Selenium

- Selenium is an open source (free) automated testing suite to test web applications. It supports different platforms and browsers.
- Selenium is a set of different software tools. Each tool has a different approach in supporting web based automation testing. It has four components namely,
    - **Selenium IDE (Selenium Integrated Development Environment**):) is a Firefox plugin. It is the simplest framework in the Selenium Suite. It allows us to record and playback the scripts
        - **Firefox** is the only browser that supports Selenium IDE to be used.
        - **Selenese** is the language which is used to write test scripts in Selenium IDE.
    - **Selenium RC (Selenium Remote Control**): Selenium RC AKA Selenium Remote control / Selenium 1. Selenium Remote Control was the main Selenium project for a long time before the WebDriver merge brought up Selenium 2. Selenium 1 is still actively supported (in maintenance mode). It relies on JavaScript for automation.
    - **Selenium WebDriver**: Selenium WebDriver AKA Selenium 2 is a browser automation framework that accepts commands and sends them to a browser. It is implemented through a browser-specific driver. It controls the browser by directly communicating with it. Selenium WebDriver supports Java, C#, PHP, Python, Perl, Ruby.
    - **Selenium Grid**: Selenium Grid can be used to execute same or different test scripts on multiple platforms and browsers concurrently so as to achieve distributed test execution. What are the advantages of Selenium Grid?
        - It allows running test cases in parallel thereby saving test execution time.
        - It allows multi-browser testing
        - It allows us to execute test cases on multi-platform
- Advantages of Selenium Automation Tool
    - Free and open source
    - Have large user base and helping communities
    - Cross-browser compatibility
    - Platform compatibility
- **Junit and TestNG** are two Open-source Frameworks supported by Selenium WebDriver.
- limitations of Selenium WebDriver
    - We cannot test windows application
    - We cannot test mobile apps
    - Limited reporting
    - Handling dynamic Elements
    - Handling page load
    - Handling pop up windows
    - Handling captcha

DRIVER INITIALIZATION –BASICS

- **Chrome** *System.setProperty("webdriver.chrome.browser", "C:\\chromedriver.exe");/**key val***
  WebDriver driver = new ChromeDriver(); //
    - WebDriver is an interface, and here we are creating an object of type WebDriver instantiating an object of ChromeDriver class. It also, create session id. For every step driver is called, two things will be sent each time: Session id and command.
    - **"SearchContext"** acts as the super interface for the Web Driver. It is the external interface which has only two methods: findElement() and findElements()
- **Firefox** *System.setProperty("webdriver.**gecko**.browser", "C:\\geckodriver.exe");*
  WebDriver driver = new FirefoxDriver();
- **IE** *System.setProperty("webdriver.internetexplorer.browser", "C:\\internetexplorerdriver.exe");*
  WebDriver driver = new InternetExplorerDriver();
- **Safari** *System.setProperty("webdriver.safari.browser", "C:\\safaridriver.exe");*
  WebDriver driver = new SafariDriver();

WEBDRIVERMANAGER

helps to download binaries/executables in an automated way. We just need to add its dependency through Maven Lunching browser:

```
WebDriverManager.chromedriver().setup();
WebDriverManager.operadriver().setup();
WebDriverManager.firefoxdriver().setup();
WebDriverManager.edgedriver().setup();
WebDriverManager.iedriver().setup();
```

specific version instead of latest driver

```
WebDriverManager.chromedriver().version("2.40").setup();
```

arch32() Force to use the 32-bit version of a given driver binary.

```
WebDriverManager.firefoxdriver().arch32().setup();
```

arch64() Force to use the 64-bit version of a given driver binary.

```
WebDriverManager.firefoxdriver().arch64().setup();
```

OBJECT LOCATORS

Locating by ID: (priority = 1)

```
driver.findElement(By.id("but1")).click();
```

Locating by Name: (priority = 2)

```
driver.findElement(By.name("but2")).click();
```

Locating by Xpath: (priority = 3)

```
//tag[@attribute='value']
driver.findElement(By.xpath("html/body/div[1]"));
```

Locating by CSS SELECTOR:

```
tag[attribute='value']
driver.findElement(By.cssSelector(".mainheading"));
```

Locating by ClassName:

```
driver.findElement(By.className("profileheader"));
```

Locating by TagName:

```
driver.findElement(By.tagName("select"));
```

Locating by LinkText: Accessing links using their exact link text. Specified as <a (Anker) then click in DOM

```
driver.findElement(By.linkText("Index")).click();
```

Locating by PartialLinkText: Accessing links using a portion of their link text

```
driver.findElement(By.partialLinkText("NextP")).click();
```

CUSTOME XPATH : XPath is used to locate the elements

- **Difference between Absolute Path and Relative Path?**
    - Absolute XPath starts from the **root node** and ends with desired descendant element's node. It starts with top **HTML** node and ends with input node. It starts with a single forward slash**(/**) as shown below
      `/html/body/div[3]/div[1]/form/table/tbody/tr[1]/td/input`
    - Relative XPath starts from **any node** in between the HTML page to the current element's nodeb(last node of the element). It starts with a double forward slash**(//)** as shown below.
      `//input[@id='email']`

Basic Xpath Using OR & AND

```
//tag[@attribute_1='value' and @attribute_2='value']
//tag[@attribute_1='value' or @attribute_2='value']
```

Basic Xpath Using text ()

```
//tag[text()='value of text']
```

Contains: it is used for dynamic attributes. The contain feature has an ability to find the element with partial text.

```
//tag[contains(@attribute,'value')]
//input[@id='username' and contains(@class,'login-email')]
```

contains text: mostly used for links

```
//tag[contains(text(),'value')]
```

starts-with: it is used for dynamic attributes. For instance, id = test_test_1234

```
//input[starts-with(@id,'test_test')]
```

ends-with: it is used for dynamic attributes. For instance, id = 1234_test_test

```
//input[ends-with(@id,'_test_test')]
```

Positioning: is used when your xpath is used for multiple items.
you put the entire xpath in parentheses and give the index number. it does not work for [0]

```
(//tagname[@attribute='value'])[1] OR
(//tagname[@attribute='value'])[position()=1]
```

Following takes you to next code line having the specific attributes

```
//*[@type='text']//following::input[1]
```

Preceding

```
input[@name='username']//preceding::div
```

- The Primary Difference Between The XPath And CSS Selectors?
    - With the XPath, we can traverse both forward and backward, whereas CSS selector only moves forward.

Travel between family

1. Find xpath of a definite object.	
2. Travel from Child to parent:	//parent::
3. Sibling to Sibling:	//preceding-sibling:: or //following-sibling::
4. Reverse travel (immediate parent)	//tagname[@attribute='value']/../../..
5. parent to **direct** child	//tagname[@attribute='value']/tagname
6. parent to **Indirect** child	//tagname[@attribute='value']//tagname

FINDELEMENT AND FINDELEMENTS

Find Element: is used to uniquely identify a (one) web element within the web page

```
WebElement loginLink = driver.findElement(By.linkText("Login"));
```

FindElements: is used to uniquely identify the list of web elements within the web page. Find Elements command takes in By object as the parameter and returns a list of web elements .
Links:

```
List<WebElement> linksList = driver.findElements(By.tagName("a"));
System.out.println("total number of links: "+ linksList.size());
        for(int i=0; i<linksList.size(); i++){
        String text = linksList.get(i).getText();
        System.out.println(text);}
```

ImagesCount and AttributeValues:

```
List<WebElement> imagesList = driver.findElements(By.tagName("img"));
System.out.println(imagesList.size());
        for(int i=0; i<imagesList.size(); i++){
        String imageUrl = imagesList.get(i).getAttribute("src");
        System.out.println(imageUrl);           }
```

- **Difference between findElement and findELements**
    - findElement
        - Returns the first most web element if there are multiple web elements found with the same locator
        - Throws exception **NoSuchElementException** if there are no elements matching the locator strategy
        - It will only find one web element
    - findElements
        - Returns a list of web elements
        - **Returns an empty list** if there are no web elements matching the locator strategy (**No exception**)
        - It will find a collection of elements whose match the locator strategy.
        - Each Web element is indexed with a number starting from 0 just like an array

OPERATIONS

Get() Launches WebPage. Example: driver.get("https://www.google.com"); //gives error without http or https //without "www", it works. it doesn't maintain the browser History and cookies so , we can't use forward and backward buttons
getCurrentUrl() gets Current URL
getTitle() Get the Actual Page Title
getPageSource() get source code of the page
getText() Fetches the inner text of the element that you specify
clear( ) used to clear text boxes of its current value
getAttribute("value"); pass the attribute to get the value—used to get the default text
sendKeys() used to enter values onto text boxes
click() used to click on the link - **sendKeys(Keys.ENTER);**
close() It closes only the browser window that WebDriver is currently controlling. Make the session ID expired or invalid.
quit() It closes all windows that WebDriver has opened. Closes the session ID. Session id be comes null.

ELEMENT VISIBILITY

They all return Boolean value. For example, always remember the "i agree with terms" + submit button in login page.
isDisplayed(); is the method used to verify presence of a web element within the webpage. the method returns a "True" value if the specified. It is capable to check for the presence of all kinds of web elements available
isEnabled(); is the method used to verify if the web element is enabled or disabled within the webpage. It is primarily used with buttons.
isSelected(); is the method used to verify if the web element is selected or not. It is used with radio buttons, dropdowns and checkboxes.

SYNCHRONIZATION (IMPLICIT, EXPLICIT & FLUENT WAIT)

- **Implicit Wait**
    - Tells web driver to wait for certain amount of time before it throws a "No Such Element Exception". The default setting is 0. Once we set the time, web driver will wait for that time before throwing an exception.
    - Applicable **ONLY for webElements**. (NOT title of page)
    - get() load the page but the implicitlyWait displays the DOM - DOM display may take place after a few seconds
    - it is a global wait. Meaning, will be applied to all elements

```
driver.manage().timeouts().implicitlyWait(10,TimeUnit.SECONDS);
E1
E2
driver.manage().timeouts().implicitlyWait(15,TimeUnit.SECONDS);
E3
E4
```

- **Explicit Wait**
    - Tells the Web Driver to wait for certain conditions (Expected Conditions) or the maximum time exceeded before throwing "TimeOutException"
    - Can be applied only for **specified elements** with **customization**
    - Used **for both WebElement and Non WebElements**

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

Webelement: *wait.until(ExpectedConditions.presenceOfElementLocated(**Bylocator**))*
NonWebElement: *wait.until(ExpectedConditions.titleContains(excededCondition));*

- **A few tips**
    - combining Exp and Imp waits: it gives you unnecessary extra waits. Hint: don't apply them together
    - pageloadtimeout: ONLY for page loading - for very slow internet we can use
    - pageloadTimeout with navigate: They work well together if used.
    - **Thread.Sleep(5000);** static (fixed) wait in milliseconds

MANAGE COMMANDS

Maximize Window: driver.manage().window().maximize();
Delete all cookies: driver.manage().deleteAllCookies();

NAVIGATE COMMANDS

Navigate to url: driver.get("http://newexample.com"); driver.navigate().to("http://newexample.com");
Refresh Page: driver.navigate().refresh();
Navigate forwards in browser history: driver.navigate().forward();
Navigate backwards in browser history: driver.navigate().back();
Differences between get() and navigate().to():
1.in get(), it allows the website to fully load and then executes next line unlike navigate().to()
2.with navigation() we can back and forward while with get() we cant.

REFRESH A PAGE

```
driver.navigate().refresh();
sendKeys(Keys.F5)
driver.get(driver.getCurrentURL());
driver.navigate.to(driver.getCurrentURL());
```

FRAMES

Frame is a web page which is embedded in another web page. The Frame is often used to insert content from another source, such as an advertisement. The <iframe> tag specifies an inline frame.
How to Identify iframe:

- Right click on the element, If you find the option like 'This Frame' then it is an iframe
- Right click on the page and click 'View Page Source' and Search with the 'iframe', if you can find any tag name with the 'iframe' then it is meaning to say the page consisting an iframe.

How to switch between frames:

- **By Index:** driver.switchTo().frame(0); Not usually used. bcz either you may not know the index or it may change.
- **By Name or Id:** driver.switchTo().frame("iframe1");
- **By Web Element:** driver.switchTo().frame(WebElement);

How to switch back to the Main Frame:

```
driver.switchTo().parentFrame();
driver.switchTo().defaultcontent();
```

ALERTS

- Alert is a small message box which displays on-screen notification to give the user some kind of information or ask for permission to perform certain kind of operation
- JS Alert ---POP ups (Confirmation, warning,...), not inspectable
- Alert is an interface NOT Class
- Selenium doesn't support windows based applications. It is an automation testing tool which supports only web application testing. We could handle windows based popups in Selenium using some third party tools such as **AutoIT, Robot class** etc.
- switch from WebDirver to alert.

accept(): To click on the 'OK' button of the alert.
dismiss(): To click on the 'Cancel' button of the alert.
sendkeys(): To send some data to alert box.
gettext(): To capture the alert message.

```
Alert alert = driver.switchTo().alert();
String text = alert.getText();
System.out.println(text);
```

AUTHENTICATION POP UP

There are times that login page is under construction and a browserwindow pops up and asks for authentication. Since we cannot inspect the browser window, we put the username and pass in the weblink:
http://**username:password@**the-internet.herokuapp.com/basic_auth

HANDLE WINDOWS-BASED ALERTS/POP-UPS

Selenium only supports web applications and doesn't provide a way to automate Windows-based applications. However, we can use the **Robot class (Java-based) utility** to simulate the keyboard and mouse actions. That's how you can handle the window based pop.

HANDLE MULTIPLE BROWSER WINDOWS

Each window has its own url and window ID
driver.getWindowHandle(): This will handle the **current window** that uniquely identifies it within this driver instance. Its return type is String .

```
String MainWindow=driver.getWindowHandle();
```

driver.getWindowHandles();
Is used to handle **all opened windows** by web driver and then we can switch window from one window to another in a web application. Its return type is Iterator<String>

```
Set<String> handler = driver.getWindowHandles();
```

Set object stores data but not in form of indexes and stores data without any particular order.
To get data from Set we need Iterator to read.

```
Iterator<String> it = handles.iterator();

String parentWindowID = it.next();
System.out.println("parent windows id is: "+ parentWindowID);

String childWindowID = it.next();
System.out.println("child windows id is: "+ childWindowID);
driver.switchTo().window(childWindowID);//switch to child window
System.out.println("child window title is: "+ driver.getTitle());
driver.close(); //close the child window pop up
driver.switchTo().window(parentWindowID); //switch to parent window
System.out.println("parent window title is: "+ driver.getTitle());
```

UPLOADING FILES

Uploading files in WebDriver is done by simply using the **sendKeys("give the address of your file")** method on the file-select input field to enter the path to the file to be uploaded. The upload button should have **<input Type="File"** **.** If not, ask the developer to add it. This is the only time we use sendkey() for a button instead of click.

```
driver.findElement(By.name("upfile")).sendKeys
("C:\\Users\\Ramzy\\Desktop\\MethodOverloadingConcept.java");
```

CHROME OPTIONS CLASS

Chrome options class is used to manipulate various properties of Chrome driver

```
ChromeOptions options = new ChromeOptions()
options.addArgument("start-maximized");
ChromeDriver driver = new ChromeDriver(options);
```

- **start-maximized:** Opens Chrome in maximize mode
- **- - incognito:** Opens Chrome in incognito mode
- **- - headless:** Opens Chrome in headless mode

ACTION CLASS

- Is used to handling special keyboard and mouse events. First, create object of Actions class and pass **WebDriver**
  `Actions actions = new Actions(driver);`
- Actions commands always ends with >>>.**build().perform();**
- The following are the most commonly used keyboard and mouse events provided by the Actions class.
    - **click(element):** performs the click with action class. only pass the WebElement. this one moves to the element and perform action. Built in movetoElement feature.
    - **sendKeys(element," ").** The difference with normal sendKeys is that this one moves to the element and perform action. Built in movetoElement feature.
    - **contextClick()** : Performs Right Click
    - **doubleClick(element)** : Performs a double-click at the current mouse location.
    - **dragAndDrop(source, target)** : Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse
    - **moveToElement(element)**: Moves the mouse to the middle of the element. Used for dynamic drop down menus
    - **clickAndHold()** : Clicks (without releasing) at the current mouse location.
    - **release()** Releases the depressed left mouse button at the current mouse location

HANDLING STATIC DROPDOWN USING SELECT CLASS

- **Condition**: it should have **select** tag.
- Initially, we need to create class Object of Select. This class has no default constructor, so we need to pass WebElement
  `Select select = new Select(element);`
- How to create method of select class

```
public void selectValueFromDropDownByText(WebElement element, String value) {
        Select select = new Select(element);
        select.selectByVisibleText(value); }
```

- Methods available in select class:
    - **selectByVisibleText("One") / deselectByVisibleText():** selects/deselects an option by its displayed text (**preferred**)
    - **selectByIndex() / deselectByIndex():** selects/deselects an option by its index **isMultiple():** returns **TRUE** if the drop-down element allows multiple selection at a time; FALSE if otherwise
    - **deselectAll():** deselects all previously selected options
    - **getAllSelectedOptions( ):** Retrieves the list options that are currently selected in the Multi-Selection Box field and assign them to List<WebElement> type variable as below:

```
List<WebElement> selected_List_Options = select.getAllSelectedOptions();
```

- **getoptions():** perform as findElements in Select Class. when used with Drop Down field retrieves the list of all the options that are available in the Drop Down field.

```
List<WebElement> selected_List_Options = select.getOptions();
```

HANDLING DYNAMIC DROPDOWN

Initially, we get the custom Xpath for dropdown Box and store in List<WebElement>

```
List<WebElement> NoOfPassList = driver.findElements(By.xpath("Custom Xpath"));
```

To print the number of items available in the list:

```
System.out.println(NoOfPassList.size());
```

To select a specific item in the list we use loop concept: here number "4"

```
for(int i = 0; i<NoOfPassList.size(); i++ ) {
if(NoOfPassList.get(i).getText().equals("4")) {
NoOfPassList.get(i).click();
break;          } }
```

TAKE SCREENSHOT

- Test cases may fail while executing the test scripts. While we are executing the test cases manually we just take a screenshot and place in a result repository. Some of the scenarios may be:
    - Application issue
    - Assertion Failure
    - Difficulty to find Webelements on the web page
    - Timeout to find Webelements on the web page
- You can also take screenshot of a specific element in Selenium 4. you need to add a dependency.
- Selenium provides an interface called ***TakesScreenshot*** which has a metod ***getScreenShotAs*** which can be used to take a screenshot of the application under test.

```
public static void takePageScreenshot(WebDriver driver, String fileName){
        File SrcFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
        try {
        FileUtils.copyFile(SrcFile, new File("./target screenshots/"+fileName+".png"));
        } catch (IOException e) {
        e.printStackTrace();        } }
```

SELENIUM WEBDRIVER EXCEPTIONS

- **ElementNotVisibleException:** This exception will be thrown when you are trying to locate a particular element on webpage that is not currently visible even though it is present in the DOM. Also sometimes, if you are trying to locate an element with the xpath which associates with two or more element.
- **NoSuchElementException:** Element not present in DOM. Thrown by findElement()
- **StaleElementReferenceException** The two reasons for Stale element reference are The element has been deleted entirely. The element is no longer attached to the DOM. We face this stale element reference exception when the element we are interacting is destroyed and then recreated again. When this happens the reference of the element in the DOM becomes stale. Hence we are not able to get the reference to the element.
- **TimeoutException:** Thrown when a command does not complete in enough time. Such as explicit wait.
- **WebDriverException :** This Exception takes place when the WebDriver is acting right after you close the browser.
- **MoveTargetOutOfBoundsException:** The target provided to the action move() method is invalid—outside of the size of the window
- **NoAlertPresentException:** When user tries to access an alert when it is not present.
- **NoSuchFrameException:** Thrown by WebDriver.swithchTo().frame() when accessing frame which is not present.
- **NosuchSessionException:** Thrown by any command being called after WebDrier.quit() session Id will become null after this.
- **NoSuchWindowException:** Thrown by WebDriver.swithchTo().window() when accessing window which is not present;

A few Points:

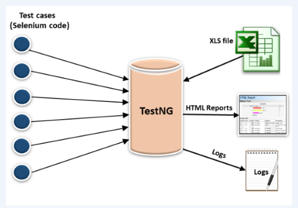
- **What is Automation Testing?** Automation testing is the process of testing a software or application using an automation testing tool to find the defects. In this process, executing the test scripts and generating the results are performed automatically by automation tools. It is required when we have huge amount of regression test cases. Some most popular tools to do automation testing are HP QTP/UFT, Selenium WebDriver, etc.,
- **What are the benefits of Automation Testing?**
    - Saves time and money. Automation testing is faster in execution.
    - Reusability of code. Create one time and execute multiple times with less or no maintenance.
    - Easy reporting. It generates automatic reports after test execution.
    - Easy for compatibility testing. It enables parallel execution in the combination of different OS and browser environments.
    - Low-cost maintenance. It is cheaper compared to manual testing in a long run.
    - Automated testing is more reliable.
    - Automated testing is more powerful and versatile. Automation tools allow us to integrate with Cross Browser Testing Tools, Jenkins, Github etc.,
    - It is mostly used for regression testing. Supports execution of repeated test cases.
    - Minimal manual intervention. Test scripts can be run unattended.
    - Maximum coverage. It helps to increase the test coverage.
- By using **JavascriptExecutor interface**, we could highlight the specified element
- **JavaScript scrollBy()** method scrolls the document by the specified number of pixels.
- **How To Resize Browser Window Using Selenium WebDriver?** To resize the browser window to particular dimensions, we use 'Dimension' class to resize the browser window
- **How to Download a file in Selenium WebDriver?** By using *AutoIT* script, we could download a file in Selenium WebDriver.
- **List some scenarios which we cannot automate using Selenium WebDriver?**
    - Automating Captcha is not possible using Selenium WebDriver
    - We can not read bar code using Selenium WebDriver
- **How do you read test data from excels?** Test data can efficiently be read from excel using JXL or POI API. POI API has many advantages than JXL.
- **How many test cases you have automated per day?** It depends on Test case scenario complexity and length. I did automate 1-2 test scenarios per day when the complexity is limited. Sometimes just 1 or fewer test scenarios in a day when the complexity is high.
- **What type of tests have you automated?** Our main focus is to automate test cases to do *Regression testing, Smoke testing, and Sanity testing.* Sometimes based on the project and the test time estimation, we do focus on End to End testing.
- **What Do You Use Selenium Grid?** Selenium Grid allows the test cases to run on multiple platforms and browsers simultaneously, and hence, supports distributed testing. This ability to parallelize the testing is what makes us use the Selenium Grid.
- **What is a Framework?** A framework defines a set of rules or best practices which we can follow in a systematic way to achieve the desired results. There are different types of automation frameworks and the most common ones are:
    - Data Driven Testing Framework
    - Keyword Driven Testing Framework
    - Hybrid Testing Framework
- **Have you created any Framework?**
    - *If you are a beginner:* You can say "No, I didn't get a chance to create framework from the scratch. I have used the framework which is already available. My contribution is mostly in creating test cases by using the existing framework."
    - *If you are a beginner but have good knowledge on creating framework:* You can say "Yes, I have involved in developing framework along with other automation tester in my company."
    - *If you are an experienced tester:* You can say "I have contributed in developing framework." or You can say "Yes, I have created framework from the scratch. There was no automation process in my previous company. I designed the framework from the scratch."

Generic Methods

```
public class Util {

        WebDriver driver;

        public Util(WebDriver driver) {
                this.driver = driver;          }

        This method is used to create the webElement on the basis of By locator.
        @param locator
        @return
        public WebElement getElement(By locator) {
                try {
                WebElement element = driver.findElement(locator);
                } catch (Exception e) {
                System.out.println("some exception occurred while creating the
                webelement....");          }
                return element; }

        public void waitForElementPresent(By locator, int timeOut) {
                WebDriverWait wait = new WebDriverWait(driver, timeOut);
                wait.until(ExpectedConditions.presenceOfElementLocated(locator));}

        public String waitForTitlePresent(String title, int timeOut) {
                WebDriverWait wait = new WebDriverWait(driver, timeOut);
                wait.until(ExpectedConditions.titleContains(title));
                return driver.getTitle();}

        This method is used to click on element
        @param locator
        public void doClick(By locator) {
                try {
                getElement(locator).click();
                } catch (Exception e) {
                System.out.println("some exception occurred while clicking on the
                webelement....");          }                }

        This method is used to pass the values in a webelement
        @param locator
        @param value
        public void doSendKeys(By locator, String value) {
                try {
                getElement(locator).sendKeys(value);
                } catch (Exception e) {
                        System.out.println("some exception occurred while
        passing value to the webelement...."); }                }
```

- You need to create the object of Util class in other classes and pass driver to call these generic methods.

TestNG



- **Open source testing framework used in automation testing**
- designed to simplify a broad range of testing needs, from unit testing to integration testing
- Provides you full control over the test cases and the execution of the test cases.
- TestNG framework came after Junit, and TestNG framework adds more powerful functionality and easier to use.
- In TestNG, NG stands for "**Next Generation**".

Advantages of TestNG

- Detailed (HTML) reports
- It also generates the Logs.
- TestNG does not extend any class. TestNG framework allows you to define the test cases where each test case is independent of other test cases.
- Parallel execution of test cases, i.e., running multiple test cases is only possible in the TestNG framework.
- It allows to assign priority to test methods
- It allows to define dependency of one test method over other method
- It allows grouping of test methods into test groups
- It has support for parameterizing test cases using @Parameters annotation
- It allows data driven testing using @DataProvider annotation
- It has different assertions that helps in checking the expected and actual results

Various ways in which TestNG can be invoked?

- TestNG can be invoked in the following ways
    - Using Eclipse IDE
    - Using ant build tool
    - From the command line
    - Using IntelliJ's IDEA

TestNG Installation and Configuration in Eclipse

- Copy link " https://beust.com/eclipse "
- In Eclipse >> Help >>install new software >> enter >> install
- Verify: right click on project>> check visibility of "TestNG"
- Add TestNG library to project path
- Add TestNG dependency to POM
- All test classes should be created under src/test/java **NOT** src/main/java
- No java main method is needed

TestNG Annotation

@BeforeSuite	launch browser	[pre-condition—runs once]
@BeforeTest	Launch URL	[pre-condition—runs once]
@BeforeClass	Login to App	[pre-condition—runs once]
@BeforeMethod	go to HomePage	[pre-condition—runs before @Test]
@Test	Home page header test	[Test case]
@AfterMethod	logout	[post– Condition—runs after @Test]
@AfterClass	go to login page	[post– Condition—runs once]
@AfterTest	disconnect with DB	[post– Condition—runs once]
@AfterSuite	close the browser	[post– Condition—runs once]

- All above annotations should be associated with a method.
- Flow of execution:
    - @BeforeMethod   go to HomePage
    - @Test   Home page header test
    - @AfterMethod   logout
    - .....
    - @BeforeMethod   go to HomePage
    - @Test   Home page footer test
    - @AfterMethod   logout
- When using @BeforeMethod and @AfterMethod, browser will be opened separately for each @test. For instance, if you have 20 test cases, browser will be launched 20 times. However, you can do all your 20 test cases in one browser by using @BeforeTest and @AfterTest. It is recommended that you use the former if you have many test cases because if something happens for any of test cases and browser crashes, the rest will NOT be executed.
- You can run each @Test individually by selecting it and run it as a TestNG

priority

When no 'priority' attribute is specified then the TestNG will run the test cases in alphabetical order. Priority determines the sequence of the execution of the test cases. The priority can hold the integer values between -5000 and 5000. When the priority is set, the lowest priority test case will run first and the highest priority test case will be executed last. Suppose we have three test cases and their priority values are 5000, 0, 15, then the order of the execution will be 0,15,5000. If priority is not specified, then the default priority will be 0.

```
@Test(priority=2)
public void apple()  {  System.out.println("I am Apple");  }
```

enabled

The 'enabled' attribute contains the boolean value. By default, its value is true. If you want to skip some test method, then you need to explicitly specify '**false**' value.

```
@Test(enabled=false)
public void jira()  {  System.out.println("JIRA is a testing tool");  }
```

Description

It is a string which is attached to the @Test annotation that describes the information about the test.

```
@Test(description="This is testcase1")
public void testcase1()  { }
```

dependsOnMethods

When the second test method wants to be dependent on the first test method, then this could be possible by the use of "**dependOnMethods**" attribute. If the first test method fails, then the dependent method on the first test method, i.e., the second test method will not run.

```
public void WebStudentLogin()  {
System.out.println("Student login through web");  }
@Test(dependsOnMethods= {"WebStudentLogin"})
public void APIStudentLogin()  { }
```

- **TestNG allows two ways to declare dependencies**
    - Using attributes dependsOnMethods in @Test annotations
    - Using attributes dependsOnGroups in @Test annotations

Groups

The 'groups' attribute is used to group the different test cases that belong to the same functionality. TestNG framework allows multiple tests to run by using the test group feature.

```
@Test(groups= {"regression-Testing", "smoke-Testing"})
public void infosys()  {  System.out.println("Infosys");  }

@Test(groups= {"regression-Testing", "smoke-Testing"})
public void wipro()  {  System.out.println("Wipro");  }
```

- Groups are specified in your **testng.xml** file and can be found either under the <test> or <suite> tag. Groups specified in the <suite> tag apply to all the <test> tags underneath.
- Groups can also include other groups (group of groups). These groups are called *MetaGroups*. For example, you might want to define a group *all* that includes *smokeTest* and *functionalTest*. Let's modify our testng.xml file as follows:

```
<groups>
  <define name="all">
 <include name="smokeTest"/>
 <include name="functionalTest"/>
  </define>
  <run>
      <include name="all" />
  </run>
</groups>
```

invocationCount

Is used to run a test case multiple times

```
@Test(invocationCount=10)
public void jira()  {  System.out.println("JIRA is a testing tool");  }
```

Assertions

- These commands allow adding a checkpoint or a verification point.
- You can have multiple assertions in a test
- **Hard Assertion:** It immediately responds with an **AssertException** and terminates the current test case. After that, the **next case in the sequence gets executed**.
    - Assert: it is an class already available in TestNG. No need to import

      `Assert.assertEquals(title, "HubSpot Login", "login page title is mismatched...");`
    - Some of the common assertions supported by TestNG are

```
assertEqual(String actual,String expected)
assertEqual(String actual,String expected, String message)
assertEquals(boolean actual,boolean expected)
assertTrue(condition)
assertTrue(condition, message)
assertFalse(condition)
assertFalse(condition, message)
```

- **Soft Assertion** It collects the errors that occurred during the test execution. Soft assert does **not throw an exception**. If such an assert fails, the control jumps to the next step.
    - SoftAssert: it is an class in TestNG. Should be imported

```
softAssert.assertEquals(title, "Reports dashboard123");
softAssert.assertAll();  //mandatory
```

- **Assert vs verify**:
    - Assert: It allows us to verify the result of an expression or an operation. If the "assert" fails, then it will abort the test execution and continues with the next case.
    - Verify: It also operates the same as the assert does. However, if the "verify" fails, then it won't abort the test instead continues with the next step. (if statements) - it cannot make the test case fail or pass

Parameterized testing in TestNG

- *Parameterized tests* allow developers to run the same test over and over again using different values.
- There are two ways to set these parameters:
    - using *testng.xml*
    - using *Data Providers*

Data Driven framework using TestNG

By using *@DataProvider* annotation, we can create a Data Driven Framework.

```
@DataProvider(name="getData")
 public Object[][] getData(){
 Object [][] data = new Object [2][2];

 data [0][0] = "FirstUid";
 data [0][1] = "FirstPWD";

 data[1][0] = "SecondUid";
 data[1][1] = "SecondPWD";

 return data; }
```

TestNG.xml

- In TestNG framework, we need to create ***testng.xml*** file to create and handle multiple test classes. We do configure our test run, set test dependency, include or exclude any test, method, class or package and set priority etc in the xml file.
- In POM, it is called TestRunner
    - Create a new "Source Folder" >> src/test/resources   >>create a packate >> create a file with extension .**xml** >> right click on the file and run it as TestNG
    - You only need to add the classes you would like to execute
    - You can have different .xml for regression, sanity or smoke tests separately.
    - Copy and paste the codes

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="Hub Spot Sanity Test Suite" verbose="4">
        <listeners>
        <listener class-name="com.qa.hubspot.listeners.ExtentReportListener" />
        <listener class-name="com.qa.hubspot.listeners.TestAllureListener" />
        </listeners>

        <test name="hub spot sanity Test">
        <classes>
                <class name="com.qa.hubspot.tests.LoginPageTest" />
                <!-- <class name="com.qa.hubspot.tests.HomePageTest" /> -->
        </classes>
        </test>
</suite>
```

- **How to exclude a particular test method from a test case execution?** By adding the exclude tag in the *testng.xml*

```
<classes>
 <class name="TestCaseName">
   <methods>
     <exclude name="TestMethodNameToExclude"/>
   </methods>
 </class>
</classes>
```

- **How to exclude a particular test group from a test case execution?** By adding the exclude tag in the *testng.xml*

```
<groups>
  <run>
 <exclude name="TestGroupNameToExclude"/>
  </run>
</groups>
```

- **Importance of TestNG.xml file?** In a Selenium TestNG project, we use *testng.xml* file to configure the complete test suite in a single file. Some of the features are as follows.
    - testng.xml file allows to include or exclude the execution of test methods and test groups
    - It allows to pass parameters to the test cases
    - Allows to add group dependencies
    - Allows to add priorities to the test cases
    - Allows to configure parallel execution of test cases
    - Allows to parameterize the test cases
- **How to pass parameter through *testng.xml* file to a test case?**
    - We could define the parameters in the *testng.xml* file and then reference those parameters in the source files.
    - Create a java test class, say, *ParameterizedTest.java* and add a test method say *parameterizedTest()* to the test class. This method takes a string as input parameter. Add the annotation @Parameters("browser") to this method.

```
public class ParameterizedTest {
 @Test
 @Parameters("browser")
 public void parameterizedTest(String browser){
 if(browser.equals("firefox")){
 System.out.println("Open Firefox Driver");
 }else if(browser.equals("chrome")){
 System.out.println("Open Chrome Driver");     }  } }
```

- The parameter would be passed a value from testng.xml, which we will see in the next step.
- We could set the parameter using the below syntax in the *testng.xml file.*

  `<parameter name="browser" value="firefox"/>`
- Here, name attribute represents the parameter name and value represents the value of that parameter.

Reporting

- We specify the time unit in test suites and test cases is in milliseconds.
- **Extent Report**: with help of testNG listener they are listening the state of each test case. They will capture those information and generate reports. To run the testNG listener we need to:
    - Add dependencies
    - Use listener template … goes to src/main/java
        - You can bring minor changes like file name or folder directory.
    - Add this code to your TestNG.xml file

```
<listeners>
        <listener class-name="com.qa.hubspot.listeners.TestAllureListener" />

</listeners>
```

- Refresh your project.
- Report will be generated in folder you already specified with html extension. Copy the address and paste it into a browser.
- It takes screenshot when a test case is failed.
- You can customize your extend report by modifying listener.

- **Index.html:** this report is generated by TestNG. It will not take screenshot.
- **Emailable-report.html**: not preferred.

21. How to skip a @Test method from execution in TestNG?
By using *throw new SkipException()*
Once *SkipException()* thrown, remaining part of that test method will not be executed and control will goes directly to next test method execution.
throw new SkipException("Skipping - This is not ready for testing ");
24. What are the different ways to produce reports for TestNG results?
TestNG offers two ways to produce a report.
Listeners implement the interface *org.testng.ITestListener* and are notified in real time of when a test starts, passes, fails, etc…
Reporters implement the interface *org.testng.IReporter*and are notified when all the suites have been run by TestNG. The IReporter instance receives a list of objects that describe the entire test run.
25. What is the use of @Listener annotation in TestNG?
TestNG listeners are used to configure reports and logging. One of the most widely used listeners in testNG is *ITestListener* interface. It has methods like *onTestStart, onTestSuccess*, *onTestFailure*, *onTestSkipped* etc. We should implement this interface creating a listener class of our own. Next we should add the listeners annotation (*@Listeners*) in the Class which was created.
26. How to write regular expression In testng.xml file to search @Test methods containing "smoke" keyword.
Regular expression to find @Test methods containing keyword "smoke" is as mentioned below.

```
<methods>
   <include name=".*smoke.*"/>
</methods>
```

30. What is the use of @Test(threadPoolSize=x)?
The *threadPoolSize* attribute tells to form a thread pool to run the test method through multiple threads.
Note: This attribute is ignored if invocationCount is not specified
@Test(threadPoolSize = 3, <code class="plain">invocationCount = </code><code class="value">10</code>) public void testCase1(){
In this example, the method *testCase1* will be invoked from three different threads
31. What does the test timeout mean in TestNG?
The maximum number of milliseconds a test case should take.
@Test(threadPoolSize = 3, invocationCount = 10, timeOut = 10000)
public void testCase1(){
In this example, the function testCase1 will be invoked ten times from three different threads. Additionally, a time-out of ten seconds guarantees that none of the threads will block on this thread forever.
32. What are @Factory and @DataProvider annotation?
@Factory: A factory will execute all the test methods present inside a test class using a separate instance of the respective class with different set of data.
@DataProvider: A test method that uses DataProvider will be executed the specific methods multiple number of times based on the data provided by the DataProvider. The test method will be executed using the same instance of the test class to which the test method belongs.
11. What is exception test in TestNG?
TestNG gives an option for tracing the Exception handling of code. You can verify whether a code throws the expected exception or not. The expected exception to validate while running the test case is mentioned using the ***expectedExceptions*** attribute value along with @Test annotation.
17. How to run test cases in parallel using TestNG?
we can use "parallel" attribute in testng.xml to accomplish parallel test execution in TestNG
The parallel attribute of suite tag can accept four values:
tests – All the test cases inside <test> tag of testng.xml file will run parallel
classes – All the test cases inside a java class will run parallel
methods – All the methods with *@Test* annotation will execute parallel
instances – Test cases in same instance will execute parallel but two methods of two different instances will run in different thread.
<suite name="softwaretestingmaterial" parallel="methods">