**CHAPTER - 1**

**INTRODUCTION TO AUTOMATION TESTING**

**What is Automation Testing?**

Automation testing or Test Automation is the process of running tests automatically, managing test data, and utilizing results to improve software quality.

It is the process of converting manual test cases into automation test scripts using automation testing tools and with the help of programming or scripting language.

**Advantages of Automation Testing:**

1. Faster feedback cycle
2. Reduce business expenses
3. Higher test coverage
4. Reusability of test suite
5. Improved accuracy
6. Quickly determine the stability of the build
7. Eliminates human error
8. Earlier detection of error
9. Data driven testing
10. Maximize ROI (Return On Investment)

**Note:** ROI is a metric that provides a numerical representation of the return you derive by incorporating automation testing into your **QA process**.

**Disadvantages of Automation Testing:**

1. Complexity: Automated tests can take longer to develop than manual tests
2. **High initial costs**
3. **It needs to be rewritten for every new environment**
4. **Generates false positives and negatives**
5. **Cannot be used on GUI elements (e.g., graphics, sound files)**
6. **Difficult to design tests that are both reliable and maintainable**

**When to start with Automation Testing?**

* When the build reaches certain level of functional stability
* When the resources (Manual test cases, automation tools, etc) are ready
* When the testing has more repetitive tasks
* When the testing has more regression cycles
* When you need to run multiple tests at once
* When we have long term and complex project

**Types of Automation Testing Tools:**

1. Functional Automation testing Tools
2. Non-Functional Automation Testing Tools

**Functional Automation Testing Tools:** The tools that are used to automate functional, integration, system and regression test cases are called Functional Automation Testing Tools.

Ex: Selenium, UFT(Unit Functional Test), SoapUI, QTP(Quick Test Professional), RFT(Rational Functional Tester), Winium, Appium, Test Complete etc

**Non-Functional Automation Testing Tools:** The tools that are used to automate non-functional test cases such as performance, security, usability, compatibility testing are called as Non-Functional Automation Testing Tools.

Ex: Apache JMeter, Neo Load, App Load, Web Load, Load Runner, Webserver Stress Tool, vPerformer, Forecast etc.

**INTRODUCTION TO SELENIUM**

Selenium is an open-source functional automation testing tool that enables and supports the automation of web browsers.

Selenium is not just a tool or an API, it is a suite comprising of several components.

**Advantages of Selenium:**

* It is an open-source tool, it is available for free.
* It is an open-source tool, so integrating selenium with any third-party tool is easy.
* Selenium is developed using java, like java it is platform independent.
* Selenium supports multiple programming languages like java, python, C-Sharp, Ruby, Perl etc.
* Selenium supports multiple browsers like chrome, firefox, edge, opera, safari etc. Browser compatibility testing is easy using selenium.
* Selenium supports multiple platforms such as Microsoft Windows, macOS and Linux. System compatibility testing is easy using selenium.

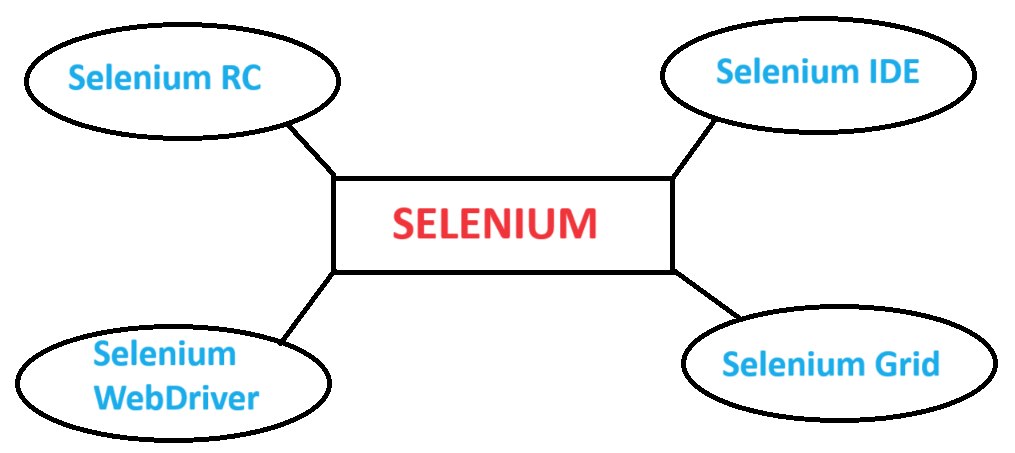
**Drawbacks of Selenium:**

* Selenium tests web applications only.
* Requires knowledge of programming language.
* No built-in reporting and test management facility, it has to be integrated with tools like TestNG to facilitate test management and reporting.

**Flavours of Selenium/ Selenium Suite Components:**

Selenium suite has four major components.

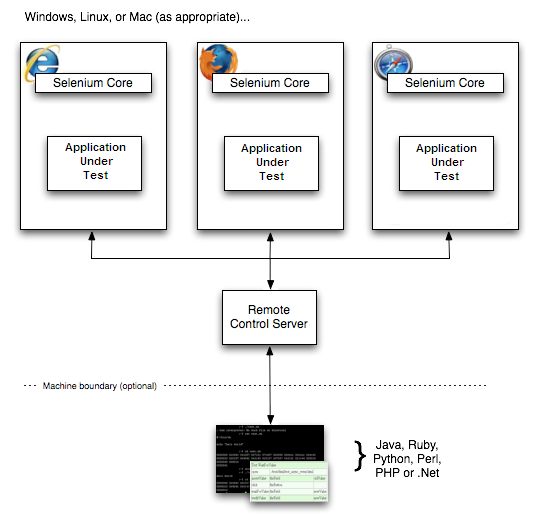
1. Selenium RC
2. Selenium IDE
3. Selenium WebDriver
4. Selenium Grid



**Selenium RC (Remote Control):**

* It is a testing framework that enables test engineer to write the test scripts in any programming language in order to automate UI tests for web applications against any HTTP website
* Selenium RC comprises of two parts:

1. Client libraries
2. Remote Control Server

­­

* RC Server acts as a mediator between the browser and the client.
* When a test script is executed, RC Server injects Javascript commands known as Selenium Core into the browser.
* Once the Selenium Core program is injected, it receives instructions from RC Server and executes these instructions as Javascript commands.
* The web browser executes all the commands given by Selenium Core and returns the test summary to the server.

**Limitations of Selenium RC:**

* Complicated Architecture
* Execution of test scripts is time consuming as Selenium RC uses Javascript commands
* No support for HTTPS protocol

**Selenium IDE (Integrated Development environment):**

* It is an open-source web automation testing tool under the Selenium Suite.
* It is basically a plugin for Firefox, Chrome and Edge browsers.
* It does not require any programming logic to write test scripts rather you can simply record your interactions with the browser to create test cases.
* Using the playback option we can re-run the test cases.
* It uses Selenese commands to execute the test scripts.
* The recorded test scripts can also be exported to programming languages like C#, Java, Ruby or Python.
* It is generally used in debugging the test scripts and for quick bug reproductive tasks.
* It is also used in automation aided exploratory testing.

**Limitations of Selenium IDE:**

* Not suitable for testing extensive data
* Cannot handle the dynamic part of web-based applications
* Does not support capturing of screenshots on test failures
* No feature available for generating result reports

**Selenium WebDriver:**

* Selenium WebDriver is an enhanced version of Selenium RC.
* It is an API, commonly referred to as WebDriver which drives a browser natively either locally or on a remote machine using the Selenium server.
* It is a collection of language specific bindings to drive a browser.
* It is a web framework which is used for testing web-based applications.
* It is used to create robust, browser-based regression automation suites and tests.
* It also allows you to perform [cross browser compatibility testing](https://www.browserstack.com/guide/cross-browser-compatibility-testing-beyond-chrome).
* It supports multiple programming languages, browsers and platforms.
* Selenium WebDriver completes the execution of test scripts faster when compared to other tools.
* More Concise API (Application Programming interface) than Selenium RC.

**Limitations of Selenium WebDriver:**

* Support for new browsers is not readily available when compared to Selenium RC.
* For the automatic generation of test results, it doesn’t have a built-in command.

**Selenium Grid:**

* It is the collection of Selenium WebDriver, Selenium RC and Grid Server libraries.
* It allows the execution of WebDriver scripts on remote machines by routing commands sent by the client to remote browser instances.
* It is used for Remote Executions (Execution of test scripts on remote devices like devices on same network or cloud servers).
* It is used for concurrent execution of test cases on different browsers, machines and operating systems simultaneously.
* This tool makes cross-browser and cross-platform compatibility testing very easy.
* It reduces the time needed to execute a test suite.

**Besides the above major components, selenium community has:**

* **For mobile based applications:**

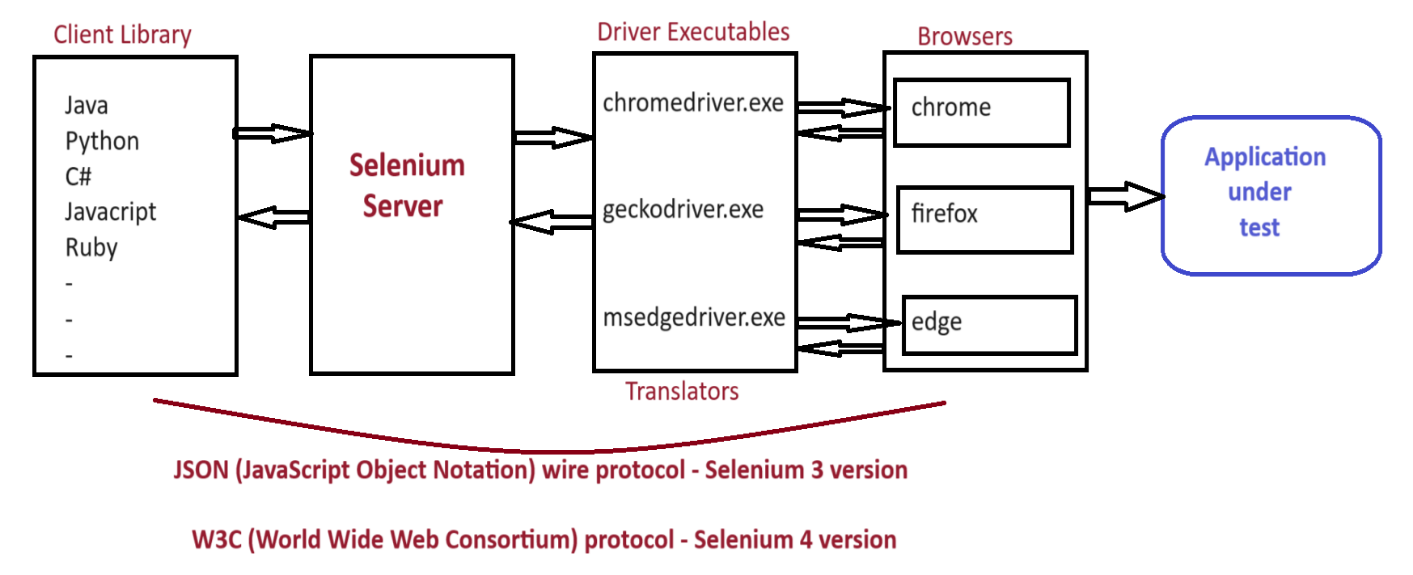
1. **Selendroid:** used to test mobile applications for Android.
2. **Appium:** used to test mobile applications for both Android and iOS.

* **Winium:** for testing and automating desktop applications on Windows.

**SELENIUM ARCHITECTURE**

Selenium architecture contains four major components:

* **Client Library:** This component provides language-specific bindings or APIs ([Java](https://www.geeksforgeeks.org/java/), [Python](https://www.geeksforgeeks.org/python-programming-language/), [Ruby](https://www.geeksforgeeks.org/ruby-programming-language/), etc. ) that allow users to write test scripts and interact with the WebDriver.
* **Selenium Server:** This component receives request from the client (user written test script) and communicates with specified driver executables.
* **Driver Executables:** This component acts as a translator between Selenium server and actual browser. They establish a communication channel between the WebDriver and the actual web browsers.
* **Browser:** It is the place where actual test execution takes place. The WebDriver interacts with these browsers through their respective browser drivers to perform actions.



* The communication between Client, Selenium server, Driver executables and Browsers happens via **JSON wire protocol in Selenium 3 version** and **W3C protocol in Selenium 4 version**.
* JSON (JavaScript Object Notation) wire protocol provides a transport mechanism for transferring data between client, server and browsers. It defines a set of commands and responses in JSON format exchanged over HTTP requests.
* W3C (World Wide Web Consortium) protocol provides a standard way to make the communication easy and direct between the client libraries and the browser drivers. Improved communication led to more stability in Selenium 4 version.

**Selenium WebDriver Installation Steps**

**Selenium Server Download:**

* Open the browser and type selenium.dev as URL
* Go to Downloads
* Scroll till “Selenium Grid” and click on latest stable version of selenium (currently 4.21.0)

**Chromedriver Executable Download:**

* First check the browser version available in your computer
* Open chrome browser
* Click on the on top right corner of the browser
* Go to “Help” and click on “About Google Chrome”
* Note the version
* Open the browser and search for chromedriver download
* Click on [Downloads | ChromeDriver - Chrome for Developers link](https://developer.chrome.com/docs/chromedriver/downloads)
* Click on  [the Chrome for Testing availability dashboard](https://googlechromelabs.github.io/chrome-for-testing/) link
* Click on “Stable” and copy the respective chromedriver link based on the OS of your computer
* Paste the copied link in the new tab and hit on Enter button in keyboard

**Java Project Creation:**

* Open the Eclipse IDE
* Click on “File” and then click on “New” and then click on “Java Project”
* Name the project and deselect “create module-info.java” checkbox
* Click on “Finish”
* Create “softwares” folder and “drivers” folder in the project

(Right click on the project => New => Folder => name it => click “Ok”)

**Adding Selenium Server to the Java Project:**

* Go to Downloads folder in the File explorer
* Copy the downloaded Selenium Server jar file
* Paste it on “softwares” folder in the Java project in Eclipse IDE
* Right click on the pasted jar file, go to “Build path” and click on “Add to build path”

(selenium server jar file will be added to “Referenced Libraries” in the project)

**Adding chromedriver.exe file to the Java Project:**

* Go to Downloads folder in the File Explorer
* Extract chromedriver.zip file
* Copy chromedriver.exe file from the extracted folder
* Paste it on “drivers” folder in the Java project in Eclipse IDE

(**Note:** If you can’t paste it directly on the “drivers” folder, right click on the “drivers” folder => go to “Properties” => click on  => open “drivers” folder and paste it and refresh the project in Eclipse IDE)

Test Scripts are written in the packages created in the “src” folder in the project.

**Script to Launch the Browser in Selenium WebDriver**

* **Launching chrome browser**

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

**public** **class** LaunchChromeBrowser {

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

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

}

}

* **Launching firefox browser**

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

**public** **class** LaunchChromeBrowser {

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

FirefoxDriver driver = **new** FirefoxDriver ();

}

}

* **Launching edge browser**

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

**public** **class** LaunchChromeBrowser {

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

EdgeDriver driver = **new** EdgeDriver ();

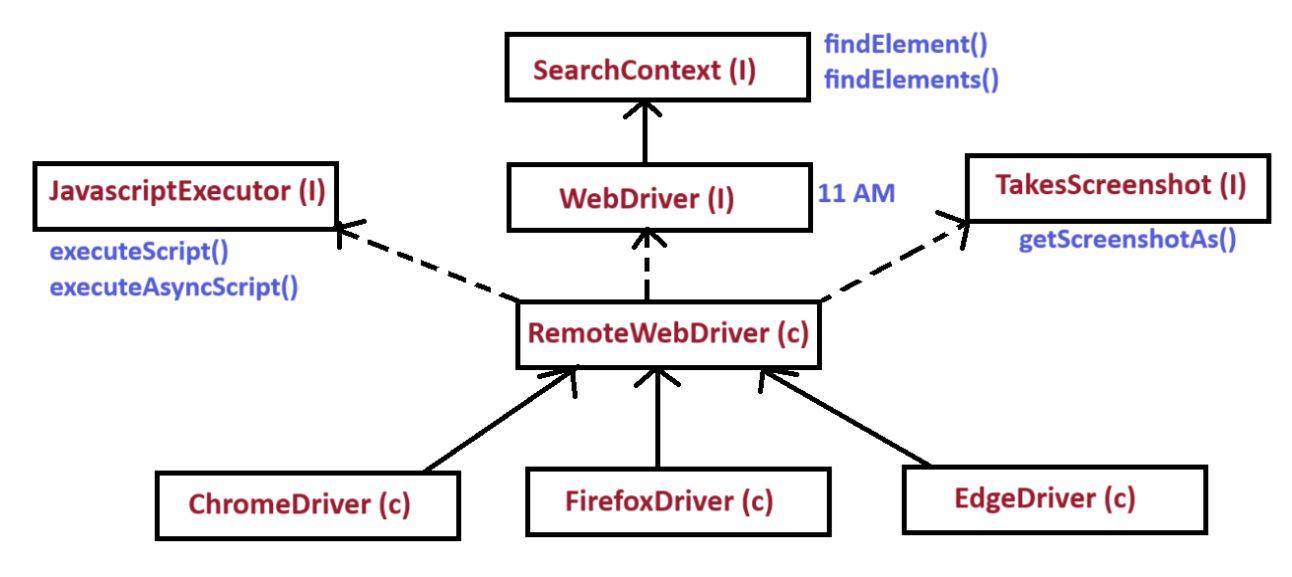
}

}

**To launch a web browser, one should just create an instance of respective browser specific class.**

**SELENIUM WEBDRIVER CLASS DIAGRAM / JAVA SELENIUM WEBDRIVER**

**ARCHITECTURE**



Selenium WebDriver class diagram is a hierarchy of interfaces and classes of Selenium WebDriver API.

**SearchContext:** It is the super most interface in Selenium WebDriver class diagram. It works like a search engine to find and fetch the elements from the web page. It has two abstract methods –

* findElement()
* findElements()

**WebDriver:** It is sub-interface of SearchContext interface. It provides driver related input actions to the web browser like launching and closing the browser, navigate to web application, maximize or minimize browser etc. It has eleven abstract methods and two abstract methods are inherited from SearchContext interface. Altogether it has thirteen abstract methods.

* get()
* getTitle()
* getCurrentUrl()
* getPageSource()
* getWindowHandle()
* getWindowHandles()
* manage()
* navigate()
* switchTo()
* close()
* quit()

**RemoteWebDriver:** It is an implementing class of WebDriver interface. It also implements two other interfaces –

* JavascriptExecutor interface
* TakesScreenshot interface

**JavascriptExecutor:** It is an interface in Selenium WebDriver which executes Javascript code. It has two abstract methods –

* executeScript()
* executeAsyncScript()

**TakesScreenshot:** It is an interface in Selenium WebDriver which captures the screenshot of a web page. It has one abstract method –

* getScreenshotAs()

**All the browser specific classes like ChromeDriver, FirefoxDriver, EdgeDriver etc., extend to RemoteWebDriver class.**

**We can launch the browser using the below statement:**

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

* With respect to selenium, the above statement launches empty chrome browser.
* With respect to java, the above statement is an upcasting statement and an example for Run Time Polymorphism.
* We create an instance of ChromeDriver class and upcast it to WebDriver interface so that we can launch different browsers using same driver reference.

**public** void launchBrowser(String browser) {

**switch** (browser) {

**case** "chrome":

driver = **new** ChromeDriver();

**break**;

**case** "firefox":

driver = **new** FirefoxDriver();

**break**;

**case** "edge":

driver = **new** EdgeDriver();

**break**;

**default**:

System.***out***.println("Invalid browser info");

}

}

* Here based on the object created respective browser is launched during run time. Hence it is an example for Run Time Polymorphism.

**WebDriver Methods**

WebDriver is an interface having eleven abstract methods

1. **get():**

* It is used to navigate to a web application and waits until the web page is loaded.
* It accepts URL in **String** datatype as argument.
* Return type is **void**.
* Usage –

**driver.get(“<url>”);**

1. **getTitle():**

* This method fetches the title of the current web page.
* It does not accept arguments.
* Return type is **String**.
* Usage –

**String title = driver.getTitle();**

1. **getCurrentUrl():**

* This method fetches the URL of the current web page.
* It does not accept any arguments.
* Return type is **String**.
* Usage –

**String url = driver.getCurrentUrl();**

1. **getPageSource():**

* This method fetches the source code of the current web page.
* It does not accept any arguments.
* Return type is **String**.
* Usage –

**String pageSource = driver.getPageSource();**

1. **close():**

* This method closes the current tab or window and exits webdriver process.
* It does not stop the server.
* It does not accept any arguments.
* Return type is **void**.
* Usage –

**driver.close();**

1. **quit():**

* This method closes all the tabs and windows and exits webdriver process.
* It stops the server.
* It does not accept any arguments.
* Return type is **void**.
* Usage –

**driver.quit();**

1. **manage():**

* This method manages all browser settings like
* Window related operations
* Timeouts related operations
* Cookies related operations
* It does not accept any arguments.
* Return type is **Options** (static interface in WebDriver).
* **Window related operations:**
* To maximize the window:

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

* To minimize the window:

**driver.manage().window().minimize();**

* To view the web page in full screen:

**driver.manage().window().fullscreen();**

* To set the size of the window:

**Dimension dimension = new Dimension(<width>, <height>);**

**driver.manage().window().setSize(dimension);**

* To set the position of the window:

**Point point = new Point(<x>, <y>);**

**driver.manage().window().setPosition(point);**

* To get the size of the window:

**Dimension dimension = driver.manage().window().getSize();**

**int height = dimension.getHeight();**

**int width = dimension.getWidth();**

* To get the position of the window:

**Point point = driver.manage().window().getPosition();**

**int x = point.getX();**

**int y = point.getY();**

* **Cookies related operations:**

Cookies are small pieces of text sent to your browser by a website you visit. They help that website remember information about your visit.

* To add a cookie:

**driver.manage().addCookie(Cookie cookie);**

* To delete a cookie:

**driver.manage().deleteCookie(Cookie cookie);**

* To delete all cookies:

**driver.manage().deleteAllCookies();**

* To get all cookies:

**Set<cookie> cookies = driver.manage().getCookies();**

1. **navigate():**

* This method is used to perform all browser navigations.
* It does not accept any arguments.
* Return type is **Navigation** (static interface).
* **Different browser navigations:**
* To navigate to before page ()

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

* To navigate to next page ()

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

* To refresh a web page ()

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

* To navigate to an application:

**driver.navigate().to(“<url>”);**

1. **findElement():**

* This method is used to fetch the first matching element from the web page.
* It accepts locator as an argument.
* Return type is WebElement (I).
* If the element is not found, it throws NoSuchElementException.
* **Usage:**

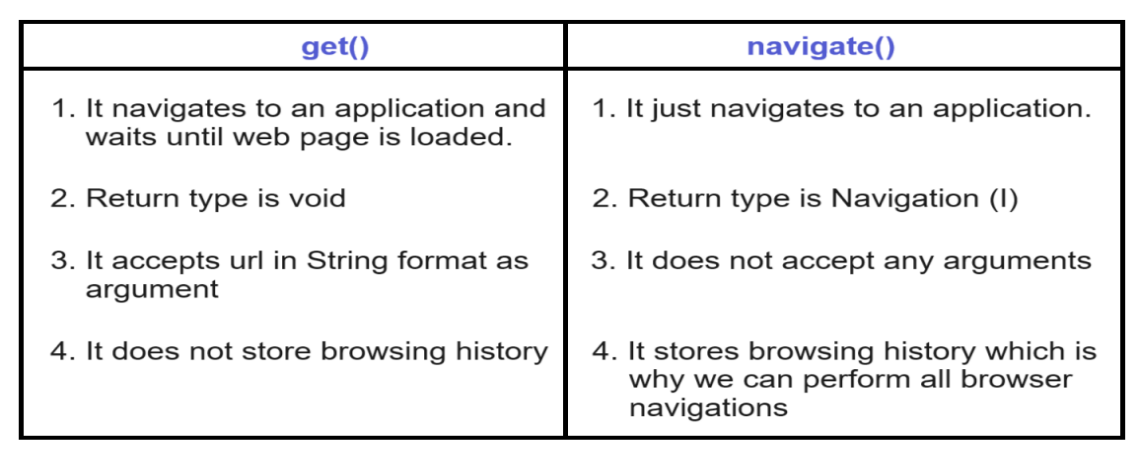
**WebElement element = driver.findElement(<locator>);**

1. **findElements():**

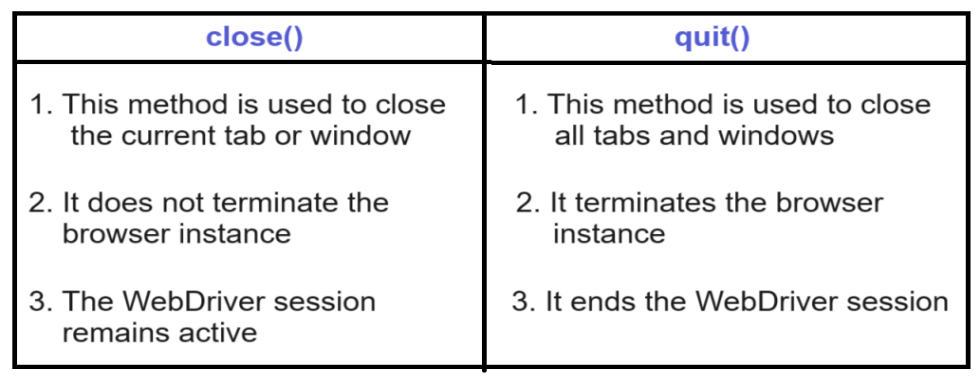
* This method is used to fetch all the matching elements from the web page.
* It accepts locator as an argument.
* Return type is List<WebElement>.
* If the elements are not found, it returns empty array list.
* **Usage:**

**List<WebElement> element = driver.findElements(<locator>);**

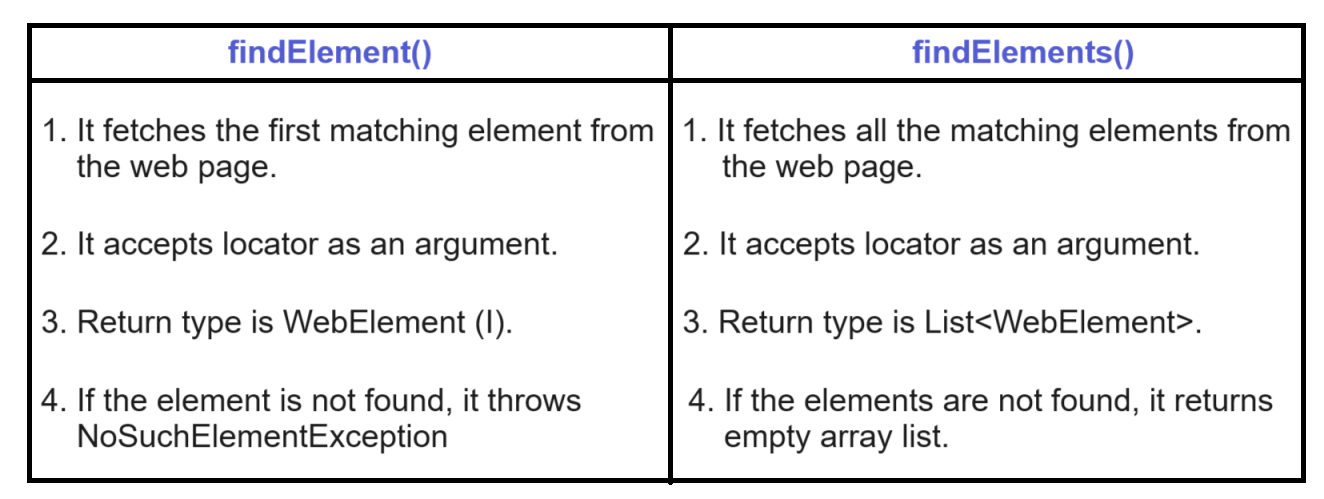
**Differences between get() and navigate():**

****

**Differences between close() and quit():**

****

**Differences between findElement() and findElements():**

****

**HTML (Hyper Text Markup Language)**

HTML is the standard markup language for creating Web pages. It describes the structure of a web page. It contains three elements:

1. Tags
2. Attributes
3. Text

HTML elements tell the browser how to display the content.

1. **Tags:** The first word enclosed within angular brackets (<>) is a tag.

Ex: <html> 🡪 first tag to start with web page design

<a> 🡪 for links

<h1>, <h2>, <h3>, <h4>, <h5>, <h6> 🡪 for page headers

<select> 🡪 for drop downs

<input> 🡪 for text fields

<button> 🡪 for buttons

<table> 🡪 for tables

<img> 🡪 for images

1. **Attributes:** The name-value pairs enclosed within angular brackets (<>) are called as attributes.

Ex:

<input **id=”username”** **name=”email”** /> 🡪 id and name are the attributes

<img **src="google.jpg" alt="Google" width="104" height="142"**>

Here src, alt, width and height are the attributes

1. **Text:** Anything which is not enclosed within angular brackets (<>) is called text.

Ex:

<a href="https://mail.google.com/mail/&amp;ogbl" target="\_top">**Gmail**</a>

**LOCATORS**

* Locators are the way to identify an HTML element on a web page.
* Locators are static methods of **By** class.
* **By** class is an abstract class in org.openqa.selenium package.

**Why Locators?**

To act on any element on the web page we should first locate the it. Locators help us to find the element on the web page.

**Types of Locators:**

1. **id**
2. **name**
3. **className**
4. **tagName**
5. **linkText**
6. **partialLinkText**
7. **cssSelector**
8. **xpath**
9. **id:** An ‘id’ locator uses id attribute to identify an element in the web page.

Usage:

**WebElement element = driver.findElement(By.id(“<id\_attribute\_value>”));**

1. **name:**  A ‘name’ locator uses name attribute to identify an element in the web page.

Usage:

**WebElement element = driver.findElement(By.name(“<name\_attribute\_value>”));**

1. **className:** A ‘className’ locator uses class attribute to identify an element in the web page.

Usage:

**WebElement element = driver.findElement(By.className(“<class\_attribute\_value>”));**

1. **tagName:** We can also locate an element using the tag name of the respective element node. Often there can be multiple elements having same tag name.

Usage:

**WebElement element = driver.findElement(By.tagName(“<tag\_name>”));**

**Or**

**List<WebElement> elements = driver.findElements(By.tagName(“<tag\_name>”));**

1. **linkText:** A ‘linkText’ locator uses the text on the hyperlinks to identify an element on the web page.

Usage:

**WebElement element = driver.findElement(By.linkText(“<text\_on \_the\_hyperlink>”));**

1. **partialLinkText:** A ‘partialLinkText’ locator uses partial/part of the text present on the hyperlinks to identify an element on the web page.

Usage:

**WebElement element = driver.findElement(By.partialLinkText(“<partial\_text>”));**

1. **cssSelector:** CSS (Cascading Style Sheet) is used to style the web elements.

CSS Selector is used to find or locate an element on the web page. It uses attributes to locate an element on the web page.

Usage:

**WebElement element = driver.findElement(By.cssSelector(“<css\_expression>”));**

**Syntax:**

* For id attribute:

**tagname#id\_attribute\_value**

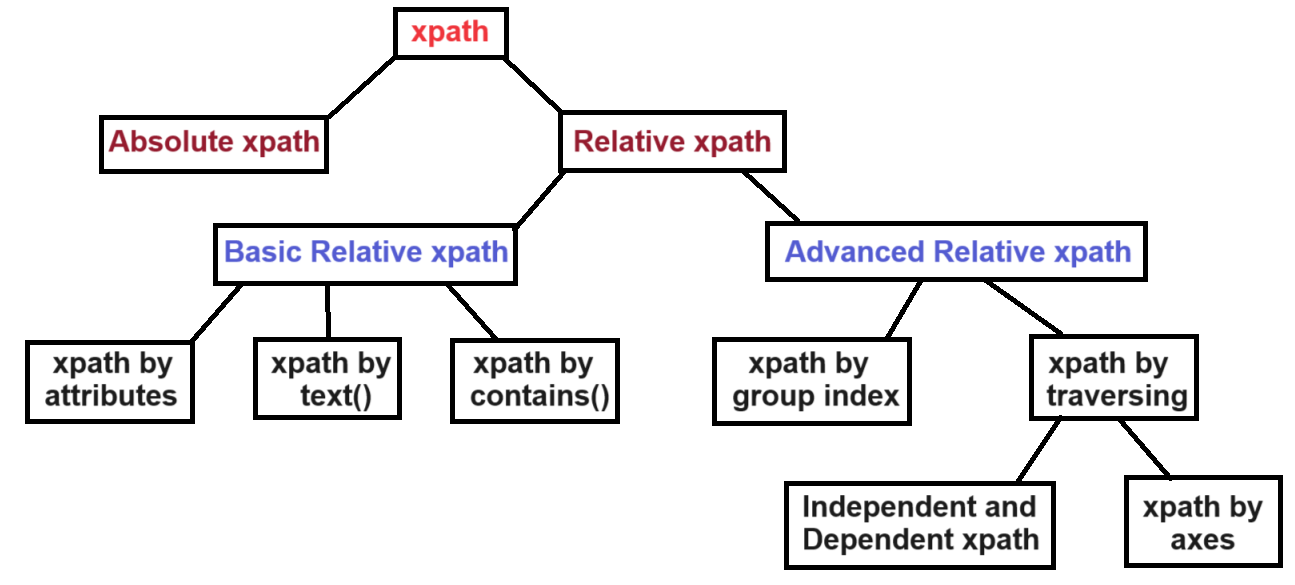
* For class attribute:

**tagname.class\_attribute\_value**

* Generic syntax:

**tagname[attribute\_name = ‘attribute\_value’]**

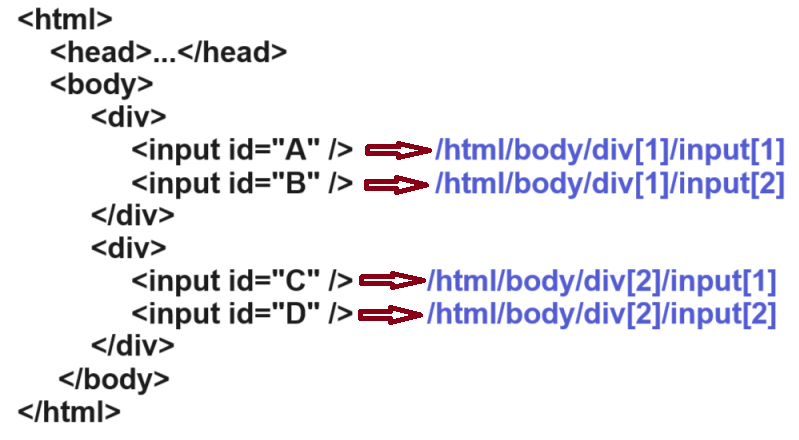
1. **xpath:** xpath stands for xml (Extensible Markup Language) path. It uses xml formats to locate an element on the web page.

****

Xpath uses either attributes or text or both to locate element. It helps in traversing through the HTML page. There are two types of xpath:

1. **Absolute xpath:** It is the complete path from the root of HTML tree till the element node. Traversing is done using ‘/’.

**Ex:**

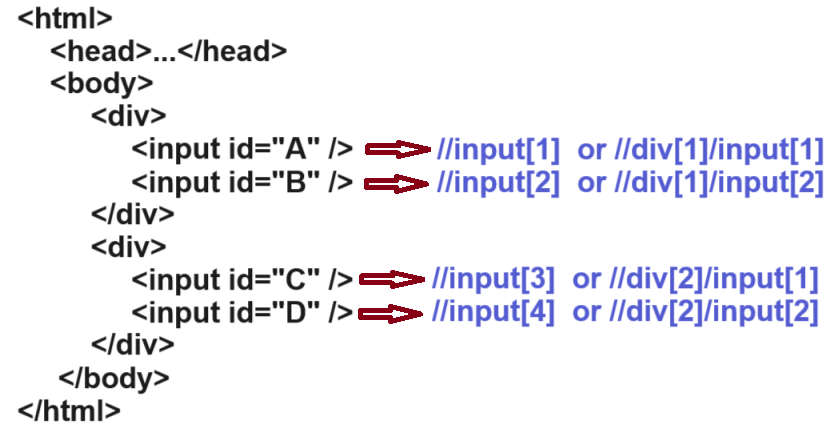
****

Drawbacks of Absolute xpath:

* Xpath is lengthy
* Time consuming
* Only forward traversing is possible
* In real time due to frequent requirement changes, GUI might change which might affect the absolute xpath.

1. **Relative xpath:** Xpath written directly to an element using ‘//’ is called Relative xpath. Using Relative xpath we can traverse from any point in the HTML tree to the required element. In Relative xpath backward traversing is also possible.

**Ex:**

****

**RELATIVE XPATH**

**Basic Relative Xpath:**

1. **Xpath by attributes:** Xpath written directly to locate an element using only attributes of the element.

**Syntax:**

**//tagname[@attribute\_name = ‘attribute\_value’]**

1. **Xpath by text():** Xpath written directly to locate an element using text on the element.

**Syntax:**

**//tagname[ text() = ‘text\_value’]**

**Or**

**//tagname[ . = ‘text\_value’]**

**.** 🡪 used for the text present on the same node or in the child nodes

1. **Xpath by contains():** Xpath written directly to locate an element using attributes or text on the element.

**Syntax:**

Using attribute 🡪 **//tagname[contains(@attribute\_name, ‘attribute\_value’)]**

Using text 🡪 **//tagname[contains(text(), ‘text\_value’)]**

Xpath by contains() handles:

* Lengthy attribute values or text values
* Partially dynamic elements

Elements whose values partially change such as software versions, build versions etc

Ex: Selenium 4.22.0 , Java JDK 22 etc

* Spaces in the text including non-breakable spaces.

Sometimes web designers or developers include spaces in the text of an element using a command ‘&nbsp’. These spaces are called as non-breakable spaces. The text having these spaces are not identified by xpath by text(). We should use xpath by contains() to handle these spaces.

**Advanced Relative Xpath:**

1. **Xpath by group index:** Whenever we have multiple matching elements, to fetch specific element we use position of that particular element. This is called as xpath by group index.

**Syntax:**

**(relative\_xpath\_expression)[position\_value]**

1. **Xpath by traversing:**

Traversing means moving from one element node to another in HTML.

Xpath by traversing is generally used to locate dynamic elements in a web page.

Steps to locate dynamic elements:

* Identify static element and write xpath expression to it
* Identify the common parent of static and dynamic elements and traverse to it
* Traverse to dynamic element and write tag name or tag name with attributes

We have two types:

1. **Independent and Dependent Xpath:**

Independent xpath 🡪 xpath of a static element

Dependent xpath 🡪 xpath of a dynamic element

Using independent and dependent xpaths we can locate a dynamic element.

1. **Forward Traversing:** Traversing from a node to its immediate child node is called forward traversing. It can be done using ‘/’.
2. **Backward Traversing:** Traversing from a node to its immediate parent node is called backward traversing. It can be done using ‘/..’
3. **Xpath by axes:**

Xpath by axes is used to optimize xpath expressions. We have different xpath axes.

* **parent:** Used to traverse from a node to its immediate parent node. It is similar to backward traversing.

**Syntax:**

**/parent::tagname**

* **child:** Used to traverse from a node to its immediate child node. It is similar to forward traversing.

**Syntax:**

**/child::tagname**

* **ancestor:** Used to traverse from a node to any parent node including immediate parent.

**Syntax:**

**/ancestor::tagname**

* **descendant:** Used to traverse from a node to any child node including immediate child.

**Syntax:**

**/descendant::tagname**

* **sibling functions:** Nodes which are at the same level having common parent are called sibling functions.
* **preceding-sibling**: Used to traverse from a node to the sibling nodes that are lying above the current node.

**Syntax:**

**/preceding-sibling::tagname**

* **following-sibling:** Used to traverse from a node to the sibling nodes that are lying above the current node.

**Syntax:**

**/following-sibling::tagname**

* **preceding:** Used to traverse from a node to the nodes that are at the same level, lying above the current node having uncommon parent.

**Syntax:**

**/preceding::tagname**

* **following:** Used to traverse from a node to the nodes that are lying below the current node having uncommon parent.

**Syntax:**

**/following::tagname**

**Xpath functions:**

1. **last():** When there are **n** number of child tags under a parent tag, then to locate the last child tag, we can use last() XPath Function. It is basically an index of the last element in the group of similar elements.

**Syntax: (xpath\_expression)[last()]**

1. **normalize-space():** This function strips leading and trailing white-space including non-breakable spaces from a string, replaces sequences of whitespace characters by a single space, and returns the resulting string.

**Syntax: //tagname[normalize-space(@attribute\_name) = ‘attribute\_value’]**

**//tagname[normalize-space(text()) = ‘text\_value’]**

1. **name():** This function handles “svg” tags. “svg” stands for scalable vector graphics.

These tags are used for graphic designing of logos and widgets of the webpages.

These tags are not identified directly. Hence we use name() to identify these tags.

**Syntax: //\*[name() = ‘svg’][@attribute\_name = ‘attribute\_value’]**

**//\*[name() = ‘svg’ and @attribute\_name = ‘attribute\_value’]**

1. **starts-with():** This function can be used to locate an element/ elements with the beginning portion of the text or beginning portion of the attribute value.

**Syntax: //tagname[starts-with(@attribute\_name, ‘attribute\_value)]**

**//tagname[starts\_with(text(), ‘text\_value’)]**

**Xpath keywords:**

1. **and:** This expression uses two conditions. Both conditions should be true for finding the element. It fails to find the element if any one condition is false.
2. **Or:** This expression uses two conditions, whether the first condition OR second condition should be true. It is also applicable if any one of the conditions is true, or maybe both. This means that any one condition should be true to find the element.

**SYNCHRONIZATION**

* Synchronization is the process of syncing the selenium script with application when interacting with the web elements on the web page.
* When we perform an action on the web page, it is expected that all components involved work together seamlessly. This collaborative process among components is referred to as synchronization.
* Synchronization ensures that the code and applications execute in more efficiently to carry out the desired operation.

**Types of wait statements:**

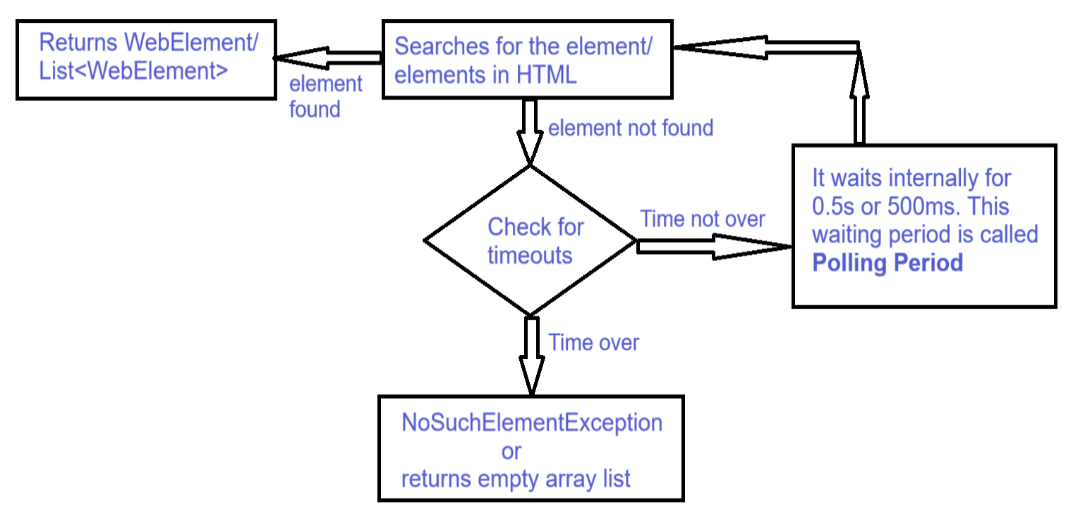
1. **Thread.sleep():**

* It is java wait statement which waits completely for specified amount of time.
* It accepts time in milliseconds in long datatype as an argument.
* Return type is void.
* It throws InterruptedException.
* Usage:

**Thread.sleep(long ms);**

1. **Implicitly wait:**

* It is selenium wait statement which synchronizes findElement() and findElements() methods only.
* It is a global wait setting **that applies to all elements in a Selenium script**.
* It waits a specified time before throwing an exception if the element is not found.The default implicitly wait value is 0.
* T**he implicit wait should be set right after initializing the**WebDriver**instance.**
* **WorkFlow:**

****

* Whenever implicitly wait is applied to the selenium script, it searches for the element / list of elements on the web page.
* If element / elements are found, it returns WebElement / List<WebElement> respectively based on the method used.
* If the element / elements are not found, it checks if the specified time is over or not.
* If the time is over, it throws NoSuchElementException or it returns empty array list respectively.
* If the time is not over, it waits internally for 0.5s or 500ms. This waiting period is called as Polling period.
* Once the polling period is done, it searches for the element / elements again.
* This process repeats until the element / elements are found or time is over.
* **Syntax:**

**For Selenium 3 version 🡪**

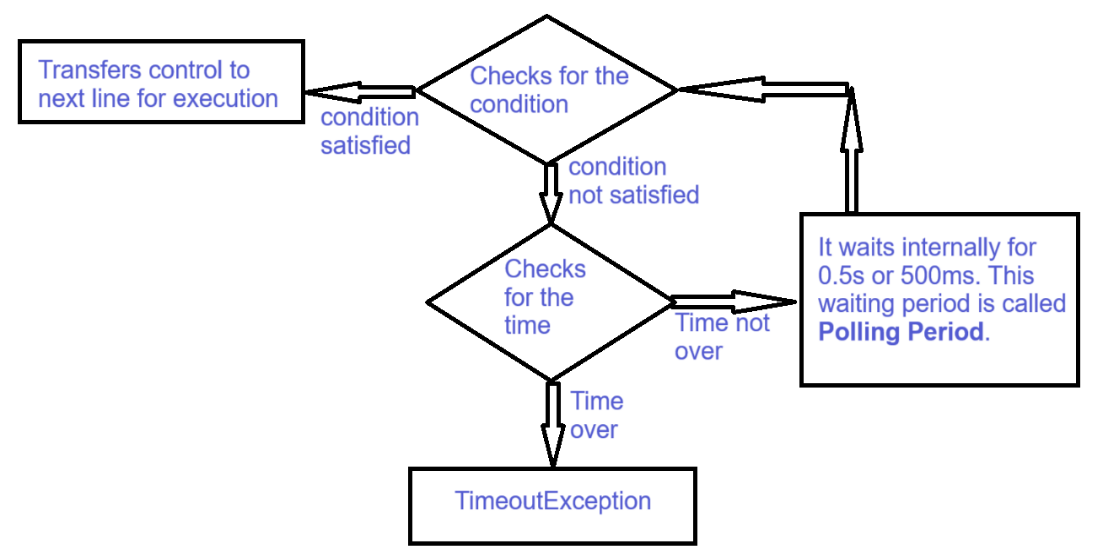
**driver.manage().timeouts().implicitlyWait(<time\_in\_sec>, TimeUnit.SECONDS);**

**For Selenium 4 version 🡪**

**driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(<time\_in\_sec>);**

1. **Explicitly wait:**

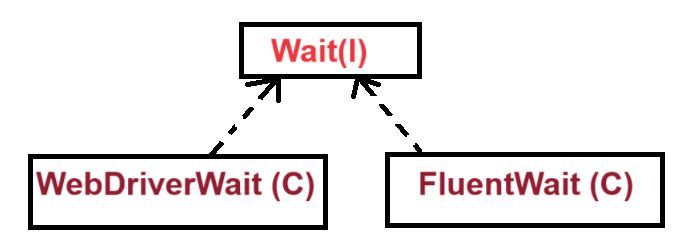
* It is selenium wait statement which synchronizes all the methods including findElement() and findElements() methods.
* It **is a more flexible wait that allows us to wait for a specific condition to be met before continuing test execution.**
* Unlike implicitly wait, explicitly wait should be given every time whenever synchronization is needed.
* **WorkFlow:**

****

* Whenever explicitly wait is applied to the selenium script, it checks for the condition.
* If the condition is satisfied, it transfers control to the next line for execution.
* If the condition is not satisfied, it checks if the specified time is over or not.
* If the time is over, it throws TimeoutException.
* If the time is not over, it waits internally for 0.5s or 500ms. This waiting period is called as Polling period.
* Once the polling period is done, it checks for the condition again.
* This process repeats until the condition is satisfied or time is over.
* Explicitly wait can be set using two classes:

1. **WebDriverWait**
2. **FluentWait**

* The above classes are the implementing classes of Wait interface.



* **WebDriverWait Syntax:**

**For Selenium 3 version 🡪**

**WebDriverWait wait = new WebDriverWait(driver, <time\_in\_sec>);**

**wait.until(ExpectedConditions.visibilityOf(<element>));**

**or**

**wait.until(ExpectedConditions.elementToBeClickable(<element>));**

**or**

**wait.until(ExpectedConditions.titleContains(<page\_title>));**

**For Selenium 4 version 🡪**

**WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(<time\_in\_sec>);**

**wait.until(ExpectedConditions.visibilityOf(<element>));**

**or**

**wait.until(ExpectedConditions.elementToBeClickable(<element>));**

**or**

**wait.until(ExpectedConditions.titleContains(<page\_title>));**

* **FluentWait:**
* It is another type of explicitly wait.
* It is used to customize polling period.
* Unlike WebDriverWait, FluentWait does not have predefined methods to provide condition. User should define a method to provide condition.
* It also ignores any exception that occurs before timeouts.

1. **Custom wait:**

* It is user defined wait statement.
* Wait statement can be defined using loops and exception handling
* Sample custom wait statement:

WebElement header = **null**;

**int** count = 0;

**while**(count < 20) {

**try** {

header = driver.findElement(<locator>));

**break**;

}**catch**(Exception e) {

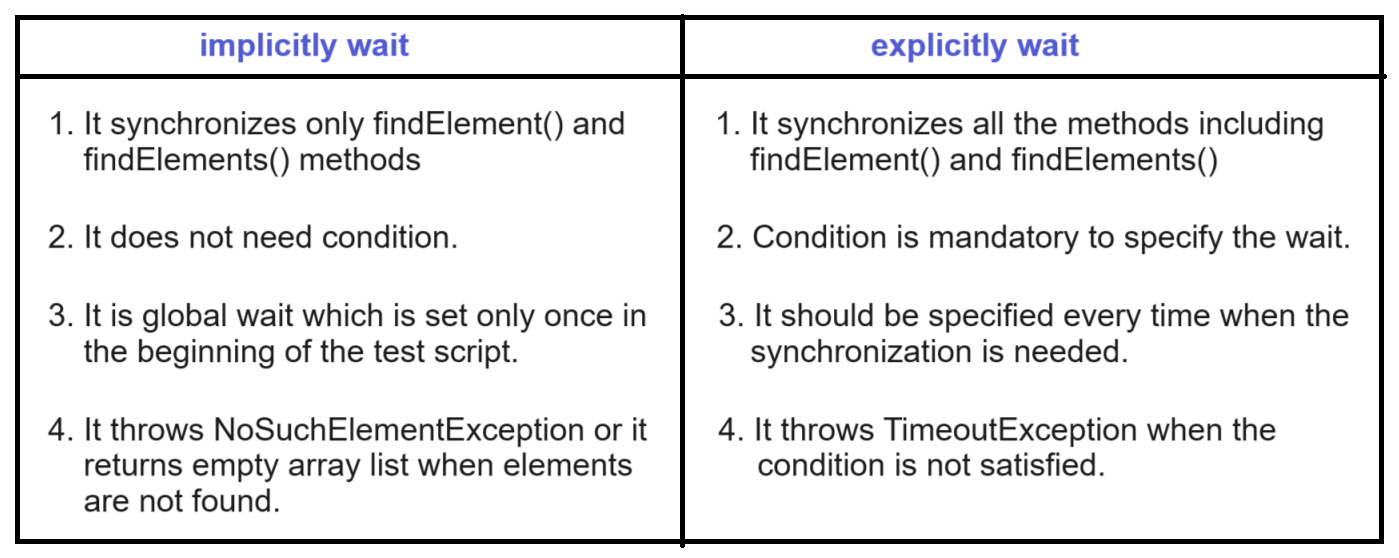
Thread.*sleep*(1000);

count++;

}

}

**Differences between implicitly wait and explicitly wait:**

****

**WEBELEMENT METHODS**

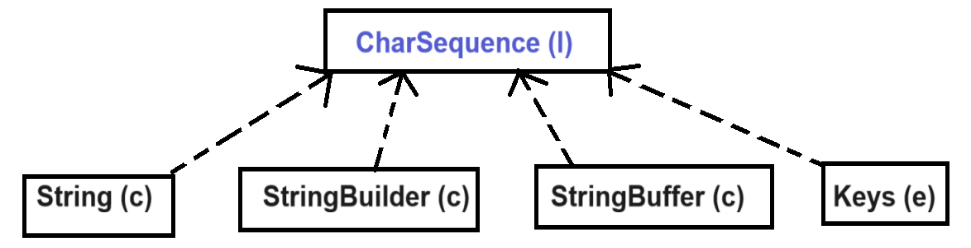
The elements which appear on the web page like text fields, drop downs, links, images etc., are called as web elements.

WebElement is an interface in org.openqa.selenium package which is used to perform different operations on web elements.

Abstract methods declared within WebElement interface are categorised into three sections.

1. Action methods
2. Getter methods
3. Validation methods
4. **Action methods:**
5. **sendKeys():**

* This method will simulate typing into the WebElement.
* It accepts CharSequence (I) type arguments.



* CharSequence is an interface in java. It is readable sequences of char values. The implementing classes of CharSequence are String, StringBuilder and StringBuffer, and Keys enum.
* Keys enum is present in org.openqa.selenium package, which is the representation of pressable keys which does not have text on it.
* Return type is void.
* If the data passed is null, this method will throw **IllegalArgumentException.**

1. **click():**

* This method clicks on any element on the web page.
* Return type is void.

1. **clear():**

* This method will reset the content on the web element.
* Return type is void.

1. **submit():**

* This method clicks on an element which is present within <form></form> and it should have type=”submit” attribute.
* Return type is void.

1. **Getter methods:**
2. **getText():**

* This method fetches the text on the web element, including sub elements.
* Return type is String.
* Usage:

**String text = element.getText();**

1. **getTagName():**

* This method fetches the tag name of the web element from the element node.
* Return type is String.
* Usage:

**String tagName = element.getTagName();**

1. **getAttribute():**

* This method fetches the attribute value of the specified attribute name from the element node.
* It accepts attribute name in the String data type as an argument.
* Return type is String.
* Usage:

**String attributeValue = element.getAttribute(“<attributeName>”);**

1. **getCssValue():**

* This method fetches the computed style or CSS (Cascading Style Sheet) properties such as color, font, background-color etc., of an element.
* It accepts CSS property key in String data type as an argument.
* Return type is String.
* Usage:

**String cssValue = element.getCssValue(“<css\_property\_key>”);**

1. **getLocation():**

* This method fetches the location of an element on the web page.
* Return type is Point class.
* Point class is present in org.openqa.selenium package. It has two non-static methods –

1. getX() – It fetches the x-coordinate of the element. It returns int.
2. getY() - It fetches the y-coordinate of the element. It returns int.

* Usage:

**Point point = element.getLocation();**

**int x = point.getX();**

**int y = point.getY();**

1. **getSize():**

* This method fetches the size of an element on the web page.
* Return type is Dimension class.
* Dimension class is present in org.openqa.selenium package. It has two non-static methods –

1. getHeight() – It fetches the height of the element. It returns int.
2. getWidth() - It fetches the width of the element. It returns int.

* Usage:

**Dimension dimension = element.getSize();**

**int height = dimension.getHeight();**

**int width = dimension.getWidth();**

1. **getRect():**

* This method fetches both the size and location of an element on the web page.
* Return type is Rectangle class.
* Rectangle class is present in org.openqa.selenium package. It has four non-static methods –

1. getX() – It fetches the x-coordinate of the element. It returns int.
2. getY() - It fetches the y-coordinate of the element. It returns int.
3. getHeight() – It fetches the height of the element. It returns int.
4. getWidth() - It fetches the width of the element. It returns int.

* Usage:

**Rectangle rect = element.getRect();**

**int height = rect.getHeight();**

**int width = rect.getWidth();**

**int x = rect.getX();**

**int y = rect.getY();**

1. **getScreenshotAs():**

* This method fetches the screenshot of an element from the web page.
* Steps to capture element screenshot:

1. **File temp = element.getScreenshotAs(OutputType.FILE);**
2. **File target = new File(“<filePath>”);**
3. **FileUtils.copyFile(temp, target); 🡪 IOException**

To copy the file to permanent memory we use external libraries called apache commons io. These libraries are used for sharing or copying files or directories.

1. **Validation methods:**
2. **isEnabled():**

* This method checks if an element is enabled to perform any operation on it.
* Return type is Boolean.

1. **isDisplayed():**

* This method checks if an element is displayed on the web page.
* Return type is Boolean.

1. **isSelected():**

* This method checks if an element is selected on the web page. Generally we use this method for checkboxes or radio buttons.
* Return type is Boolean.

**CHAPTER – 2 : HANDLING WEB ELEMENTS**

**Auto Suggestions:**

Auto suggestions or dynamic drop downs are the web elements which try to predict the values of typed text based on the input value provided.

Auto suggestions can be handled by collecting all the suggestions and traversing through this list of the suggestions and performing necessary action.

We use **findElements()** method of WebDriver interface to store the list of auto suggestions.

Example Script:

**public** **class** Practice1 {

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

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

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

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

driver.manage().timeouts().implicitlyWait(Duration.*ofSeconds*(10));

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

List<WebElement> elements = driver.findElements(By.*xpath*("//li[contains(.,'selenium')]"));

**for** (WebElement e : elements) {

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

}

driver.quit();

}

}

**Actions class:**

The Action class in Selenium WebDriver is a utility class that enables advanced user interactions such as mouse and keyboard events.

By using the Actions class, you can interact with a website in the same way a user would, allowing you to test more complex interactions that can’t be accomplished with just simple method calls.

Actions class methods are broadly divided into two categories:

1) Mouse actions

2) Keyboard actions

1. **Mouse actions:**

* **click():** performs a single mouse click on the specified element.
* **clickAndHold():** holds down the left mouse button on the specified element.
* **contextClick():** performs a right-click on the specified element.
* **doubleClick():** performs a double-click on the specified element.
* **dragAndDrop():** performs a drag and drop operation between two elements.
* **release():** releases the left mouse button on the specified element.
* **moveToElement():** moves the mouse cursor to the middle of the specified element.

1. **Keyboard actions:**

* **sendKeys(CharSequence… keysToSend):** sends a series of key presses to the specified element.
* **keyDown(Keys theKey):** holds down the specified key.
* **keyUp( Keys theKey):** releases the specified key.

**Implementation of Action class methods:**

Step 1: Create an instance of Actions class

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

Step 2: Use the methods of Actions class to perform actions on the web page

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

**actions.doubleClick(element).perform();**

**actions.contextClick(element).perform();**

**actions.dragAndDrop(element, target).perform(); etc…**

**Note: perform()** method is mandatory method which actually performs specified actions on the web page.

**build()** method generates a composite action containing all actions so far, ready to be performed.

**Drop downs:**

Dropdowns are the web elements which when clicked on provides list of options on the website.

Dropdowns can be **static** and **dynamic**.

* **Static** dropdowns have fixed options. The static dropdowns can be **select** and **non-select** dropdowns.
* **Select** dropdowns have <select> tag in the element node. These dropdowns are of two types:

1. **Single select dropdown:**

Select dropdowns which allows user to select only one option at a time are called Single select dropdowns. Deselection is not possible in single select dropdown.

1. **Multi select dropdown:**

Select dropdowns which allows user to select more than one option at a time are called Multi select dropdowns. Deselection is possible in multi select dropdown.

* **Non-select** dropdowns do not have <select> tag in the element node.
* **Dynamic** dropdowns have options which are not fixed.

**Any kind of dropdown can be handled using *findElements()* method by storing all the options in a list and iterating through the list.**

**Select Dropdowns can be handled using *Select* class.**

**Select class:**

Select class is present in org.openqa.selenium.support.ui package. It is used to select or deselect an option in a dropdown. It has several non-static methods to handle select dropdowns.

**Implementation of Select class methods:**

1. **WebElement dropdown = driver.findElement(<locator>);**
2. **Select select = new Select(dropdown);**
3. **Different operations that can be performed are:**

* **To select an option from dropdown:**
* **select.selectByIndex(int index);**
* **select.selectByValue(String value);**
* **select.selectByVisibleText(String text);**

**(Return type of the above three methods is void)**

* **To verify if the dropdown is multi select**
* **select.isMultiple(); (return type is Boolean)**
* **To get all the options from the dropdown**
* **List<WebElement> options = select.getOptions();**
* **To get first selected option**
* **WebElement firstOption = select.getFirstSelectedOption();**
* **To get all selected options**
* **List<WebElement> selectedOptions = select.getAllSelectedOptions();**
* **To deselect an option from the dropdown**
* **select.deselectByIndex(int index);**
* **select.deselectByValue(String value);**
* **select.deselectByVisibleText(String text);**

**(Return type of the above three methods is void)**

* **To deselect all the options at once**
* **select.deselectAll(); (return type is void)**

**iFrames:**

 iFrame is a HTML document embedded inside an HTML document. iFrame is defined by an *<iframe></iframe>* tag in HTML.

**Identifying frames on the web page:**

* When you right click on a frame we can view an option having the text – “view frame source”
* When you inspect a frame you can see the following HTML code:

**<iframe id=”” name=”” . . . >**

**#document**

**<html>**

**<head>…</head>**

**<body>…</body>**

**</html>**

**</iframe>**

**Handling the frames on the web page:**

* To handle the frames the control should be shifted to the frame from main web page.
* To switch to the frame using index:

**driver.switchTo().frame(int index);**

* To switch to the frame using name or id attribute value of the iframe tag

**driver.switchTo().frame(String idOrName);**

* To switch to the frame using frame element reference

**driver.switchTo().frame(WebElement frameElement);**

* To switch the control back from frame

**driver.switchTo().defaultContent();**

**switchTo():**

* It is one of the WebDriver interfaces’ abstract methods.
* It is used to switch the control to the target location like frames or windows or pop ups.
* Return type is TargetLocator (static interface in WebDriver).
* TargetLocator interface is used to select a frame or window.

**frame():**

* It is an abstract method in TargetLocator interface.
* It selects a frame based on index or id or name or frame element reference.
* It is an overloaded method. It accepts one of the three arguments:
* frame(int index)
* frame(String idOrName)
* frame(WebElement frameElement)
* Return type is WebDriver.
* It throws NoSuchFrameException if frame is not found.

**defaultContent():**

* It is an abstract method in TargetLocator interface.
* It selects either a first frame or the main document on the web page.
* Return type is WebDriver.

**Screenshot:**

* Screenshot of a web page can be captured with the help of TakesScreenshot interface and apache commons io libraries.
* Apache commons io libraries contains utility classes which are used to copy or share files or directories.

**Steps to capture screenshot:**

1. Typecast WebDriver reference to TakesScreenshot interface reference

**TakesScreenshot ts = (TakesScreenshot) driver;**

1. Call getScreenshotAs() method to create image file

**File temp = ts.getScreenshotAs(OutputType.FILE);**

1. Create a permanent location to store the captured image

**File target = new File(“<permanent\_file\_path>”);**

1. Copy the image from temporary memory to the permanent file

**FileUtils.copyFile(temp, target);**

FileUtils class is present in org.apache.commons.io package. copyFile() is a static method of FileUtils class. The above statement throws IOException.

**JavascriptExecutor (I):**

* It is an interface in Selenium WebDriver which executes Javascript code on selenium.
* It has two abstract methods –
* **executeScript():** Executes JavaScript in the context of the currently selected frame or window. Return type is Object class.
* **executeAsyncScript():** Executes JavaScript in the context of the currently selected frame or window. Return type is Object class.

**Implementation of JavascriptExecutor:**

1. **JavascriptExecutor js = (JavascriptExecutor) driver;**
2. **js.executeScript(“<Javascript\_code>”, args);**

**Actions that can be performed using JavascriptExecutor:**

* Handles scroll bar
* **js.executeScript(“window.scrollTo(<x>, <y>)”);**

The scrollTo() method scrolls the document to specified coordinates.

* **js.executeScript(“window.scrollBy(<x>, <y>)”);**

The scrollBy() method scrolls the document by the specified number of pixels.

* **js.executeScript(“arguments[0].scrollIntoView(true)”, element);**

The scrollIntoView() method scrolls an element into the visible area of the browser window.

* Navigates to an application

**js.executeScript(“window.location=arguments[0]”, <url>);**

* Fetches title and URL of a web page
* To fetch title

**js.executeScript(“return document.title”);**

* To fetch URL

**js.executeScript(“return document.URL”);**

* Refresh a web page

**js.executeScript(“history.go(0)”);**

* Performing type action

**js.executeScript(“arguments[0].value=arguments[1]”, element, “<data>”);**

* Performing click action

**js.executeScript(“arguments[0].click()”, element);**

* Handles disabled elements

**js.executeScript("arguments[0].removeAttribute('disabled','disabled')", element);**

**Pop ups:**

Popup is a window that displays or pops up on the screen due to some activity.

There are different kinds of pop ups.

1. **Javascript alerts, prompts and confirmations pop up:**

* **Javascript alerts** shows a custom message and a single button which dismisses the alert, labelled as OK. It can also be dismissed in most browsers by pressing the close button.
* **Javascript confirmation pop up** is similar to an alert, except the user can also choose to cancel the message.
* **Javascript prompt pop** up is similar to confirm pop up, except they also include a text input.
* Javascript pop ups are neither inspectable nor movable.
* These pop ups can be handled using **Alert interface**.
* Alert interface is present in org.openqa.selenium package.

**Steps to handle Javascript pop up:**

1. **Alert a = driver.switchTo().alert();**
2. To click OK 🡪 **a.accept();**

To click on Cancel 🡪 **a.dismiss();**

To fetch the text from pop up 🡪 **a.getText(); (returns String)**

To pass text to pop up 🡪 **a.sendKeys(“<text>”);**

1. **Hidden Division or Calendar pop up:**

* These pop ups are inspectable but not movable.
* Since these pop ups are inspectable they can be handled using findElement() method.

1. **Authentication pop up:**

* Authentication popups often appear when accessing secure areas of a web application, requiring valid credentials to proceed.
* These popups can disrupt the flow of automated tests if not properly managed, leading to incomplete or failed test executions.
* These pop ups are neither inspectable nor movable.
* The simplest way to handle this pop up is by directly passing username and password in the URL separated by colon.
* Syntax:

**https://<username>:<password>@<domain>**

**Ex:** [**https://admin:admin@the-internet.herokuapp.com/basic\_auth**](https://admin:admin@the-internet.herokuapp.com/basic_auth)

1. **Window or Child Browser pop up:**

* Browser window popup is a window popup that suddenly pops up on after clicking on some link or on selecting an option.
* It just likes another web page.
* This is having its own title and url.
* These pop ups are inspectable and movable.
* These pop ups can be handled using **getWindowHandle()** and **getWindowHandles()** methods.
* **getWindowHandle():**
* It will get the handle of the page the webDriver is currently controlling.
* This handle is a unique identifier for the web page.
* This is different every time you open a page even if it is the same URL.
* Return type is String.
* **getWindowHandles():**
* It will get all the handles for all the pages that the web driver understands are open.
* They are listed in the order that they have been opened.
* Return type is Set<String>
* **Handling Child Windows in Selenium:**
* Capture the handles of both the parent and child windows.
* Use getWindowHandles() to retrieve all handles.
* Iterate through the handles to switch to the desired window.

1. **File Upload pop up:**

* File upload dialog appears when you wanted to upload a file by clicking on upload button.
* This pop up is a desktop application. Hence it is not inspectable but movable.
* These pop ups can be handled using:

1. **sendKeys():**

It is used to upload the file even without opening file dialog provided the element is an **input** element with **type = ”file”** attribute.

1. **Robot class:** The Robot class in Selenium is a Java-based utility class that provides a way to simulate keyboard and mouse events on the screen.

The Robot class in Selenium works by creating a virtual robot on the screen that can perform keyboard and mouse actions. The robot can simulate key presses, mouse clicks, mouse movement, and other user interactions.

Steps to upload a file using Robot class:

1. Select the file path

**StringSelection str = new StringSelection(“<file\_path>”);**

1. Set the file path to system clipboard

**Toolkit.getDefaultToolkit().getSystemClipboard()**

**.setContents(str, null);**

1. Create Robot instance

**Robot r = new Robot();**

1. Paste the path to the file dialog box

**r.keyPress(KeyEvent.VK\_CONTROL);**

**r.keyPress(KeyEvent.VK\_V);**

**r.keyRelease(KeyEvent.VK\_CONTROL);**

**r.keyRelease(KeyEvent.VK\_V);**

5. Press ENTER button

**r.keyPress(KeyEvent.VK\_ENTER);**

**r.keyRelease(KeyEvent.VK\_ ENTER);**

For macOS:

1. Select the file path

**StringSelection str = new StringSelection(“<file\_path>”);**

1. Set the file path to system clipboard

**Toolkit.getDefaultToolkit().getSystemClipboard()**

**.setContents(str, null);**

1. Create Robot instance

**Robot r = new Robot();**

1. Cmd + Tab is needed since it launches a Java app and the browser looses focus

**robot.keyPress(KeyEvent.VK\_META);**

**robot.keyPress(KeyEvent.VK\_TAB);**

**robot.keyRelease(KeyEvent.VK\_META);**

**robot.keyRelease(KeyEvent.VK\_TAB);**

1. Open Goto window CMD+SHIFT+G

**robot.keyPress(KeyEvent.VK\_META);**

**robot.keyPress(KeyEvent.VK\_SHIFT);**

**robot.keyPress(KeyEvent.VK\_G);**

**robot.keyRelease(KeyEvent.VK\_META);**

**robot.keyRelease(KeyEvent.VK\_SHIFT);**

**robot.keyRelease(KeyEvent.VK\_G);**

1. Paste the clipboard value CMD+V

**robot.keyPress(KeyEvent.VK\_META);**

**robot.keyPress(KeyEvent.VK\_V);**

**robot.keyRelease(KeyEvent.VK\_META);**

**robot.keyRelease(KeyEvent.VK\_V);**

1. Press Enter key to close the Goto window and Upload window

**robot.keyPress(KeyEvent.VK\_ENTER);**

**robot.keyRelease(KeyEvent.VK\_ENTER);**

**robot.delay(500);**

**robot.keyPress(KeyEvent.VK\_ENTER);**

**robot.keyRelease(KeyEvent.VK\_ENTER);**

1. **Notifications pop up:**

* These pop ups are browser specific pop ups.
* They are neither inspectable nor movable.
* These pop ups are handled using browser specific classes like

ChromeOptions for chrome

FirefoxOptions for firefox

EdgeOptions for edge

* Steps to handle these pop ups:

1. Create an instance for ChromeOptions class

**ChromeOptions options = new ChromeOptions();**

1. Disable notifications or geolocation

**options.addArguments(“—disable-notifications”);**

**options.addArguments(“—disable-geolocation”);**

1. Configure options parameter to ChromeDriver

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



**CHAPTER - 3**

**Data Driven Testing**

The process of driving the data from external resources like Properties file or Excel file to the test scripts and performing test execution is called as **Data Driven Testing**.

**Why Data Driven Testing?**

According to the Automation rule, the data should not be hardcoded in to the test scripts because maintenance and modification of the data becomes tedious job. So, we store data in the external files and utilize it in the test scripts for execution.

**Advantages of Data Driven Testing:**

* Avoids hard coding of the data.
* Maintenance and modification of the data is easy.
* Allows testing of the application with multiple sets of data values during [regression testing](https://www.browserstack.com/guide/regression-testing).
* Allows reusing of Actions and Functions in different tests.
* Generates test data automatically.
* Allows execution of test cases several times which helps to reduce test cases and scripts
* Cross browser /platform testing is easy (means change the browser in property File)
* Running test scripts in different Environment is easy
* Running test scripts in different credentials is easy
* We can create the test data prior the Suite execution (we can also get the data from testData team)

**The data is of two types:**

1. **Common Data:** The data which is common to all the test scripts like URL, username, password, browser name.

* Property File is more preferred to store common data
* Property File is light weight
* Property file stores the data in key value pair
* Easy to read the data
* Disadvantage: Not organized hence its preferred only for small amount of data

1. **Test Data:** The data which is specific to only to the test case

* To store test data, we prefer Excel sheet.
* Excel sheet stores data in the form of rows and columns.
* Data is stored in a well-organized way so that data retrieval becomes easy.
* Data maintenance is easy.
* Disadvantage is Excel sheet is slow compared to property File

**Properties File**

Properties file is a java feature file in which data is stored in the form of key and value pairs.

Both key and value should be in String data type.

Properties file has **“.properties”** as extension.

The key and value are separated using **“=”** or **“:”** or **a single space** while storing in properties file.

**Why Properties file?**

Property file is light weight and faster to read the data compared to any other file. Java has own Class (Properties class in java.util package) to read the data from properties file.

**Steps to read data from Properties file:**

1. Convert physical file into java readable object

**FileInputStream fis = new FileInputStream(“<properties\_file\_path>”);**

1. Create an instance of Properties class

**Properties property = new Properties();**

1. Load all key-value pairs from file object (fis) to Properties object (property)

**property.load(fis);**

1. Fetch the data

**String data = property.getProperty(“<key>”);**

* Properties class internally utilizes Map interface.
* When “**property.load(fis);”** is given, it internally creates HashTable (underlying data structure of HashMap) and loads all the key-value pairs from the file object into HashTable. This table is used as reference to fetch data.

**Drawbacks of Properties file:**

* The data is not organized since every thing is stored in key-value pairs.
* It allows to read only single data at a time.
* The keys must be remembered.
* It is suitable only to store small amount of data.

**Excel File**

In Excel the data is stored in the tabular form in rows and cells.

Apache Poi is the open-source libraries used to get & write data from all Microsoft documents like excel, docx, ppt etc.

In real time most the company preferred the keep the test script data in Excel, because data will be in well-organized manner, so that modification and maintenance is easier.

**Steps to add Apache POI dependency to the project:**

* Open pom.xml file in the project
* Open the browser and search for ‘Maven Repository’
* Click on the first link
* Search for “apache poi” in mvnrepository.com
* Click on the second option – Apache poi for OPC and OOXML schemas
* Click on the latest version and copy the dependency
* Paste it within dependencies tag in pom.xml and save and update the project.

**Steps to read data from excel:**

1. Convert physical file in to java readable object

**FileInputStream fis = new FileInputStream(“<excel\_file\_path>”);**

1. Open Excel Workbook

**Workbook wb = WorkbookFactory.create(fis);**

1. Get control over sheet

**Sheet sheet = wb.getSheet(“<sheet\_name>”);**

1. Get control over row

**Row row = sheet.getRow(<row\_number>);**

1. Get control over cell

**Cell cell = row.getCell(<cell\_number>);**

1. Fetch data from cell

**String data = cell.getStringCellValue(); (if the data is string)**

**Or**

**double data = cell.getNumericCellValue(); (if the data is number)**

1. Close the workbook

**wb.close();**

**Drawback:** It is slow to read data from excel compared to properties file

**POM (Page Object Model)**

**Object Repository/Element Repository:**

Object repository is a dedicated place to store all the web elements and their locators. Object Repository can be designed with excel sheet, property file, Json File and POM Class. Most preferred way of developing object Repository is POM Classes.

**Why Object Repository?**

As per the rule of the automation, we should not hardcode(fix) the elements within test scripts. Instead, we should get elements from Object Repository, because in Agile process due to frequent requirement changes, modification and maintenance of elements is tedious job.

**Advantages of Object Repo:**

* Element and locator modification is easy.
* Frequent UI changes can be handled easily.
* No need to identify the same element multiple times.
* Test script development will be faster.
* All the elements are stored in specific place hence maintenance will easy.
* Test script will be optimized.
* Code readability is increased.

**POM (Page Object Model):**

* It is Java Design Pattern used to develop object repository as recommended by Google.
* POM classes are created page wise and all the web elements are stored with the help of rules of POM.
* Total number of POM Classes = Total number of web pages in your application.
* All pom class name should be same as page name and appended with 'Page' like LoginPage, HomePage, OrganizationsPage.

**Why POM?**

* It’s a Java design pattern, Java classes are created and utilized, hence no external libraries are required.
* Elements are directly available with the objects.
* POM is a perfect fit for Agile.
* POM will handle **StaleElementReferenceException**.

**Rules of creating POM Class:**

1. Create a separate java class for a web page- class name should be same as page name.
2. Identify all the web elements present in a particular page with @FindBy and make it as private web element (Element declaration)

**@FindBy(locatorName = “locatorValue”)**

**private WebElement / List<WebElement> element;**

1. Initialize these web elements with a constructor (Element initialization)

**public Name\_Of\_Class(WebDriver driver)**

**{**

**PageFactory.initElements(driver, this);**

**}**

1. Provide getters to access these Elements - Encapsulation (Element utilization)
2. Provide business library to optimize the test script.

**What is StaleElementReferenceException?**

It is a selenium exception which occurs when we try to access an element with the old or expired reference or address.

Whenever the browser is launched a driver reference will be created. When the web page is refreshed, selenium will try to identify the web element, but the element is no longer attached to the GUI because the page got refreshed. Element not attached to DOM during that execution, or element reference might be old. During these situations, selenium will throw **StaleElementReferenceException** meaning the, Reference for that element is Stale or old.

**How Does POM handle StaleElementReferenceException?**

* POM will identify the web elements with annotation @FindBy instead of driver.findElement() / driver.findElements().
* And all the web elements identified will be initialized to driver current reference with **PageFactory.initElements(driver, this);**

**@FindBy:**

* It is a selenium annotation.
* Will identify one element with one locator.
* Usage:

**@FindBy(locatorName = “locatorValue”)**

**private WebElement / List<WebElement> element;**

**What is encapsulation? How POM achieves encapsulation?**

Encapsulation is hiding the data/ limiting access of a particular information. In POM, all the web elements are stored as private web elements, and only getter access is provided through which setter access is denied. This makes the web elements secure and no unnecessary changes in element locators are entertained. This is how encapsulation is achieved.

**@FindBy 🡪 Cannot handle dynamic elements because this annotation will identify the elements before execution. All dynamic elements should be handled with traditional methods of driver.findElement() or driver.findElements().**

**What is PageFactory?**

PageFactory is a class in org.openqa.selenium.support package. It is an extended design pattern of POM, which is used to create an Object to POM classes. At the time of object creation it will execute all @FindBy annotation then initialize all the elements.

**Advantages of POM**

1. Handles StaleElementReferenceException.
2. We can achieve encapsulation via POM.
3. POM is a java design pattern; no external libraries are required.
4. POM stores elements page wise, hence debugging is easy.
5. Perfect fit for AGILE to handle frequent UI changes.
6. Code reusability.
7. Test script is optimized via business libraries.
8. Element and locator maintenance is easy.
9. Increased test script readability.
10. Test script development is faster.

**TestNG**

* **TestNG** is the combination of JUnit and NUnit with some additional functionalities.
* It is basically an open-source Unit Testing Framework Tool used by both Development Engineers and Automation Test Engineers.
* Other Unit Testing Frameworks available in market are:
* JUnit - Java
* NUnit - .Net
* PyTest/PyDev - Python
* Rspsc - ruby
* Jasmine/mocha - Javascript
* TestNG - Java, .Net
* TestNG is a headless tool (no UI), IDEs like Eclipse or IntelliJ provide user interface for testNG.
* **TestNG usage for Developers:** Developers use TestNG as a Unit Testing Tool, where a test script is written to test the source code of the application.
* **TestNG usage for Automation Testers:** Automation Testers use TestNG as a Unit Testing Framework for developing the test scripts from manual test cases and executing them.
* TestNG offers different features:
* TestNG flags
* Annotations
* We can perform batch, group and parallel executions
* Assertions
* Reporting
* Listeners
* Data Providers
* Re-running the failed test scripts

**TestNG flags:**

1. **priority:**

* Used to change order of execution of test scripts.
* It accepts integers as value.
* It follows integer number line order of execution i.e, lowest priority executes first.
* Default priority is 0.
* If priority of multiple test scripts is same, it considers the ASCII values of the method names and executes in ascending order.
* Usage:

**@Test (priority = int)**

**public void methodName()**

**{**

**// script**

**}**

1. **invocationCount:**

* Used to execute test scripts multiple times.
* It accepts integers as value.
* Default invocationCount is 1.
* If invocationCount is 0 or negative, it considers the test script in the execution queue but it will not execute the script.
* Usage:

**@Test (invocationCount = int)**

**public void methodName()**

**{**

**// script**

**}**

1. **enabled:**

* Used to enable or disable the test script.
* It accepts boolean as value.
* Default enabled is true.
* If enabled is false, it does not consider the test script in the execution queue and it will not execute the script.
* Usage:

**@Test (enabled = boolean)**

**public void methodName()**

**{**

**// script**

**}**

1. **dependsOnMethods:**

* Used to create dependency between the test methods.
* Example:

@Test

public void createCustomerTest ()

{

System.out.println("customer created");

}

@Test (dependsOnMethods = "createCustomerTest")

public void modifyCustomerTest ()

{

System.out.println("customer modified");

}

@Test (dependsOnMethods ={"createCustomerTest","modifyCustomerTest"}) public void deleteCustomerTest ()

{

System.out.println("customer deleted");

}

* In the above example, only if **createCustomerTest** is passed, **modifyCustomerTest** and **deleteCustomerTest** will be executed. If createCustomerTest failed then both dependent test scripts will not execute at all.

**Annotations:**

* It is java template
* It provided using a symbol @
* It is meta data/ information to JVM
* No need to call
* TestNG offers following annotations:

@Test

@BeforeMethod

@AfterMethod

@BeforeClass

@AfterClass

@BeforeSuite

@AfterSuite

@BeforeTest

@AfterTest

@DataProvider

@Listnener

@Parameters

* **@Test:**
* It is a basic TestNG annotation which is responsible for all the executions.
* @Test acts like a main method for the JVM to start the execution.
* The return type of @Test annotation method is always void and access specifier is always public.
* Usage:

@Test

public void demoTest ()

{

// script

}

* We can have any number of @Test annotations inside a single Test class.
* Every @Test annotation is individual test script.
* Test class which consists of @Test annotations should be always written in src/test/java in any package.
* Test class name should be always module name ending with Test.
* Test Method name should always be manual test case name with Test.
* In general, a test class contains 15+ @Test annotations/Test Method.
* Default execution order of all the @Test annotations in a test class is an ASCII.
* **@BeforeSuite:**
* It is executed before the tag in the suite xml file.
* It is executed only once per execution as there will be only one <suite> tag per xml file.
* It is used for establishing database connection.
* **@AfterSuite:**
* It is executed after the closing of suite tag in suite xml file.
* It is executed only once per execution as there will be only one <suite> tag per xml file.
* It is used for closing database connection.
* **@BeforeTest:**
* It is executed before the opening of <test> tag in suite xml file.
* The number of times it will execute depends on number <test> tags.
* This is mostly used for parallel executions as it creates multiple threads.
* **@AfterTest:**
* It is executed after the closing of </test> tag in suite xml file.
* The number of times it will execute depends on number tags.
* This is mostly used for parallel executions as it creates multiple threads.
* **@BeforeClass:**
* It will execute before opening of every <class> in suite xml file or simply we can tell before every test class.
* The number of times it will execute depends on the number of <class> tags or test classes.
* It is used for launching browser, maximizing browser and applying implicitly wait statement.
* **@AfterClass:**
* It will execute after closing of every </class> in suite xml file or simply we can tell after every test class.
* The number of times it will execute depends on the number of <class> tags or test classes.
* It is used for closing browser.
* **@BeforeMethod:**
* It will execute before every @Test annotation.
* The number of times it will execute depends on number of @Test.
* It is used for navigating to an application and login to an application.
* **@AfterMethod:**
* It will execute after every @Test annotation.
* The number of times it will execute depends on number of @Test.
* It is used for logout of Application.

**Steps to convert java class to xml file:**

1. Right click on the .java class file in Package Explorer.
2. Go to **TestNG** and click on **Convert to TestNG**
3. Rename the xml file name and click on **Finish**.
4. Suite file / xml file and order of execution of the configuration annotations are as follows:

<!-- BeforeSuite -->

<suite name="Suite">

<!-- BeforeTest -->

<test thread-count="5" name="Test">

<classes>

<!-- BeforeClass -->

<class name="testng.SampleTest">

<methods>

<!-- BeforeMethod -->

<!-- Test --> <include name="demo"></include> <!-- AfterMethod -->

</methods>

</class>

<!-- AfterClass -->

</classes>

</test>

<!-- AfterTest -->

</suite>

<!-- AfterSuite -->

**Executions in TestNG:**

Once the test scripts are developed, they have to be executed to check the correctness of the code - running the program.

**NOTE:** For every new feature/build, we will execute the recent/old Framework to check the regression issues.

1. **Batch Execution:**

* Executing all the existing test script sequentially/one after another.
* Batch execution is also called as FULL REGRESSION testing.
* All the test scripts are loaded inside one suite xml file.
* In one suite xml file / testNg.xml file we can invoke any number of classes but all the classes should be present inside the project.
* **Steps to achieve batch execution:**

1. Convert all the java class files to single xml file.
2. Execute the xml file.
3. **Group Execution:**

* Executing the similar kind of test script under a group.

Ex: Smoke Suite, Regression Suite.

* All type test script belongs either to smoke suite or to Regression suite
* To achieve group execution, every @Test should be included in the group. @Test (groups = "RegressionSuite")

@Test (groups = {"SmokeSuite”, “RegressionSuite"})

* Groups should be specified in 3 places:

1. @Test In test script

@Test (groups = "RegressionSuite")

1. Base Class in all configuration annotations

@BeforeSuite (groups = {"SmokeSuite”, “RegressionSuite"})

1. Suite xml file

<suite name="Suite">

<groups>

<run>

<include name="RegressionSuite"></include>

<exclude name="SmokeSuite"></exclude>

</run>

</groups>

<test thread-count="5" name="Test">

<classes>

<class name="groupExecution.TestClass1" />

<class name="groupExecution.TestClass2" />

<class name="groupExecution.TestClass3" />

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

1. **Parallel Execution:**

Parallel execution means multiple threads start the execution simultaneously. We have three types in parallel executions.

1. Distributed Parallel execution
2. Cross Browser Parallel Execution
3. Cross Platform Parallel Execution

**Distributed Parallel execution:**

* We prefer distributed parallel execution when we have to reduce the total execution time taken by a suite.
* We distribute the total number of existing test scripts among multiple threads in Suite xml file with <test> and start the execution.
* Browser will not be changed here. Only the test scripts are distributed.
* Different threads - Different test scripts - Same browser launched in all threads.
* One thread can have any number of test classes and all the test class inside the <test> and </test> will be executed sequentially.

<suite name="Suite" parallel="tests" thread-count="2">

<test name="Test Runner 1">

<classes>

<class name="distributedParallel.Test3" />

<class name="distributedParallel.Test2" />

<class name="distributedParallel.Test1" />

</classes>

</test> <!-- Test -->

<test name="Test Runner 2">

<classes>

<class name="distributedParallel.Test6" />

<class name="distributedParallel.Test5" />

<class name="distributedParallel.Test4" />

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

* **Steps to achieve Distributed Parallel Execution:**
* Distributed Parallel execution is a form of TestNG executions which is used to reduce the execution time required when all the test scripts are executed as Batch.
* We will create a separate suite xml file and give parallel mode as Tests and provide required thread count (max thread count is 5).
* Later create a separate <test> </test> and load the required classes which should execute in different threads.
* Run the execution.

**Cross Browser Parallel Execution:**

* Cross browser execution means executing the same set of test scripts in multiple browsers to ensure they are compatible.
* In cross browser execution, same set of test scripts are executed over different browsers in different threads.
* Different Threads - Different Browsers - same set of Test scripts.
* Since during run time we have to choose the browser for execution, we have to provide browser name from suite xml file instead of property file.
* <parameter> is used to set the name and value which will pass this data to @Parameter annotation in base class
* **In suite file:**

<suite name="Suite" parallel="tests" thread-count="3">

<test name="Test Runner 1">

<parameter name="BROWSER" value="chrome"></parameter>

<classes>

<class name="crossBrowserParallel.Test2" />

<class name="crossBrowserParallel.Test1" />

</classes>

</test> <!-- Test -->

<test name="Test Runner 2">

<parameter name="BROWSER" value="firefox"></parameter>

<classes>

<class name="crossBrowserParallel.Test2" />

<class name="crossBrowserParallel.Test1" />

</classes>

</test> <!-- Test -->

<test name="Test Runner 3">

<parameter name="BROWSER" value="edge"></parameter>

<classes>

<class name="crossBrowserParallel.Test2" />

<class name="crossBrowserParallel.Test1" />

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

* **In BaseClass:**

**public** **class** BaseClass {

WebDriver driver;

@Parameters("BROWSER")

@BeforeClass

**public** **void** classConfig(String browser) {

**if**(browser.equalsIgnoreCase("chrome"))

driver = **new** ChromeDriver();

**else** **if**(browser.equalsIgnoreCase("firefox"))

driver = **new** FirefoxDriver();

**else** **if**(browser.equalsIgnoreCase("edge"))

driver = **new** EdgeDriver();

**else**

System.***out***.println("Invalid browser info");

}

@AfterClass

**public** **void** classteardown() {

driver.quit();

}

}

* **In Test Class:**

**public** **class** Test1 **extends** BaseClass {

@Test

**public** **void** google() {

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

System.***out***.println(driver.getTitle());

}

}

* **How to perform cross browser execution / compatibility testing?**
* Choose the test scripts which have to execute in multiple browsers.
* Create a separate suite xml file and choose parallel mode as tests and set thread count as required.
* Create separate <test> and load the same class in both threads.
* Provide browser details using <parameter>

<parameter name="BROWSER" value="edge"></parameter>

* Use @Parameters annotation to read the parameter value from suite xml file in base class

@Parameters("BROWSER")

* Parameterize annotation method to capture the browser details

public void bcConfig (String BROWSER) {}

* Execute / run the test script through suite xml file

**Cross Platform Parallel Execution:**

* Cross platform execution means executing the same set of test scripts in multiple platforms or operating systems to ensure they are compatible.
* In cross platform execution, same set of test scripts are executed over different platforms in different threads.
* Different Threads - Different Platforms - same set of Test scripts.
* This is achieved through **Selenium Grid**.

**Assertions:**

* Automation rule says validation is mandatory for every test script.
* Validations are very important part of test script development which is completely depended on individual automation engineering.
* Every test script should be validated for the correctness of result based on expected and actual result.
* Java If-Else blocks are not recommended for validations of test script as if- else block do not have the capability to fail the test script.
* Depending on if condition:

if true: if block will be executed

if false: else block will be executed.

* So, no failure will occur and since in any case either If or Else is getting executed, further lines in the test script will be executed. Hence this validation is not recommended.
* **Assertions is a feature of TestNG which helps in validation. if expected result doesn’t match with actual result, Assertions will give Assertion Error Exception and fail the test script providing the line number of the failure.**
* **Hard Assert:**
* Class name is **Assert**.
* All the methods present here are **static** **methods** hence accessed via Class name.
* **Assert.assertTrue();**
* **Assert.assertFalse();**
* **Assert.assertEquals();**
* **Assert.assertNotEquals();**
* **Assert.assertNull();**
* **Assert.assertNotNull();**
* **Assert.fail();**
* These are the few methods present in Assert Class.
* Hard assert will stop the execution of further lines if the assertion is failed.
* Whenever hard assert test script fail, hard assert will generate **AssertionError** Exception along with line number of failure and error message and it will stop the current test script execution and continue other test scripts in the same test class.
* Whole test script gets failed if one hard assert is failed.
* Hence Hard assert is used to validate mandatory fields in the test script.
* **Soft Assert:**
* Class name is **SoftAssert**.
* All the methods present here are **non-static methods** hence accessed by creating object.

**SoftAssert sa = new SoftAssert ();**

**sa. assertTrue ();**

**sa. assertFalse ();**

**sa. assertEquals ();**

**sa. assertNotEquals ();**

**sa. assertNull ();**

**sa. assertNotNull ();**

**sa. assertAll ();**

* These are the few methods present in SoftAssert Class.
* Soft assert will not stop the execution of further lines if the assertion is failed.
* Whenever soft assert test script fail, soft assert will generate **AssertionError** Exception along with line number of failure and error message and it will continue the current test script execution.
* **assertAll ()** is a special method which will sum up the status of all soft asserts used in the entire test script and provide it in console at the end.
* If assertAll () is not used at the end of all soft assert statements, no failures will be logged.
* Hence Soft assert is used to validate non mandatory fields in the test scripts.

**Reporting:**

* Reports are a very important part of Framework because they act like proof for all the execution done by automation tester.
* Customers usually ask for execution report.
* Reports have to be provided for Developers also.
* Usually after every execution, reports will be sent to high offs like Product Owner, Automation Leads, Technical consultants and customers.
* There are two types of reports:

1. **High Level Reports:** TestNG supports HTML report which provides basic information like

* Total number of test scripts executed
* Number of test scripts Passed/Failed/Skipped
* Reason for the failure.

1. **Low Level Reports:** These specify the log level analysis. In TestNg we have "Reporter.log();”

* **Steps to view testNG reports:**

1. Execute xml / suite file
2. Refresh the project
3. Open test-output folder
4. Open emailable-report.html

**Listeners:**

* Listeners is a feature in TestNG used to monitor the run time events of a test script like pass or fail or skip statuses and based on the events, we can perform necessary actions with abstract methods of **ITestListener** and **ITestResult** interfaces.
* **ITestListener:**
* It is an inbuilt interface available in TestNG
* ITestListener has abstract methods which will capture all the run time events of a test script like Pass, fail, Skip.
* As per the rule ListenerImplementation class should implements ITestListener Interface and should override all the abstract methods present.
* Some of the important abstract methods are:

**1. onStart() - Start of current suite execution**

**2. onFinish() - end of current suite execution**

**3. onTestStart() - start of current test script execution (@Test)**

**4. onTestSuccess() - current test script execution is passed (@Test)**

**5. onTestFailure() - current test script execution is failed (@Test)**

**6. onTestSkipped() - current test script execution is skipped (@Test)**

* **ITestResult:** It is an interface available in TestNG and it is used as an argument for every abstract method in ListenerImplementation class which will capture the pass / fail / skip status of the current test script during run time.
* **ListenerImplementation class:**

**public** **class** ListenerImplementation **implements** ITestListener{

@Override

**public** **void** onStart(ITestContext context) {

System.***err***.println("onStart");

}

@Override

**public** **void** onTestStart(ITestResult result) {

System.***err***.println("onTestStart");

}

@Override

**public** **void** onTestSuccess(ITestResult result) {

System.***err***.println("onTestSuccess");

}

@Override

**public** **void** onTestFailure(ITestResult result) {

System.***err***.println("onTestFailure");

}

@Override

**public** **void** onTestSkipped(ITestResult result) {

System.***err***.println("onTestSkipped");

}

@Override

**public** **void** onFinish(ITestContext context) {

System.***err***.println("onFinish");

}

}

* **Test Class:** We have to provide listeners for every class using @Listeners because each class should be monitored

@Listeners(listenerPractice.ListenerImplementation.**class**)

**public** **class** TestClass **extends** BaseClass {

@Test

**public** **void** test1() {

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

//Assert.*fail*();

}

}

* **In Suite file:** Listeners can be given at Suite xml file also using <listeners>

<suite name="Suite">

<listeners>

<listener class-name="listener.ListenerImplementation"></listener>

</listeners>

<test thread-count="5" name="Test">

<classes>

<class name="listener.TestClass" />

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

**Data Providers:**

* To run the same test script multiple times with different sets of data we use Data Provider.
* Data provider is an annotation which will load multiple data for every execution.
* Total number of test runs = Total number of data present in Data provider.
* Test class can have multiple @DataProviders but @Test can read only one data provider at once.
* The data in the data provider is stored in Object[][], where first array represents number of sets of data and second array represents number of arguments in each set.
* Return type of @DataProvider annotation method is always Object [][].
* Sample Program to illustrate Data Providers:

**public** **class** DataProvidersTest {

@DataProvider

**public** Object[][] data() {

Object[][] obj = **new** Object[3][2];

obj[0][0] = "India";

obj[0][1] = "Scotland";

obj[1][0] = "India";

obj[1][1] = "Canada";

obj[2][0] = "India";

obj[2][1] = "New Jersey";

**return** obj;

}

@Test(dataProvider = "data")

**public** **void** bookTicketTest(String from, String to) {

System.***out***.println("From: "+ from + "\tTo: "+ to);

}

}

**Re-running the failed test scripts:**

* We can re-run the failed test scripts as follows:

1. Once suite execution is done, refresh the project
2. Open test-output folder
3. Open testing-failed.xml

This suite xml file consists of failed test scripts in the current execution and it is auto updated.

1. Run this suite xml file, all the failed scripts in current execution will run



**CHAPTER - 4**

**MAVEN**

* It is also called as **Build management Tool, build testing Tool** and **Build Dependency Tool**.
* Maven is tool used which has become popular for the build management instead ANT and GRADLE.
* **Why Maven used in Development?**
* In case of Development Maven is used as **Build Process Tool [Build creation, build testing and Build Deployment]**.
* Various Maven commands are used for these build related actions like
* **Build Creation process:** Process of converting the source code into any executable format ***“mvn package”*** is the maven command used.
* **Build Testing Process:** Since multiple Developers will be contributing to the projects, we have to test the project for compilation issues, or/and if the build is broken or not***."mvn Compile", "mvn Test"*** is the maven command is used.
* **Build Deployment Process:** Process of installing the build into Testing server/any other Environment,***"mvn Deploy","mvn install***" is the maven command is used.
* **Why Maven used in Automation Testing?**
* In Automation Maven is used for Testing the Application with the current/recent Framework by executing the test scripts.
* When Multiple automation testers are working with same framework, there might be chances that the changes made by one Automation engineering might affect the entire framework, in order to overcome this issue we run the Maven life cycle for every build ***mvn clean, mvn validate, mvn compile and mvn test*** are the maven command used.
* **Softwares in Maven:**

1. **Maven Eclipse Plugin:** This is inbuilt plugin available in IDEs like Eclipse. It helps to create Maven project and provides folder structure.

**To create maven Project:**

Group id: Organization/Community Name

Artifact Id: Project Name Inside Maven

Project: Framework folder structure

* src/main/java: reusable components like
* GenericUtilities
* ObjectRepository
* src/main/resources: driver executables
* ChromeDriver.exe
* FirefoxDriver.exe
* IEDriver.exe
* src/test/java
* Test scripts
* src/test/resources: external files
* commonData.properties
* TestData.xlsx
* pom.xml
* Dependencies
* plugins

**pom.xml:**

* It is called **Project Object Model** or **Project Configuration file**.
* It is the heart of any maven project.
* It is created only once during the project creation.
* If pom.xml is corrupted, then we have to discard the entire project and create another.
* The components of pom.xml are dependencies and Plugins.

**Dependencies in Maven:**

* Dependency is advance feature in maven which is used to get all the required jars from the Global repository(http://mvnrepository.com) to Local repository (C://user/name/.m2) & attach all jars to the project automatically.

1. **Maven Command Line Plugin:**

* This is a Maven software which helps us to run the maven project without eclipse.
* We have to download Maven to local system and set the environment variables.
* To check if its set properly: mvn --version.
* **Build Life Cycle/maven Life cycle:**
* mvn clean: It is a maven command used to clean all the old reports from target folder.
* mvn validate: It is a maven command used to check if all the necessary jar are downloaded for all the dependencies added, if not it will automatically download.
* mvn compile: It is maven command to will check the compilation issues in the framework.
* mvn test: It is a maven command which will identify all the test classes whose class name end with "test" and execute them.
* **Advantages of MAVEN**
* Check the integration & compilation issues between the framework components. Whenever multiple engineers working with same framework, there might be possibilities one engineer modification, addition & deletion might affect the entire build.
* Handle dependencies jars.
* Create framework configuration quick setup for new engineers.
* Provide framework folder structure.
* We can run the test script in command Line without Eclipse used to check framework build is broken or not
* Maven majorly supports Jenkins for CD/CI.

**FRAMEWORK**

**Definition:** Framework is a well-organized structure of reusable components where one driver (.xml) file will take care of the execution without any manual intervention.

Framework is collection of reusable components that makes automation development execution and modification easier and faster.

Framework is a set of instruction followed by every organization that makes automation test engineer life easy.

**Framework Approaches:** Broadly there are two types of Framework approaches:

1. **TDD - Test Driven Development**

* Test cases are mandatory.
* @Test is the driving factor for test script development.
* TestNG, Junit are used for any TDD approach-based framework.
* These Frameworks are easy to develop.
* Maximum number of features can be made re-usable.

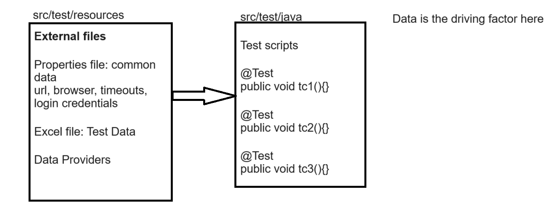
1. **BDD - Behavior Driven Development**

* Test Scenarios are mandatory, anybody who has application knowledge can write scenario
* @Given, @When and @Then are used for test script development
* Cucumber is the Tool used for BDD approach
* Given, when, Then, And are the Gherkins keywords which help to develop the feature file, which is later used to develop the step definitions.
* Cannot achieve good code re usability.

**Types of Frameworks:**

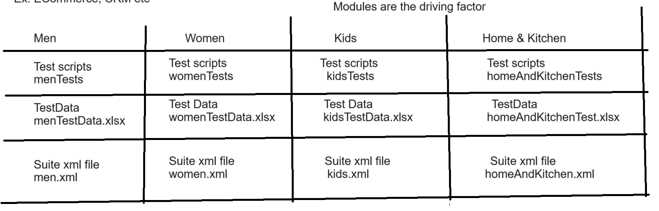
1. **Data Driven Framework:**

* Reading the data from any external resources or using data provider is called as data driven Framework.
* In Data driven framework, Data is the driving factor.
* Why? - Automation rule says never hardcode the data, read it from external resources.
* When? - Whenever the test data is huge compared to test scenarios, we prefer data driven framework.
* Examples - Ecommerce, banking, Finance, Travel



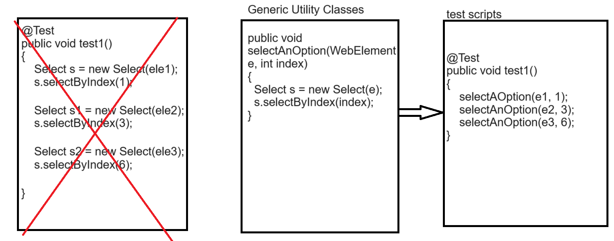
1. **Modular Driven Framework:**

* Maintaining the test scripts, test data and suite xml file module wise in order to make the debugging process easy is called as modular driven framework.
* Here module is the driving factor for framework.
* Why? - Code Modification and maintenance is easy, Debugging is easy.
* When? - Whenever the application is very huge and has lot of modules, maintaining all the modules together will be difficult, hence we prefer modular driven framework.
* Examples - CRM, Ecommerce



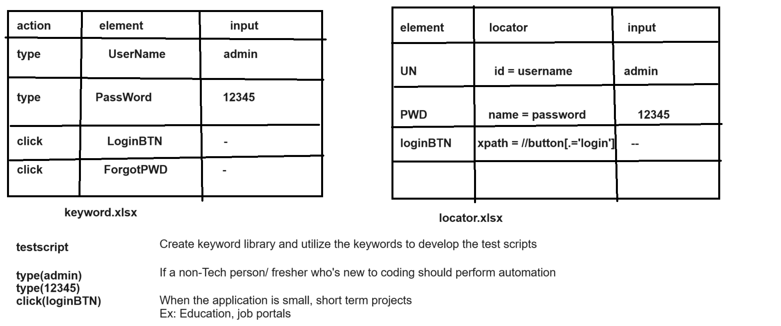
1. **Method Driven Framework:**

* Developing the re-usable methods for all the repetitive action/functionalities in the application and calling these methods in the test script is called as method driven framework.
* Here re-usable methods are the driving factors for framework.
* Why? - Method re usability, Test script Optimization.
* When? - Whenever the application contains more repeated functionalities like too many dropdowns too many frames, more windows, we prefer to write generic method and use them multiple times.
* Examples - Ecommerce, CRM, HealthCare



1. **Keyword Driven Framework:**

* Creating a keyword library and utilizing these keywords to develop the test script is called as keyword driven framework.
* Here keywords are the driving factors for framework.
* Why? - If Non coder should perform automation/ freshers.
* When? - Whenever manual testers, Freshers, have to perform automation, they would not be good at coding. Hence, we prefer Keyword driven framework where all the actions are converted into methods with user friendly method names so that they can just call and perform the actions.
* Examples - short Term projects - Education, job portals.



1. **Hybrid Framework:**

* Combination of two or more above discussed frameworks is called as Hybrid framework.
* Why? Some applications will have huge Data and also more number modules, so we have combined feature of two framework to make the Framework user friendly.
* When? - Whenever the application demands to use more than one framework, we use Hybrid by Combining the features of multiple frameworks.
* NOTE:
* Hybrid = Data Driven + Modular Driven
* Hybrid = Data Driven + Modular Driven + Method Driven ---
* Hybrid = Data Driven + Method Driven ---
* Hybrid = Data Driven + keyword Driven
* Hybrid = Data Driven + Keyword Driven + Method Driven
* Examples - All long-term projects like Ecommerce, CRM

**Advantages of Framework:**

1. Test script development is faster and easier because of reusability Modification and maintenance of data is easy because data is stored in external resource.
2. Modification and maintenance of element is easy because we have used.
3. POM design pattern to maintain the elements in well-organized way.
4. POM design pattern is the perfect fit for agile Process.
5. Framework provide automatic screenshot for failed script.
6. Framework provides flexibility to achieve cross browser testing, distributed environment testing, smoke testing, regression testing, regional regression testing.
7. Framework provide accurate execution report for every execution.
8. Framework provides generic reusable utility for all actions like Excel Utility, base class.
9. Test script can be re used for every new build.
10. Test script is optimized.

**Disadvantages of Framework:**

1. Should be good in programming.
2. Initial Framework development cost and time is high.

**Phases of Framework:** There are 3 phases in Frame work

* **Choice of Framework:** In this Phase is based on CRS, technical leads, BA, PO, automation Consultants will decide which type of framework should be chosen then that framework design phase will start.
* **Framework design:** In this phase framework developer will design the framework which contain common utility like generic libraries, test data template, POM classes with partial elements. This phase is executed in Sprint-1 or Release-1.
* **Framework Implementation**: This phase starts from Sprint-2, all automation engineer participates in test script development implementation.
* **Framework Execution:** Execution phase is always handled by Jenkins tool, that is whenever we get a new build to testing environment JENKINS will automatically execute and send email to the concerned engineer.

**GITHUB**

**Definition:** It is a distributed cloud based decentralized repository where we can maintain our source Code / Automation Framework / CRS doc /build of the application in one place.

**Softwares in GitHub** There are 2 Software in GitRepsitory

1. **Git HUB:** Cloud based repository(software), which is used maintain the source code in one place, in order to use it just create an account with <https://github.com>

2. **Git [Git client]:** it is a software should installed in client machine, which is used to communicate to GITHUB.

Example:

* **EGit:** Eclipse Git, Installed with Eclipse, we can execute git commands through eclipse.
* **GitDeskTop / GitBash:** Should be installed externally to execute git commands through cmd line.

**Advantages of GitHub Cloud:**

1. Since its cloud-based repository, no need to maintenance team to maintain the Software / Hardware.

2. Cloud means pay rent for what you use

3. Cloud software always access via internet

4. Cloud System / sever physically not present within the Organization, but present virtually

5. Initial investment is not required for Software/ Physical location

6. Scale UP / Scale Down is easy

7. File Share between the team members is easier

8. It provides remote access. It means anywhere contributors can access via internet

9. Provide History for changes made by users & backup facility

10. Jenkins Always get the latest framework from the Git for batch Execution

**Why Git is Decentralized Repository?**

Git is Decentralized Repository because, in Git before pushing any Code to git Hub, we have to commit the code to local repository first, make sure code is working in Local Repository then push Code to GITHUB (Global Repo)

There are Three stages in Git =>

start with **"Working Directory"->Local Repo->Global Repo**

**Usage of github at different stages:**

1. **Developers use github for:** storage of source code

2. **DevOps use github for:** storage of build versions

3. **Manual Testers use github for:** storage of CRS and Manual test cases.

4. **Automation Testers use github for**: storage of Framework

**Steps to Commit and Push the project to GitHUB:**

1. **Create local repository (.git folder):**

* Right click on the project 🡪 Team 🡪 Share Project
* Select “Use or create repository in parent folder” checkbox
* Click on the project name and click on “Create Repository” and Finish.

1. **Create global repository:**

* Login to <https://github.com/>
* Click on “+” in top right options 🡪 New Repository
* Name the repository and click “Create Repository”

1. **Generate github token:**

* Login to <https://github.com/>
* Click on profile icon on top right corner 🡪 Settings
* Click on Developer Settings
* Click on Personal Access Tokens 🡪 Tokens classic
* Click on Generate New Token 🡪 Generate New Token Classic
* Give the “Note” about why the token is generated
* Select “repo” checkbox under “Scopes” and click on “Generate Token”
* Copy the token and save it in NotePad.

1. **Commit the project to local repository:**

* Right click on the project 🡪 Team 🡪 Commit
* Open Git Staging and click on “++” button (All the files will be moved to Staged Changes from Un-staged Changes)
* Provide commit message and click on Commit button

1. **Push the project to global repository:**

* Once the project is committed to local repository, click on Push Head
* Copy the URI of the global repository from github.com
* Provide Project URI, GitHUB Username and GitHUB Token as Password
* Click on Preview 🡪 Preview 🡪 Push
* Provide GitHUB Username and GitHUB Token as Password again

**JENKINS**

Jenkins is a **CI/CD** Tool used by Developers, DevOps and Automation Testers.

**CI: Continuous Integration**

**CD: Continuous Development, Continuous Deployment, Continuous Delivery**

Basically, Jenkins automates

* Process of build creation: Continuous development.
* Process of installing the build into testing environment: Continuous deployment.
* Process of checking the integration issues between old feature and new features: continuous integration.
* Process of delivering the tested the build to the production environment: Continuous delivery.

**Why Jenkins is required in development?**

* **Continuous development:** continuous monitor the git source code repository & create a new build if any changes happened in the git source code.
* **Continuous deployment:** get the latest build from git location & deploy the build in to testing environment.
* **Continuous Delivery:** get the latest build from git location & deploy the build in to UAT environment/Production environment.

**Why Jenkins is required in Automation?**

* **Continuous Integration:** Continuous execution of the selenium test scripts in testing environment to check the integration issues.
* Automation Engineers use Jenkins for Continuous Integration.
* Continuous Integration means checking the integration issues between the old build and the new feature by executing the old framework.
* If the test scripts get failed, then with the failure we will get to know the impact of new feature on the old build. Hence, we can analyse the failure, debug the failed test scripts, if any product issues/bugs are found we will rise the defect in Jira, if it’s a test script issue we will correct it update the framework and re run the framework.

**Steps to download and install Jenkins:**

* Search for Jenkins Download in the browser
* Click on the first link
* Under “Download Jenkins 2.462.1 LTS” click on “Generic java package (war)”
* Open the terminal and change the directory to the folder where the jenkins.war file is downloaded
* Run the following command: **java -jar jenkins.war**
* Copy the Administrator Password
* Open the browser and type the URL – [**http://localhost:8080/**](http://localhost:8080/)
* Paste the Administrator Password and click on Continue
* Click on Install Suggested Plugins
* Once the plugins are installed create Jenkins credentials and then click on “start using Jenkins”

**Configuring Jenkins:**

* **Steps to install plugins:**
* Go to Dashboard 🡪 Manage Jenkins 🡪 Plugins
* Click on Available Plugins
* Search for Maven Integration and select the Maven Integration checkbox and click on install
* Again, click on Available Plugins
* Search for GitHub Integration and select the GitHub Integration checkbox and click on install
* **Steps to do path settings:**
* Go to Dashboard 🡪 Manage Jenkins 🡪 Tools
* **JDK path setting:**
* Click on Add JDK
* Provide name as JAVA\_HOME
* Give the path of JDK 17 / JDK 21
* **GIT path setting:**
* Search for Git download in browser
* Click on the first link
* Download the git file and install
* In Jenkins, Provide name as GIT
* Give the path of git.exe
* **Maven path setting:**
* Click on Add Maven
* Provide name as MAVEN\_HOME
* Select install automatically check box
* **Click on Apply and then click on Save**

**Creating a new Job in Jenkins and executing it:**

* Login to Jenkins
* New Item -> Name of the job -> Choose Maven Project and Click on OK
* Click on Source Code Management and Select Git radio button and provide the global git location where the project is stored
* Click on Build and provide “test” command in Goals and Options
* Click on Apply and then Click on Save
* To trigger the build, click on Build Now

