* Selenium is open-source test automation tool
* Key components of selenium are

selenium web driver-🡪now we are using this

selenium IDE---🡪it contains less features

*selenium grid-🡪it is used to multiple machines and* *browsers*

*->exclusively used for testing web-application on multiple Browsers, like Firefox, Chrome, IE, Edge, Safari, Opera*

*->selenium supports multiple operating systems like windows, MacOS, c#, java script*

*->selenium supports multiple programming languages like java, python, c#, java script, ruby*

**Selenium architecture**

**Selenium browser result**

**Client library driver browsers**

***Java chrome chrome***

***Python Mozilla Mozilla***

***Ruby <----🡪json wire🡨---🡪 internet 🡨---🡪 internet***

***protocol***

***C# safari safari***

***java script opera opera***

***->****whatever we are writing java code it converts into Json wire protocol over http transfer using any of the web browser like chrome, Firefox then it transfer to the result browser we are seeing outputs in result browser, whatever errors are coming*

*the result browser transfer errors to the browser-driver then it send to the client-library*

*selenium web driver is an API means application programming interface*

*client*

*library interface browser*

*selenium 🡨-🡪 web driver 🡨-🡪 chrome*

*with java*

*what is maven dependency*

*for example, a project contains group of peoples are there one person using new version jars and other using older version the new version code will not execute on older version environment they all are need to install new versions, to overcome this issue*

*maven dependency management tool comes into picture.*

*maven is very popular and heavily used management tool*

***maven project contains:***

*src\main\java: -in this we are writing main functionalities & common utilities*

*src\main\resources: -here external resources perpous we are using*

*src\test\java: -whatever we are writing all the testcases are written here*

*src\test\resources: -excel and csv file*

***pom.xml****: -for managing all the decencies all the decencies are written here*

* *Whatever you want all the decencies are available in* ***maven repository***

for opening chrome browser

syntax

**System.setProperty(“webdriver. chrome. Driver”,” path”);**

**driver = new chromedriver ();**

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

Path means where your chromedriver.exe file is located that path

Other approach

**Webdrivermanager.chromedriver(). setup ();**

**driver = new chromedriver ();**

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

by using this approach whenever our chromedriver is updated no need to install the latest version chromedriver automatically webdrivermanager install the chromedriver

Other approach

**driver TestBrowser.openchromebrowser();**

**TestBrowser** is a thirdy-party tool by using this we are using browsers whatever you want

for opening firefox browser

syntax

**System.setProperty(“webdriver. firefox. Driver”,” path”);**

**driver = new Firefoxdriver ();**

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

Path means where your chromedriver.exe file is located that path

Other approach

**Webdrivermanager. Firefoxdriver (). setup ();**

**driver = new Firefoxdriver ();**

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

by using this approach whenever our chromedriver is updated no need to install the latest version chromedriver automatically webdrivermanager install the chromedriver

**driver TestBrowser.openfirefoxbrowser();**

**TestBrowser** is a thirdy-party tool by using this we are using browsers whatever you want

**Working on multiple browsers without using data provider the code is**

Public static string browser = “firefox”🡪whatever we want

Public static webdriver driver;

If (browser. Equals(“firefox”))

{

Webdrivermanager. Firefoxdriver (). setup ();

driver = TestBrowser .OpenFirefoxdriver ();

}

Else

If (browser. Equals(“chrome”))

{

Webdrivermanager. Firefoxdriver (). setup ();

driver = TestBrowser. OpenFirefoxdriver ();

}

**Working on multiple browsers using data provider the code is**

@Test(dataProvider="TC01\_Add\_Nationality")

**public** **void** Login Test (String Browser) **throws** Exception

{

**if** (Browser.equalsIgnoreCase("Chrome"))

{

*driver*=TestBrowser.*OpenChromeBrowser*();

}

**if** (Browser.equalsIgnoreCase("FireFox"))

{

*driver* = TestBrowser. *OpenFirefoxBrowser* ();

Thread.*sleep*(10000);

}

**if** (Browser.equalsIgnoreCase("IE"))

{

*driver* =TestBrowser.*OpenIEBrowser*();

Thread.*sleep*(10000);

}

**if** (Browser.equalsIgnoreCase("RemoteWebDriver1\_Chrom"))

{

*driver* =TestBrowser.*RemoteWebDriver*();

Thread.*sleep*(10000)

}

The browser details whatever browser we are selecting that browser details available in excel by using the data provider concept

**What Types of Testing Can Be Done Using Selenium?**

There are many [types of testing](https://www.perfecto.io/resources/types-of-testing) that can be done with Selenium. You can do smoke testing, sanity, testing, UI testing, [regression testing](https://www.perfecto.io/blog/what-is-regression-testing), and more.

Now we are using testng

### How Is Selenium Used for Automated Testing?

Selenium is an automated testing framework. The framework executes test scripts against different web browsers, like Chrome, Firefox, and Safari.

TestNG is **an automation testing framework** in which NG stands for "Next Generation". TestNG is inspired from JUnit which uses the annotations (@). TestNG overcomes the disadvantages of JUnit and is designed to make end-to-end testing easy. ... The TestNG in Selenium provides an option, i.e., testng-failed.

**TESTNG ANNOTATIONS**:

**@BeforeTest**:->whatever we are writing in this it will be executed first mostly openingchromebrowser like that code we are writing here

**@Test:->**all the remaining code written here

**@AfterTest:->**here the code will be executed after test block mostly here we are writing closing browsers like driver. Close ();

@BeforeSuite

@BeforeTest

@BeforeClass

@BeforeMethod

@Test

@AfterSuite

@AfterTest

@AfterClass

@AfterMethod

If our programme contains 2 **@Test** methods the testing annotations will be executed 2 times

**What is a Selenium Locator?**

Selenium Locators are used for identifying the web elements on the web page. To access the HTML elements from a web page locator are used. In Selenium, we can use locators to perform actions on text boxes, checkboxes, links, radio buttons, list boxes, and other web elements. Locators help us in identifying the objects.

1.**id**🡪driver. find Element (By.id(“id-name”))

2.**Name**🡪 driver. find Element (By.name(“name”))

3.**CSSSelector**🡪 click on ...dots>copy>copy CSS Selector

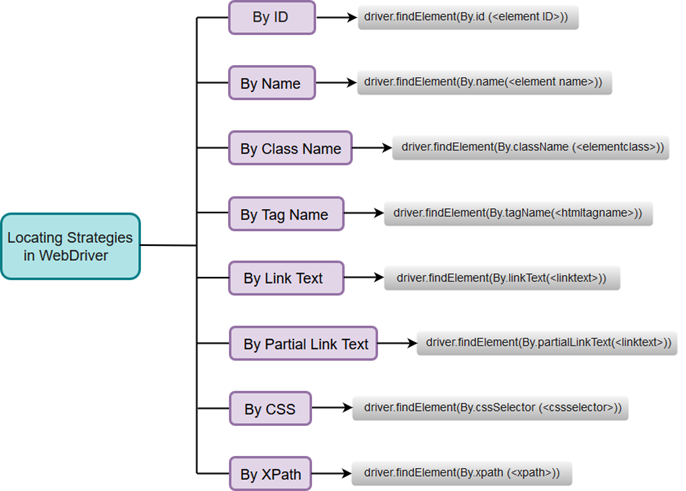
4.**xpath** 🡪 click on ...dots>copy>copy xpath

5**.linkText** 🡪 must has to start with tag Name as <a, static Hyperlink

6.**Partial link Text**--> must has to start with tag Name as <a, Dynamic Hyperlink

7.**className**

8.**TagName**🡪 Whenever u want to find group of similar web Elements on a webpage



**Ex**:

<input type="submit" name="Submit" class="button" id="btnLogin" value="LOGIN">

To writing any text we are using following driver instructions

driver. findElement (By.*id*("txtUsername")).sendKeys("Admin");

For clicking any button, we are using following driver

instructions

driver. findElement (By.*name*("Submit")). click ();

for navigate one site to another site the code is

driver. Navigate().to("https://gmail.com/");

driver. Navigate().to("https://google.co.in");

driver. Navigate (). back (); in this case its coming to Gmail

driver. Navigate ().forward (); here its navigate to google

to open any of the website we are using driver instruction like below

driver. Get("https://opensource-demo.orangehrmlive.com/");

**for highlighting web element border**

**public** Web Element findElement (By by) **throws** Exception

{

WebElement elem = *driver*.findElement(by);

**if** (*driver* **instanceof** JavascriptExecutor)

{

((JavascriptExecutor)*driver*).executeScript("arguments[0].style.border='3px solid red'", elem);

}

**return** elem;

}

Drop Down:

In selenium dropdown can be handled in 3 ways

1. SelectByIndex
2. SelectByValue
3. SelectByVisibleText

**SelectByIndex:**

For every dropdown developer will give index we are using that index by using SelecByIndex, index will start with ‘0’.

First, we need to create object for dropdown box

**Select dropdown1 = new Select (*driver*. findElement (By.*id*("location\_country")));**

**Dropdown1.selectByIndex(1);**

**🡪whatever index value you want u can**

**2.SelectByValue**

For every visible text developer will give value, first we need to create object for dropdown

**Select dropdown1 = new Select (*driver*. findElement (By.*id*("location\_country")));**

**Dropdown1.selectByValue(“al”);**

Whatever developer will give that value we are assign

**3.SelectByVisibleText**

**Select dropdown1 = new Select (*driver*. findElement (By.*id*("location\_country")));**

**Dropdown1.selectByVisibleText(“India”);**

<option value=’AF’>Afghanistan</option>

Afghanistan🡪visible text

’AF’🡪value

**Data provider:**

What is a Data Provider in TestNG?

Similar to TestNG Parameters, Data Providers are **a means to pass data to test scripts in TestNG**. Using Data Provider in TestNG, we can easily inject multiple values into the same test case. It comes inbuilt in TestNG and is popularly used in data-driven frameworks.

**We can handle data provider in 3 ways.**

1. Providing-data-by-using-Array
2. Providingdataindataprovider
3. Providing data in excel

1. Providing-data-by-using-Array

@DataProvider(name="rohitha1")

**public** Object [][] Test2() **throws** Exception {

Object data [][] = **new** Object [2][5];

data [0][0] ="Chrome";

data [0][1] ="https://opensource-

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

data [0][3] ="admin123";

data [0][4] ="Indian2027";

//second input data

data [1][0] ="Firefox";

data [1][1] ="https://opensource-

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

data [1][3] ="admin123";

data [1][4] ="Indian2028";

**return** data;

}

--------------------------------------------------------------2 providing-data-in-data provider-only

@DataProvider(name="rohitha")

**public** **static** Object [][] test1() **throws** Exception {

**return** **new** Object [][]{

{"Chrome","https://opensource-demo.orangehrmlive.com/index.php/auth/login","Admin","admin123","Indian12027"},

{"Firefox","https://opensource-demo.orangehrmlive.com/index.php/auth/login","Admin","admin123","Indian12028"}

};

}

**--------------------------------------------------------------------------**

3.Providing data in excel

@DataProvider(name="rohitha1")

**public** **static** Object [][] Test2() **throws** Exception {

ExcelApiTest4 eat = **new** ExcelApiTest4();

Object [][] testObjectArray = eat. gettable Array ("C:\\HTML Report\\OrangeHRM6\\TC01\_Nationality.xls","sheet1");

**return** (testObjectArray); }

**What is emailable report in TestNG?**

emailable-report. html: This is **the summarize report of the current test execution which contains Test Case message** in green (for pass test cases) and red (for failed test cases) highlight.

**Reporter.*log* ("Pass- open Orangehrm1");**

By writing above line, we can see our output in **test-output**

Of **emailable-report.html**

**What is Extent report in TestNG?**

With the **Extent Reports** library, you can create beautiful, interactive and detailed reports for your tests. Add events, screenshots, tags, devices

By capturing screenshots, testers can better identify what went wrong when the software acted erroneously during a test. Capture screenshots only when a test fails, since they consume a lot of memory.

//step1: we need to import below lines

**import** com. aventstack. extentreports.ExtentReports;

**import** com.aventstack.extentreports.ExtentTest;

**import** com.aventstack.extentreports.MediaEntityBuilder;

**import** com.aventstack.extentreports.reporter.ExtentHtmlReporter;

//step2: we need to declare below lines globally in below webdriver driver

ExtentTest logger;

ExtentReports extent;

String screenShotPath;

**public** **static** String *TestScriptName* = "TC01\_Add\_Employee";

**public** **static** String *TestName*;

//step3: Extends Reports starts here

SimpleDateFormat sdfDate = **new** SimpleDateFormat("yyyy\_MMM\_dd\_h\_mm\_ss\_SSS\_a");

Date now = **new** Date ();

String strDate = sdfDate.format(now);

*TestName*=*TestScriptName*+"\_"+strDate+".html";

*TestScriptName*=*TestScriptName*+"\_"+strDate;

String TestHtmlName="C:/HTML Report/test-output/ExtentReportScreenShots/"+ *TestScriptName* +"/"+*TestName*;

//String TestHtmlName=TestName;

ExtentHtmlReporter reporter=**new** ExtentHtmlReporter(TestHtmlName);

System.***out***.println("Html Report path is : "+TestHtmlName);

extent=**new** ExtentReports();

extent.attachReporter(reporter);

logger=extent.createTest(*TestName*);

then writing below code after driver instruction

screenShotPath = ExtentReport.*capture*(driver,*TestScriptName*);

logger.pass("Login Page - Entered user Name",MediaEntityBuilder.*createScreenCaptureFromPath*(screenShotPath).build());

by using extent report, we can see our output in whatever path we are given in screenShotPath in that path we can see our output

**What is customize report in TestNG?**

**Reporter** is mainly used to generate the final **report** for the test execution. The extension can be used to generate XML, HTML, XLS, CSV, or text format files ...

TestNG has provided an ability to implement '**IReporter**' an interface which can be implemented to generate a customized report by users. It has 'generateReport()' method which will be invoked after all the suite has completed its execution and gives the report into the specified output directory.

In this we need to use **reporter1** class it is a thirdy party class

First we need to write **Reporter1 R1** globally in below webdriver driver

And create object for reporter1 class

String str= "TC01\_Login";

R1= **new** Reporter1(*driver*, str);

After that writing below code after driver instructions

R1.TakeScreenShotAuto(*driver*,"Opened Orange HRM","Pass");

## **What is Katalon Recorder?**

Katalon Recorder is a free, Selenium IDE alternative, lightweight web extension for automating actions and automated testing on the browser. Currently, we are available on **Chrome**, **Firefox** and **Edge**.

## **What do people use Katalon Recorder for?**

People reportedly use Katalon Recorder to:

* Save time by automating the repetitive tasks on browsers, such as generating reports, filling forms, automating games, etc.
* Ensure quality of your work by testing before handing your work over to the QA team.
* Ensure end-to-end quality of your software by automated testing.

After Katalon Automation Recorder is installed, **open the extension from Chrome's main toolbar and click the Settings button** to specify the preferred port to communicate with Katalon Studio

Katalon recorder is a playback tool.

**what is @parameters in testng**

Parameterized tests allow developers to run the same test multiple times with different input values. TestNG lets you pass parameters directly to your test methods in two different ways − With **testng.xml**. With Data Providers.

@Parameters({"Browser1","UserName1","Password1","Nationality1"})

**Data Driven Testing**

 is important because testers frequently have multiple data sets for a single test and creating individual tests for each data set can be time-consuming. Data driven testing helps keeping data separate from test scripts and the same test scripts can be executed for different combinations (like different browsers) of input test data and test results can be generated efficiently.

@Parameters({"Browser"})

@DataProvider (name = "TC01\_Sample")

**public** **static** Object [][] Authentication1() **throws** Exception {

ExcelApiTest4 eat = **new** ExcelApiTest4();

Object [][] testObjArray = eat.getTableArray("C://HTML Report//OrangeHRM6//TC01\_Sample.xlsx", "Sheet1");

System.***out***.println(testObjArray.length);

**return** (testObjArray);

}

By using these above codes, we can run our code in multiple browsers by changing the browser value in testng.xml

**invocation count in testng**

By using of invocationCount we can execute a test multiple times. Invocation count **determines how many times test will execute.**

@Test(invocationCount=3)

**Where can we apply Parallel Test execution in TestNG?**

1. Methods: This will run the parallel tests on all @Test methods in TestNG.
2. Tests: All the test cases present inside the <test> tag will run with this value.
3. Classes: All the test cases present inside the classes that exist in the XML will run in parallel.

**Parallel methods**  
TestNG provides multiple ways to execute tests in separate threads. In testng. xml, if we set 'parallel' attribute on the tag to 'tests', **testNG will run all the '@Test' methods in tag** in the same thread, but each tag will be in a separate thread. This helps us to run test methods / classes / tests in parallel.

Methods: If aTestNG Class contains multiple test methods.

<suite name=*"Suite” thread*-count=*"2"* parallel=*"methods” >*

<test name=*"Test"*>

<classes>

<class name=*"Day\_016\_ParallelMethods.AllTests"*/>

</classes>

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

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

Tests: If aTestNG.xml contains multiple Tests.

<suite name=*"Suite"* parallel=*"tests"* thread-count=*"5"*>

<test name =*"Test1"* >

<classes>

<class name=*"Day\_015\_ParallelTests.OrangeHRMTest"*/>

</classes>

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

<test name=*"Test2"* >

<classes>

<class name=*"Day\_015\_ParallelTests.MercuryTest"*/>

</classes>

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

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

Classes: if a TestNG.xml contains multiple TestNG classes

<suite thread-count=*"4"* name=*"Suite"* parallel=*"classes"*>

<test name=*"Test"* >

<classes>

<class name=*"Day\_016\_ParallelClasses.OrangeHRMTest"*/>

<class name=*"Day\_016\_ParallelClasses.MercuryTest"*/>

</classes>

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

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

Multiple Suites

**TestNG** enables you want to execute test methods. It allows us to run **multiple** test **suites** at the same time across **multiple** environments.

<suite name=*"allSuites"*>

<suite-files>

<suite-file path=*"./suite1.xml"* />

<suite-file path=*"./suite2.xml"* />

</suite-files>

</suite>

Priority Test:

We can write multiple @Test methods in our program by using @Test(priority=) we can run our program in priority order like below

PASSED: Test6\_closebrowser-------->@Test(priority=6)

PASSED: Test4\_Addnatialities----> @Test(priority=5)

PASSED: Test5\_Logout-------------> @Test(priority=-1)

PASSED: Test3\_Login--------------> @Test(priority=1)

PASSED: Test2\_OpenOrangeHRM------ @Test(priority=0)

PASSED: Test1\_OpenChromeBrowser--@Test(priority=0)

* It will execute from lower priority to higher priority (-1 first finally 6)
* if priority is same, it will be executed on alphabetical order of method name

Waits:

Waits are 3 types

1.implicit wait

2.Explicit wait

3.Fluent wait

**1.implicit wait:** It is a Global rule applicable for all the web Elements where ever it has performance issues, and driver will wait for the web element as per the implicit time specified.

***driver*. Manage (). timeouts ().implicitlyWait(10, TimeUnit.*SECONDS*);**

**2.Explicit wait:** Some web Elements can't be loaded with in the default implicit time,

such specific web elements can be handled explicitly by using explicit wait

**WebDriverWait wait= new WebDriverWait(*driver*,120);**

**wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.*linkText*("Logout")));**

3.Fluent wait: it is also used to handle specific web elements only on the webpage.

The webdriver will look for the web element at regular intervals as per the pooling frequency time.

**WebDriverWait wait = new WebDriverWait (driver,60);**

**Wait.PollingEvery(20, TimeUnit.SECONDS);**

**Wait.until(ExpectedConditions.VisibilityOfElementLocated(By.id(“welcome”)));**

**-----------------------------------------------------------------------------------------**

**Frames:** Frame is used to call another html page in the current web page.

Frames can be handled in 3 ways

1**.**Switch into frame by Index:

2. Switch into frame by name:

3.Switch into frame by web element reference:

**1.Switch into frame by Index:** first we need to find out the number of frames available in current webpage by using like this

**List<WebElement> frames = driver. findElements (By. tagName("iframe"));**

**System.out.println("Number of frames: " + frames. Size ());**

**driver.switchTo().frame(frames.get(0));**

**driver.switchTo().defaultContent();->**by using this we can move to frame to current webpage

**driver.switchTo().frame(frames.get(1));**

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

**2. Switch into frame by name:** by using the frame name we can switch

**driver. switchTo (). frame("Google\_ContactForm");**

**3.Switch into frame by webelement reference:** by using the webelement reference we can switch to frame

**driver. switchTo().Frame**

**(driver. findElement (By.id("GoogleContactForm")));**

**JavascriptExecutor:**

Selenium WebDriver can encounter problems interacting with a few web elements. For instance, the user opens a URL and there is an unexpected pop-up that will prevent the WebDriver from locating a specific element and produce inaccurate results. This is where JavascriptExecutor comes into the picture.

by using JavaScriptexecutor we can perform following operations

1. **We can launch new tab:**

((JavascriptExecutor)driver).executeScript("window.open()");

1. **Scroll at particular web Element:**

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("arguments[0].scrollIntoView();",Connect\_withUs);

1. **SendKeys:**

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("arguments[0].setAttribute('value','Admin')", username);

1. **Click:**

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("arguments[0].click();", loginbutton);

1. **Highlight a web Element border**

JavascriptExecutor js = (JavascriptExecutor)driver;

js.executeScript("arguments[0].style.border='3px solid red'", username);

Action class: Actions class is **an ability provided by Selenium for handling keyboard and mouse events**. In Selenium WebDriver, handling these events includes operations such as drag and drop, clicking on multiple elements with the control key

1. Mouse Hover action

2. Double click

3. Right Click

4. Drag and drop

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

**1.actions.moveToElement(Users).click().build().perform();**

**2. actions.doubleClick(Copy\_Text\_Button).perform();**

**3. actions.contextClick(RButton).perform();**

**4. actions.dragAndDrop(Source\_Drag\_Button, Target\_Drag\_Button).perform();**

**File Upload:** by using file upload concept we can upload photos and files

Wherever we want to upload file or photo we can use below code

StringSelection str = **new** StringSelection(PhotoPath);

Toolkit.*getDefaultToolkit*().getSystemClipboard().setContents(sel,**null**);

**Robot robot = new Robot();**

**// press keyboard**

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

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

**// release keyboard**

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

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

**// press enter**

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

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

**File Download:**

Wherever we want to download file in that place we can use below code, in this we want use movefile class (thirdyparty)

//download file locator

String str1="//\*[@id='tblAttachments']/tbody/tr/td[2]";

//driver instructions for download file

WebElement Element=findElement(By.*xpath*(str1));

String fname = Element.getText();

findElement(By.*linkText*(fname)).click();

//source and destination paths

String SrcFile="C:\\Users\\USER\\Downloads\\"+fname;

String DestinationFile="C:\\HTML Report\\"+fname;

*moveFile*(SrcFile, DestinationFile);

Advanced X-Paths:

**1. Absolute xpath:**

Copy full xapth--identifying the xpath from root node to current node is nothing but full xpath

**2. Relative xpath:**

Copy xapth-->identifying the xpath based on object properties is known as relative xpath

Relative xpath can be handled in following ways

**By id:**

*Xpath=*("//\*[@id='txtUsername']"));

**Using and/or:**

Xpath="//\*[@type='submit' **or** @name='btnReset']";

Xpath1="//input[@type='submit' **and** @id='btnLogin']";

**ByText:**

**//**this is used for static xpath

Xpath *=*(“//\*[text() =’PIM’]”));

//this is used for dynamic xpath

Xpath*=*(“//\*[contains(text(),’Nationalit’)]”);

**Following**

"//\*[@id='txtUsername']🡪current webelement xpath

"//\*[@id='txtUsername']//following::input== 1 of 2

*xpath*("//\*[@id='txtUsername']//following::input[1]");

*xpath*("//\*[@id='txtUsername']//following::input[2]");

**preceding**

"//\*[@id='txtUsername']//preceding::input== 1 of 7

"//\*[@id='txtUsername']//preceding::input[1]");

"//\*[@id='txtUsername']//preceding:input[2]");

**parent**

"//\*[@id='divUsername']//child:input->1 of 2

"//\*[@id='divUsername']//child:input[1]

"//\*[@id='divUsername']//child:span[1]

"//\*[@id='divUsername']//desendant:input[1]

"//\*[@id='divUsername']//descendant:span[1]

**child**

"//\*[@id='txtUsername']

"//\*[@id='txtUsername']//parent::div

**Ancestor: parent+grand parents**

"//\*[@id='txtUsername']//ancestor::div

"//\*[@id='txtUsername']//ancestor::label

"//\*[@id='txtUsername']//ancestor::a

**Following-sibling’s**

"//\*[@id='divUsername']//following-sibling

**Starts with**

"//\*[starts-with(text(),'Nationa')]"

**ExcelUtil\_Apache\_POI:** without using dataprovider we can able read excel data by using this concept

For this first we need to import following lines

**//step1**

**import** java.io.FileInputStream;

**import** org.apache.poi.xssf.usermodel.XSSFWorkbook;

**import** org.apache.poi.xssf.usermodel.XSSFSheet;

**import** org.apache.poi.xssf.usermodel.XSSFRow;

**import** org.apache.poi.xssf.usermodel.XSSFCell;

we need to declare these variables globally

**public** FileInputStream fis = **null**;

**public** XSSFWorkbook workbook = **null**;

**public** XSSFSheet sheet = **null**;

**public** XSSFRow row = **null**;

**public** XSSFCell cell = **null**;

we need to create object for current class

TC02\_Excel\_Test\_xlsx eat = **new** TC02\_Excel\_Test\_xlsx();

TestURL=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1",1,0);

UserName=eat.getCellData("C://HTML Report//OrangeHRM6//TC01\_Nationality1.xlsx","Sheet1",1,1);

Like all variables we use above code and we need to use getcelldata class(thirdy-party)then we are writing driver instructions as per your need

The above code we are using .xlsx files

For .xls files same as above but instead of **xssf** we are using **hssf**

**We are inserting data into excel by using putcelldata concept**

**Same as above instead of writing getcelldata we are writing putcelldata**

**For putcelldata we need to use** FileoutputStream also

**ScreenShot:**

A **Screenshot in Selenium Webdriver** is used for bug analysis. Selenium webdriver can automatically take screenshots during the execution. But if users need to capture a screenshot on their own, they need to use the TakeScreenshot method which notifies the WebDrive to take the screenshot and store it in Selenium.

*takeSnapShot*(driver, "C:\\HTML Report\\test-output\\Test1\\ScreenShot1.png");

writing above code after driver-instructions and we need to use ***takeSnapShot*** *class (thirdy-party)*

**Alerts:** An **Alert in Selenium** is a small message box which appears on screen to give the user some information or notification. It notifies the user with some specific information or error, asks for permission to perform certain tasks and it also provides warning messages as well.

Alert interface provides the below few methods which are widely used in [Selenium Webdriver](https://www.guru99.com/introduction-webdriver-comparison-selenium-rc.html).

**// To click on the ‘Cancel’ button of the alert.**

driver. switchTo (). alert (). dismiss ();

**// To click on the ‘OK’ button of the alert.**

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

**// To capture the alert message.**

driver. switchTo (). alert (). getText ();

**//To send some data to alert box.**

driver. switchTo(). alert().sendKeys("Text");

**Switching to alert**

Alert alert = *driver*. switchTo (). alert ();

**Web Table:** by using this concept we can get all the web table data like rows and columns information into excel by using web table concept

First, we need to find number of rows and columns information into some values like **emprows** and **empcolumns**

We need to declare irow globally

**public** **int** iRow;

String str=String.*valueOf*(iRow);

String Empcoloums=" //\*[@id='resultTable']/thead/tr/th";

List<WebElement> columns = driver.findElements(By.*xpath*(Empcoloums));

System.***out***.println("No of columns in WebTable : " + columns.size());

String EmpRows="//\*[@id='resultTable']/tbody/tr/td[2]";

List<WebElement> rows = driver.findElements(By.*xpath*(EmpRows));

System.***out***.println("No of rows in WebTable : " + rows.size());

Then we need to create object for ExcelApiTest3 class

ExcelApiTest3 eat = **new** ExcelApiTest3();

By using for loops we can get our web table data

**for** ( **int** i=1 ; i<=rows.size() ;i++)

{

**for** ( **int** j=2 ,k=0; j<=columns.size() ;j++,k++)

String str1="//\*[@id='resultTable']/tbody/tr[" + i + "]" + "/td" + "[" + j +"]";

WebElement Ele=findElement(By.*xpath*(str1));

String WebElementText = Ele.getText();

**if** (WebElementText !=**null**)

eat.PutCellData(“C://HTML Report//OrangeHRM6//TC01\_EMPExport.xls","Sheet4",i,k,WebElementText);

**else**

eat.PutCellData(“C://HTML Report//OrangeHRM6//TC01\_EMPExport.xls","Sheet4",i,k,"Blank Data");

**PageObjectModel:** by using this we can call all the classes into one class

**public** **class** Customers

{

WebDriver driver;

**public** **void** Customers(WebDriver driver)

{

**this**.driver = driver;

}

**public** void AddCustomers () **throws** Exception

{

In this we are writing driver instructions as per need

}

Like that we create all the methods and finally in main class

**@BeforeTest**

Writing openchromebrowser and URL code

**@Test**

Customers S1= **new** Customers();

S1.Customers(driver);

S1. AddCustomers ();

Like that as per your need we can write number of classes

**@AfterTest**

Here we can write close browser code

**Page\_Factory**

**public** **class** HomePage{

WebDriver driver;

**public** **void** HomePage(WebDriver driver)

{

**this**.driver = driver;

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

}

@FindBy(id="welcome")

WebElement welcome1;

@FindBy(linkText="Logout")

WebElement Logout1;

**public** **void** Logout() **throws** Exception

{

welcome1.click();

Logout1.click();

}

After that the final class create code is same as pageobjectmodel

**@BeforeTest**

Writing openchromebrowser and URL code

**@Test**

Customers S1= **new** Customers();

S1.Customers(driver);

S1. AddCustomers ();

Like that as per your need we can write number of classes

**@AfterTest**

Here we can write close browser code

**return\_single\_value:**

by using this we can written single value and multiple values like key and value pair concept

first we create object for current class

str2= h1. addstrings("Hello");

System.***out***.println("updated string value is " +str2);

**int** dd1= h1.Add(30000,400);

System.***out***.println("sum of a,b " +dd1);

**int** e= h1.multiplication(50,40);

System.***out***.println("multiplication of a,b " +e);

**int** f= h1.subtract(50,60);

System.***out***.println("subtract of a,b " +f);

Then create methods for all above

**public** String addstrings(String str)

{

String str1=str + "World";

**return** str1;

}

**public** **int** add(**int** a,**int** b)

{

dd1= a+b;

Return dd1;

}

**public** **int** subtract(**int** a,**int** b)

{

f= a-b;

Return f;

}

**public** **int** multiplication(**int** a,**int** b)

{

**int** e=a\*b;

return e;

}

**Return-multiple-values by using key and value pair**

**public** **void** Test1() **throws** Exception

{

Map<String, Integer> map1 = **new** HashMap<>();

map1=getHashMap ();

System.***out***.println(map1);

}

**public** Map<String, Integer> getHashMap () **throws** Exception

{

Map<String, Integer> map = **new** HashMap<> ();

map.put ("Vishal", 10);

map.put ("Sachin", 30);

map.put ("Vaibhav", 20);

System.***out***.println("Size of map is: - "+ map. Size ());

**return** map;

}

**what is testng groups**

TestNG Groups **allow you to perform groupings of different test methods**. Grouping of test methods is required when you want to access the test methods of different classes. ... If the <groups> tag is specified inside the <suite> tag, then it is applied to all the <test> tags of XML file.

@Test(groups = { "checkintest" })

@Test (groups = {“functest”})

@Test (groups = {“checkintest","GoogleGroup”})

<suite name=*"Suite"*>

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

<groups>

<run>

<include name=*"checkintest"* />

<exclude name=*"functest"*/>

<exclude name=*"GoogleGroup"*/>

</run>

</groups>

<classes>

<class name=*"Day\_032\_TestNG\_Groups.AllTests"*/>

</classes>

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

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

**FailedTestCases**

If any test case is failed in test-output folder the suite fail is generated so we need to identify the error and re-run suite fail instead of running entire testng.xml we can run that suite fail only, after running that suite fail it generates failed. Suite fails in our output

**what is testng listeners**

One of the important features of the TestNG framework is listeners. It is an interface that listens to predefined events in test scripts and modifies the default behaviour of the TestNG tool. Whether you want to customize reports or generate logs for specific tests, TestNG listeners help you to do so. Testers would also want to define prerequisites and clean-up configurations for each test project. Automate the process of setting up the prerequisites before starting your testing process. Performing clean-up operations after the testing finishes with a TestNG listener.

There are different types of Listeners in TestNG.

Now we are using ITestListner

**public** **class** ListenerTest **implements** ITestListener

{

// When Test case get Skipped, this method is called.

@Override

**public** **void** onTestFailure(ITestResult Result)

{

System.***out***.println("The name of the testcase failed is:"+Result.getName());

Reporter.*log* ("The name of the testcase failed is:"+Result.getName());

}

// When Test case get Skipped, this method is called.

@Override

**public** **void** onTestSkipped (ITestResult Result)

{

System.***out***.println("The name of the testcase Skipped is:"+Result.getName());

}

@Test()

**public** **void** Test2()

{

System.***out***.println("This method to test fail");

Assert.*assertTrue*(**false**);

}

Forcefully failed this test as to verify listener.

<suite name=*"Suite"*>

<listeners>

<listener class-name=*"Day\_034\_TestNG\_Listeners.ListenerTest"*/>

</listeners>

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

<classes>

<class name=*"Day\_034\_TestNG\_Listeners.TestCases"*/>

</classes>

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

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