name="method2.RetryListnerClass"></listener>
</listeners>
```

Final testng.xml file should looks like below:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite">

<listeners>

<listener class-name="method2.RetryListnerClass"></listener>

</listeners>

  <test name="Test">

    <classes>

      <class name="method2.ClassA"/>

    </classes>

  </test> <!-- Test -->

</suite> <!-- Suite -->
```

Execute the testng.xml. Here is the output which I got. You could see in the below mentioned result that the Test 2 is executed three times as we have mentioned ‘maxCount = 2’. Even though we have just 2 tests, we could find total test runs are 4 in the result.

Output:

```
this is testcase1
this is testcase2
```

Retrying testCase2 again and count is **1**
this is testcase2
Retrying testCase2 again and count is **2**
this is testcase2
===
Suite

Total tests **run: 4**, **Failures: 1**, **Skips: 2**

===

Listeners in TestNG

At times, test cases may fail while running automated test scripts. The reason may be anything (say, Network issue, System issue or browser issue) but as an automation tester, you need to analyze the result and need to execute the test scripts again. Here is a solution to run failed test cases using TestNG in Selenium.

Why Test fails:
The reason for the failures can be anything

- Application Failure – not getting expected output
- Network issue – wifi, LAN issue etc.
- Server is not responding
- Scripting issue – Locator change due to new functionality
- Application is down
- Browser/browser driver issue

How to execute failed test cases?
We can execute failed test cases using two methods

- By running “testng-failed.xml”
- By Implementing TestNG IRetryAnalyzer interface

Lets have a look the first method:

1. By running “testng-failed.xml"

We have created two classes ClassA and ClassB. Each class has two test methods of which we are deliberately failing one of the tests.

```
public class ClassA {

 @Test

 public void testCase1() {

  System.out.println("this is testcase1");

  Assert.assertTrue(false);

 }
 @Test

 public void testCase2() {
  System.out.println("this is testcase2");

  Assert.assertTrue(true);
 }

}
public class ClassB {

 @Test

 public void testCase3() {

  System.out.println("this is testcase3");

  Assert.assertTrue(true);

 }
 @Test

 public void testCase4() {

  System.out.println("this is testcase4");

  Assert.assertTrue(false);
 }
}
```

testng.xml

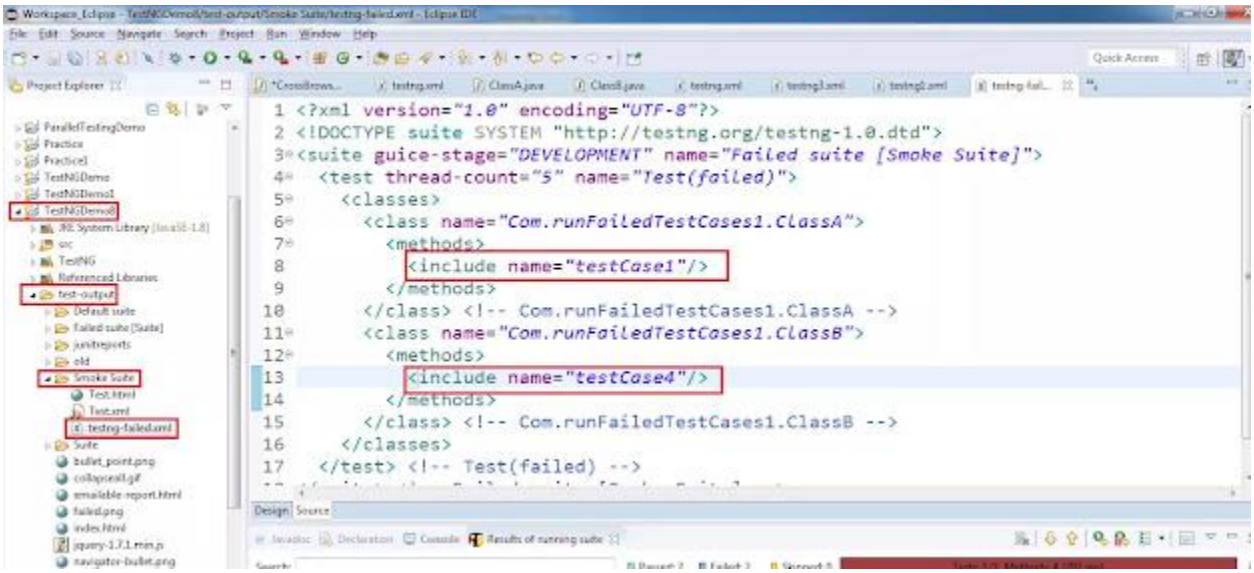
```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Smoke Suite">

  <test name="Test">

    <classes>

      <class name="Com.runFailedTestCases1.ClassA"/>

      <class name="Com.runFailedTestCases1.ClassB"/>

    </classes>

  </test> <!-- Test -->

</suite> <!-- Suite -->
```

Output

```
this is testcase1
this is testcase2
this is testcase3
this is testcase4
===============================================
Suite
Total tests run: 4, Failures: 2, Skips: 0

===============================================
```

If you want to execute only failed test cases through the Eclipse, then first refresh the project. Now navigate to test-output folder and then navigate to Suite folder (in our case suite name is Smoke Suite) Expand that, you will be able to see testng-failed.xml

Open the xml and you will see failed test cases in xml file.
Right-click on this file and click on run as and select the option called "testNG suite".



testng-failed.xml

Suppose if you have three test cases if all the test cases are executed successfully means you are not able to see this folder under the test-output folder. This folder will appear only when one of the test case is failed. Then run this file, it will going to run only failed test cases.

You can create Runner file to execute your failed test cases through script.Please refer below program.

```
public class FailTestRunner {

 @AfterTest

 public void runFailTestCases() {

  TestNG obj= new TestNG();

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

  list.add("D:\\Workspace_Eclipse\\TestNGDemo8\\test-output\\Suite\\testng-failed.xml");

  obj.setTestSuites(list);

  obj.run();
 }
}
```

2. By Implementing TestNG IRetryAnalyzer interface

Create a class to implement IRetryAnalyzer. Here I am creating a class (say, RetryAnalyzerTest) and implementing IRetryAnalyzer.

```
import org.testng.IRetryAnalyzer;
import org.testng.ITestResult;

public class RetryAnalyzerTest implements IRetryAnalyzer {

 int count=0;

 int maxCount=2;

 public boolean retry(ITestResult result) {

  if(count<maxCount) {

   System.out.println("Retrying " +result.getName()

   +" again and count is " +(count+1));

   count++;

   return true;
  }
  return false;
 }

}
```

Now create another class ‘RetryListnerClass’ by Implementing ‘IAnnotationTransaformer’ interface. transform method is called for every test during test run. A simple implementation of this ‘IAnnotationTransformer’ interface can help us set the ‘setRetryAnalyzer’ for ‘ITestAnnotation’. Add the above class name (RetryAnalyzerTest .class) in the below program. This interface does its work in run time by adding annotation to the test methods.

```
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import org.testng.IAnnotationTransformer;
```

```
import org.testng.annotations.ITestAnnotation;

public class RetryListnerClass implements IAnnotationTransformer {

 public void transform(ITestAnnotation annotation, Class tesClass,

   Constructor testConstructor, Method testMethod) {

  annotation.setRetryAnalyzer(RetryAnalyzerTest.class);
 }
}
```

Let us see the example by executing simple tests below

```
public class ClassA {

 @Test()

 public void testCase1() {
  System.out.println("this is testcase1");
  Assert.assertTrue(true);

 }
 @Test
 public void testCase2() {
  System.out.println("this is testcase2");
  Assert.assertTrue(false);

 }
}
```

We have two test cases in above program and one of the test case (testCase2) is getting failed. First lets include below mentioned Listener to testng.xml file. Below mentioned syntax is to add Listener for RetryListnereClass.

```
<listeners>
<listener class-name="method2.RetryListnerClass"></listener>
</listeners>
```

Final testng.xml file should looks like below:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="Suite">

<listeners>

<listener class-name="method2.RetryListnerClass"></listener>

</listeners>

  <test name="Test">

    <classes>

      <class name="method2.ClassA"/>

    </classes>

  </test> <!-- Test -->

</suite> <!-- Suite -->
```

Execute the testng.xml. Here is the output which I got. You could see in the below mentioned result that the Test 2 is executed three times as we have mentioned ‘maxCount = 2’. Even though we have just 2 tests, we could find total test runs are 4 in the result.

Output:

```
this is testcase1
this is testcase2
Retrying testCase2 again and count is 1
this is testcase2
Retrying testCase2 again and count is 2
this is testcase2
===============================================
Suite

Total tests run: 4, Failures: 1, Skips: 2
```

Assertions in TestNG

What is Assertion?

- Assertion determines the state of the application whether it is the same what we are expecting or not.
- While using Selenium for automated testing of web applications, we need to add validations in our tests to report them as pass or fail.
- Assertions give you a way (other than If-Else blocks) to test conditions.

There are two types of Assertions in TestNG: Hard Assertion(Assert) and Soft Assertion(Verify

Hard Assertions – It is the default assert mechanism built into TestNG's "org.testng.Assert" package. We use it when a test has to stop immediately after the assertion fails.
Hard Assertion is an Assertion that throws the AssertException when the test case is failed. In the case of Hard Assertion, you can handle the error by using a catch block like a java exception. A Hard Assertion contains the following methods:

- assertEquals
- assertNotEquals
- assertTrue
- assertFalse
- assertNull
- assertNotNull

Example: In below example we will understand Hard Assertion, in below test class we have two test methods and testCase1 test method is getting failed at 3rd line. Once exception occurs remaining code will not be executed in testCase1. However testCase2 will be executed successfully. Look at the output

```
import org.testng.Assert;
import org.testng.annotations.Test;

public class HardAssertionDemo {

 @Test
 public void testCase1() {
  System.out.println("================================");
  System.out.println("Test1 Start");
  Assert.assertTrue(false);
  System.out.println("Test1 End");
  System.out.println("================================");
```

```
 }

 @Test
 public void testCase2() {
  System.out.println("==================================");
  System.out.println("Test2 Start");
  Assert.assertTrue(true);
  System.out.println("Test2 End");
  System.out.println("==================================");
 }

}
```

Output:

```
==================================

Test1 Start

==================================

Test2 Start

Test2 End

==================================

PASSED: testCase2

FAILED: testCase1

java.lang.AssertionError: expected [true] but found [false]

 at org.testng.Assert.fail(Assert.java:96)
```

The disadvantage of Hard Assert – It marks method as fail if assert condition gets failed and the remaining statements inside the method will be aborted.

And sometimes we want to execute the whole script even if the assertion fails. This is not possible in Hard Assertion. To overcome this problem, we need to use a soft assertion in testng.

Soft Assertions (Verify):

- It is a custom assert mechanism supported by TestNG's "org.testng.asserts.Softassert" package. We use it when a test has to continue execution even after an assertion fails in the sequence.
- Soft Assert does not throw an exception when an assert fails and would continue with the next step after the assert statement.

If there is any exception and you want to throw it then you need to use assertAll() method as a last statement in the @Test and test suite again continue with next @Test as it is.

Example:

```
import org.testng.Assert;
import org.testng.annotations.Test;
import org.testng.asserts.SoftAssert;

public class SoftAssertionDemo {

 SoftAssert obj= new SoftAssert();

 SoftAssert obj1= new SoftAssert();


 @Test
 public void testCase1() {
  System.out.println("================================");
  System.out.println("Test1 Start");
  obj.assertTrue(false);
  obj.assertEquals("Hi", "Hi");
  System.out.println("Test1 End");
  System.out.println("================================");
  obj.assertAll();

 }
 @Test
 public void testCase2() {
  System.out.println("================================");
  System.out.println("Test2 Start");
  obj1.assertTrue(true);
  obj1.assertEquals("HELLO", "HELLO");
  System.out.println("Test2 End");
```

```
  System.out.println("==================================");
  obj1.assertAll();

 }

}
```

Output:

==========

Test1 Start

Test1 End

==========

==========

Test2 Start

Test2 End

==========

PASSED: testCase2

FAILED: testCase1

java.lang.AssertionError: The following asserts **failed:**

 expected [**true**] but found [**false**]

Maven Tutorials:

- ✓ **Maven Introduction and Setup**

What is a build tool?

The build tool is used to set up everything which is required to run your java code

independently. It generates source code, compiling code, packaging code to a jar etc.

Maven is a build / project management tool, based on the concept of a project object model (POM) contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

Maven provides a common platform to perform these activities which makes programmer's life easier while handling the huge project.

Common problems:
1. Problem of multiple jars – let's say we are using a couple of frameworks in our code, for instance selenium server, testNG , JXL or POI library etc. In order to use these frameworks within my application I need to include all the required jars in my application, I need to make them available.

Sometimes we miss out on some jars, sometimes we don't know what jars are needed. So MAVEN helps us on this aspect.

2. Dependencies and Versions – Let's say we have a particular jar and that jar has a dependency on other jar. Then I need to make sure that all my dependencies are closed, I need to make sure I have supplied all the required dependencies.

Maven provides pom.xml which is the core of any project. This is the configuration file where all required information's are kept.

Maven downloads the dependency jar from a central repository. Most of the commonly used libraries are available in https://mvnrepository.com/repos/central.
The user has to configure the remote repository in pom.xml to download from the remote repository. Below is the example of configuring a remote repository to pom.xml file. Provide id and URL of the repository where libraries are stored.

```
<repositories>
   <repository>
      <id>libraryId</id>
      <url>http://comanyrepositryId</url>
   </repository>
</repositories>
```

Uses in Selenium:
We can create Maven project for writing script and create dependency-using POM.xml once dependency is set Maven will download all the dependent jar files automatically and in future if any update comes from Selenium or TestNG side it will simply update all the required changes.

General Phrases used in Maven:
groupId: Generally groupId refers to domain id. For best practices company name is used as groupId. It identifies the project uniquely.
artifactId: It is basically the name of the Jar without version.

version: This tag is used to create a version of the project.
Local repository: Maven downloads all the required dependencies and stores in the local repository called m2.

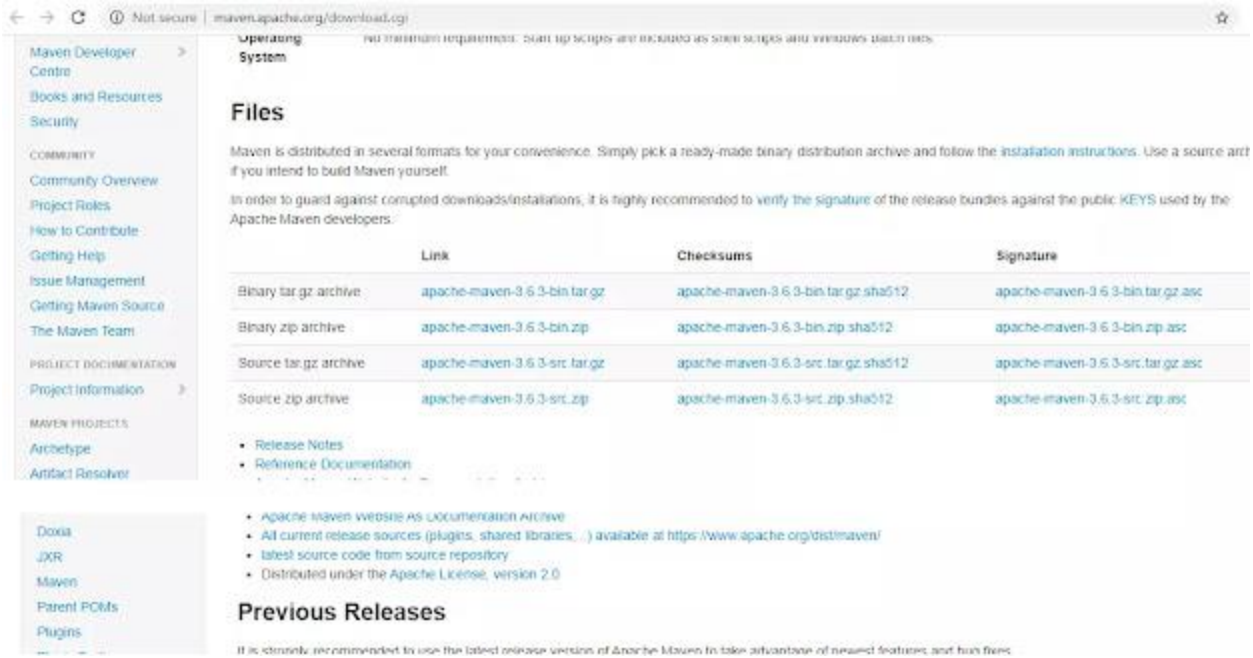
Maven Build Life Cycle:
Basic maven phases are used as below.

- **clean**: deletes all artifacts and targets which are created already.
- **compile**: used to compile the source code of the project.
- **test**: test the compiled code and these tests do not require to be packaged or deployed.
- **package**: package is used to convert your project into a jar or war etc.
- **install**: install the package into the local repository for use of another project.

How to Install and Setup Maven

1. To setup Maven, download the maven's latest version form Apache depending upon different OS.
Link: http://maven.apache.org/download.cgi

2. Download required as per OS.

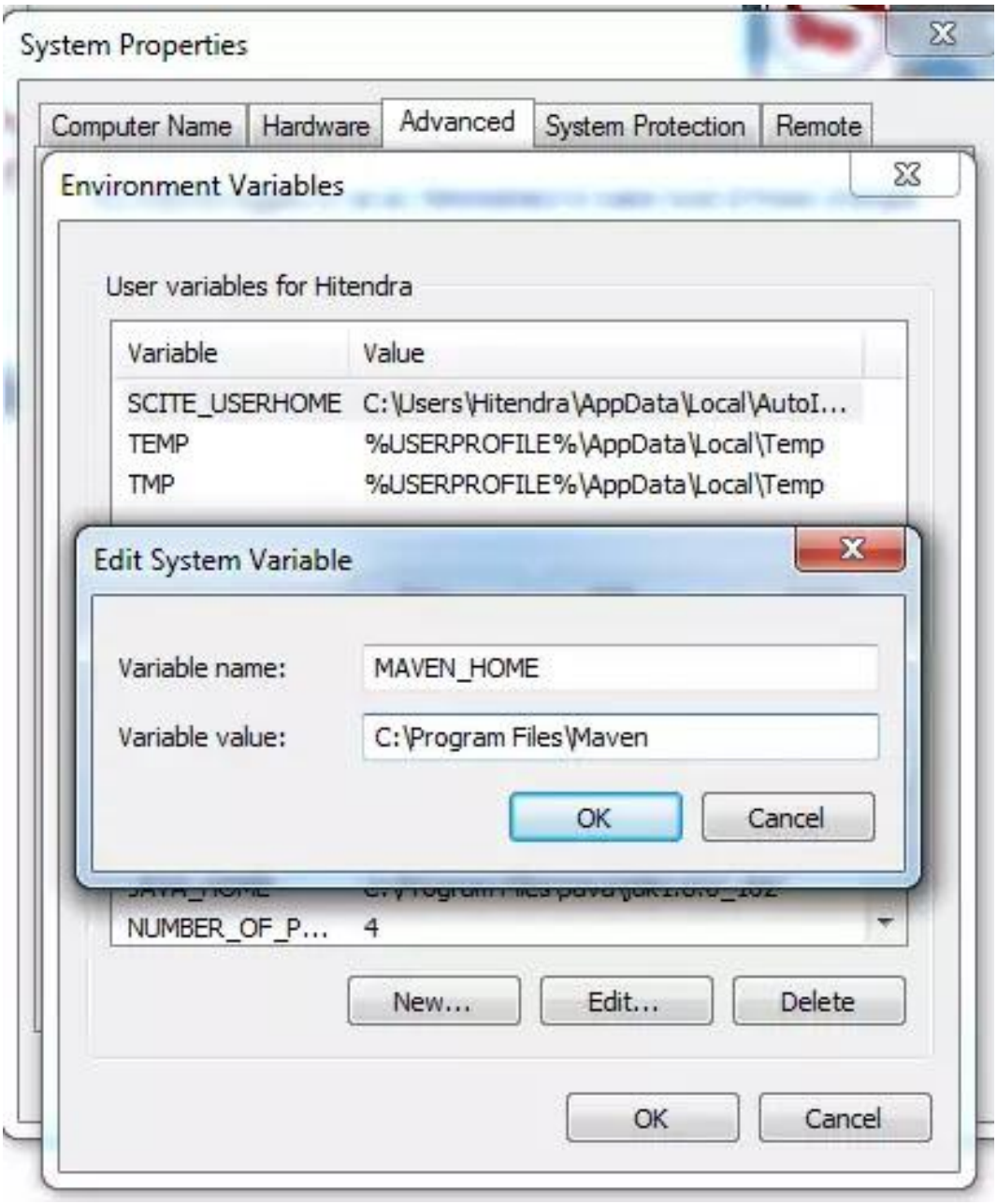


Apache Maven

3. Extract the File, and place it in C: drive.
Rename the file from Apache Maven (version) to simple Maven for just usability.
C:\Program Files\Maven.

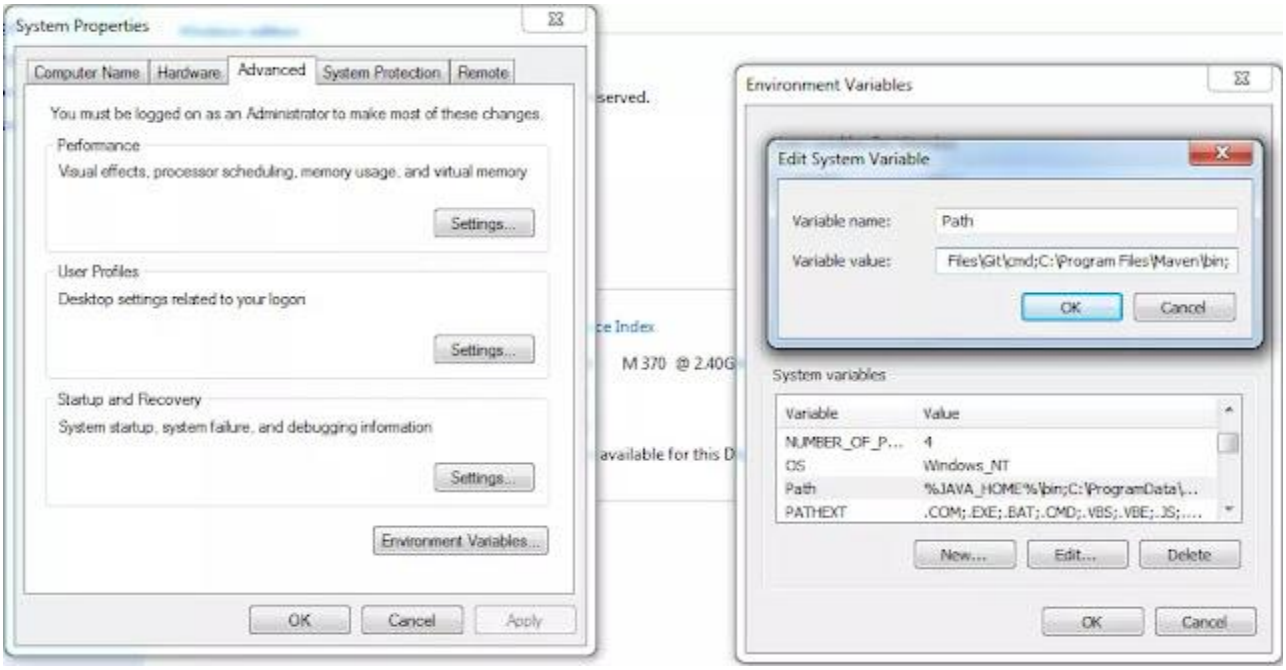
4. Once we have downloaded, all that we are now left to do before start using MAVEN is set a couple of environment variables. To do that, right click Computer >> Properties >> Advanced >> Environment Variables.

5. First you need to set the new variable as "MAVEN_HOME" and provide the path where the Maven is stored in C drive (as above mentioned C:\Program Files\Maven).

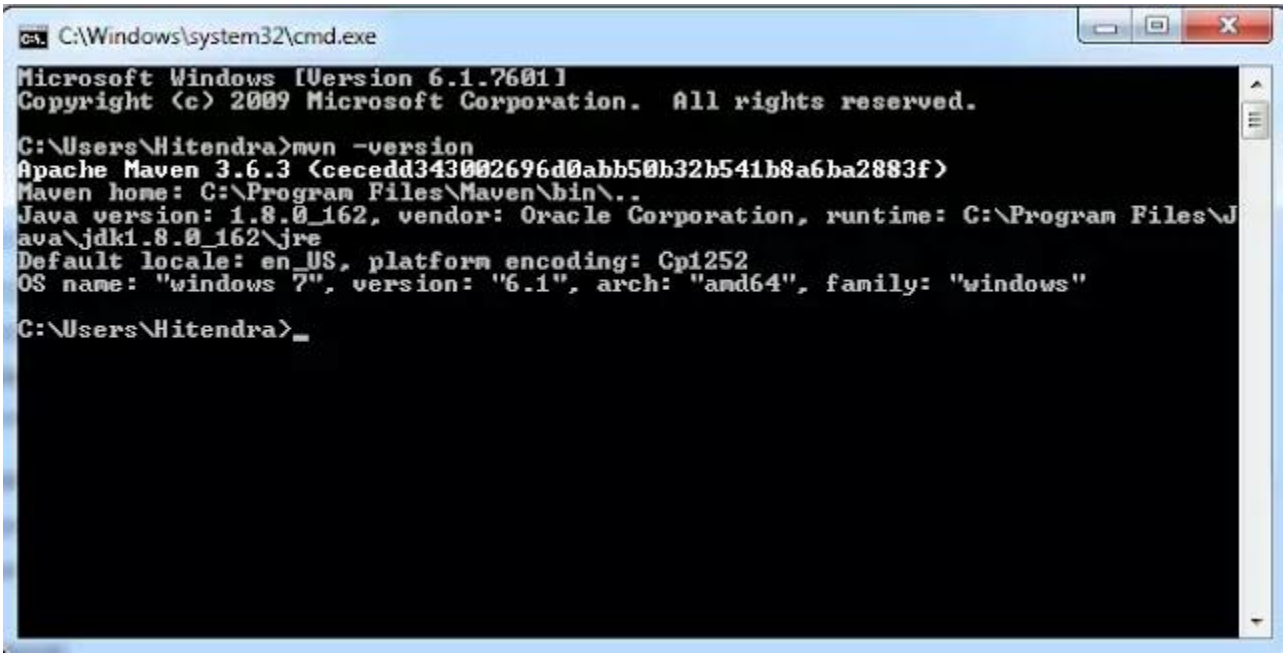


Next we need to add the bin folder to the Path. Select the Path in System variables and Click on Edit.

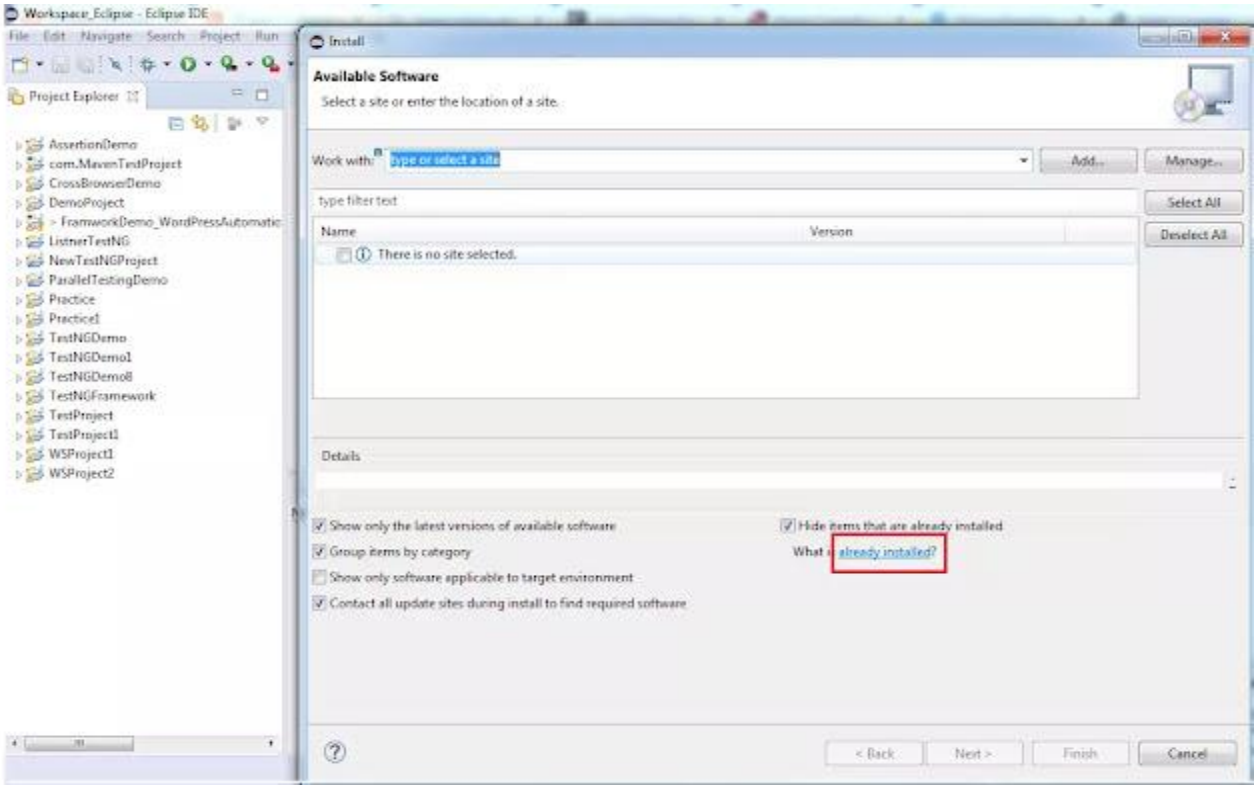
Add the path of the Maven Build Folder as %MAVEN_HOME%/bin or simply as (C:\Program Files\Maven\bin).

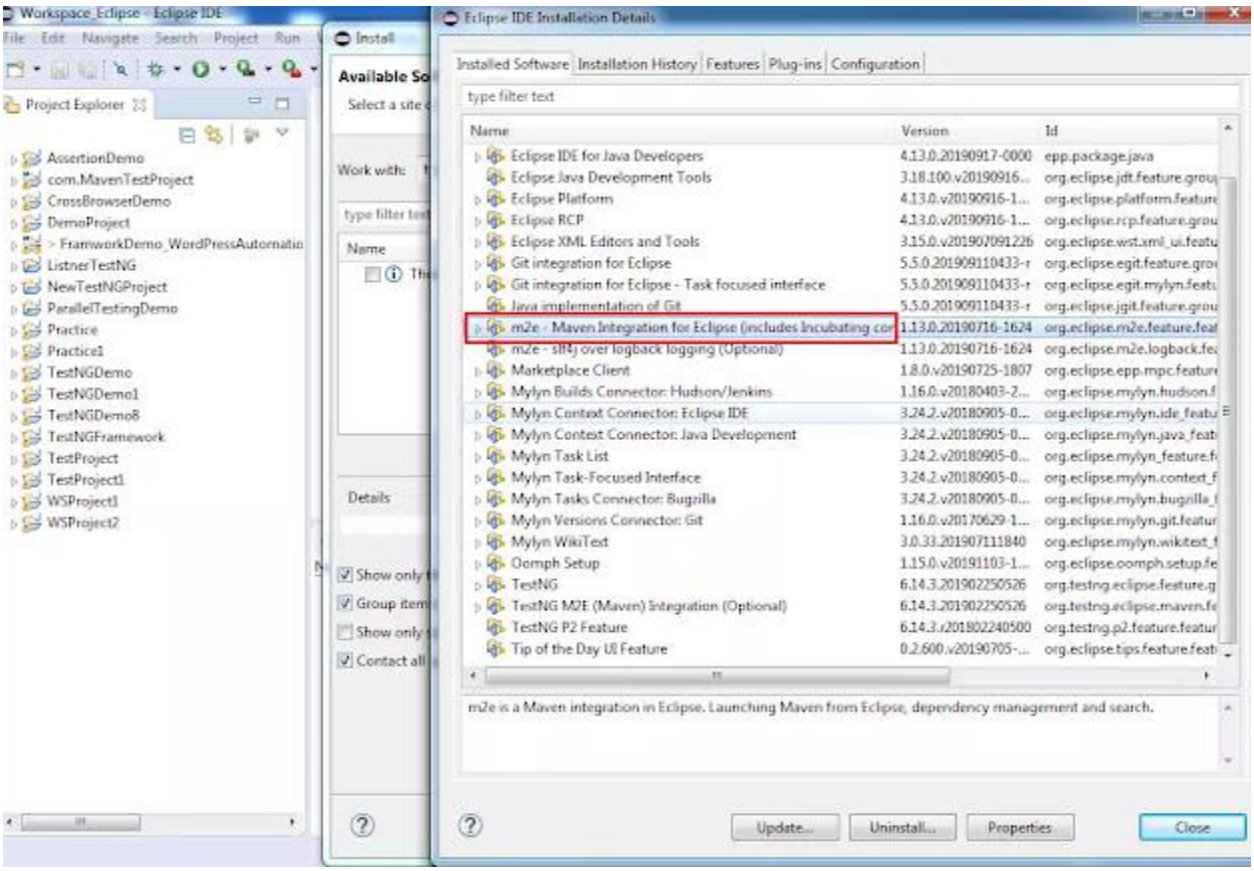


6. There we go, we are set and ready to hit MAVEN commands. So, firstly lets test to see if MAVEN is working fine. A simple MAVEN command is mvn --version.



7. How to verify Maven eclipse plugin is installed:
In Eclipse Navigate to Help--> Install new software
And as per below image click on already installed link and verify the eclipse maven plugin is present as per second image.


Workspace_Eclipse - Eclipse IDE
File Edit Navigate Search Project Run
Project Explorer
AssertionDemo
com.MavenTestProject
CrossBrowserDemo
DemoProject
FramworkDemo_WordPressAutomatic
ListnerTestNG
NewTestNGProject
ParallelTestingDemo
Practice
Practice1
TestNGDemo
TestNGDemo1
TestNGDemo8
TestNGFramework
TestProject
TestProject1
WSProject1
WSProject2
Install
Available Software
Select a site or enter the location of a site.
Work with:
type or select a site
Add...
Manage...
type filter text
Select All
Deselect All
Name
Version
There is no site selected.
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Show only software applicable to target environment
Contact all update sites during install to find required software
< Back
Next >
Finish
Cancel

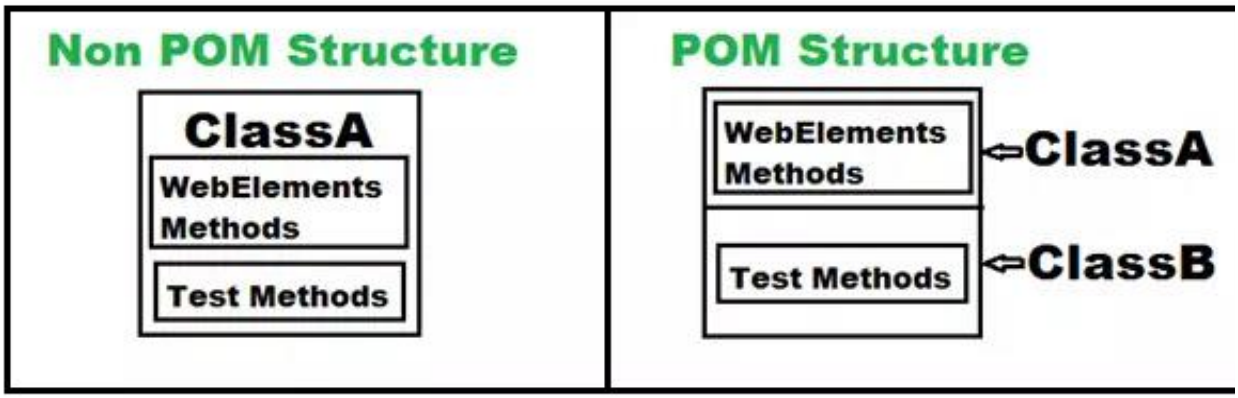

Workspace_Eclipse - Eclipse IDE
Project Explorer
Install
Eclipse IDE Installation Details
Installed Software
Installation History
Features
Plug-ins
Configuration
type filter text
Name
Version
Id
Eclipse IDE for Java Developers
Eclipse Java Development Tools
Eclipse Platform
Eclipse RCP
Eclipse XML Editors and Tools
Git integration for Eclipse
Git integration for Eclipse - Task focused interface
Java implementation of Git
m2e - Maven Integration for Eclipse (includes Incubating
m2e - slf4j over logback logging (Optional)
Marketplace Client
Mylyn Builds Connector: Hudson/Jenkins
Mylyn Context Connector: Eclipse IDE
Mylyn Context Connector: Java Development
Mylyn Task List
Mylyn Task-Focused Interface
Mylyn Tasks Connector: Bugzilla
Mylyn Versions Connector: Git
Mylyn WikiText
Oomph Setup
TestNG
TestNG M2E (Maven) Integration (Optional)
TestNG P2 Feature
Tip of the Day UI Feature
m2e is a Maven integration in Eclipse. Launching Maven from Eclipse, dependency management and search.
Update...
Uninstall...
Properties
Close

Eclipse Maven Plugin

POM, DataDriven FrameWork:

- **Introduction to Page Object Model(POM)**

What is Page Object Model?
Page Object Model is a design pattern to create Object Repository for web UI elements. Under this model, for each web page in the application, there should be corresponding page class. This Page class will find the Web Elements of that web page and also contains Page methods which perform operations on those Web Elements.



Non POM structure Vs POM Structure

POM Advantages and Disadvantages:
Advantages:

- **Object Repository:** You can create an Object Repository of the fields segmented page-wise. This as a result provides a Page Repository of the application as well. Each page will be defined as a java class. All the fields in the page will be defined in an interface as members. The class will then implement the interface.
- **Reusability:** We can reuse the page class if required in different test cases .
- **Functional Encapsulation:** All possible functionality or operations that can be performed on a page can be defined and contained within the same class created for each page. This allows for clear definition and scope of each page's functionality.
- **Low maintenance:** Any User Interface changes can swiftly be implemented into the interface as well as class.
- **Programmer Friendly:** Robust and more readable. The Object-oriented approach makes the framework programmer friendly.
- **Low Redundancy:** Helps reduce duplication of code. If the architecture is correctly and sufficiently defined, the POM gets more done in less code.

- **Efficient & Scalable:** Faster than other keyword-driven/data-driven approaches where Excel sheets are to be read/written.

Disadvantages:

- **High Setup Time & Effort:** Initial effort investment in development of Automation Framework is high. This is the biggest weight of POM in case of web applications with hundreds/thousands of pages. It is highly suggested that if this model is decided to be implemented, then it should be done parallel to development of the application.
- **Skilled labour:** Testers not technically sound or aware of programming best practices are a nightmare in this case.

There are two ways to implement Page Object Model.
Basic Approach
Page Factory approach (@FindBy Annotation)

Steps to be followed to create page classes

- Create a Java class for every page in the application.
- In each class, declare all the web Elements as variable.
- Implement corresponding methods acting on the variables.

Steps to be followed to create test cases

- Initialize the driver and open the application.
- Create an object of page layer class and pass the driver instance.
- Using the object call the method from page layer class
- Repeat step#3 until all actions are performed and at the end close the browser.

Below implementation given using basic approach:

LoginPage Class:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class LoginPage {
 WebDriver driver;
   //UI Elements
 By username = By.id("txtUsername");
 By password = By.name("txtPassword");
 By loginBtn = By.xpath("//*[@id=\"btnLogin\"]");
```

```
 By logo = By.xpath("//*[@id=\"divLogo\"]/img");

 //Constructor to initialize current class objects
 public LoginPage(WebDriver driver) {
  this.driver=driver;
 }
 //User Actions methods
 public boolean validateLogo() {
  driver.findElement(logo).isDisplayed();
  return true;
 }
 public HomePage login(String uname, String pswd) {
  driver.findElement(username).sendKeys(uname);
  driver.findElement(password).sendKeys(pswd);
  driver.findElement(loginBtn).click();
  return new HomePage();
 }
}
```

LoginPageTest Class:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
import com.orangeHRM.pages.HomePage;
import com.orangeHRM.pages.LoginPage;

/**
 * @author Kebede B.
 *
 */
public class LoginPageTest {
 public WebDriver driver;
 LoginPage loginPage;
 HomePage homepage;

 @BeforeMethod
 public void setUp() {
```

```
 System.setProperty("webdriver.chrome.driver", " chromedriver.exe path");

 driver= new ChromeDriver();
 loginPage= new LoginPage(driver);
 driver.manage().window().maximize();
 driver.get("https://opensource-demo.orangehrmlive.com/index.php/auth/validateCredentials");
}
@Test
public void varifyLogo() {

 boolean flag=loginPage.validateLogo();
 Assert.assertTrue(flag);
}
@Test
public void varifyLogin() {
 homepage=loginPage.login("admin", "admin123");
 String actualURL= driver.getCurrentUrl();
 String expectedURL= "https://opensource-demo.orangehrmlive.com/index.php/dashboard";
 Assert.assertEquals(actualURL, expectedURL);
}
@AfterMethod
public void tearDown() {
 driver.close();
 }
}
```

- **Page Object Model(POM) Using PageFactory**

In last post we have seen Page Object Model with basic approach, in this post we will see Page Object Model using Page Factory in Selenium.

What is PageFactory?
PageFactory Class in Selenium is an extension to the Page Object design pattern. It is used to initialize the web elements of the Page. It is used to initialize elements of a Page class without having to use 'FindElement' or 'FindElements'.

Here we follow the below steps to implement Page Object Model using Page Factory:

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>POMPageFactory</groupId>
  <artifactId>POMPageFactory</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <dependencies>
  <dependency>
   <groupId>org.testng</groupId>
   <artifactId>testng</artifactId>
   <version>7.1.0</version>
   <scope>test</scope>
  </dependency>
  <dependency>
   <groupId>org.seleniumhq.selenium</groupId>
   <artifactId>selenium-java</artifactId>
   <version>3.141.59</version>
  </dependency>
 </dependencies>
</project>
```

Steps to be followed to create Base Class

- Declare web driver as static and Initialize the web driver and open the application using setup method.
- Create an object of page layer class(Only need to create LoginPage object inside setup method).
- Close the browser using tearDown method.
- You can have some other common methods as well which will be used across framework like implicit waits, taking screen shots etc.

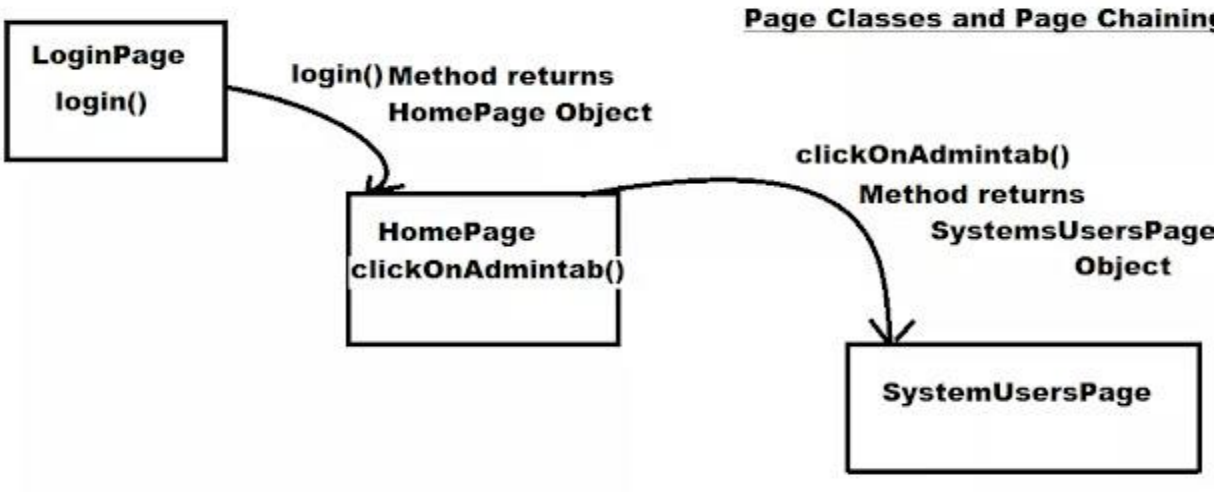
BaseClass.java

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import com.OrangeHRM.pages.LoginPage;
```

```
/**
 * @author Kebede B.
 *
 */
public class BaseClass {

 public static WebDriver driver;
 public LoginPage loginPage;

 @BeforeMethod
 public void setup() {
  System.setProperty("webdriver.chrome.driver",
    "C:\\Users\\Hitendra\\Downloads\\chromedriver_win32 (1)\\chromedriver.exe");
  driver= new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://opensource-demo.orangehrmlive.com/index.php/auth/validateCredentials");
  loginPage=new LoginPage();
 }
 @AfterMethod
 public void tearDown() {
  driver.close();
 }
}
```

Steps to be followed to create page classes

- Create a Java class for every page in the application (This class will extend BaseClass).
- In each class, declare all the web Elements as variable using @FindBy.
- Implement corresponding methods acting on the variables.
- Construct page chaining logic - lets say in LoginPage.java we have login method and when we perform actions like entering username, password and clicking on login button then user lands HomePage. So login method will return object of HomePage.java class.
- Add constructor to initialize the web elements initElements method.



Page Chaining Model

LoginPage.java

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

import com.OrangeHRM.base.BaseClass;

public class LoginPage extends BaseClass {

 @FindBy(id="txtUsername")
 WebElement userName;

 @FindBy(name="txtPassword")
 WebElement password;

 @FindBy(xpath="//*[@id=\"btnLogin\"]")
 WebElement loginBtn;

 @FindBy(xpath="//*[@id=\"divLogo\"]/img")
 WebElement logo;

 public LoginPage() {
  PageFactory.initElements(driver, this);
 }

 public boolean validateLogo() {
  logo.isDisplayed();
  return true;
```

```
 }
 //This method will return object of HomePage class as we are landing on
 //HomePage using this method
 public HomePage login(String uname, String pswd) {
  userName.sendKeys(uname);
  password.sendKeys(pswd);
  loginBtn.click();
  return new HomePage();
 }
}
```

HomePage.java

```
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
import com.OrangeHRM.base.BaseClass;

public class HomePage extends BaseClass {

 @FindBy(xpath="//*[@id=\"menu_admin_viewAdminModule\"]/b")
 WebElement adminTab;

 public HomePage() {
  PageFactory.initElements(driver, this);
 }
 //This method will return object of SystemUsersPage class as we are landing on
 //SystemUsersPage using this method
 public SystemUsersPage clickOnAdminTab() {
  adminTab.click();
  return new SystemUsersPage();
 }
}
```

SystemUsersPage.java - Just created this page class without writing anything inside it.

```
public class SystemUsersPage {

}
```

Steps to be followed to create test cases

- Create test class for every corresponding page class. Like LoginPageTest, HomePageTest etc. (This class will extend BaseClass).
- Using the object reference, call the method from page layer class.
- Insert the assert in the test method to validate the scenarios.
- Repeat step#3 until all actions are performed.

LoginPageTest.java

```
import org.testng.Assert;
import org.testng.annotations.Test;
import com.OrangeHRM.base.BaseClass;
import com.OrangeHRM.pages.HomePage;

public class LoginPageTest extends BaseClass {
 HomePage homePage;

 @Test(priority = 1)
 public void logoTest() {
  boolean flag=loginPage.validateLogo();
  Assert.assertTrue(flag);
 }
 @Test(priority = 2)
 public void loginTest() {
  homePage=loginPage.login("admin", "admin123");
  String expectedURL="https://opensource-demo.orangehrmlive.com/index.php/dashboard";
  String actualURL=BaseClass.driver.getCurrentUrl();
  Assert.assertEquals(actualURL, expectedURL);
 }
}
```

HomePageTest.java

```
import org.testng.Assert;
import org.testng.annotations.Test;
import com.OrangeHRM.base.BaseClass;
import com.OrangeHRM.pages.HomePage;

public class HomePageTest extends BaseClass {
 HomePage homePage;

 @Test(priority = 3)
```

```
 public void clickOnAdminTab() throws InterruptedException  {
  homePage=loginPage.login("admin", "admin123");
  homePage.clickOnAdminTab();
  Thread.sleep(2000);
  String expectedURL="https://opensource-
demo.orangehrmlive.com/index.php/admin/viewSystemUsers";
  String actualURL=BaseClass.driver.getCurrentUrl();
  Assert.assertEquals(actualURL, expectedURL);
 }
}
```

testng.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="com.OrangeHRM.testcases.LoginPageTest"/>
      <class name="com.OrangeHRM.testcases.HomePageTest"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

- **Read excel file in Selenium using Apache POI**

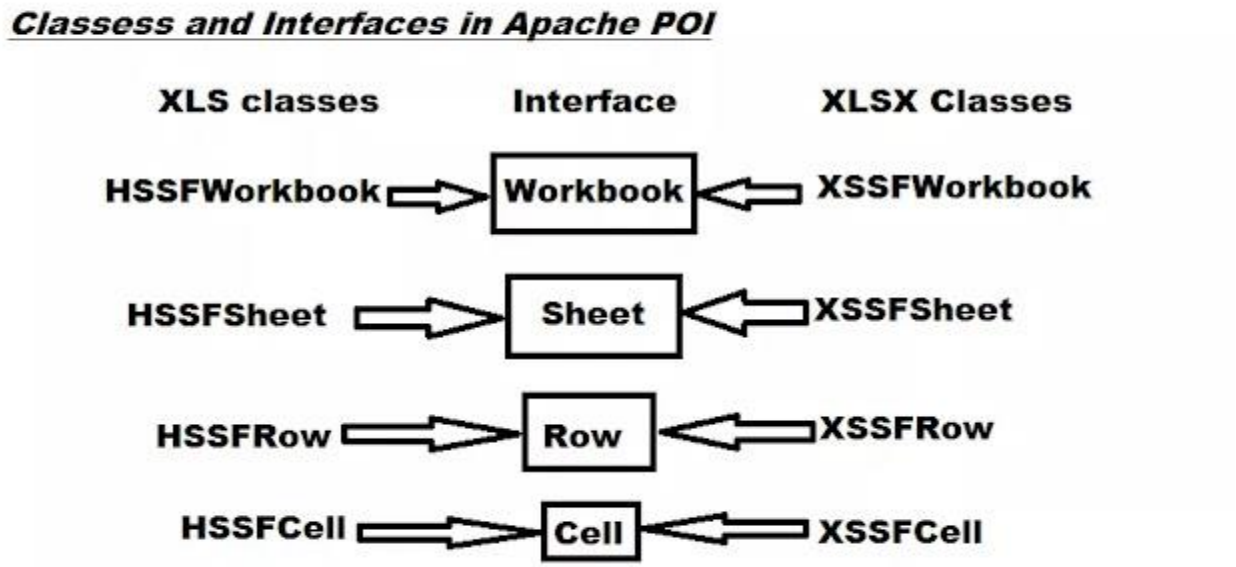
How to Read/Write Excel File?

- To Read and Write excel files in Selenium we have to take help of third party API like JXL and Apache POI.
- To read or write an Excel,Apache provides a very famous library POI. This library is capable enough to read and write both XLS and XLSX file format of Excel.

Difference between JXL and Apache POI

- Most significant difference is that Java JXL does not support ".xlsx" format; it only supports ".xls" format.
- JXL doesn't support Conditional Formatting, Apache POI does.
- JXL doesn't support rich text formatting, i.e. different formatting within a text string.

Different Classes and Interfaces in Apache POI:



Classes and Interfaces

Explanation of classes and interfaces POI for reading XLS and XLSX file

- **Workbook**: XSSFWorkbook and HSSFWorkbook classes implement this interface.
- **XSSFWorkbook**: Is a class representation of XLSX file.
- **HSSFWorkbook**: Is a class representation of XLS file.
- **Sheet**: XSSFSheet and HSSFSheet classes implement this interface.
- **XSSFSheet**: Is a class representing a sheet in an XLSX file.
- **HSSFSheet**: Is a class representing a sheet in an XLS file.
- **Row**: XSSFRow and HSSFRow classes implement this interface.
- **XSSFRow**: Is a class representing a row in the sheet of XLSX file.
- **HSSFRow**: Is a class representing a row in the sheet of XLS file.
- **Cell**: XSSFCell and HSSFCell classes implement this interface.
- **XSSFCell**: Is a class representing a cell in a row of XLSX file.
- **HSSFCell**: Is a class representing a cell in a row of XLS file.

Download Apache POI Jars and configure in your project to work with excel.

1. If you are using plain java project then download the jars from below link and configure in your project's build path.

http://poi.apache.org/download.html

2. And if you are using maven use below dependencies in your POM.xml file.

```
                <dependency>
  <groupId>org.apache.poi</groupId>
  <artifactId>poi</artifactId>
  <version>4.1.2</version>
 </dependency>
 <dependency>
```

```
<groupId>org.apache.poi</groupId>
<artifactId>poi-ooxml</artifactId>
<version>4.1.2</version>
</dependency>
```

We have a excel file in project directory and it contains below data:

	A	B	C
1	FirstName1	LastName1	
2	FirstName2	LastName2	
3	FirstName3	LastName3	
4	FirstName4	LastName4	
5	FirstName5	LastName5	
6	FirstName6	LastName6	
7	FirstName7	LastName7	
8	FirstName8	LastName8	
9	FirstName9	LastName9	
10	FirstName10	LastName10	
11			

Please refer below program to read above excel file in selenium:

```
import java.io.File;
import java.io.FileInputStream;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.testng.annotations.Test;

public class ReadExcel {

@Test
public void readExcel() throws Exception {

String excelPath="D:\\Workspace_Eclipse\\ReadExcel\\TestData\\TestData.xlsx";
String fileName="TestData";
String sheetName="Test";
//Create the object of File Class to get the excel path
File file= new File(excelPath);
//To read the file
FileInputStream fis= new FileInputStream(file);
XSSFWorkbook wb= new XSSFWorkbook(fis);
```

```
XSSFSheet sheet=wb.getSheet(sheetName);
//Get Total Rows in Excel Sheet
int rowCount=sheet.getLastRowNum();
System.out.println("Total Rows: "+(rowCount+1));
//Print a particular cell value
String data=sheet.getRow(0).getCell(1).getStringCellValue();
System.out.println("Particular cell value: "+data);

//Loop to print all values of the excel sheet
for(int i=0; i<=rowCount;i++) {
Row row=sheet.getRow(i);
for(int j=0; j<row.getLastCellNum();j++) {
String data1=sheet.getRow(i).getCell(j).getStringCellValue();
System.out.print(data1+" ");
}
System.out.println();
}
wb.close();
}
}
```

Output:

```
[RemoteTestNG] detected TestNG version 6.14.3
Total Rows: 10
Particular cell value: LastName1
FirstName1 LastName1
FirstName2 LastName2
FirstName3 LastName3
FirstName4 LastName4
FirstName5 LastName5
FirstName6 LastName6
FirstName7 LastName7
FirstName8 LastName8
FirstName9 LastName9
FirstName10 LastName10
PASSED: readExcel
```

How to create Excel Library so that we can use it inside out test cases. For that please refer below program.

Excel Library class:

```
import java.io.File;
import java.io.FileInputStream;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class ExcelLibrary {

 XSSFWorkbook wb;
 XSSFSheet sheet;
 //Below Constructor is to load the excel configuration
 public ExcelLibrary() throws Exception {
  String excelPath="D:\\Workspace_Eclipse\\ReadExcel\\TestData\\TestData.xlsx";
  File file= new File(excelPath);
  FileInputStream fis= new FileInputStream(file);
  wb= new XSSFWorkbook(fis);
 }
 public String readData(String sheetName, int row, int col) {
  sheet=wb.getSheet(sheetName);
  String data=sheet.getRow(row).getCell(col).getStringCellValue();
  return data;
 }
}
```

TestClass:

```
import org.testng.annotations.Test;
import com.readExcel.ExcelLibrary;

public class ReadExcelTest {

 @Test
 public void readExcelTest() throws Exception {
  ExcelLibrary obj= new ExcelLibrary();
  //Call readData method from ExcelLibrary class to get the value of Particular cell
  String datString=obj.readData("Test", 5, 1);
  System.out.println("The data is: "+datString);
 }
}
```

- **Write data into excel file in Selenium using Apache POI**

Below is the code to write data into excel sheet:

Library class to write data into excel sheet

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class WriteExcel {

 public void writeExcel(String sheetName, String cellvalue, int row, int col) throws Exception {

  String excelPath="D:\\Workspace_Eclipse\\ReadExcel\\TestData\\TestData.xlsx";

  File file= new File(excelPath);

  FileInputStream fis= new FileInputStream(file);

  XSSFWorkbook wb= new XSSFWorkbook(fis);

  XSSFSheet sheet= wb.getSheet(sheetName);

  sheet.getRow(row).createCell(col).setCellValue(cellvalue);

  FileOutputStream fos= new FileOutputStream(new File(excelPath));

  wb.write(fos);

  wb.close();

  fos.close();
 }
}
```

Sample Test Class

```
import org.testng.annotations.Test;
import com.writeExcel.WriteExcel;

public class WriteExcelTest {

 WriteExcel obj= new WriteExcel();

 @Test
 public void writeExcelTest() throws Exception {
  obj.writeExcel("Test", "Male", 0, 2);
 }

 @Test
 public void writeExcelTest1() throws Exception {
  obj.writeExcel("Test", "Female", 1, 2);
 }

}
```

- **Complete Excel Library**

Using this excel library we can perform following manipulations on excel sheet.

- How to get total row count - int getRowCount(String sheetName)
- How to read the value - String getCellData(String sheetName,String colName,int rowNum), String getCellData(String sheetName,int colNum,int rowNum).
- How to set the value - boolean setCellData(String sheetName,String colName,int rowNum, String data)
- How to add a New work sheet - boolean addSheet(String sheetname)
- How to remove a work sheet - boolean removeSheet(String sheetName)
- How to add a column - boolean addColumn(String sheetName,String colName)
- How to remove a column - boolean removeColumn(String sheetName, int colNum)
- How to check if a sheet is exist or not - boolean isSheetExist(String sheetName)
- How to get total column count - int getColumnCount(String sheetName)
- How to get cell row number - int getCellRowNum(String sheetName,String colName,String cellValue)

Complete Excel Library:

```
package com.utility;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.Calendar;
import org.apache.poi.hssf.usermodel.HSSFDateUtil;
import org.apache.poi.xssf.usermodel.XSSFCell;
import org.apache.poi.xssf.usermodel.XSSFCellStyle;
import org.apache.poi.xssf.usermodel.XSSFRow;
import org.apache.poi.xssf.usermodel.XSSFSheet;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;

public class NewExcelLibrary {

 public static String path = System.getProperty("user.dir") + "/TestData/TestData.xlsx";

// public  String path;
 public FileInputStream fis = null;
 public FileOutputStream fileOut = null;
 private XSSFWorkbook workbook = null;
 private XSSFSheet sheet = null;
 private XSSFRow row = null;
 private XSSFCell cell = null;

 public NewExcelLibrary() {

  // this.path=path;
  try {
   fis = new FileInputStream(path);
   workbook = new XSSFWorkbook(fis);
   sheet = workbook.getSheetAt(0);
   fis.close();
  } catch (Exception e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
  }
 }

 public NewExcelLibrary(String path) {
```

```
 this.path = path;
 try {
  fis = new FileInputStream(path);
  workbook = new XSSFWorkbook(fis);
  sheet = workbook.getSheetAt(0);
  fis.close();
 } catch (Exception e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
 }
}

// returns the row count in a sheet
 public int getRowCount(String sheetName){
  int index = workbook.getSheetIndex(sheetName);
  if(index==-1)
   return 0;
  else{
  sheet = workbook.getSheetAt(index);
  int number=sheet.getLastRowNum()+1;
  return number;
  }

 }

 public String getCellData(String sheetName,String colName,int rowNum){
  try{
   if(rowNum <=0)
    return "";

  int index = workbook.getSheetIndex(sheetName);
  int col_Num=-1;
  if(index==-1)
   return "";

  sheet = workbook.getSheetAt(index);
  row=sheet.getRow(0);
  for(int i=0;i<row.getLastCellNum();i++){
   //System.out.println(row.getCell(i).getStringCellValue().trim());
```

```
     if(row.getCell(i).getStringCellValue().trim().equals(colName.trim()))
      col_Num=i;
    }
    if(col_Num==-1)
     return "";

    sheet = workbook.getSheetAt(index);
    row = sheet.getRow(rowNum-1);
    if(row==null)
     return "";
    cell = row.getCell(col_Num);

    if(cell==null)
     return "";
    //System.out.println(cell.getCellType());
    if(cell.getCellType().name().equals("STRING"))
      return cell.getStringCellValue();
    else if(cell.getCellType().name().equals("NUMERIC") ||
cell.getCellType().name().equals("FORMULA") ){

      String cellText  = String.valueOf(cell.getNumericCellValue());
      if (HSSFDateUtil.isCellDateFormatted(cell)) {
            // format in form of M/D/YY
       double d = cell.getNumericCellValue();

       Calendar cal =Calendar.getInstance();
       cal.setTime(HSSFDateUtil.getJavaDate(d));
             cellText =
              (String.valueOf(cal.get(Calendar.YEAR))).substring(2);
             cellText = cal.get(Calendar.DAY_OF_MONTH) + "/" +
                       cal.get(Calendar.MONTH)+1 + "/" +
                       cellText;

             //System.out.println(cellText);

           }

      return cellText;
     }else if(cell.getCellType().name().equals("BLANK"))
        return "";
```

```
    else
     return String.valueOf(cell.getBooleanCellValue());

   }
   catch(Exception e){

    e.printStackTrace();
    return "row "+rowNum+" or column "+colName +" does not exist in xls";
   }
  }

  // returns the data from a cell
  public String getCellData(String sheetName,int colNum,int rowNum){
   try{
    if(rowNum <=0)
     return "";

   int index = workbook.getSheetIndex(sheetName);

   if(index==-1)
    return "";


   sheet = workbook.getSheetAt(index);
   row = sheet.getRow(rowNum-1);
   if(row==null)
    return "";
   cell = row.getCell(colNum);
   if(cell==null)
    return "";

    if(cell.getCellType().name().equals("STRING"))
     return cell.getStringCellValue();
    else if(cell.getCellType().name().equals("NUMERIC") ||
cell.getCellType().name().equals("FORMULA") ){

     String cellText  = String.valueOf(cell.getNumericCellValue());
     if (HSSFDateUtil.isCellDateFormatted(cell)) {
           // format in form of M/D/YY
      double d = cell.getNumericCellValue();
```

```
   Calendar cal =Calendar.getInstance();
   cal.setTime(HSSFDateUtil.getJavaDate(d));
         cellText =
          (String.valueOf(cal.get(Calendar.YEAR))).substring(2);
         cellText = cal.get(Calendar.MONTH)+1 + "/" +
                  cal.get(Calendar.DAY_OF_MONTH) + "/" +
                  cellText;

        // System.out.println(cellText);

       }

  return cellText;
 }else if(cell.getCellType().name().equals("BLANK"))
    return "";
 else
  return String.valueOf(cell.getBooleanCellValue());
 }
 catch(Exception e){

 e.printStackTrace();
 return "row "+rowNum+" or column "+colNum +" does not exist  in xls";
 }
}

// returns true if data is set successfully else false
public boolean setCellData(String sheetName,String colName,int rowNum, String data){
 try{
 fis = new FileInputStream(path);
 workbook = new XSSFWorkbook(fis);

 if(rowNum<=0)
  return false;

 int index = workbook.getSheetIndex(sheetName);
 int colNum=-1;
 if(index==-1)
  return false;
```

```
sheet = workbook.getSheetAt(index);

row=sheet.getRow(0);
for(int i=0;i<row.getLastCellNum();i++){
 //System.out.println(row.getCell(i).getStringCellValue().trim());
 if(row.getCell(i).getStringCellValue().trim().equals(colName))
  colNum=i;
}
if(colNum==-1)
 return false;

sheet.autoSizeColumn(colNum);
row = sheet.getRow(rowNum-1);
if (row == null)
 row = sheet.createRow(rowNum-1);

cell = row.getCell(colNum);
if (cell == null)
       cell = row.createCell(colNum);

   // cell style
   //CellStyle cs = workbook.createCellStyle();
   //cs.setWrapText(true);
   //cell.setCellStyle(cs);
   cell.setCellValue(data);

   fileOut = new FileOutputStream(path);

workbook.write(fileOut);

   fileOut.close();

}
catch(Exception e){
 e.printStackTrace();
 return false;
}
return true;
}
```

```
// returns true if sheet is created successfully else false
public boolean addSheet(String  sheetname){

 FileOutputStream fileOut;
 try {
   workbook.createSheet(sheetname);
   fileOut = new FileOutputStream(path);
   workbook.write(fileOut);
      fileOut.close();
 } catch (Exception e) {
  e.printStackTrace();
  return false;
 }
 return true;
}

// returns true if sheet is removed successfully else false if sheet does not exist
public boolean removeSheet(String sheetName){
 int index = workbook.getSheetIndex(sheetName);
 if(index==-1)
  return false;

 FileOutputStream fileOut;
 try {
  workbook.removeSheetAt(index);
  fileOut = new FileOutputStream(path);
  workbook.write(fileOut);
     fileOut.close();
 } catch (Exception e) {
  e.printStackTrace();
  return false;
 }
 return true;
}

// returns true if column is created successfully
public boolean addColumn(String sheetName,String colName){
 //System.out.println("***************addColumn**********************");

 try{
```

```
 fis = new FileInputStream(path);
 workbook = new XSSFWorkbook(fis);
 int index = workbook.getSheetIndex(sheetName);
 if(index==-1)
  return false;

XSSFCellStyle style = workbook.createCellStyle();
//style.setFillForegroundColor(HSSFColor.GREY_40_PERCENT.index);
//style.setFillPattern(HSSFCellStyle.SOLID_FOREGROUND);

sheet=workbook.getSheetAt(index);

row = sheet.getRow(0);
if (row == null)
 row = sheet.createRow(0);

//cell = row.getCell();
//if (cell == null)
//System.out.println(row.getLastCellNum());
if(row.getLastCellNum() == -1)
 cell = row.createCell(0);
else
 cell = row.createCell(row.getLastCellNum());

      cell.setCellValue(colName);
      cell.setCellStyle(style);

      fileOut = new FileOutputStream(path);
 workbook.write(fileOut);
    fileOut.close();

 }catch(Exception e){
 e.printStackTrace();
 return false;
 }

 return true;

}
```

```
// removes a column and all the contents
public boolean removeColumn(String sheetName, int colNum) {
 try{
 if(!isSheetExist(sheetName))
  return false;
 fis = new FileInputStream(path);
 workbook = new XSSFWorkbook(fis);
 sheet=workbook.getSheet(sheetName);
 XSSFCellStyle style = workbook.createCellStyle();
 //style.setFillForegroundColor(HSSFColor.GREY_40_PERCENT.index);
 //XSSFCreationHelper createHelper = workbook.getCreationHelper();
 //style.setFillPattern(HSSFCellStyle.NO_FILL);

 for(int i =0;i<getRowCount(sheetName);i++){
  row=sheet.getRow(i);
  if(row!=null){
   cell=row.getCell(colNum);
   if(cell!=null){
    cell.setCellStyle(style);
    row.removeCell(cell);
   }
  }
 }
 fileOut = new FileOutputStream(path);
 workbook.write(fileOut);
    fileOut.close();
 }
 catch(Exception e){
  e.printStackTrace();
  return false;
 }
 return true;

}
 // find whether sheets exists
public boolean isSheetExist(String sheetName){
 int index = workbook.getSheetIndex(sheetName);
 if(index==-1){
  index=workbook.getSheetIndex(sheetName.toUpperCase());
   if(index==-1)
```

```
    return false;
   else
    return true;
  }
 else
  return true;
 }

// returns number of columns in a sheet
public int getColumnCount(String sheetName){
 // check if sheet exists
 if(!isSheetExist(sheetName))
  return -1;

 sheet = workbook.getSheet(sheetName);
 row = sheet.getRow(0);

 if(row==null)
  return -1;

 return row.getLastCellNum();

 }

 public int getCellRowNum(String sheetName,String colName,String cellValue){

 for(int i=2;i<=getRowCount(sheetName);i++){
     if(getCellData(sheetName,colName , i).equalsIgnoreCase(cellValue)){
      return i;
     }
    }
 return -1;

 }
}
```

- **Data Driven Framework/Testing Part 1**

What is Data Driven Testing?

Data-driven is a test automation framework which stores test data in a table or spreadsheet format. This allows automation engineers to have a single test script which can execute tests for all the test data in the table.

Why Data Driven Testing?

Example 1:
We want to test the login system with multiple input fields with 100 different data sets.
To test this, you can take following different approaches:
Approach 1) Create 100 scripts one for each dataset and runs each test separately one by one.
Approach 2) Manually change the value in the test script and run it several times.
Approach 3) Import the data from the excel sheet. Fetch test data from excel rows one by one and execute the script.
In the given three scenarios first two are laborious and time-consuming. Therefore, it is ideal to follow the third approach.
Thus, the third approach is nothing but a Data-Driven framework.

Example 2: Test different products in e-commerce application.

Advantages of Data Driven Framework

- Advantages of using Data Driven Test Framework
- Re-usability of code
- Improves test coverage
- Faster Execution
- Less maintenance
- Permits better error handling

How to Implement?

Using DataProvider in TestNG
An important features provided by TestNG is the testng DataProvider feature. It helps you to write data-driven tests which essentially means that same test method can be run multiple times with different data-setsTo test this.

To use the DataProvider feature in the tests, you have to declare a method annotated by @DataProvider and then use the said method in the test method using the ‘dataProvider‘ attribute in the @Test annotation.
Data provider returns a two-dimensional JAVA object and test method will invoke M times in a M*N type of object array.

Lets have a look Data driven example without using excel sheet just to understand the data driven test -- Data driven testing of login functionality.

Base Class: In this base class we have created following methods
setup() - to launch the browser and navigate to application
tearDown() - quit the browser
getData() - to setup the test data using 2-D Array.

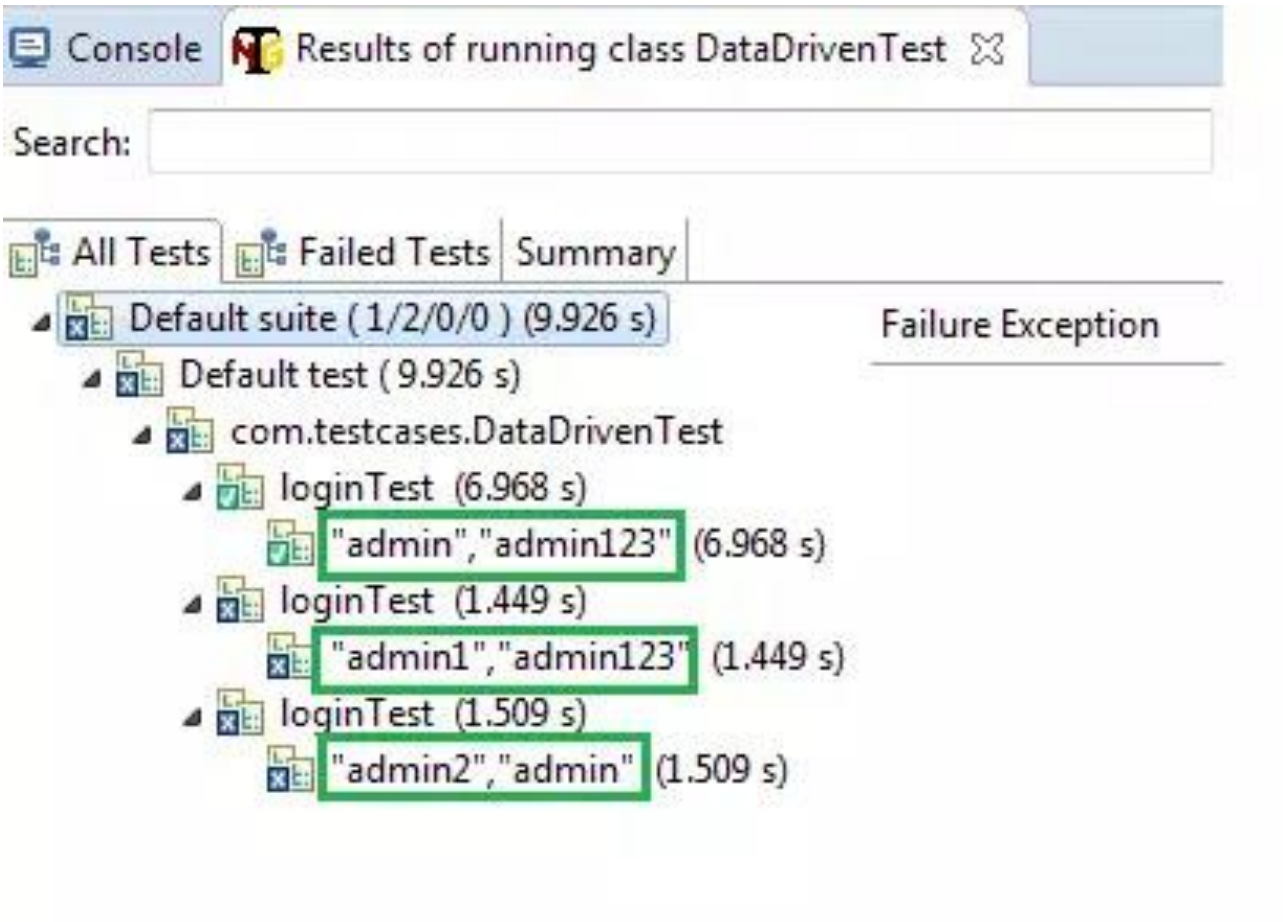
```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.DataProvider;
import com.utility.NewExcelLibrary;

public class BaseClass {
 public WebDriver driver;
 NewExcelLibrary obj= new
NewExcelLibrary("D:\\Workspace_Eclipse\\DataDriven\\TestData\\User.xlsx");

 @BeforeMethod
 public void setup() {
  System.setProperty("webdriver.chrome.driver",  " chromedriver.exe path");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://opensource-demo.orangehrmlive.com/");
 }

 @AfterMethod
 public void tearDown() {
  driver.quit();
 }

 @DataProvider(name = "Credentials")
```

```
 public Object[][] getData() {

  Object[][] data = new Object[3][2];

  data[0][0] = "admin";
  data[0][1] = "admin123";

  data[1][0] = "admin1";
  data[1][1] = "admin123";

  data[2][0] = "admin2";
  data[2][1] = "admin";

  return data;
 }
}
```

Test Class: Supply the dataProvider name in test method

```
import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.Test;
import com.base.BaseClass;

public class DataDrivenTest extends BaseClass {

 @Test(dataProvider = "Credentials")
 public void loginTest(String username,String password) {

  driver.findElement(By.id("txtUsername")).sendKeys(username);
  driver.findElement(By.id("txtPassword")).sendKeys(password);
  driver.findElement(By.id("btnLogin")).click();
  String actualURL=driver.getCurrentUrl();
  String expectedURL="https://opensource-demo.orangehrmlive.com/index.php/dashboard";
  Assert.assertEquals(actualURL, expectedURL);
 }
}
```

Output: Refer the below output, you can see our test executed with different set of test data

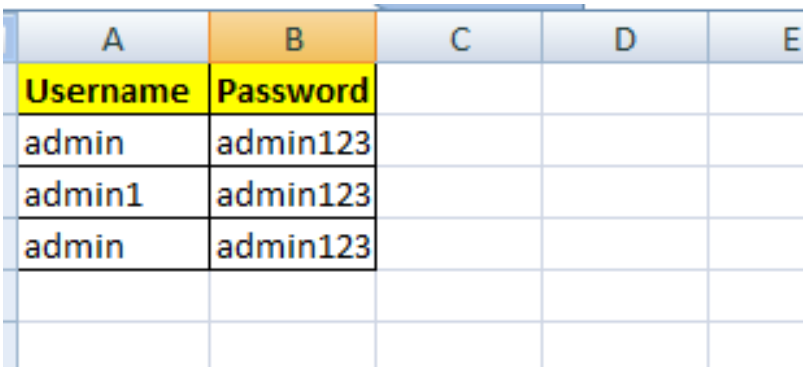


- **Data Driven Framework/Testing Part 2**

Data Driven Framework Part 2

Data driven example with excel sheet - Please refer below source codes to understand the implementation of Data Driven Testing/Framework using excel sheet.

In this example we are passing the username and password using excel sheet to test the login functionality.
Excel Sheet with user credentials:



A	B	C	D	E
Username	Password			
admin	admin123			
admin1	admin123			
admin	admin123			

Base Class: In this base class we have created following methods
setup() - to launch the browser and navigate to application
tearDown() - quit the browser
getExcelData() - to get the excel test data and store using 2-D Array.
and we have created the object of NewExcelLibrary to use the different methods to access the excel sheet. Complete excel library you can find here.

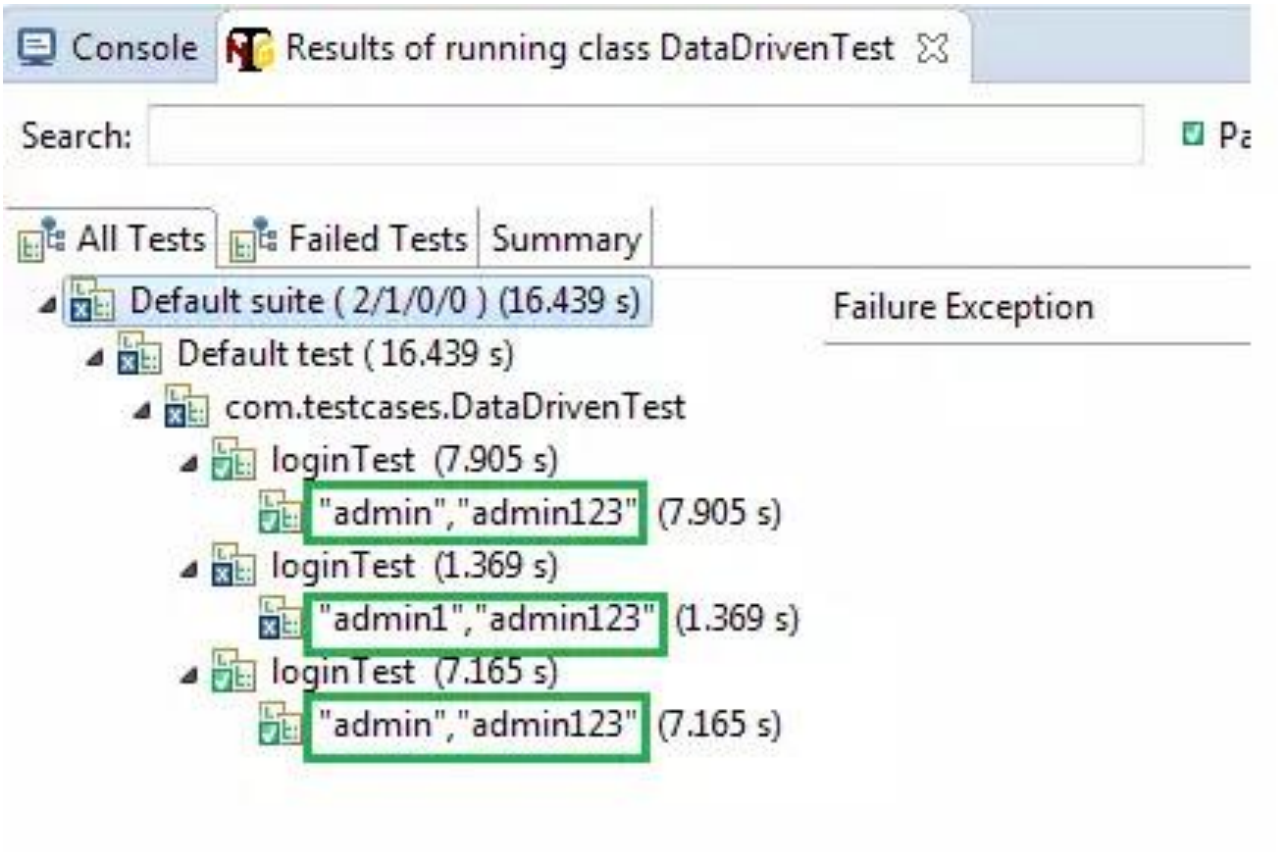
```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.DataProvider;
import com.utility.NewExcelLibrary;

public class BaseClass {
 public WebDriver driver;
 NewExcelLibrary obj= new
NewExcelLibrary("D:\\Workspace_Eclipse\\DataDriven\\TestData\\User.xlsx");

 @BeforeMethod
 public void setup() {
  System.setProperty("webdriver.chrome.driver", " chromedriver.exe path ");
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("https://opensource-demo.orangehrmlive.com/");
 }

 @AfterMethod
 public void tearDown() {
  driver.quit();
 }
```

```
 @DataProvider(name ="Credentials1")
 public Object[][] getExcelData() {
  //Totals rows count
  int rows=obj.getRowCount("Data");
  //Total Columns
  int column=obj.getColumnCount("Data");
  int actRows=rows-1;

  Object[][] data= new Object[actRows][column];

  for(int i=0;i<actRows;i++) {
   for(int j=0; j<column;j++) {
    data[i][j]=obj.getCellData("Data", j, i+2);
   }
  }
  return data;
 }
}
```

TestClass: Supply the dataProvider name in test method

```
import org.openqa.selenium.By;
import org.testng.Assert;
import org.testng.annotations.Test;
import com.base.BaseClass;

public class DataDrivenTest extends BaseClass {

 @Test(dataProvider = "Credentials1")
 public void loginTest(String username,String password) {

  driver.findElement(By.id("txtUsername")).sendKeys(username);
  driver.findElement(By.id("txtPassword")).sendKeys(password);
  driver.findElement(By.id("btnLogin")).click();
  String actualURL=driver.getCurrentUrl();
  String expectedURL="https://opensource-demo.orangehrmlive.com/index.php/dashboard";
  Assert.assertEquals(actualURL, expectedURL);
 }
}
```

Output: Refer the below output, you can see our test executed with different set of test data



Git and GitHub:

- ✓ **Selenium Integration with Git and GitHub**

Selenium Integration with Git and GitHub

In this post we will discuss about following things:

- What do you mean by CM and what is SCM tool?
- Why do we use SCM tools and advantages
- What is Git and GitHub?
- Basic Architecture
- Steps to work with Git and GitHub
- How to install Git
- Demo – execute different commands using cmd

What is CM and what are diff SCM tool?

- Configuration management(CM) is managing the configuration of all of the project's key products and assets.

- SCM stands for Source Code Management is an integral part of any project in the IT world.
- Important component in DeveOps culture.
- Source Code Management or Version Control Systems in any project ensure all the members of a team stay on top of the source code changes.
- SCM practices include revision control and the establishment of baselines.

Top SCM Tools:

- MS Team Foundation Server (TFS):
- Kallithea - Open Source
- GitLab - Continuous Integration (CI), Continuous Delivery (CD) is an integral part of GitLab
- Bitbucket Server:
- Subversion (SVN):
- Git and GitHub

Why do we use version control System?

- **Collaboration**- Without a SCM in place, you're probably working together in a shared folder on the same set of files. It's extremely error prone, someone will overwrite someone else's changes. With a SCM, everybody on the team is able to work absolutely freely - on any file at any time.
- **Storing Versions**
- **Restoring Previous Versions**
- **Understanding What Happened:** Every time you save a new version of your project, your SCM requires you to provide a short description of what was changed. This helps you understand how your project evolved between versions.
- **Backup**

Now lets talk about Git and GitHub

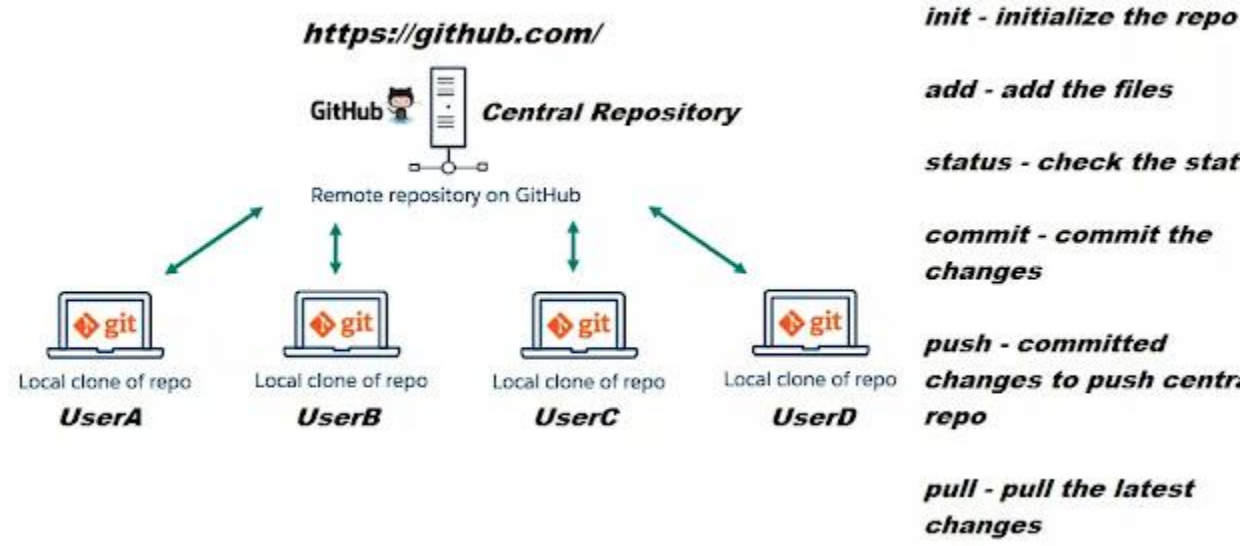
Git and GitHub

- Git – initially developed by Linus Torvalds is a version control system.
- Git is a version control system that lets you manage and keep track of your source code history.
- GitHub is a cloud-based hosting service that lets you manage Git repositories.

Different Terminologies with GitHub

Repository: You can simply, treat it as a storage area of your workplace that contains all your documentation files and the history of changes.
Clone: Clones are literally clones (copies) of a repository that sit on the developer's computer instead of a server elsewhere.
Commit: Whatever the changes you make in your files will come under commit. Every change is saved under a particular name or ID which is also called "revision".
Push: Pushing refers to sending your committed changes to a remote repository such as GitHub.com.

Pull Request: If you have made changes in code/script, to show the other collaborators you send a pull request.
Fork: It is a copy of other's repository in your account in which you can make changes and it won't affect the original code.
Branching: When you extract a portion /section of code from the main or remote track of your software, then it is called 'branch' and the process is known as Branching.
Fetch: Fetching refers to getting the latest changes from an online repository (like GitHub.com) without merging them in.
Merge: Merging takes the changes from one branch (in the same repository or from a fork), and applies them into another.



Git and GitHub

Steps to work with Git and GiHub

- **Download and Install Git** - https://git-scm.com/download/win , https://git-scm.com/download/mac (Please refer below YouTube video at 27:00 minute for download/installation of Git)
- Create a GitHub Account

- Using Command Prompt - by using different git commands
- Using Eclipse - we have inbuilt Git plugin in eclipse

Please refer below different git commands:

1. git config

This command sets the author name **and** email address respectively to be used **with** your commits.

git config –**global** user.name "[name]"

git config –**global** user.email "[email address]"

Example:

git config user.name "Hitendra Kuamar Verma"

git config user.email "Hitendra@Hitendra-PC"

2. git init

This command **is** used to start a new repository.

git init [repository path

Example: navigate to repo path **and**

git init

3. git clone

This command **is** used to obtain a repository **from an existing URL.**

git clone [url]

Example: navigate to your repo path where you want to clone **and** write below command **in** cmd

git clone https://github.com/hverma22/Test2

4. git add

This command adds a file to the staging area.

git add [file]

Example:

git add [file]

git add *

git add .

5. git commit

git commit -m "[commit message]"

This command records **or** snapshots the file permanently **in** the version history.

Example:

git commit -m "First Commit"

6. git diff

This command shows the file differences which are **not** yet staged.

Example:
git diff –-staged
git diff [first branch] [second branch]

7. git reset
This command unstages the file, but it preserves the file contents.
git reset [file]
git reset [commit]
git reset –hard [commit]

8. git status
git status
This command lists all the files that have to be committed.

9. git rm
This command deletes the file **from your working directory and stages the deletion.**
git rm [file]

10. git branch
git branch
This command lists all the local branches **in** the current repository.
Example:
git master

11. git log
git log
This command **is** used to list the version history **for** the current branch.
Example:
git log --online

12. git merge
git merge [branch name]
This command merges the specified branch's history into the current branch.

13. git remote add [variable name] [Remote Server Link]
This command **is** used to connect your local repository to the remote server.
Example:
git remote add origin https://github.com/hverma22/Test5

14. git push

git push [variable name] master
This command sends the committed changes of master branch to your remote repository.
Example:
git push origin master
git push origin master --force

15. git pull
git pull [Repository Link]
This command fetches **and** merges changes on the remote server to your working directory.
Example:
git pull https://github.com/hverma22/Test2.git

16. git show
git show [commit]
This command shows the metadata **and** content changes of the specified commit.
Command: git show <ChangeID>:<FilePath>
Example:
git show **45**dhfg56:/src/test/newtest.xml

17. Checkout
git checkout [branch name]
This command **is** used to switch **from one branch to another or You can get the specific previous version.**
Command: git checkout <ChangeID> <filePath **with** extenion>
Example:
git checkout **6475**fgh5 pom.xml

Please refer below video to understand better. We have covered the following scenarios:

- Create a new Project and Add your fresh project into Github (into existing repository)

Add all project files
Make the change in the code and push the code

- How to pull the code when someone make the changes.
- How to checkout the particular version.
- How to work with existing repository- Clone a existing repository

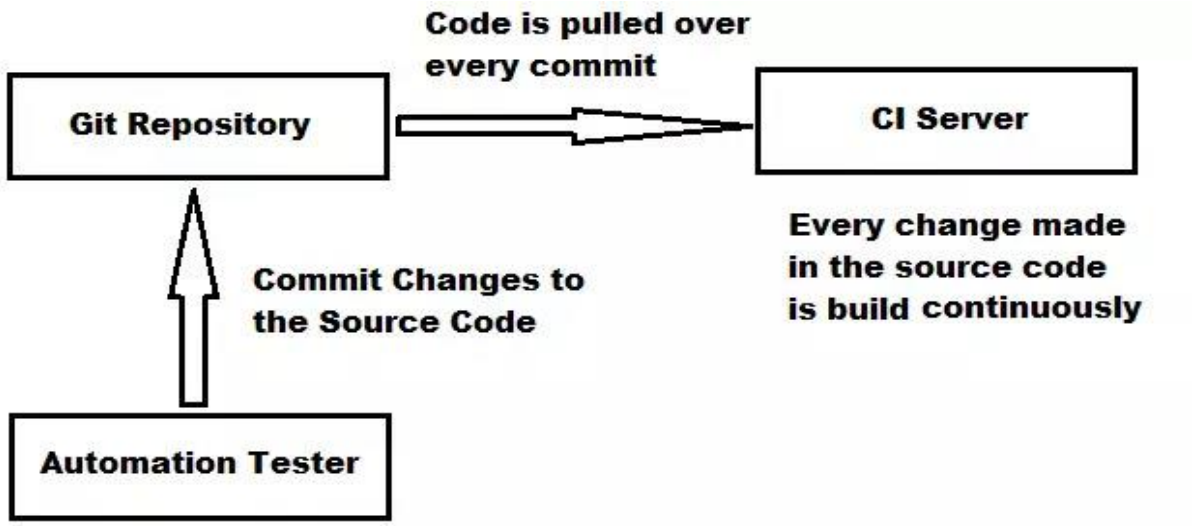
Continuous Integration with Jenkins:

- **Selenium Integration with Jenkins**

In this post we will discuss about following points:

- What is Continuous Integration?
- What is Jenkins
- Advantages of Jenkins
- Other CI tools
- Complete Jenkins installation Process

What is Continuous Integration?
Continuous Integration is a development practice in which the Automation Testers are required to commit changes to the source code in a shared repository several times a day or more frequently. Every commit made in the repository is then built. This allows the teams to detect the problems early. Continuous Integration is the most important part of DevOps that is used to integrate various DevOps stages. Jenkins is the most famous Continuous Integration tool.



What is Jenkins?
Jenkins is an open source automation tool written in Java with plugins built for Continuous Integration purpose. Jenkins is used to build and test your software projects continuously making it easier for developers to integrate changes to the project, and making it easier for users to obtain a fresh build. It also allows you to continuously deliver your software by integrating with a large number of testing and deployment technologies. Jenkins integrates development life-cycle processes of all kinds, including build, document, test, package, stage, deploy, static analysis and

much more. Jenkins achieves Continuous Integration with the help of plugins. Plugins allows the integration of Various DevOps stages. If you want to integrate a particular tool, you need to install the plugins for that tool. For example: Git, Maven, HTML publisher etc.

Advantages:

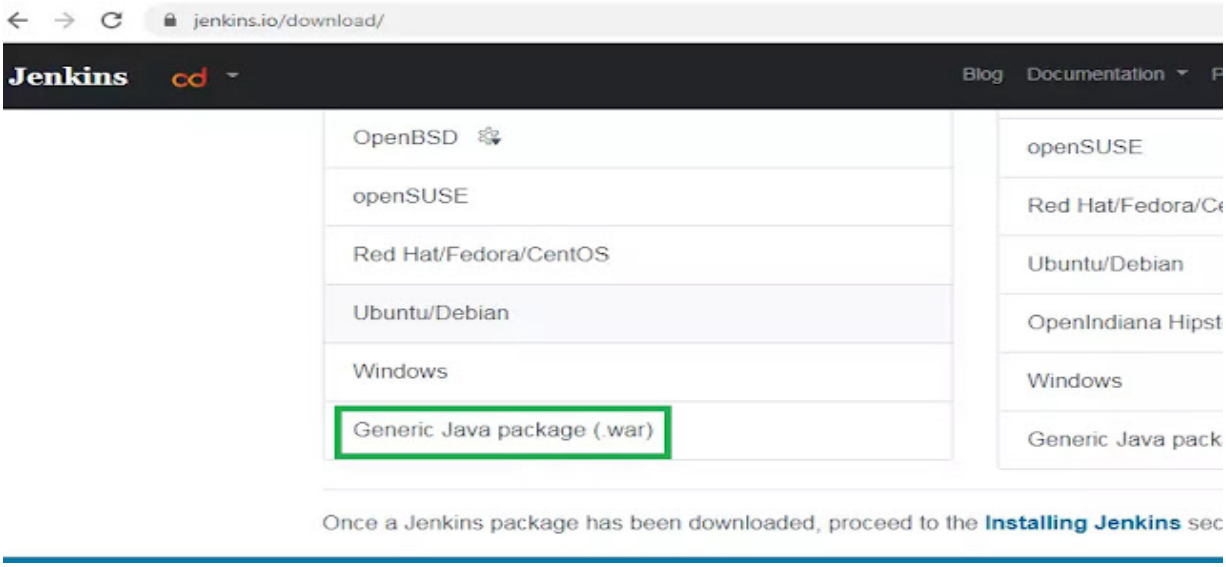
- Advantages of Jenkins include:
- It is an open source tool with great community support.
- It is easy to install.
- It has 1000+ plugins to ease your work. If a plugin does not exist, you can code it and share with the community.
- It is built with Java and hence, it is portable to all the major platforms.

Apart from Jenkins, we have many more tools in the market such as:

- Anthill
- Bamboo
- Cruise Control
- Team City

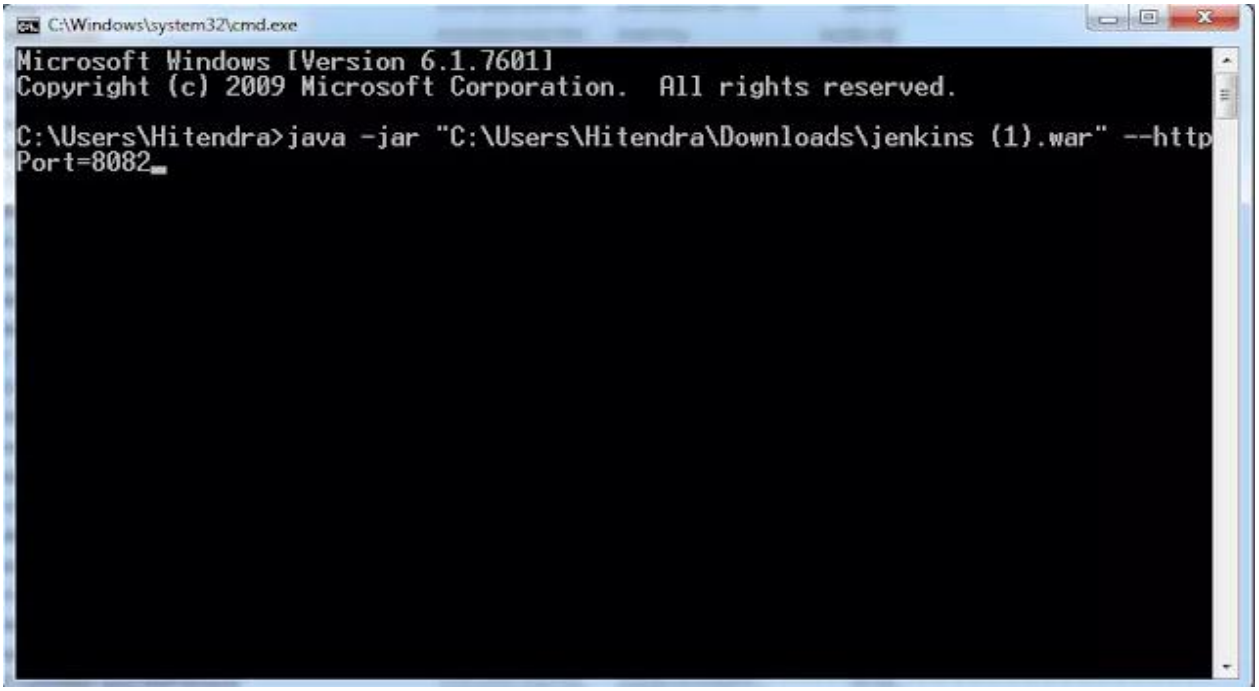
Complete Jenkins Installation Process
URL: https://www.jenkins.io/download/

1. Download the WAR file from https://www.jenkins.io/download/

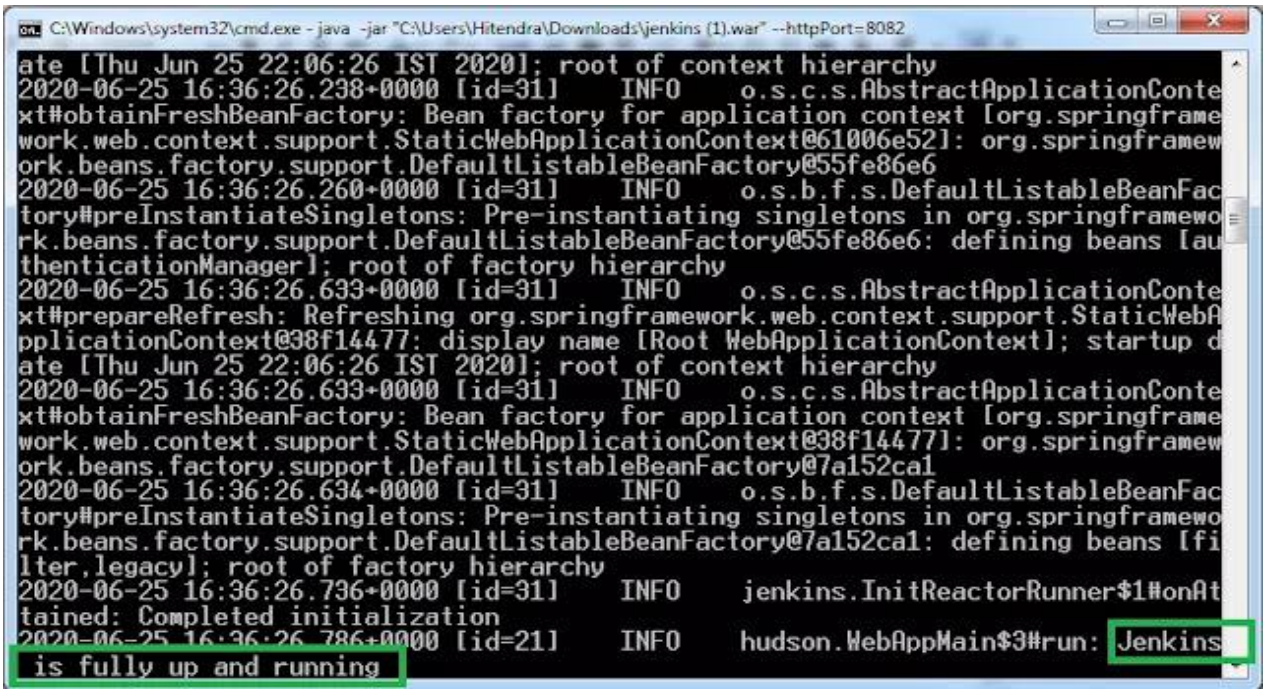


2. Start the Jenkins using below commands:

>C:\Users\Hitendra>java -jar <Path of WAR file> --httpPort=8082
Default port is 8080, no need to specify port if your jenkins running on 8080 port

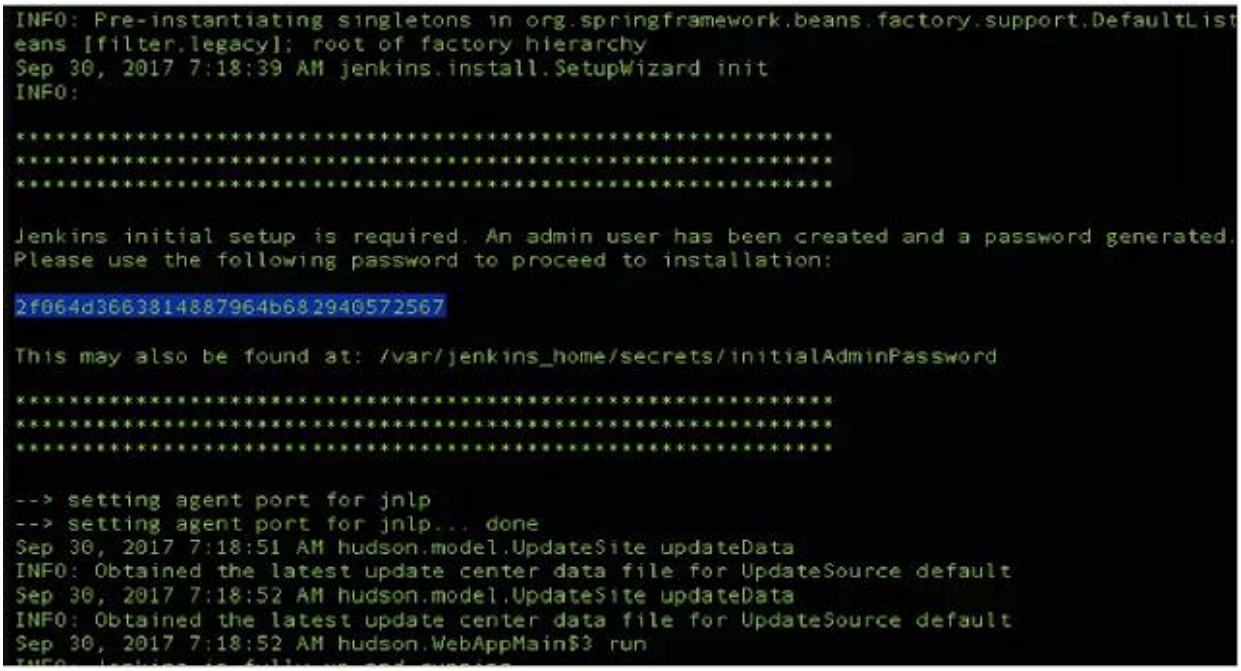


Press Enter



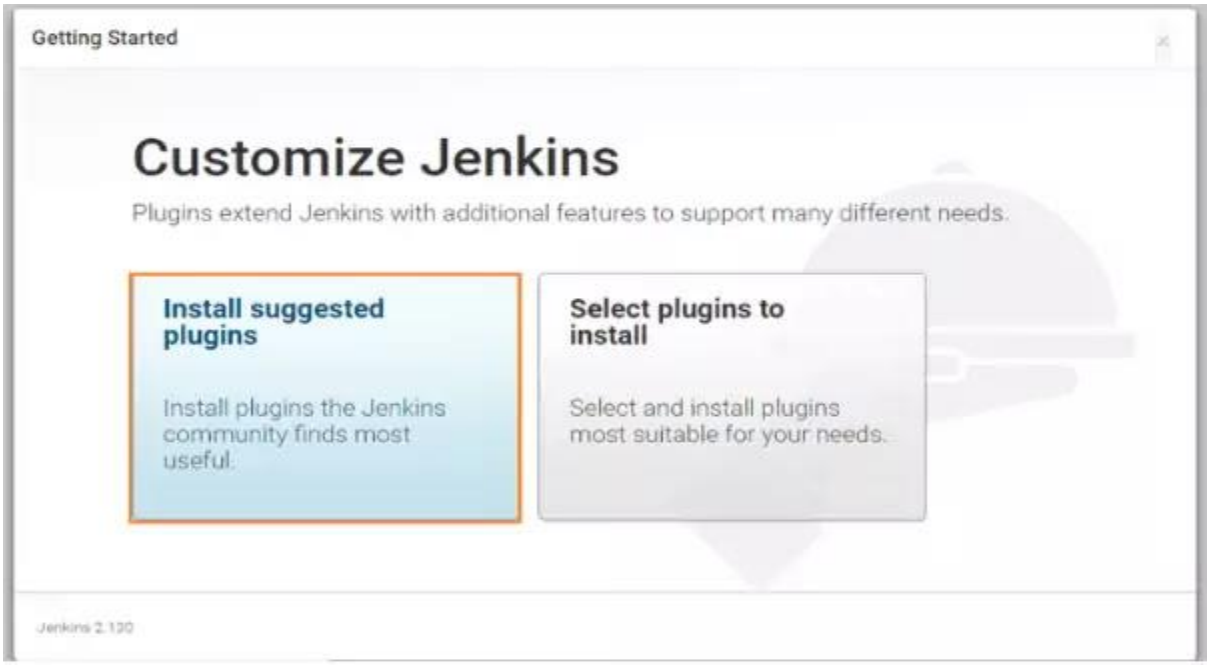
3. Browse to http://localhost:8080 (or whichever port you configured for Jenkins when installing it) and wait until the Unlock Jenkins page appears.

4. From the Jenkins console log output, copy the automatically-generated alphanumeric password from the cmd or from given path.

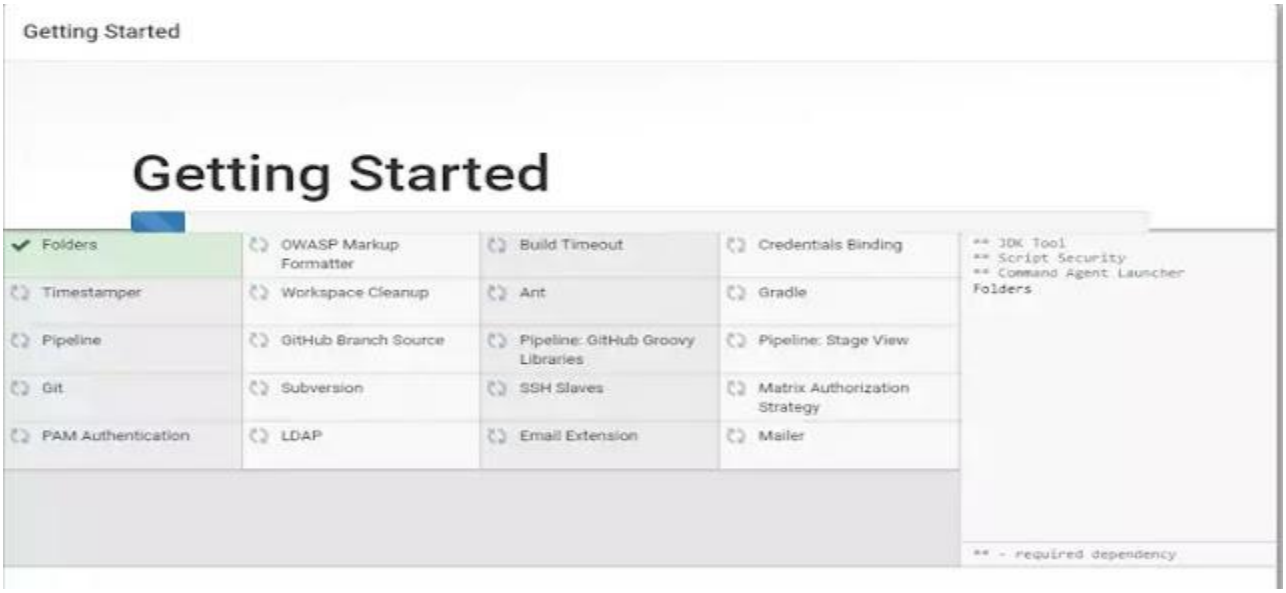


5. On the Unlock Jenkins page, paste this password into the Administrator password field and click Continue.

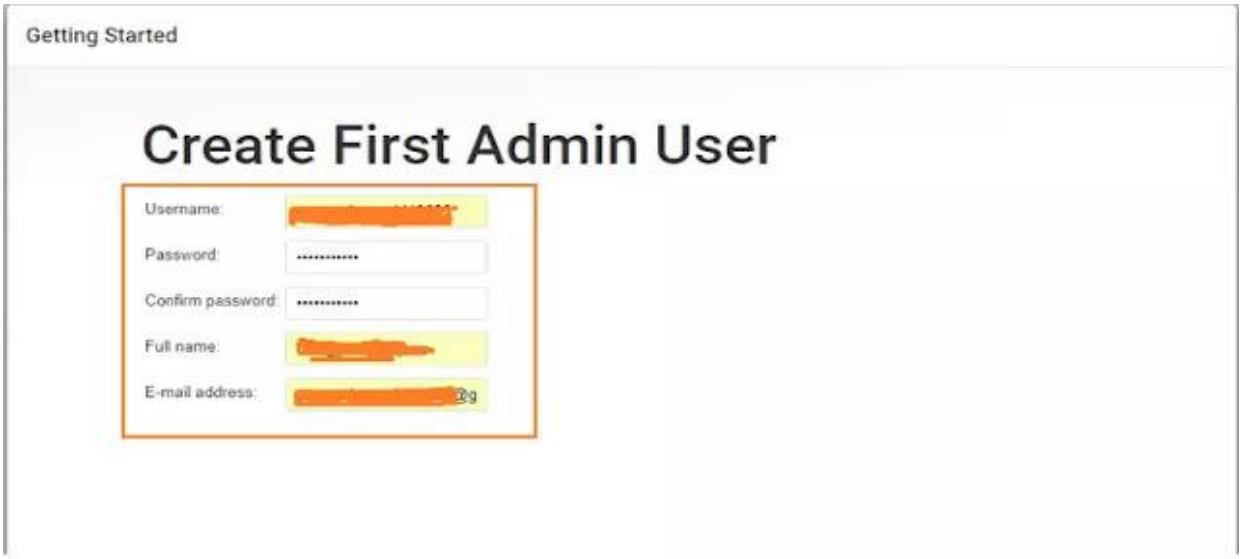
6. You can install either the suggested plugins or selected plugins you choose. To keep it simple, we will install the suggested plugins.



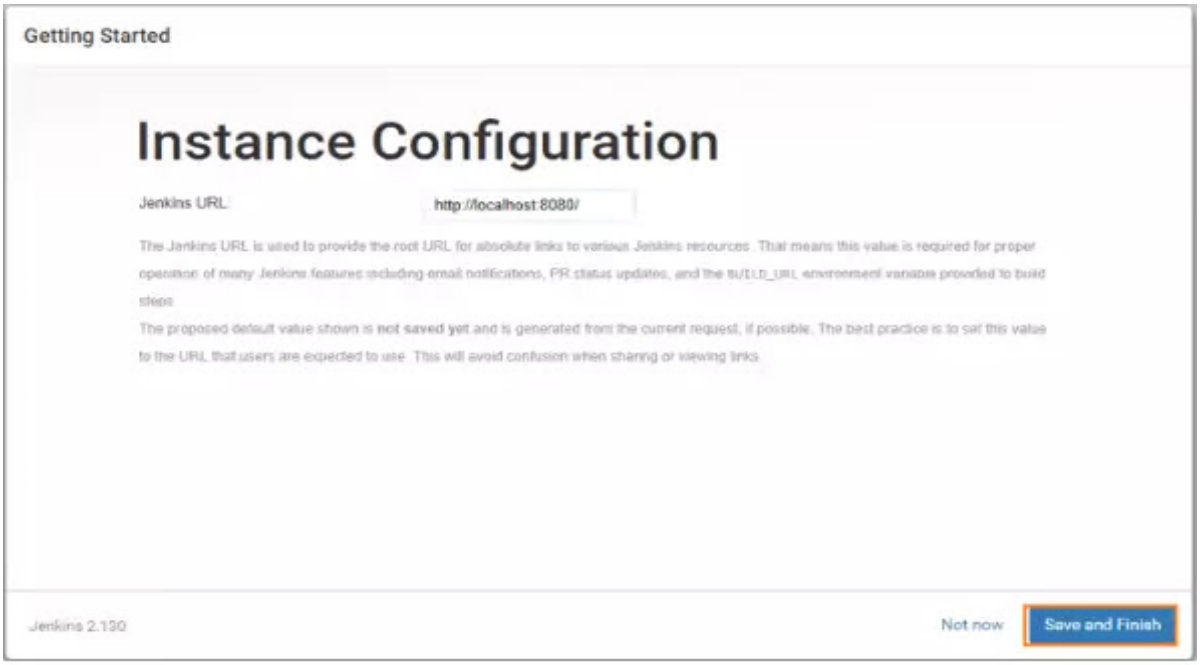
7. Wait until the plugins are completely installed.



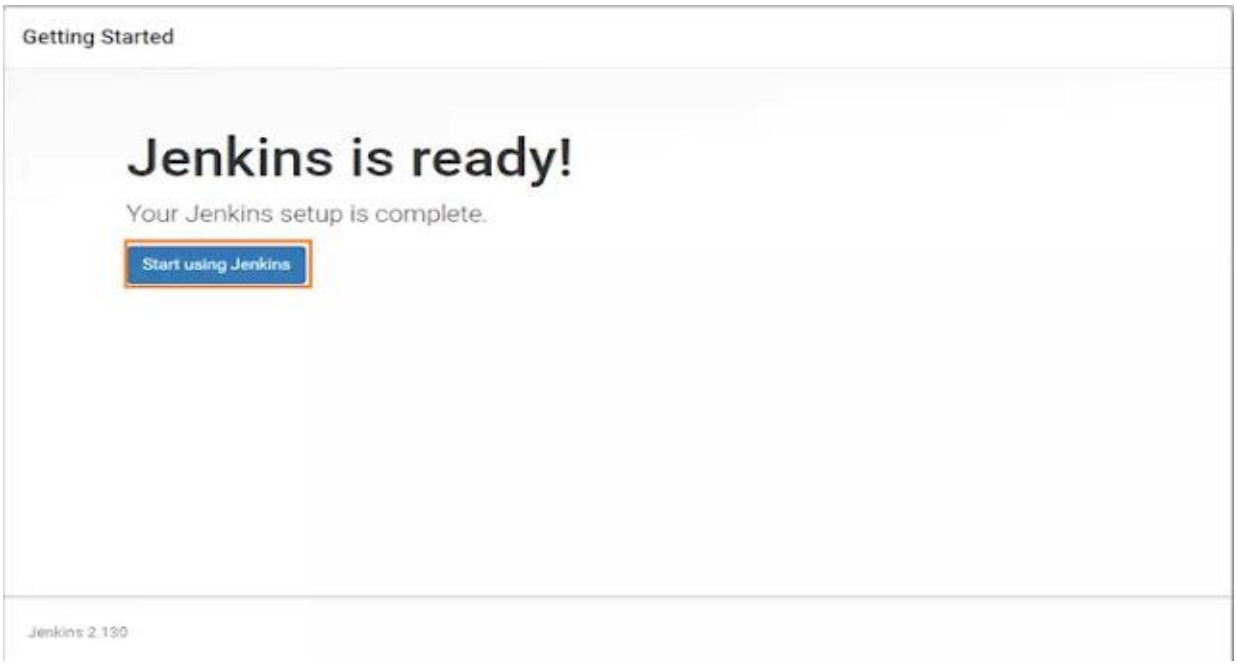
8. The next thing we should do is create an admin user for Jenkins. Put in your details and click “Save and Continue”.



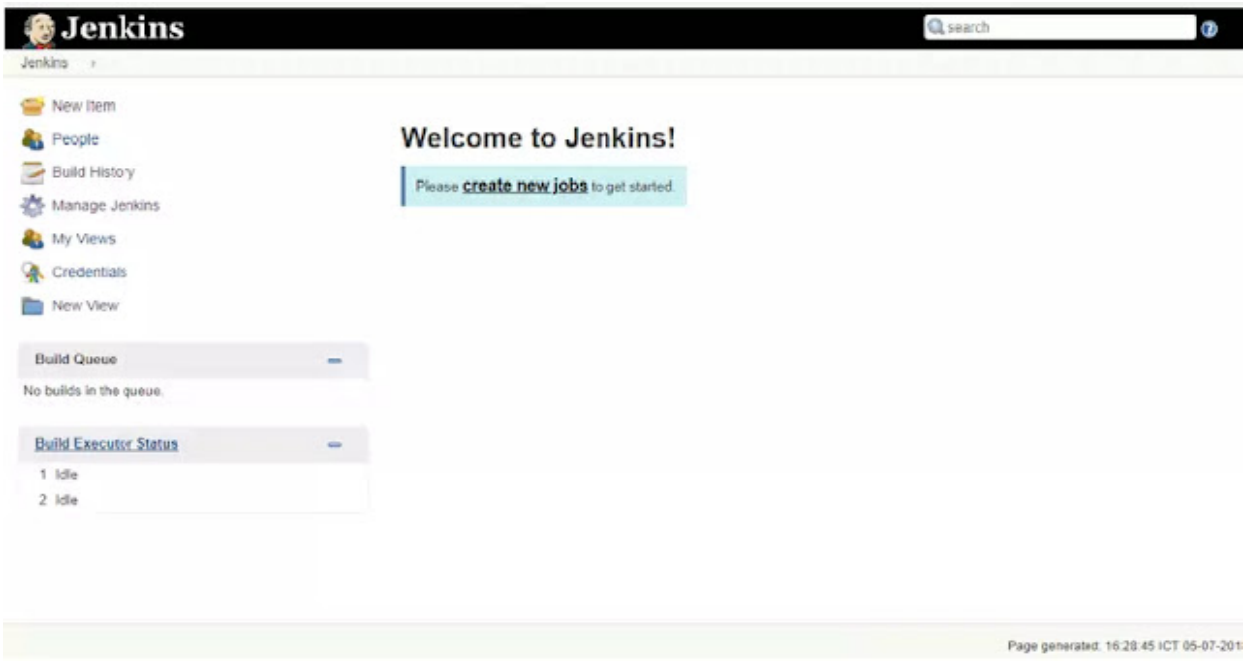
9. Click “Save and Finish” to complete the Jenkins installation.



10. Now, click “Start using Jenkins” to start Jenkins.



11. Finally, here is the default Jenkins page.



Please refer below YouTube videos to understand the Different Configurations on Jenkins

- **Demo 1**: Create a batch file and run on Jenkins (Integration of Eclipse Java Project + Jenkins)
- Steps to create the batch file

```
set projectLocation=D:\Workspace_Eclipse\TestNGJenkinsDemo
cd %projectLocation%
```

```
set classpath=%projectLocation%\bin;%projectLocation%\lib\*
java org.testng.TestNG %projectLocation%\testng.xml
pause
```

Save this file with extension (.bat).
Please follow below video on how to use this command in Jenkins

Demo2 : Installation of different plugins and Integration of (Selenium + Maven + Jenkins) and Selenium + Maven + GitHub + Jenkins Integration with Parameters
Install different plugins – Green Balls, Email Extensions plugin, HTML publisher, Maven Integration, TestNG Results Plugin

Demo3 : Selenium + Maven + GitHub + Jenkins + TestNG Report + Extent Report + Email +Email with Attachment + Schedule Jobs
In this, we will discuss about following things.
1. How to generate HTML report in Jenkins
2. How to generate Extent report in Jenkins
3. How to send email
4. How to send email with an attachment
5. How to run job periodically in Jenkins

Please refer below YouTube Video for complete understanding:

https://www.youtube.com/watch?v=GAjxnkH6Tt4&list=PLsGOlyTzNH6f_pZuGZDFjI5IeCWGQqiwF&index=75

Demo4 : Add logo in extent report and attach screen shot for failed test case-

- Logo in Extent Report and ScreenShot attachment in Extent Report
- Attachment in the Email and email body
- Run job in headless mode

Sample Email Body Template:

```
Hi Team,
<br/>
<br/>
Please find the below Automation Testing Results for Project:
<b>$PROJECT_NAME</b>.
<br/>
<br/>
<b>Project Name:</b> $PROJECT_NAME
<br/>
<b>Build#</b>: $BUILD_NUMBER
```

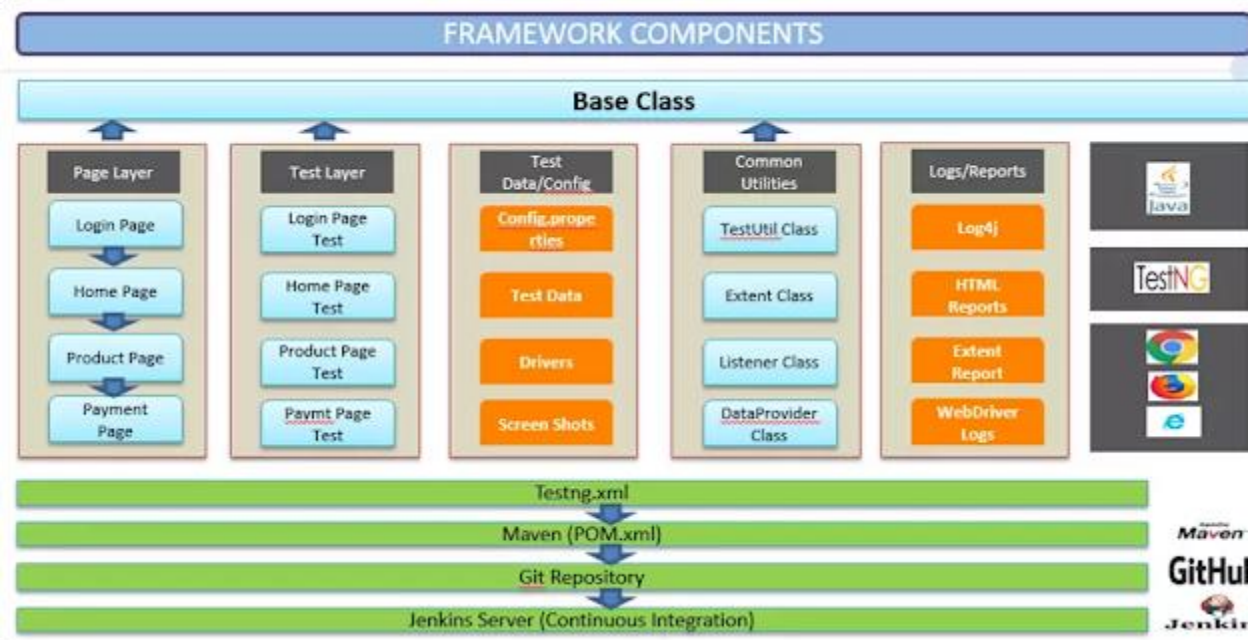
```
<br/>
<b>Build Status</b>: $BUILD_STATUS.<br/>
<br/>
Check <a href='http://localhost:8082/job/Demo4/ws/ExtentDemo/test-
output/ExtentReport/MyReport.html'>Extent Report</a> to view full
results.<br/>
<br/>
Please find below <b>Colsole Logs</b> for your reference<br/>
<br/>
--LOG-BEGIN--<br/>
<pre style='line-height: 22px; display: block; color: #333; font-family:
Monaco,Menlo,Consolas,"Courier New",monospace; padding: 10.5px; margin: 0 0
11px; font-size: 13px; word-break: break-all; word-wrap: break-word; white-
space: pre-wrap; background-color: #f5f5f5; border: 1px solid #ccc; border:
1px solid rgba(0,0,0,.15); -webkit-border-radius: 4px; -moz-border-radius:
4px; border-radius: 4px;'>
${BUILD_LOG, maxLines=500, escapeHtml=true}
</pre>
--LOG-END--
<br/>
<br/>
Thank You
Kebede B.
```

Selenium Framework:

- **Selenium Data Driven Framework with POM**

In this article we are going to discuss following points:

- Brief Introduction about Selenium
- What is a TEST Automation Framework?
- What is Data Driven Testing and why we are using it?
- POM and its Pros and Cons
- Framework Components
- Key Enhancements in Framework
- Framework Demo
- Steps to Create framework from Scratch
- How to explain Framework
- Implementation of OOP Concept in Selenium Framework



Different Components in a Framework

Selenium and It's history

Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms. Selenium is not just a single tool but a suite of software's, each catering to different testing needs of an organization. It has four components:

- Selenium Integrated Development Environment (IDE)
- Selenium Remote Control (RC)
- WebDriver
- Selenium Grid

Read More about **Selenium and It's Components**

Selenium Supporting Environment

Operating Environment: MS Windows, Linux, Apple OX

Programming Languages:

Language Supported by Selenium

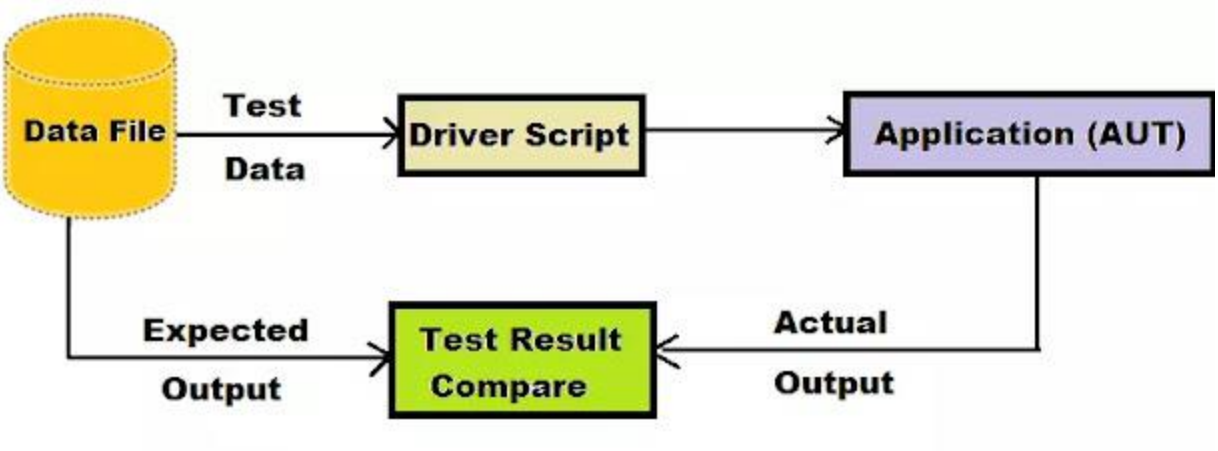
Application Environment: Web Based and Mobile Based web applications which has web forms. For mobile native apps we are using third party tool i.e. Appium.

What is a TEST Automation Framework?

A set of guidelines like coding standards , test-data handling , object repository treatment etc... which when followed during automation scripting produce beneficial outcomes like increase code re-usage , higher portability , reduced script maintenance cost etc. Mind you these are just guidelines and not rules; they are not mandatory and you can still script without following the guidelines. But you will miss out on the advantages of having a Framework.

What is Data Driven Testing?

Data-driven is a test automation framework which stores test data in a table or spreadsheet format. This allows automation engineers to have a single test script which can execute tests for all the test data in the table.



Data-Driven Framework

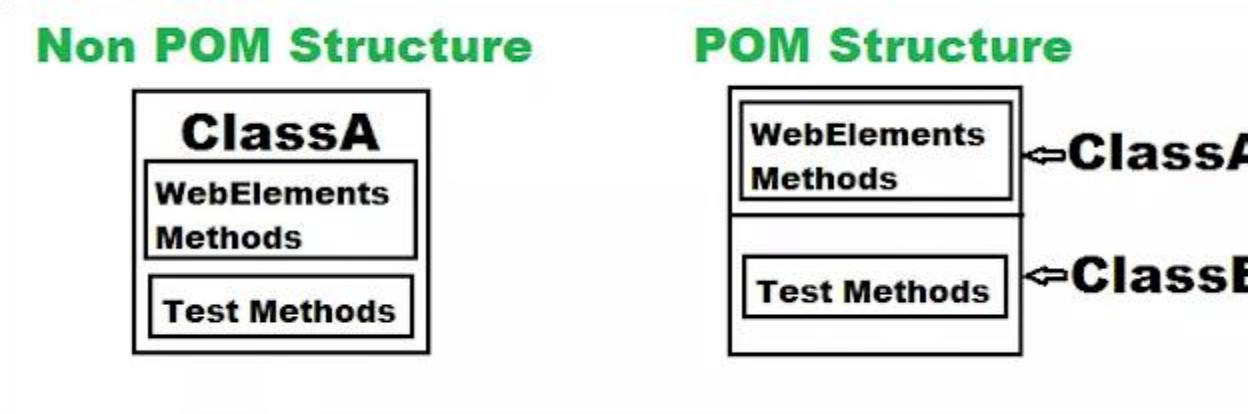
Why Data Driven Testing?

Look at the following Example:

We want to test the login system with multiple input fields with 1000 different data sets.
To test this, you can take following different approaches:
Approach 1) Create 1000 scripts one for each data-set and runs each test separately one by one.
Approach 2) Manually change the value in the test script and run it several times.
Approach 3) Import the data from the excel sheet. Fetch test data from excel rows one by one and execute the script.
In the given three scenarios first two are laborious and time-consuming. Therefore, it is ideal to follow the third approach. Thus, the third approach is nothing but a Data-Driven framework.

Page Object Model

Page Object Model is a design pattern to create Object Repository for web UI elements. Under this model, for each web page in the application, there should be corresponding page class. This Page class will find the Web Elements of that web page and also contains Page methods which perform operations on those Web Elements.



POM Vs Non POM Structure

POM Advantages and Disadvantages:

Advantages

Object Repository: You can create an Object Repository of the fields segmented page-wise. This as a result provides a Page Repository of the application as well. Each page will be defined as a java class.

Functional Encapsulation: All possible functionality or operations that can be performed on a page can be defined and contained within the same class created for each page. This allows for clear definition and scope of each page's functionality.

Low maintenance: Any User Interface changes can swiftly be implemented into the interface as well as class.

Programmer Friendly: Robust and more readable. The Object-oriented approach makes the framework programmer friendly.

Low Redundancy: Helps reduce duplication of code. If the architecture is correctly and sufficiently defined, the POM gets more done in less code.

Efficient & Scalable: Faster than other keyword-driven/data-driven approaches where Excel sheets are to be read/written.

Disadvantages

High Setup Time & Effort: Initial effort investment in development of Automation Framework is high. This is the biggest weight of POM in case of web applications with hundreds/thousands of pages. It is highly suggested that if this model is decided to be implemented, then it should be done parallel to development of the application.

Skilled labor: Testers not technically sound or aware of programming best practices are a nightmare in this case.

Specific: Not a generic model. Automation Framework developed using POM approach is specific to the application. Unlike keyword-driven/data-driven frameworks, it is not a generic framework.

Framework Demo Overview– WordPress Automation

Project/Application: WordPress
No. of Test Cases: 10 Test Cases
Automation Tools: Selenium Web Driver, TestNG framework and Java

Key Points:

- Selenium Webdriver is being used as the core automation engine.
- Eclipse IDE is used to develop the automated scripts.
- Build tool Maven is used for build, execution and dependency purpose.
- TestNG framework is used for organizing the scripts.
- Page Factory are created to store the element definitions.
- Test data is read from Excel sheet at run time.
- Git and Github is used for version control management.
- CI tool Jenkins is used to run the scripts.
- Extent Reports test results are generated for each run.

Test Cases Covered:
Smoke Test: Check for the links are directing to desired pages
Functional Tests: Search Posts
Regression Tests: Add New User, Publish a New Posts

Key Components in this Framework

- Config.Properties file for Constants
- Extent Report
- TestNG with Groups and Priorities
- Simplified script writing
- Cross Browser Testing
- New Selenium Customized functions
- Maven for Jar files version control
- Created dependencies for Jar files using Maven
- Git configuration
- Jenkins Integration
- Ready to use Frame work

Selenium Framework Integration

Maven: Using Maven for build, execution and dependency purpose. Integrating the TestNG dependency in POM.xml file and running this POM.xml file using Jenkins.
Version Control Tool: We use Git as a repository to store our test scripts.
Jenkins: By using Jenkins CI (Continuous Integration) Tool, we execute test cases on daily basis

and also for nightly execution based on the schedule. Test Result will be sent to the peers using Jenkins.

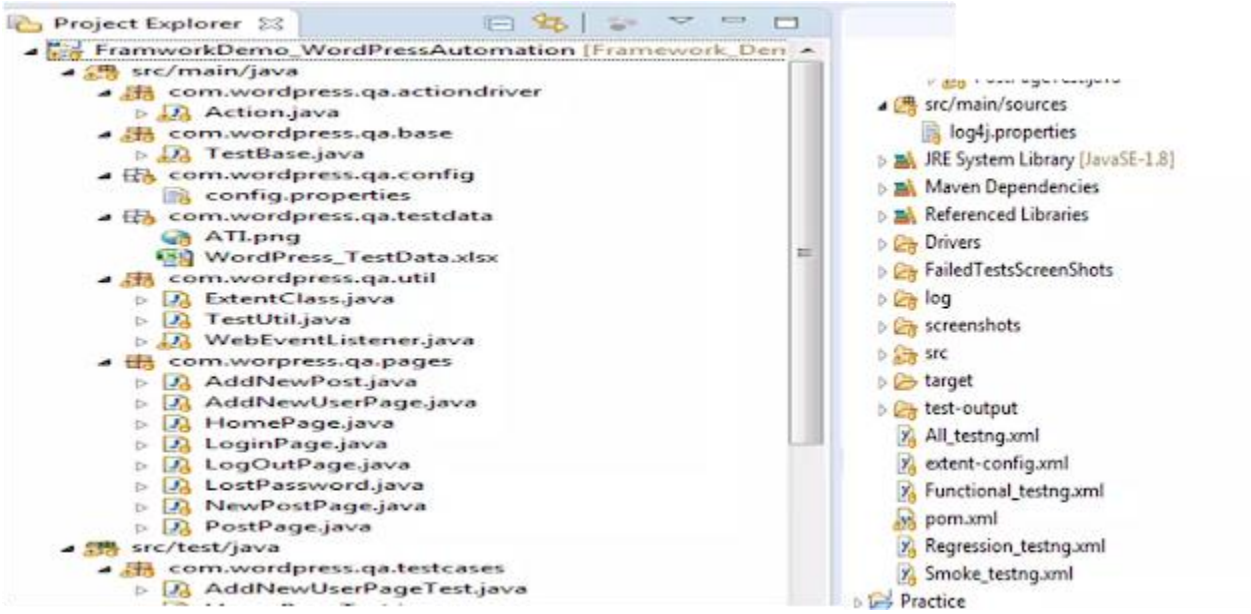
MAVEN Configuration - POM File

```
    <url>http://maven.apache.org</url>

    <!-- <properties> <suiteXmlFile>/Sakriya_WordPressAutoma
        <project.build.sourceEncoding></project.build.source
    <dependencies>
        <dependency>
            <groupId>org.seleniumhq.selenium</groupId>
            <artifactId>selenium-java</artifactId>
            <version>3.11.0</version>
        </dependency>
        <dependency>
            <groupId>org.testng</groupId>
            <artifactId>testng</artifactId>
            <version>6.14.3</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi</artifactId>
```

Sample POM file

Project Structure:

Complete Project structure shown as below:



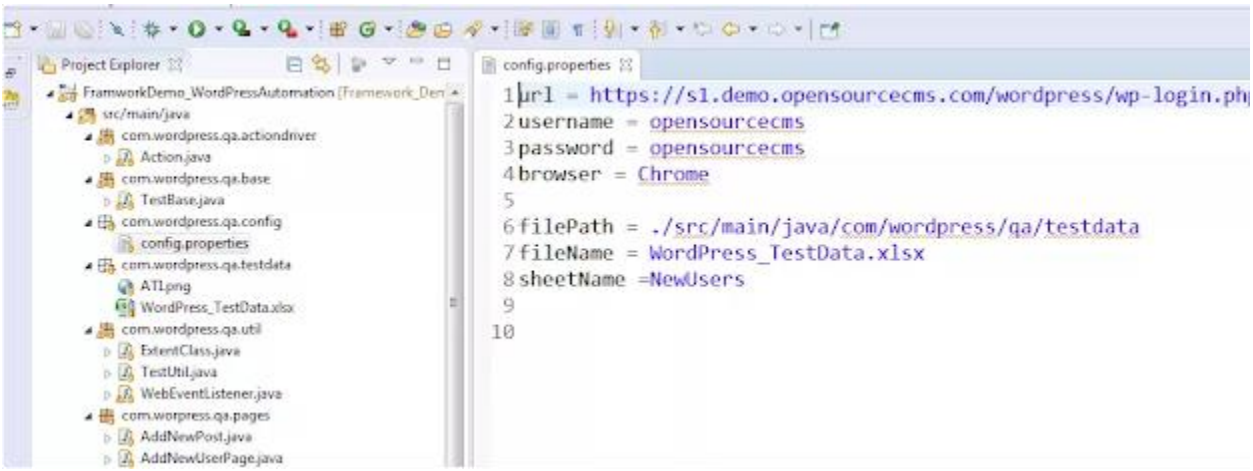
Sample Project Structure

Properties File

This file (config.properties) stores the information that remains static throughout the framework such as browser specific information, application URL, Test Data path etc.

All the details which change as per the environment and authorization such as URL, Login

Credentials are kept in the config.properties file. Keeping these details in a separate file makes easy to maintain.



Properties file

TestBase Class

Test Base class (TestBase.java) deals with all the common functions used by all the pages. This class is responsible for loading the configurations from properties files, Initializing the WebDriver and also to create the object of FileInputStream which is responsible for pointing towards the file from which the data should be read.

TestBase Class

Page Class Package

This package contains all the objects of different pages.

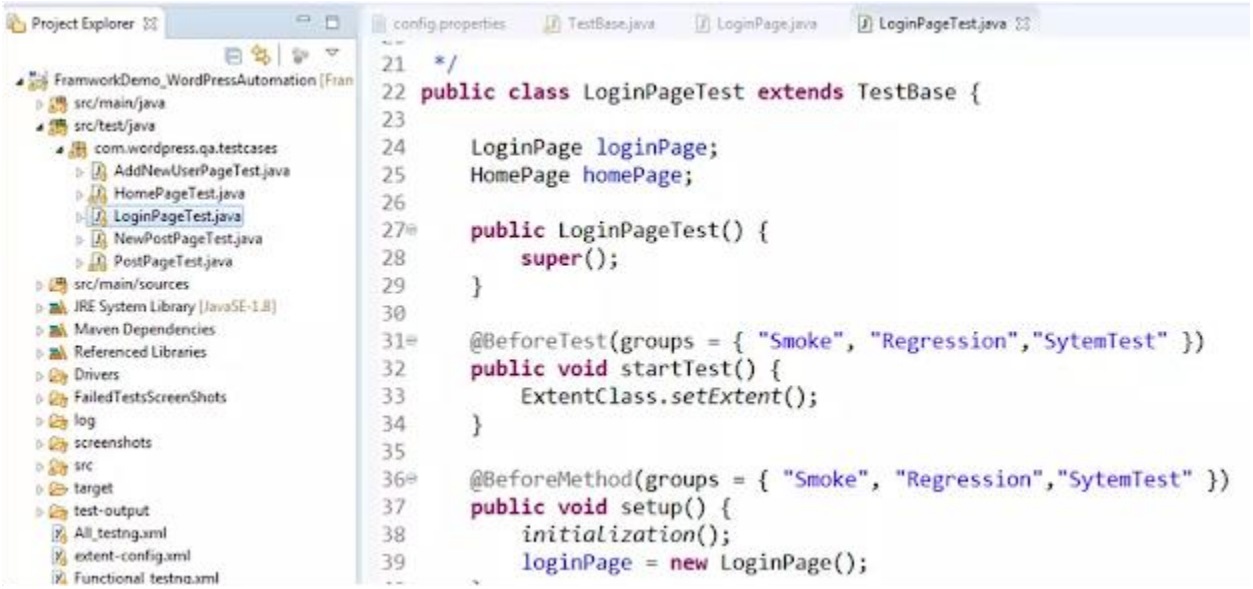
Advantage : If the object locator is changed, and the same object is used in more than 10 test cases, then you can change in one object page , the updated locator will be called in all test cases.

Page Classes

Test Class Package

We have separate packages for Pages and Tests. All the web page related classes come under Pages package and all the tests related classes come under Tests package.

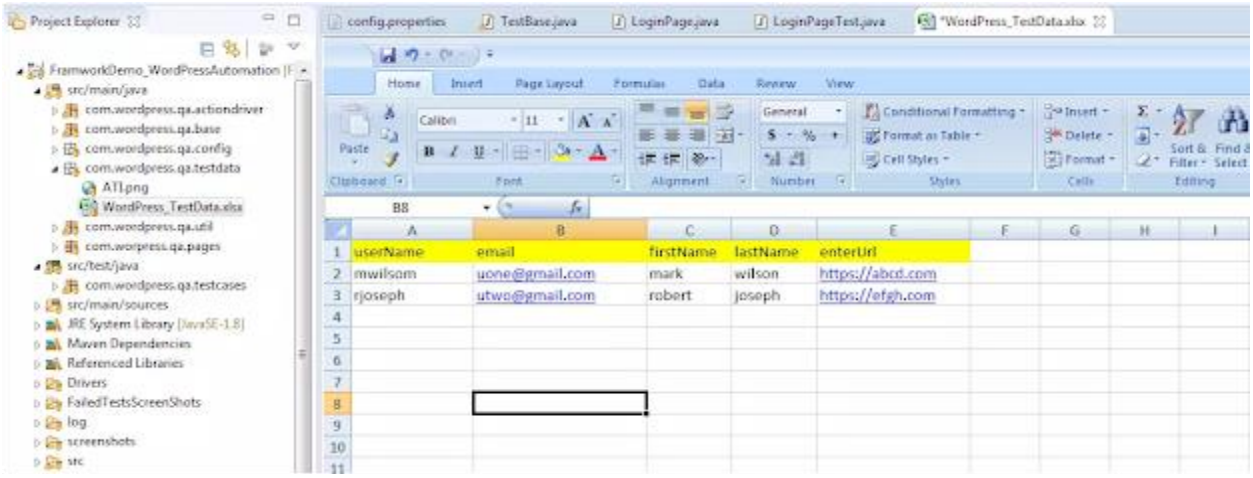
For example, Home Page and Login Page have a separate classes to store element locators. For the login test there would be a separate class which calls the methods from the Home Page class and Login Page class.



Test Classes

Test Data Package

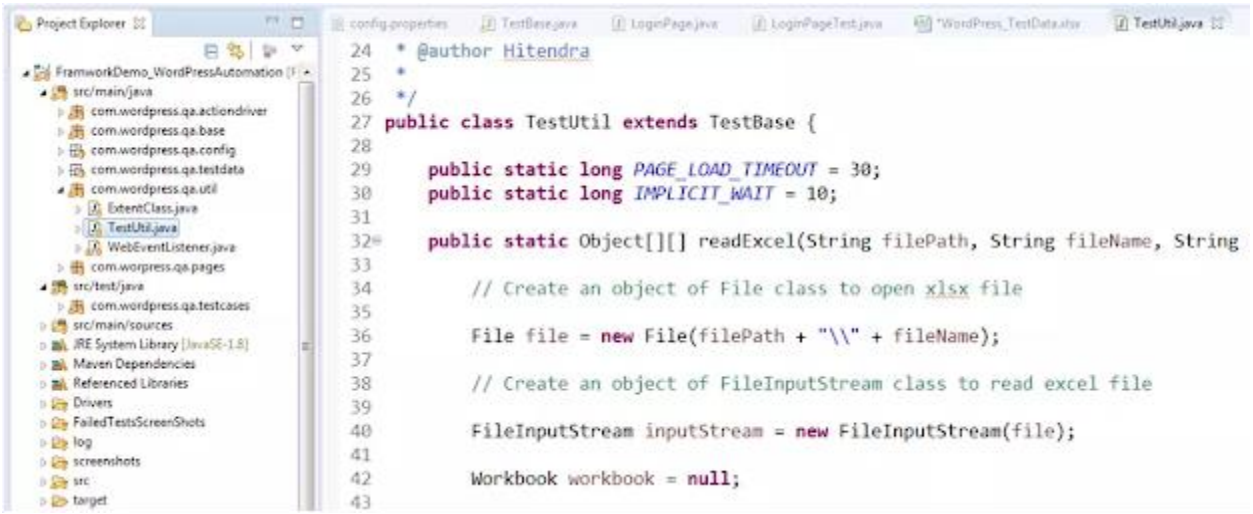
All the test data will be kept in excel sheet (TestData.xlsx). By using ‘TestData.xlsx’, we pass test data and handle data driven testing. We use Apache POI to handle excel sheets.



Test Data Sheet

Utility Package

Utility class (TestUtil.java) stores and handles the functions (The code which is repetitive in nature such as waits, actions, capturing screenshots, accessing excels, sending email etc.,) which can be commonly used across the entire framework. The reason behind creating utility class is to achieve re-usability. This class extends the TestBase class to inherit the properties of TestBase in TestUtil.



Utility Class

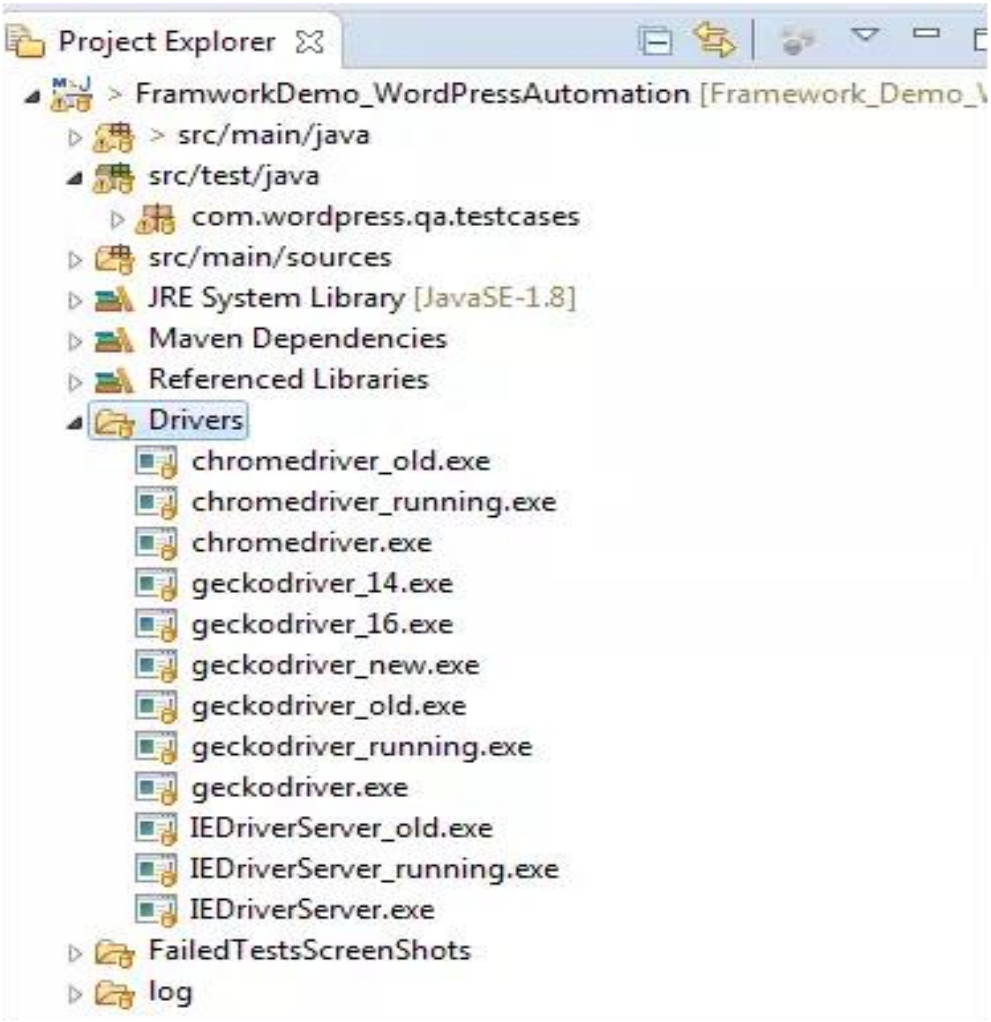
Action Driver Package

ActionDriver class contains all generic functions/ customized functions like clicking on an element, verifying element or text, Waitforelementpresent, mousehover and rightclick etc.
For Example we can write customized commands scrollByVisibilityOfElement and click shown below:

Action Driver Class

Browser Driver Folder

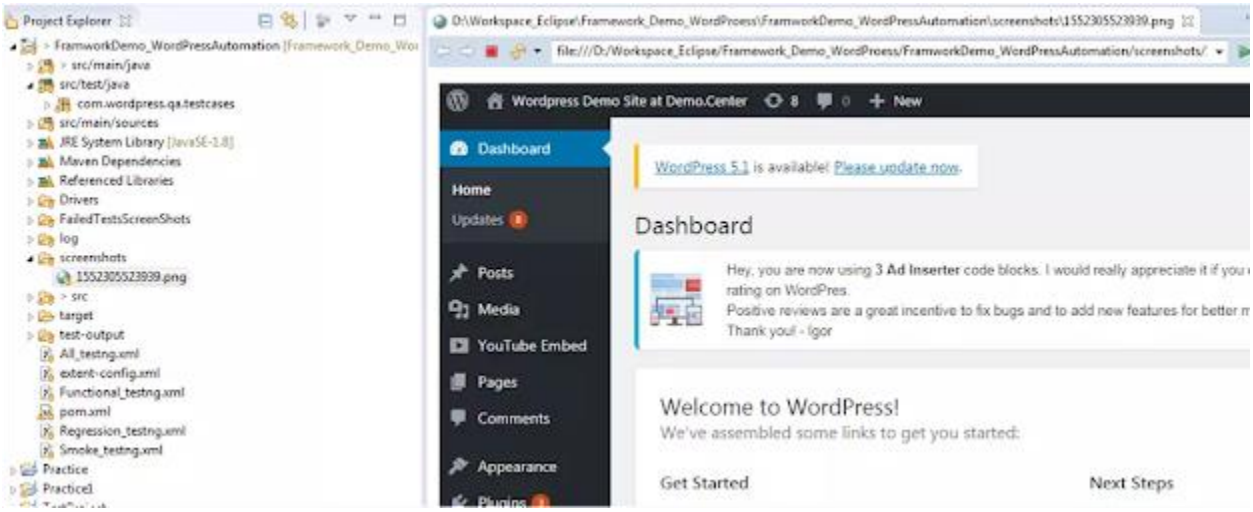
This folder contains IEDriverServer.exe , chromedriver.exe and geckodriver.exe.



Driver Folder

Screen Shots Folder

Screenshots will be captured and stored in a separate folder and also the screenshots of a failed test cases will be added in the extent reports.



Screen Shots

TestNG.xml

This testNG.xml suite file contains groups and classes of the modules.

TestNG.xml

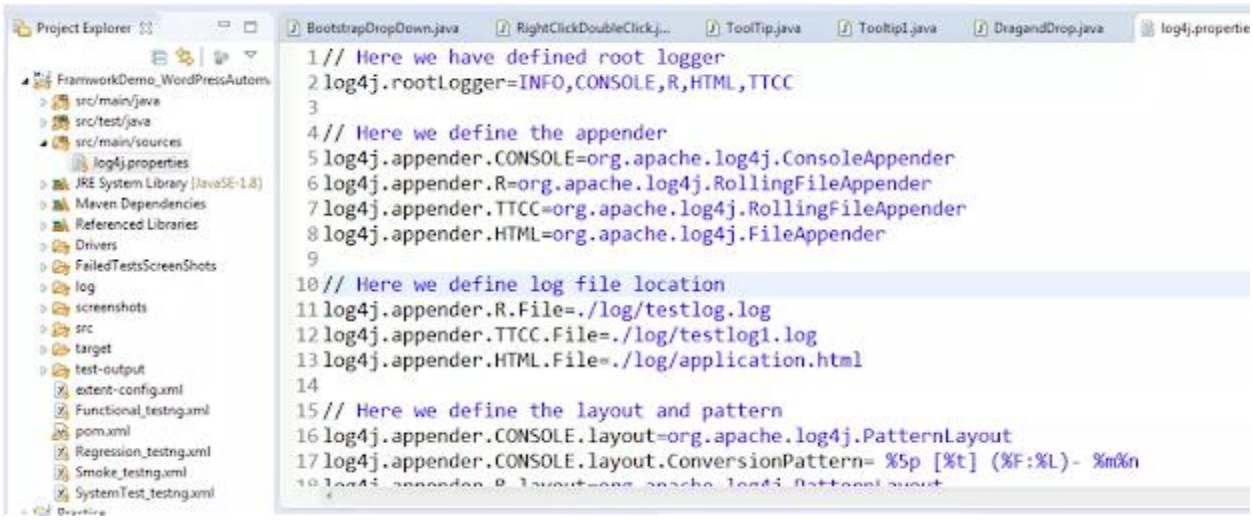
Reports

Extent Reports is being used for reporting purpose.

Extent Report

Logs – log4j

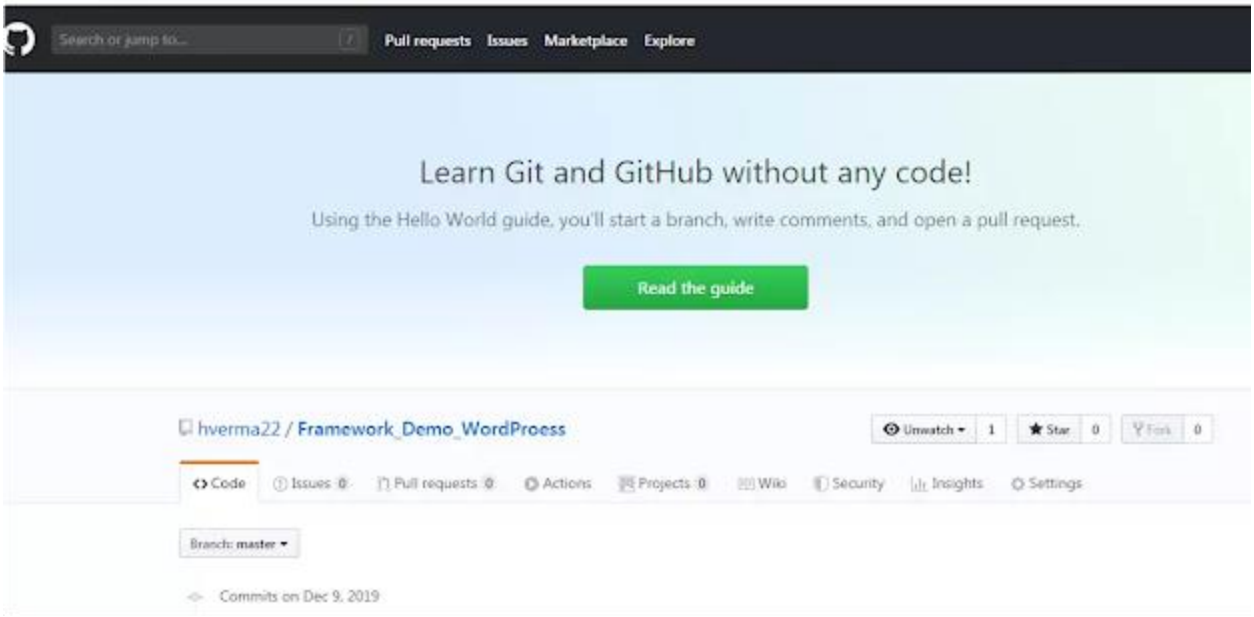
Grants a complete understanding of test suites execution. Logs are an exceptional assistant in debugging the program execution issues and failures.



log4j

Version Control tool

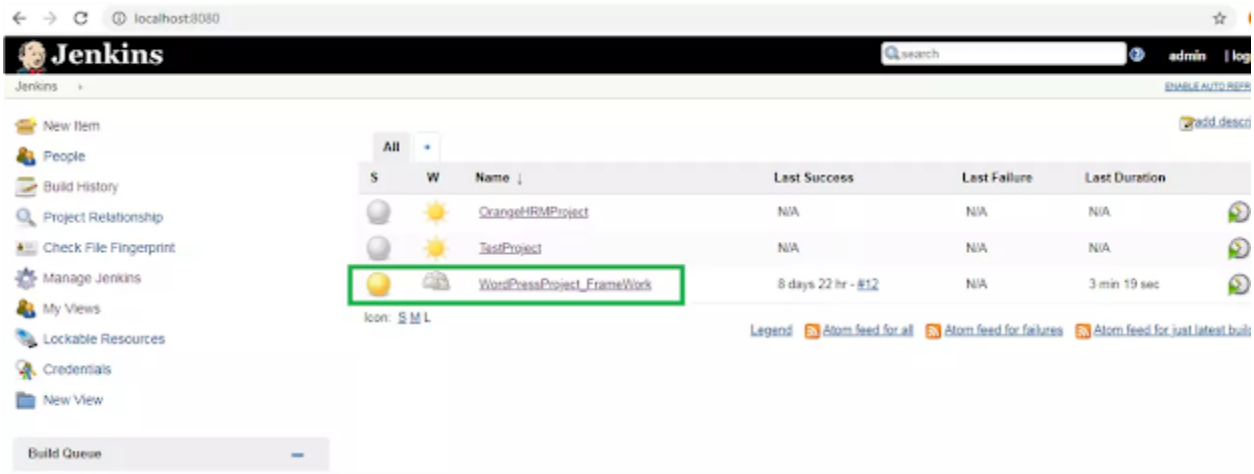
GitHub is a web-based service for version control using Git.



GitHub

Selenium Framework Integration with CI Tool

Jenkins is used as CI tool



Jenkins

Steps to Create Framework from Scratch

You can follow below Steps:

- Create Maven Project
- Update pom.xml
- Create Page Objects using page classes
- Create Basic Test Case
- Add logs to test case
- Read common values from properties file
- Run test cases on desired browser
- Add extent report
- Create data driven test case
- Adding new test cases.

How to Explain Framework?

1. **Framework Overview** -- What kind of Framework are you using?
We are using Page Object with Page Factory framework with functional/structural implementation.
Make sure you know what is functional/structural implementation in this framework.
2. **High Level overview of your framework** -- Different Components with Architecture.
Language - We use java because it is known to most people when we started automation.
Build Tool - We are using standardized maven project for build, execution & dependency management.
Data Driven - For handling data driven cases we are passing data using java properties file/xls file /csv file.
Make sure you know about libraries like openCSV,JXL/APACHE POI/Java Properties class
Testing Framework - For ordering tests we are using testng framework.
Configuration management - Git and Github - We check in our code into client repository using a version controlling tool git bash on windows system.
3. Make sure you know what is functional/structural implementation in this framework.
We have maintained a page class for every page in our application and a page test class to maintain test for that pages.
e.g. Product listing page,Add to cart page,Payment page,Invoice generation page.
4. We have maintained separate package for page and page test e.g.
com.companyName.page1 com.companyName.pageTest1
Maintaining different packages is always a good practice to follow.
5. We also have a base page class for common functions use by all the pages.
Make sure you know why we have Base Page class in page object.
6. **Logging Mechanism** - We are using log4j library to maintain logging of our project. We are using all kinds of logging statements like
INFO,DEBUG,ERROR etc. We have maintained a separate class for it in com.companyName.main package
7. **Reporting** - We are using Extent Report for reporting purpose. It is a third party report and it is easily available at maven central repo.

We are using maven postman plugin / JAVA API to send generated extent reports as an attachment to client Distribution list.

Implementation of OOP Concept in Selenium Framework

INTERFACE:

An interface in the Java programming language is an abstract type that is used to specify a behavior that classes must implement. An interface also contain methods and variables just like the class but the methods declared in interface are by default abstract.
To understand this the very basic statement we write in Selenium
WebDriver driver = new Chromedriver();

In this Case WebDriver itself is an Interface. So based on this statement, WebDriver driver = new Chromedriver(); we are initializing chrome browser using Selenium WebDriver. It means we are creating a reference variable (driver) of the interface (WebDriver) and creating an Object. Here WebDriver is an Interface as mentioned earlier and Chromedriver is a class.

ABSTRACTION:

Abstraction is a process of hiding the implementation details from the user and showing only relevant details to them. It also helps to reduce programming complexity and effort.

In our Automation Framework whenever we Use Page object Model, we write all the locator in page class and use this locator in our test it means we are hiding our implementation from the user this is the simple example of using abstraction in framework.

INHERITANCE:

The process by which one class acquires the properties (instance variables) and functionalities of another class is called inheritance.

When We create a Base Class in our automation Framework to initialize WebDriver interface, waits,loggers,reports etc. and when we extend this Base Class in other classes such as Tests and Utility Class. In this case extending one class into other class is example of implementing Inheritance.

ENCAPSULATION:

Encapsulation is a mechanism of wrapping data (variables) and code together as a single unit. All the classes which we write in our automation framework are an example of Encapsulation.

For e.g In Page object model classes, in which we declare the WebElement locator using @FindBy and initialization of this data members will be done using Constructor to utilize those in test methods

METHOD OVERLOADING:

If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.

In Implicit wait when we use different time stamps such as SECONDS, MINUTES, HOURS etc is one of the possible example of method overloading..

METHOD OVERRIDING:

If subclass or child class has the same method as declared in the parent class, it is known as method overriding in Java.

Whenever we use a method which was already implemented/written in another class by changing its parameters this is the example of method overriding.

Basic Selenium Frame Work