

# Chapter 1

## Introduction to selenium

---

### **W**hat is Selenium?

The Selenium is the most popular testing tool in automation . across different browsers and platforms Selenium is a free (open source) automated testing suite for web applications .Selenium supports web based applications testing on wide range of browsers and platforms. Since it is open source it has become one of the most accepted tools amongst the testing professionals.

### **Advantages of selenium**

- it is free and open source
- Supports execution of repeated test cases
- It helps us in testing a large test matrix
- Enables parallel execution
- Unattended execution can be easily detected

- 
- Reduces human generated errors & improves accuracy
  - Selenium supports various languages that include Java, Perl, Python, C#, Ruby, Groovy, Java Script, and VB Script. etc.
  - Selenium supports different operating systems like Windows, Macintosh, Linux, Unix etc.
  - Selenium supports different browsers like Internet explorer, Chrome, Firefox, Opera, Safari etc. Saves time and money

### **Disadvantages of selenium**

- Selenium needs experienced resources.
- Selenium does not support windows based application , it only supports web based applications
- It is quite difficult to test Images in the application.
- It takes more time to create Selenium script.
- CAPTCH and Bar code cannot be tested using Selenium.
- to generate Reports we need to depend on third party tools like TestNG or Junit.
- Since Selenium is a free tool, there is no direct technical support, we need to rely on its communities and forums.
- prior programming language knowledge is needed in order to implement the selenium. Why the name selenium

### **Why the name Selenium :**

Selenium was invented by Jason Huggins in 2004 in Thought works. prior to Selenium there was a tool by

### 3 Fundamentals of Selenium WebDriver

---

Mercury and it was paid. now if you know the periodicity of the element you might observe the Mercury and Selenium are the element of the university included in the periodicity of elements. Mercury has a poisonous nature and in order to get rid of this poison the element Selenium can help us, so actually it was a joke made by Jason Huggins email that in order to remove the poison by Mercury we will need Selenium supplements



# Chapter 2

## Selenium Basic Methods

---

- **Opening a browser** : you should download the particular driver and keep it into your project first. for example if you are using chrome browser you need to use chromedriver. (given on selenium download page)
- `System.setProperty("webdriver.chrome.driver","path-to-chrome-driver");`
- `WebDriver driver= new ChromeDriver();`
- **Maximize** : to maximize the window use
- `driver.manage().window().maximize();`
- **get()** :It will load a new web page in the current browser window.
- e.g. `driver.get("http://gmail.com")`
- **getCurrentUrl()**: Gets a string representing the current URL that the browser is opened.
- e.g. `driver.getCurrentUrl();`
- **getTitle()**: Gets the title of the current web page.
- e.g. `driver.getTitle();`

- 
- **findElements()** : This method finds all elements within the current page using the given locator
  - **findElement()**: This method find the first WebElement using the given locator.
  - **getPageSource()**:Get the source of the currently loaded page.
  - **To(URL)** : This methods works like get() method only but it is generally used for redirecting to another link on the same page e.g.

```
System.setProperty("webdriver.chrome.driver","path-to-chrome-driver");
```

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

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

- **Back()** : To move back a single step in the web browser's history

e.g.

```
System.setProperty("webdriver.chrome.driver","path-to-chrome-driver");
```

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

```
String URL="yahoo.com"; driver.navigate().to(URL);
driver.findElement(By.linkText("Mail")).click();
driver.navigate().back();
```

- **Forward()** : To move a single step forward in the web browser's history.

```
System.setProperty("webdriver.chrome.driver", "path-to-chrome-driver");
```

```
WebDriver driver= ChromeDriver();  
driver.get("http://gmail.com");
```

```
String URL="http://yahoo.com";  
driver.navigate().to(URL);  
driver.navigate().back();  
driver.navigate().forward();
```

- **Refresh()** : It refreshes the current web page

```
System.setProperty("webdriver.chrome.driver", "path-to-chrome-driver")
```

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

```
String URL="https://login.yahoo.com/";
```

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

```
driver.findElement(By.linkText("Sign up")).click();
```

```
Thread.sleep(1000);
```

```
driver.findElement(By.id("username-reg-firstName")).sendKeys("amol");
```

```
driver.navigate().refresh();
```

- **close()** : Close the current window, if there are multiple windows, it will close the current window which is active and quits the browser if it's the last window opened currently.

---

e.g. `driver.close();`

- **quit():** Quits this driver instance, closing every associated window which is opened.

e.g. `driver.quit();`



# Chapter 3

## Locators in Selenium

---

To locate a web element we use `findElement` method e.g. `driver.findElement(By.id("idname"))`. Here the `id` is a locator there total 8 locators supported by selenium



Locator	HTML Tag	Accessing using selenium
id	<code>&lt;input type="text" id="email" /&gt;</code>	<code>WebElement Element = driver.findElement(By.id("email"));</code>
name	<code>&lt;input type="text" name="email" /&gt;</code>	<code>WebElement Element = driver.findElement(By.name("email"));</code>
LinkText	<code>&lt;a href = "<u>http://google.com</u>" &gt; My link &lt;/a&gt;</code>	<code>WebElement Element = driver.findElement(By.linkText("My Link"));</code>
Partial	<code>&lt; a href=<u>http://google.com</u>&gt; My link &lt;/a&gt;</code>	<code>WebElement Element = driver.findElement(By.partialLinkText ("My "));</code>





LinkText		
TagName	<code>&lt;input type="text" name="email" /&gt;</code>	<code>WebElement Element = driver.findElement(By.tagName ("input"));</code>
ClassName	<code>&lt;input type="text" name="email" class="css"/&gt;</code>	<code>WebElement Element = driver.findElement(By.className ("css"));</code>
css selector	<code>css=input[id=email]</code>	<code>WebElement Element = driver.findElement(By.cssSelector("input[id=email]"));</code>
xpath	<a href="#"><u>Click here for xpath tutorial</u></a>	<code>WebElement Element = driver.findElement(By.xpath("//tr/td "));</code>

# Chapter 4



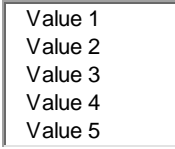
## Operations on HTML Elements.

In every web application there are inputs tags, Selects Options, Checkboxes ,Text areas links & buttons. lets see how to interact with all these HTML elements (Web Controls). WebDriver treats all those elements as a WebElement.

Sr. No.	Name	Control	HTML Syntax
1	Hyper Link	<u>My Simple Link</u>	<code>&lt;a href="#"&gt;My Simple Link&lt;/a&gt;</code>
2	Input Button		<code>&lt;input id="myInpButton" type="button" value="Click Me" /&gt;</code>
3	Button		<code>&lt;button id="myButton"&gt;Click Me&lt;/button&gt;</code>

4	Submit Button		<pre>&lt;input id="mySubmit" type="Submit" /&gt;</pre>
5	Text Box		<pre>&lt;input type="text" id="txt" value="Some Text" /&gt;</pre>
6	Password		<pre>&lt;input type="password" id="pass" value="somepass" /&gt;</pre>
7	Multiline Textbox		<pre>&lt;textarea id="tArea"&gt;this is Firstline.  This is Secondline&lt;/textarea&gt;</pre>
8	Checkbox	<div><input type="checkbox"/> checkbox 1</div> <div><input type="checkbox"/> checkbox 2</div>	<pre>&lt;input name="check1" type="checkbox" /&gt; checkbox 1  &lt;input name="check2" type="checkbox" /&gt; checkbox 2</pre>
9	Radio	<div><input type="radio"/> Option 1</div>	<pre>&lt;input name="options"</pre>

## 13 Fundamentals of Selenium WebDriver

	Button	 Option 2	<pre>type="radio"  value="option1"/&gt;Option 1  &lt;input      name="options" type="radio"  value="option1"/&gt;Option 2</pre>
10	Combo Box		<pre>&lt;select id="mycombo"&gt;  &lt;option&gt;Value 1&lt;/option&gt;  &lt;option&gt;Value 2&lt;/option&gt;  &lt;option&gt;Value 3&lt;/option&gt;  &lt;/select&gt;</pre>
11	Multi Select ListBox		<pre>&lt;select multiple="multiple" size="5"  id="myList"&gt;  &lt;option&gt;Value 1&lt;/option&gt;  &lt;option&gt;Value 2&lt;/option&gt;</pre>

			<pre>&lt;option&gt;Value 3&lt;/option&gt; &lt;option&gt;Value 4&lt;/option&gt; &lt;option&gt;Value 5&lt;/option&gt; &lt;/select&gt;</pre>
--	--	--	---

While testing we need to do several operations on these web elements like click on link or buttons entering values to textboxes or text areas selecting a value from drop down list or selecting checkbox or radio button. Let's see one by one.

1. **Hyper link** : to click on this controls we can use **click()** Method . below is the way to click on hyperlink.

```
System.setProperty("webdriver.chrome.driver","path-to-chrome-driver")
WebDriver driver=ChromeDriver();

driver.get("http://scriptinglogic.com/software-testing/selenium/operations-on-html-elements/");
driver.findElement(By.id("link")).click();
```

2. **Input Button** : to click on this controls we can use **click()** Method. below is the way to click on Input Button.

```
System.setProperty("webdriver.chrome.driver", "path-to-chrome-driver")
```

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

```
driver.get("http://scriptinglogic.com/index.php/software-testing/selenium/operations-on-html-elements/");  
driver.findElement(By.id("button")).click();
```

**3. Button :** to click on this controls we can use **click()** Method. below is the way to click on tag name Button.

```
System.setProperty("webdriver.chrome.driver", "path-to-chrome-driver")  
WebDriver driver= ChromeDriver();  
driver.get("http://scriptinglogic.com/index.php/software-testing/selenium/operations-on-html-elements/");  
driver.findElement(By.id("mybutton")).click();
```

**4. Submit Button :** to click on this controls we can use **click()** below is the way to click on Input Button.

```
System.setProperty("webdriver.chrome.driver", "path-to-chrome-driver")  
WebDriver driver= ChromeDriver();  
  
driver.get("http://scriptinglogic.com/index.php/software-testing/selenium/operations-on-html-elements/");  
driver.findElement(By.id("mySubmit")).click();
```

Now to check whether the web element is enabled or not we use **isEnabled()** method it returns true if the element is enabled and false if not enabled. lets see below example.

```
System.setProperty("webdriver.chrome.driver","path-to-chrome-driver")
```

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

```
driver.get("http://scriptinglogic.com/index.php/software-testing/selenium/operations-on-html-elements/");  
if(driver.findElement(By.id("myButton")).isEnabled()){  
driver.findElement(By.id("myButton")).click(); }
```

### **Text Box | Password | Multi line Text Box**

For these web elements we need to write or enter some value or text inside it. we can achieve this using sendkeys("string to be set") method. lets see below code

```
System.setProperty("webdriver.chrome.driver","path-to-chrome-driver")
```

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

```
driver.get("http://scriptinglogic.com/index.php/software-testing/selenium/operations-on-html-elements/");
```

```
// Enter the text in text box
```

```
driver.findElement(By.id("txt")).sendKeys("hi text");
```

```
// Enter the text in Password
```

```
driver.findElement(By.id("pass")).sendKeys("password");
```



```
// Enter the text inside multiline text box
```

```
driver.findElement(By.id("tArea")).sendKeys("This is line  
no.1\nThis is line no. 2");
```

Now to get the value from above HTML elements we have a method `getText()`. `clear()` Method clears the text. let's see how it works

```
System.setProperty("webdriver.chrome.driver", "path-to-  
chrome-driver")
```

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

```
driver.get("http://scriptinglogic.com/index.php/software-  
testing/selenium/operations-on-html-elements/");
```

```
//Printing the text inside the text field  
System.out.println(driver.findElement(By.id("txt")).getTex  
t());
```

```
// Clear the Text field
```

```
driver.findElement(By.id("pass")).clear();
```

### Check Box | Radio Button

check box and radio button are the web elements where we do click operation so we use `click()` Method.

to verify the CheckBox is checked we use `.isSelected()` Method

same is true for Radio buttons.

```
System.setProperty("webdriver.chrome.driver","path-to-chrome-driver")
```

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

```
driver.get("http://scriptinglogic.com/index.php/software-testing/selenium/operations-on-html-elements/");
```

```
//Check the checkbox
```

```
driver.findElement(By.name("check1")).click();
```

```
//click on the checkbox if it is not checked
```

```
if(driver.findElement(By.name("check2")).isSelected())
```

```
{ driver.findElement(By.name("check2")).click();
```

```
}
```

### **Combo Box & Multi Select ListBox**

to select the specific option from Combo Box & Multi Select ListBox Selenium WebDriver has a Special class `org.openqa.selenium.support.ui.Select`, we need to import this class lets see how to select a particular option.

```
System.setProperty("webdriver.chrome.driver","path-to-chrome-driver")
```

---

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

```
driver.get("http://scriptinglogic.com/index.php/software-  
testing/selenium/operations-on-html-elements/");
```

```
-----
```

```
// For combo box using selectByVisibleText
```

```
Select          comboBox          =          new  
Select(driver.findElement(By.id("myCombo")));  
comboBox.selectByVisibleText("Value 2");
```

```
-----
```

```
// For combo box using selectByValue
```

```
Select comboBox = new  
Select(driver.findElement(By.id("myCombo")));  
comboBox.selectByValue("value2");
```

```
// value inside value tag
```

```
-----
```

```
// For combo box using selectByIndex
```

```
Select comboBox = new  
Select(driver.findElement(By.id("myCombo")));  
  
comboBox.selectByIndex(2);
```

```
// index of the option
```

```
// index starts from 0
```

-----

```
// For Multi select list
```

```
Select listBox = new  
Select(driver.findElement(By.id("myList")));  
listBox.selectByVisibleText("Value 1");
```

```
listBox.selectByVisibleText("Value 2"); // deselect the  
specific item
```

```
listBox.deselectByValue("Value 1"); // De-select all the  
items from the list
```

```
listBox.deselectAll();
```

# Chapter 5

## Understanding XPATH

---

### **Xpath**

XPath is used to navigate through elements and attributes in an XML document. XPath is designed to allow the navigation of XML documents, with the purpose of selecting individual elements, attributes, or some other part of an XML document for specific processing.

### **What is XML?**

The Extensible Markup Language (XML) is the context in which the XML Path Language, XPath, exists. XML provides a standard syntax for the markup of data and documents. XML documents contain one or more elements. If an element contains content, whether other elements or text, then it must have a start tag and an end tag. The text contained between the start tag and the end tag is the element's content.

### **Absolute XPath**

Absolute XPath starts with the root node or a forward slash (/).

**Example:**

If the Path we defined as

1. `html/head/body/table/tbody/tr/th`

If there is a tag that has added between body and table as below

2. `html/head/body/form/table/tbody/tr/th`

The first path will not work as 'form' tag added in between

**Relative Xpath**

A relative xpath is one where the path starts from the node of your choice – it doesn't need to start from the root node.

It starts with Double forward slash(//)

**Syntax:**

`//table/tbody/tr/th`

**Selecting Elements:**

XPath uses path expressions to select nodes in an Web document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Sr. No.	Expression	Description
1	nodename	Selects all nodes with the name "nodename" Example div, a, input, button etc.
2	Unknown Elements	XPath wildcard character * can be used to select unknown Web elements.

## 23 Fundamentals of Selenium WebDriver

3	/	Selects child from the current node
4	//	Selects Descendants from the current element that match the selection no matter where they are
5	.	Selects the current node
6	..	Selects the parent of the current node
7	@	Selects attributes

### Predicates

Predicates are used to find a specific node or a node that contains a specific value [and/or based on some condition]. Predicates are always written inside square brackets.

Sr. No.	Path Expression	Result
1	//div[1]	Selects the first div element from the root
2	//div[@id='mainMenu']	Selects all the <b>div</b> having id as <b>mainMenu</b>























3	last()	Matches the Last position from the list
	//li[last()]	Selects the last <b>li</b> from list of <b>li</b>
4	text()	Get the text node of element
	//a[text()='Home']	Selects all the <b>a</b> (anchor) having text <b>Home</b>
5	position()	Matches the particular position from the list
	//li[position()<3]	Selects the first two <b>li</b> from list of <b>li</b>
6	contains(string1,string2)	Matches if the string2 contains in string1
7	starts-with(string1,string2)	Matches if the String1 starts with String2
8	ends-with(string1,string2)	Matches if the String1 ends with String2



9	matches(string,pattern)	Matches if the String1 match with given regular expression pattern
10	upper-case(string)	Convert the string to Uppercase
11	lower-case(string)	Convert the String to Lowercase

### Selecting a specific row of a table.

You may come across a situation where you might need to click (please refer below screen shot) on either edit or delete button of a row in a table. It might depend upon specific id or some name.

#	Function	First Name	Surname 1	Surname 2	Login	Type
1.	  	Aakansha	Yadav	Pavshe	aakansha	Doctor
2.	  	Benito	Camelas	Unmont�n	benito	Administrative
3.	  	Deepak	sharma		deepak2793	Doctor
4.	 ** 	John	Doe	Smith	admin	Administrative
5.	  	RENU	KHATE	JADHAV	renuka	Administrative
6.	  	renu	khate	JADHAV	jyop	Administrative
7.	  *	Renuka	khate			Administrative
8.	  	Renuka	Khate	khate	renu	Administrative

**Step 1:** first select the table say you have this table , like  
 //table (if it is the only table, if there are other table then use  
 predicates for selecting this table e.g.  
 //table[@class='classname'])

**Step 2:** then use //tr to select all the tr in this table. You  
 must use // because there could be a node th or tbody, in  
 order to skip those use // and not /

**Step 3:** Now say I want to select a row whose login name  
 is deepak2793. Looking at step 1 & to we have come up to  
 xpath //table//tr. It will select all the row of the table now to  
 select the specific row use  
 //table//tr[td[text()='deepak2793']] it will select the row  
 whose login name is deepak2793.

**Step 4:** now to select an icon of edit/delete simply append  
 td[2] to //table//tr[td[text()='deepak2793']] for edit and  
 td[3] to //table//tr[td[text()='deepak2793']] for delete

So final xpath becomes

**//table//tr[td[text()='deepak2793']]//td[2] for edit**

**//table//tr[td[text()='deepak2793']]//td[3] for delete**

# Chapter 6

## Junit

---

**G**enerally the entry point of any JAVA program is main function. and it is inside a class. in selenium the entry point can be any function having annotation @Test. consider below example. where in driver is initialized, then browser is maximized and a url is opened.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.junit.Test;
```

```
public class MyTest
{
```

```
    @Test
    public void LoginTest()
    {
```

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

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

---

```
driver.get("http://clinic.scriptinglogic.com/auth/login_form.php");  
}  
}
```

## What is junit

JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development. JUnit is perfect unit test framework for java programming language. Open source Java testing framework used to write and run repeatable automated tests.

```
/*
```

```
* This class prints the given message on console.
```

```
*/
```

```
public class MessageUtil {
```

```
private String message;
```

```
//Constructor
```

```
//@param message to be printed
```

```
public MessageUtil(String message){
```

```
this.message = message;
```

```
}
```

```
// prints the message

public String printMessage(){

System.out.println(message);

return message;

}

}
```

### Create Test Case Class

```
import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class TestJunit {

String message = "Hello World";

MessageUtil messageUtil = new MessageUtil(message);

@Test

public void testPrintMessage()

{

assertEquals(message,messageUtil.printMessage());

}
```

---

```
}
```

### Annotations in JUnit –

- @Before –

```
public void method()
```

The @Before annotation indicates that the attached method will be run **before** any test in the class. It is mainly used to setup some objects needed by your tests.

- @BeforeClass –

```
public static void method()
```

The BeforeClass annotation indicates that the static method to which is attached must be executed once and before all tests in the class. That happens when the test methods share computationally expensive setup (e.g. connect to database).

- @After –

```
public void method()
```

Method that is marked with @**After** gets executed after execution of every test. If we need to reset some variable after execution of every test then this annotation can be

used with a method that has the needed code. It is used to cleanup the test environment (e.g., delete temporary data, restore defaults).

- `@AfterClass` –

`public static void method()`

In the same way “***@AfterClass***” annotation can be used to execute a method that needs to be executed after executing all the tests in a JUnit Test Case class. It is used to perform clean-up activities, for example, to disconnect from a database.

```
import org.junit.*;

import static org.junit.Assert.*;

import java.util.*;

public class JunitTest1 {

    private Collection collection;

    @BeforeClass
```

---

```
public static void oneTimeSetUp() {
```

```
// one-time initialization code
```

```
System.out.println("@BeforeClass - oneTimeSetUp");
```

```
}
```

```
@AfterClass
```

```
public static void oneTimeTearDown() {
```

```
// one-time cleanup code
```

```
System.out.println("@AfterClass - oneTimeTearDown");
```

```
}
```

```
@Before
```

```
public void setUp() {
```

```
collection = new ArrayList();
```

```
System.out.println("@Before - setUp");
```

```
}
```

```
@After
```

```
public void tearDown() {
```



```
collection.clear();
```

```
System.out.println("@After - tearDown");
```

```
}
```

```
@Test
```

```
public void testEmptyCollection() {
```

```
    assertTrue(collection.isEmpty());
```

```
    System.out.println("@Test - testEmptyCollection");
```

```
}
```

```
@Test
```

```
public void testOneItemCollection() {
```

```
    collection.add("itemA");
```

```
    assertEquals(1, collection.size());
```

```
    System.out.println("@Test - testOneItemCollection");
```

```
}
```

```
}
```

**OUTPUT –**

@BeforeClass – oneTimeSetUp

---

@Before – setUp

@Test – testEmptyCollection

@After – tearDown

@Before – setUp

@Test – testOneItemCollection

@After – tearDown

@AfterClass – oneTimeTearDown

### **What is Assert?**

JUnit provides static methods in the Assert class to test for certain conditions. These *assert statements* typically start with assert and allow you to specify the error message, the expected and the actual result. An *assertion method* compares the actual value returned by a test to the expected value, and throws an AssertionError if the comparison test fails.

### **EXAMPLE –**

```
import com.sun.webkit.Disposer;
import org.junit.Assert;
import org.junit.Test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.ie.InternetExplorerDriver;

import java.util.concurrent.TimeUnit;

public class MyJUnit {
    @Test
    public void loginTest()
    {
```

```

System.setProperty("webdriver.ie.driver",
"IEDriver/IEDriverServer.exe");
WebDriver driver = new InternetExplorerDriver();
driver.manage().window().maximize();

driver.get("http://localhost/openclinic/auth/login_form.php");
driver.findElement(By.xpath("//input[@id='login_session']"))
).sendKeys("admin");
driver.findElement(By.xpath("//input[@id='pwd_session']"))
).sendKeys("admin1");
driver.findElement(By.xpath("//input[@id='login']")).click();
driver.manage().timeouts().implicitlyWait(20,
TimeUnit.SECONDS);

boolean result=false;
try {
    result =
driver.findElement(By.xpath("//a[@href='../auth/logout.php']"))
).isDisplayed();
}

catch(Throwable t)
{
    result=false;
}

Assert.assertEquals("Test failed",true,result);

}

}

```

OUTPUT –

---

java.lang.AssertionError: Test failed

Expected :true

Actual :false

# Chapter 7

## TestNG

---

### 1. What is TestNG

**Ans.** – TestNG is an open source automated frame work used to drive all kinds of testing like Functional, unit, integration, system, regression..etc. NG stands for next generation

- Its build on the base line of JUnit and NUnit which were previously used as frame works.
- TestNG overcomes limitations of JUnit like poor configuration control,static programming model (forces you to recompile unnecessarily),the management of different suites of tests in complex projects can be very tricky..etc

Some features of TestNG are listed as follows-

- 1.Supports annotations.

2. Uses OOP concepts like interfaces, inheritance, abstraction, polymorphism..etc

3. Allows separation of compile-time test code from run-time configuration/data info.

4. Flexible plug-in API.

5. Support for multi threaded testing.

6. Grouping of tests.

## 2. **Explain with example**

@BeforeSuite

@AfterSuite

@BeforeTest

@AfterTest

@BeforeGroups

@AfterGroups

@BeforeClass

@AfterClass

@BeforeMethod

@AfterMethod

Ans.

Annotations – Annotations are used in TestNG to schedule the test methods, i.e. it gives user the facility to determine which test in the test class(test suite) should run first and then the other tests as desired.

@BeforeSuite – The annotated method will be run only once before all tests in this suite have run.

@AfterSuite – The annotated method will be run only once after all tests in this suite have run.

@BeforeTest – The annotated method will be run only once before the first test method in the current class is invoked.

@AfterTest – The annotated method will be run only once after all the test methods in the current class have run.

---

### @BeforeGroups –

The list of groups that this configuration method will run before. This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked.

### @AfterGroups

- The list of groups that this configuration method will run after. This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

@BeforeClass – The annotated method will be run only once before the first test method in the current class is invoked.

@AfterClass – The annotated method will be run only once after all the test methods in the current class have run.

@BeforeMethod – The annotated method will be run before each test method.

@AfterMethod – The annotated method will be run after each test method.

For E.g



```
public class TestClass1 {

    @BeforeSuite

    public void suiteSetup1() {

        System.out.println("testClass1.suiteSetup1:    before
suite");

    }

    @BeforeTest

    public void beforeTest() {

        System.out.println("testClass1: before test");

    }

    @Test

    public void unitLevel1() {
```

---

```
        System.out.println("testClass1: Unit level1 testing");

    }

    @Test

    public void unitLevel2() {

        System.out.println("testClass1: Unit level2 testing");

    }

    @BeforeMethod

    public void beforeMethod() {

        System.out.println("testClass1: before method");

    }
```

@AfterMethod

```
public void afterMethod() {  
  
    System.out.println("testClass1: after method");  
  
}
```

@BeforeClass

```
public void beforeClass() {  
  
    System.out.println("testClass1: before class");  
  
}
```

@AfterClass

```
public void afterClass() {  
  
    System.out.println("testClass1: after class");  
  
}
```

```
}
```

```
@AfterSuite
```

```
public void cleanupSuite() {
```

```
    System.out.println("testClass1.cleanupSuite:    after  
suite");
```

```
}
```

```
}
```

**3.What is priority how it is implemented explain with example**

**Ans** – In TestNG “Priority” is used to schedule the test cases. When there are multiple test cases, we want to execute test cases in order. Like First we need to execute a test case “Registration” before login.

E.G

```
import org.testng.annotations.Test;
public class testNGPriorityExample {
    @Test
    public void registerAccount()
    {
        System.out.println("First register your account");
    }
    @Test(priority=2)
    public void sendEmail()
    {
        System.out.println("Send email after login");
    }
}
```

#### 4.What are groups explain with example

**Ans.** TestNG allows us to perform sophisticated groupings of test methods.

Using TestNG we can execute only set of groups while excluding another set. This gives us the maximum flexibility in dividing tests and doesn't require us to recompile anything if you want to run two different sets of tests back to back.

Groups are specified in testng.xml file and can be used either under the `<or>` tag. Groups specified in the tag apply to all the tags underneath.

E.G

```
package com.example.group;
import org.testng.annotations.Test;
```

---

```
public class groupExamples {
    @Test(groups="Regression")
    public void testCaseOne()
    {
        System.out.println("Im in testCaseOne – And in Regression
        Group");
    }
    @Test(groups="Regression")
    public void testCaseTwo(){
        System.out.println("Im in testCaseTwo – And in
        Regression Group");
    }
}
```

## 5. How the annotation @parameter is implemented

**Ans.** TestNG allows the user to pass values to test methods as arguments by using parameter annotations through testng.xml file.

```
package com.parameterization;
```

```
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;
```

```
public class TestParameters {
```

```
    @Parameters({ "browser" })
```

```
    @Test
```

```
    public void testCaseOne(String browser) {
```

```
        System.out.println("browser passed as :- " + browser);
```

```
    }
```

6. **Explain testng.xml ,different tags used in it and their significance with example.**

**Ans:-**In testng.xml file we can specify multiple name (s) which needs to be executed if any project

there may be many classes, but we want to execute only the selected classes.

The below is the example testng.xml which will execute the specific packages.

Example:

```
<?xml          version="1.0"          encoding="UTF-8"?>
<suite  name="example  suite  1"  verbose="1"  >
  <test    name="Regression    suite    1"    >
    <packages>
      <package name="com.first.example" />

    <class name= "sample">

  </class>
    </packages>
  </test>
</suite>
```

- A suite is represented by one XML file. It can contain one or more tests and is defined by the <suite> tag.
- Tag <test> represents one test and can contain one or more TestNG classes.

- 
- `<class>` tag represents a TestNG class. It is a Java class that contains at least one TestNG annotation. It can contain one or more test methods



# Chapter 8

## Waits in Selenium

---

### **Implicit Waits :**

An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available. Once set, the implicit wait is set for the life of the WebDriver object instance.

```
WebDriver driver = new FirefoxDriver();  
driver.get("https://www.google.co.in");  
driver.manage().timeouts().implicitlyWait(10,  
TimeUnit.SECONDS);  
WebElement myDynamicElement =  
driver.findElement(By.linkText("Gmail"));
```

### **Explicit Waits**

Explicit waits are used to halt the execution till the time a particular condition is met or the maximum time has elapsed. Unlike Implicit waits, Explicit waits are applied for a particular instance only. WebDriver introduces classes

like `WebDriverWait` and `ExpectedConditions` to enforce Explicit waits into the test scripts.

## Object Instantiation for `WebDriverWait` class

```
WebDriverWait wait = new WebDriverWait(driver,30);
```

We create a reference variable “wait” for `WebDriverWait` class and instantiate it using `WebDriver` instance and maximum wait time for the execution to layoff. The maximum wait time quoted is measured in “seconds”.

<b>Expected</b>	<b>Condition</b>
<pre><i>wait.until(ExpectedConditions.visibilityOfElementLocated( By.xpath("//div[contains(text(),'COMPOSE')]"))); driver.findElement(By.xpath("//div[contains(text(),'COMP OSE')]")).click();</i></pre>	

The above command waits for a stipulated amount of time or an expected condition to occur whichever occurs or elapses first.

## Types of Expected Conditions

`ExpectedConditions` class provides a great help to deal with scenarios where we have to ascertain for a condition to occur before executing the actual test step.

`ExpectedConditions` class comes with a wide range of expected conditions that can be accessed with the help of the `WebDriverWait` reference variable and `until()` method.

### Let us discuss a few of them at length:

**1) `elementToBeClickable()`** – The expected condition waits for an element to be clickable i.e. it should be present/displayed/visible on the screen as well as enabled.

---

**Sample** **Code**

```
wait.until(ExpectedConditions.elementToBeClickable(By.xpath("//div[contains(text(),'COMPOSE')])));
```

**2) textToBePresentInElement()** – The expected condition waits for an element having a certain string pattern.

**Sample** **Code**

```
wait.until(ExpectedConditions.textToBePresentInElement(By.xpath("//div[@id='forgotPass']"), "text to be found"));
```

**3) alertIsPresent()**- The expected condition waits for an alert box to appear.

**Sample** **Code**

```
wait.until(ExpectedConditions.alertIsPresent()) != null;
```

**4) titleIs()** – The expected condition waits for a page with a specific title.

**Sample** **Code**

```
wait.until(ExpectedConditions.titleIs("gmail"));
```

**5) frameToBeAvailableAndSwitchToIt()** – The expected condition waits for a frame to be available and then as soon as the frame is available, the control switches to it automatically.

**Sample** **Code**

```
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id("newframe")));
```

---

## **Example to understand implicit and explicit waits**

Consider Below steps

1. Launch the web browser and open the “gmail.com”
2. Enter a valid username
3. Enter a valid password
4. Click on the sign in button
5. Wait for Compose button to be visible after page load

lets automate above mentioned scenario using selenium.

```
import static org.junit.Assert.*;

import java.util.concurrent.TimeUnit;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

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

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

public class WaitDemo {

    // created reference variable for WebDriver

    WebDriver driver;
```

@Before

```
public void init() throws InterruptedException {  
  
    driver=new FirefoxDriver();  
  
    driver.get("https://gmail.com");  
  
    driver.manage().window().maximize();  
  
    driver.manage().timeouts().implicitlyWait(  
    TimeUnit.SECONDS);  
  
}
```

@Test

```
public void test() throws InterruptedException {  
    driver.findElement(By.id("Email")).sendKeys("username")  
    ;  
  
    driver.findElement(By.id("Passwd")).sendKeys("password"  
    );  
  
    driver.findElement(By.id("signIn")).click();  
  
    // explicit wait - to wait for the compose button to be click-  
    able  
  
    WebDriverWait wait = new WebDriverWait(driver,30);  
    wait.until(ExpectedConditions.visibilityOfElementLocated(  
    By.xpath("//div[contains(text(),'COMPOSE')]")));  
    // click on the compose button as soon as the "compose"  
    button is visible  
  
    driver.findElement(By.xpath("//div[contains(text(),'COMP  
    OSE')]")).click();  
}
```

---

```
}
```

```
@After
```

```
public void end() {
```

```
    driver.quit();
```

```
}
```

```
}
```

# Chapter 9

## Data Provider

---

**A** major advantage of automating testing is its ability to test large amount of data on the system quickly. We can achieve this using data provider. Follow below steps in order to understand and implement data provider method.

- Write the annotation `@DataProvider` before the data provider method and its return type is 2 dimensional object array
- Define an object `Object[][] data = new Object[row][column];`

Now

**Rows** – Number of times your test has to be repeated.

**Columns** – Number of parameters in test data.

**For example**

```
Object[][] data = new Object[3][2];
```

---

```
// 1st row
```

```
data[0][0] ="Vaishali";
```

```
data[0][1] = "vaishali";
```

```
// 2nd row
```

```
data[1][0] ="admin";
```

```
data[1][1] = "renu";
```

```
// 3rd row
```

```
data[2][0] ="";
```

```
data[2][1] = "";
```

3. Return this object g. return data;
4. Now before writing test method write data providers method name in front of @Test tag.

e.g. @Test(dataProvider="getData"). Get data is the data providers method name.

5. We need to pass parameters to this test case. Lets understand this with the example.
6. Below is the example given for testing a login page using data provider. The 2 parameters are passed username and passwords. This test takes the input from data providers and the test method takes this inputs and runs the test multiple times depending upon the rows defined for the object returning in the data providers method.



**Example code for Data providers**

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.testng.Assert;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;
import java.util.Date;

import java.util.concurrent.TimeUnit;

public class DataProviders
{

    @Test(dataProvider="getData")
    public void LoginTest(String username,String password)

    {

        WebDriver driver = new FirefoxDriver();

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

        driver.get("http://clinic.scriptinglogic.com/auth/login\_form.php");

        driver.manage().timeouts().implicitlyWait(10,
        TimeUnit.SECONDS);

        driver.findElement(By.xpath("//*[@id='login_session'
        ]")).sendKeys(username);
```

---

```
driver.findElement(By.xpath("//*[@id='pwd_session']  
")).sendKeys(password);
```

```
driver.findElement(By.xpath("//*[@id='login']")).click();
```

```
}
```

```
@DataProvider
```

```
public Object[][] getData()
```

```
{
```

```
    //Rows - Number of times your test has to be repeated.
```

```
    //Columns - Number of parameters in test data.
```

```
    Object[][] data = new Object[4][2];
```

```
    // 1st row
```

```
    data[0][0] = "Vaishali";
```

```
    data[0][1] = "vaishali";
```

```
    // 2nd row
```

```
    data[1][0] = "admin";
```

```
    data[1][1] = "renu";
```

```
    // 3rd row
```

```
    data[2][0] = "";
```

```
    data[2][1] = "";
```

```
    // 4th row
```

```
    data[3][0] = "admin";
```

```
    data[3][1] = "admin";
```

```
    return data;
```

```
}  
  
}
```

**Example Code for reading excel file :** To implement this you must have POI Library. download it from <https://poi.apache.org/>

data provider example with excel sheet

```
import java.io.File;  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
import java.io.IOException;  
  
public class LoginWithExcelData {  
  
    @Test(dataProvider = "getData")  
    public void loginTest(String user,String pass) {  
        WebDriverManager.chromedriver().setup();  
        WebDriver driver = new ChromeDriver();  
        driver.manage().window().maximize();  
        driver.get("http://billing.scriptinglogic.net");  
  
        driver.findElement(By.name("email")).sendKeys(user);  
  
        driver.findElement(By.name("password")).sendKeys(pass  
);  
        driver.findElement(By.name("btn_login")).click();  
    }  
  
    @DataProvider  
    public Object[][] getData() throws IOException {
```

```
FileInputStream fileInputStream = new
FileInputStream("Data/Book2.xlsx");

XSSFWorkbook workbook = new
XSSFWorkbook(fileInputStream);

XSSFSheet worksheet =
workbook.getSheet("Sheet2");

int rowCount=
worksheet.getPhysicalNumberOfRows();

Object[][] data = new Object[rowCount][2];

for(int i = 0 ; i<rowCount;i++) {

    XSSFRow row = worksheet.getRow(i);

    XSSFCell user = row.getCell(0);

    data[i][0] = user.getStringCellValue();

    XSSFCell pass = row.getCell(1);
    data[i][1] = pass.getStringCellValue();

}
return data;
}
}
```

# Chapter 10

## Report Generation

---

**R**eport generation: the report generation can be better implemented in testing To generate report you have to add listeners below your suite tag.

For e.g. in below example you can see the listener *EmailableReporter*.

```
<suite name="My clinic - Project" verbose="1" >

<listeners>
<listener                                     class-
name="org.testng.reporters.EmailableReporter2" />
</listeners>

<test name="Add staff" >
<classes>
<class
name="com.com.vaidashree.selenium.TestNGExample" />
</classes>
</test>

</suite>
```

When you run the below code in testing.xml, a test-output folder will be created and there will be a file emailable-reports.html and it will contain the formatted report as shown below.

Test	# Passed	# Skipped	# Failed	Time (ms)	Included Groups	Excluded Groups
My clinic - Project						
<a href="#">Add staff</a>	1	0	0	78.202		

Class	Method	Start	Time (ms)
My clinic - Project			
Add staff — passed			
com.deepak.testng.Myclinic	<a href="#">mylogin1</a>	1463900153478	15203

### Add staff

`com.deepak.testng.Myclinic#mylogin1`

[back to summary](#)

## Reports using reportNG.

In order to get well formatted report there is a plug-in reportNG. In order to implement this you should import below 3 jars in to your library.

1. guice-3.0.jar
2. reportng-1.1.4.jar
3. velocity-dep-1.4.jar

Now you have to simply add its listeners below your suite tag. See the example below

```
<suite name="My clinic - Project" verbose="1" >
  <listeners>
    <listener class-
name="org.uncommons.reportng.HTMLReporter"/>
    <listener class-
name="org.uncommons.reportng.JUnitXMLReporter"/>
  </listeners>
  <test name="Add staff" >
    <classes>
      <class
name="com.com.vaidashree.selenium.TestNGExample" />
    </classes>
  </test>
</suite>
```

When you run the below code in testing.xml, you will see a html folder inside test-output folder and there will be few css, js and html files. Open the index.html file and it will contain the well formatted report as shown below.

Test Results Report

Overview

My clinic - Project  
Add staff

Add staff

Test duration: 103.859s

Failed Tests

com.com.valdasthree.selenium.TestingExample  
DoctorInfo 30.554s

```
org.openqa.selenium.NoSuchElementException: Unable to locate element: "[name='select']/../  
[type='content/1/a72121a']"  
Build info: version: '3.141.59', revision: '53a252c', time: '2018-08-15 15:57:40'  
System info: host: 'valsharif', ip: '192.168.0.105', os.name: 'Windows 8.1', os.arch: 'x86', os.version: '6.3', java.version: '1.8.0_20'  
Driver info: org.openqa.selenium.firefox.FirefoxDriver  
Capabilities {applicationCacheEnabled=true, rotateable=false, handlesAlerts=true, databaseEnabled=true,  
version=46.0.1, platform=WINDOWS, nativeEvents=false, acceptsSslCerts=true, webStorageEnabled=true,  
locationContextEnabled=true, browserName=firefox, takesScreenshot=true, javascriptEnabled=true,  
cssSelectorsEnabled=true}  
Session ID: 2f52d015-5efb-4b64-896a-3f4411ea3b8b  
*** Element info: [using=xpath, value="//*[type='content/1/a72121a']"
```

Passed Tests

com.com.valdasthree.selenium.TestingExample  
administrativeInfo 11.395s