**JAVA SELENIUM**

Reliance Transitions Systems Inc

**(RT Systems Inc)**

[training@rtsysinc.com](mailto:training@rtsysinc.com)

(925) 322 3131

**Selenium Introduction**

**Test Automation for Web Applications**

* Frequent regression testing
* Rapid feedback to developers
* Virtually unlimited iterations of test case execution
* Support for Agile and extreme development methodologies
* Disciplined documentation of test cases
* Customized defect reporting
* Finding defects missed by manual testing

**To Automate or Not to Automate?**

* User Interface changes considerably.
* Not enough time to automate

History

* 2004 - Jason Huggins **–** Thoughtworks
* Selenium IDE - Shinya Kasatani
* He developed a JavaScript library that could drive interactions with the page, allowing him to automatically rerun tests against multiple browsers.
* Selenium RC - Patrick Lightbody and Paul Hammant
* Selenium RC and Selenium IDE
* 2006 **–** Simon Stewart **–** Google
* WebDriver API **–** Direct native methods
* Selenium Core + WebDriver = Selenium2
* Selenium1 = Selenium RC **–** Remote Control
* Selenium IDE **–** Rapid Prototyping Tool
* Selenium-Grid **–**

Selenium-Grid allows the Selenium RC solution to scale for large test suites and for test suites that must be run in multiple environments. Selenium Grid allows you to run your tests in parallel, that is, different tests can be run at the same time on different remote machines

Browsers Supported

**–** *ChromeDriver*

**–** *InternetExplorerDriver*

**–** *FirefoxDriver*

**–** *OperaDriver*

**–** *HtmlUnitDriver - Browser without UI, Fastest way to test programs.*

**–** *AndroidDriver*

**–** *IPhoneDriver*

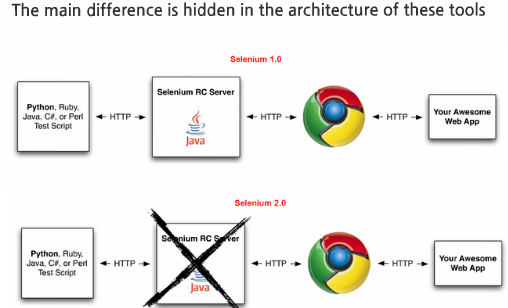
Selenium Components

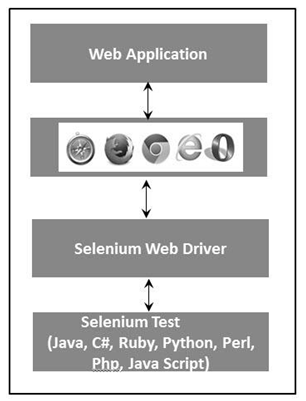
➢ Selenium IDE

➢ Selenium Remote Control (Selenium 1.0)

➢ Selenium GRID

➢ WebDriver (Selenium 2.x)





Selenium **–** Why?

* Test tool for web applications
* Runs in any mainstream browser
* Supports tests in many languages
* Selenese (pure HTML, no backend required)
* Java, C#, Perl, Python, Ruby
* Record/playback (Selenium IDE)
* Open Source with corporate backing
* [www.seleniumhq.org](http://www.seleniumhq.org)
* Testing tool for browser-based testing of web applications. It can be used both for **functional, compatibility** (it has extensive cross-browser support) and **regression testing.**

Section – 8

Selenium IDE

**Topics:**

**Introduction to Selenium IDE**

**Recording, Running and Debugging Script**

**Xpaths & Installing Firebug**

**Verifications & Assertions**

**Command Reference for IDE**

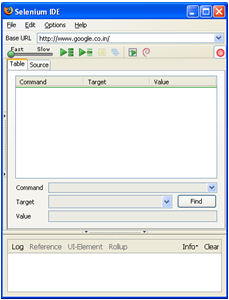
**Examples & Exercise**

**Introduction to Selenium IDE**

* The Selenium-IDE (Integrated Development Environment) is the tool you use to develop your Selenium test cases.
* **It’s** an easy-to-use Firefox plug-in and is generally the most efficientway to develop test cases.
* Context Menu on Firefox
* **Download & DEMO**
* Selenium IDE allows you to save tests as HTML, Java, Ruby scripts, or any other format
* It allows you to automatically add assertions to all the pages
* It contains **Context Menu** provides few Selenium commands.
* Allows to create Test Suites

**URL to Download – In FireFox:**

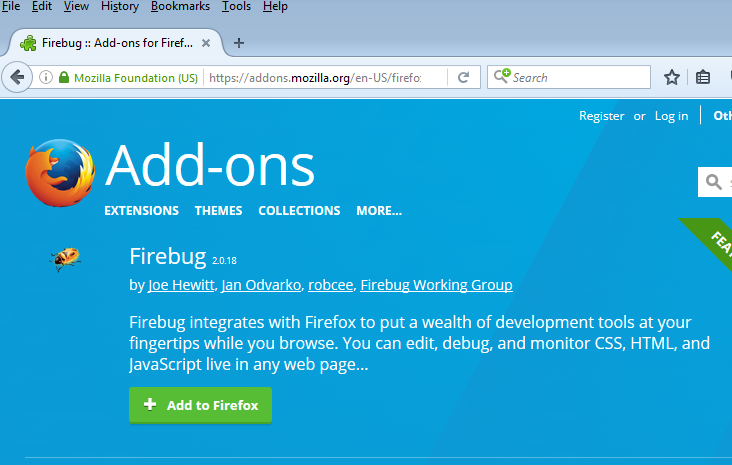
<https://addons.mozilla.org/en-us/firefox/addon/selenium-expert-selenium-ide/>



Firebug and FirePath

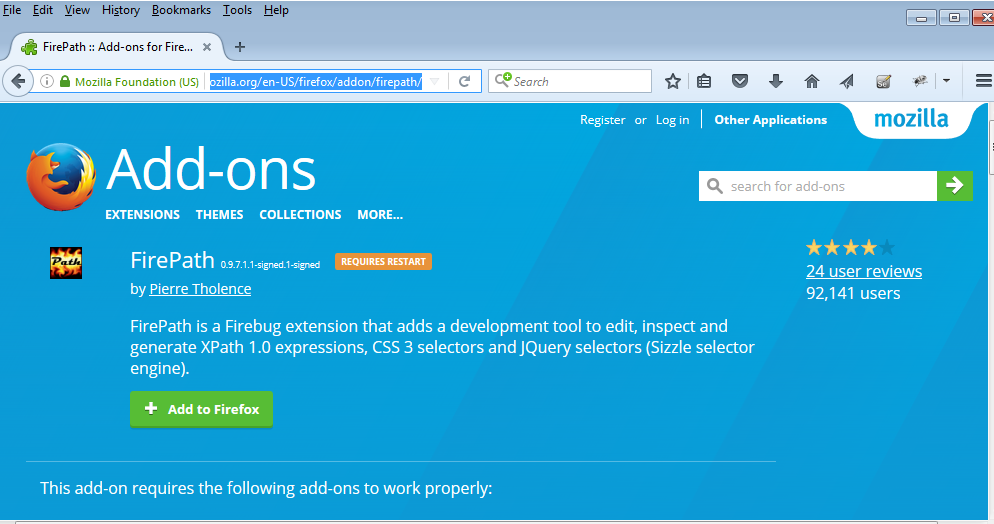
* **Firebug** has become the defacto tool for web developers as it allowsdevelopers to find elements on the page by using the find functionality.

[**https://addons.mozilla.org/en-US/firefox/addon/firebug/**](https://addons.mozilla.org/en-US/firefox/addon/firebug/)



* **FirePath**: A very good tool for testing out XPath and CSSon the page. It will highlight all elements on the page that match the selector to your element location.

<https://addons.mozilla.org/en-US/firefox/addon/firepath/>



Verifications and Assertions

An “assert” will fail the test and abort the current test case, whereas a “verify” will fail the test and continue to run the test case.

The best use of this feature is to logically group your test commands, and start each group with an “assert” followed by one or more “verify” test commands. An example follows:

|  |  |  |
| --- | --- | --- |
| **Command** | **Target** | **Value** |
| Open | /download/ |  |
| assertTitle | Downloads |  |
| verifyText | //h2 | Downloads |
| assertTable | 1.2.1 | Selenium IDE |
| verifyTable | 1.2.2 | June 3, 2008 |
| verifyTable | 1.2.3 | 1.0 beta 2 |

The above example first opens a page and then “asserts” that the correct page is loaded by comparing the title with the expected value. Only if this passes will the following command run and “verify” that the text is present in the expected location. The test case then “asserts” the first column in the second row of the first table contains the expected value, and only if this passed will the remaining cells in that row be “verified”.

Commonly Used Selenium Commands

To conclude our introduction of Selenium, we’ll show you a few typical Selenium commands. These are probably the most commonly used commands for building tests.

**open**

opens a page using a URL.

**click/clickAndWait**

performs a click operation, and optionally waits for a new page to load.

**verifyTitle/assertTitle**

verifies an expected page title.

**verifyTextPresent**

verifies expected text is somewhere on the page.

**verifyElementPresent**

verifies an expected UI element, as defined by its HTML tag, is present on the page.

**verifyText**

verifies expected text and its corresponding HTML tag are present on the page.

**verifyTable**

verifies a table’s expected contents.

**waitForPageToLoad**

pauses execution until an expected new page loads. Called automatically when clickAndWait is used.

**waitForElementPresent**

pauses execution until an expected UI element, as defined by its HTML tag, is present on the page.

## Exercise and Review questions:

Can you complete Selenium Testing with IDE?

What browser IDE supports?

What is the language Selenium Core written? What is the language used by IDE to save the script?

What are three attributes of a Selenium step?

What is Test Suite and Test Case in IDE?

Record a Test Case to open www.bing.com page and search for “Pleasanton”.

Section – 9

Selenium Web Driver

**Topics:**

**Introduction to Selenium WebDriver**

**Locators**

**Handling web elements**

**Sample Scripts**

**Execution of Scripts on Different Browsers**

**WebDriver API**

**Introduction to Selenium WebDriver**

Selenium WebDriver fits in the same role as RC did, and has incorporated the original 1.x bindings. It refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just "WebDriver" or sometimes as Selenium 2.

**Selenium 1.0 + WebDriver = Selenium 2.0**

• Selenium Core + WebDriver API

• Selenium Server is not needed.

• WebDriver makes direct calls

• Selenium Grid works only with Selenium Server.

• WebDriver is a clean, fast framework for automated testing of web apps.

• Firefox - WebDriver is implemented as an extension

• Interface WebDriver

WebDriver is the name of the key interface against which tests should be written in Java, the implementing classes one should use are listed as below:

ChromeDriver, EventFiringWebDriver, FirefoxDriver, HtmlUnitDriver, InternetExplorerDriver, PhantomJSDriver, RemoteWebDriver, SafariDriver

## Locators:

* Finding elements by ID
* Finding elements by name
* Finding elements by link text
* Finding elements by Xpath
* Finding Elements by using CSS

To get the WebElement object write the below statement:

***WebElement element = driver.findElement(By.id(“UserName“));***

**By ID:**

driver.findElement(By.id(“123”));

**By Name:**

driver.findElement(By.name(“username”));

**By Link Text:**

driver.findElement(By.linkText(“Sign In and Register”));

By Partial Link Text:

driver.findElement(By.partialLinkText(“Sign In”));

**By CSS:**

driver.findElement(By.cssSelector(“.bUserBar.LikeRealBtn”));

**By XPATH:**

XPath is the language used when locating XML (Extensible Markup Language) nodes. Since HTML can be thought of as an implementation of XML, we can also use XPath in locating HTML elements.

Advantage: It can access almost any element, even those without class, name, or id attributes.

Disadvantage: It is the most complicated method of identifying elements because of too many different rules and considerations.

driver.findElement(By.xpath(”//input[@class=’Page\_eRadioButton’]”));

**By Class Name:**

driver.findElement(By.className(”Page\_eRadioButton”));

**By Tag Name:**

driver.findElement(By.tagName(”option”));

**XPath Usage:**

* If Two object has same attributes
* The object attribute changing dynamically
* If Element has no attributes.

**Syntax for Locator Usage**

|  |  |  |
| --- | --- | --- |
| **Method** | **Target Syntax** | **Example** |
| By ID | id=*id\_of\_the\_element* | id=email |
| By Name | name=*name\_of\_the\_element* | name=username |
| By Name Using Filters | name=*name\_of\_the\_elementfilter*=*value\_of\_filter* | name=tripType value=oneway |
| By Link Text | link=*link\_text* | link=REGISTER |
| Tag and ID | css=*tag*#*id* | css=input#email |
| Tag and Class | css=*tag*.*class* | css=input.inputtext |
| Tag and Attribute | css=*tag*[*attribute*=*value*] | css=input[name=lastName] |
| Tag, Class, and Attribute | css=*tag*.*class*[*attribute*=*value*] | css=input.inputtext[tabindex=1] |

Examples:

<div id="register">

<label for="email" class="required">Email:</label>

<**input id="email" name="register"** class="required" type="text"/>

<label for="fullname" class="optional">Full Name:</label>

<input id="fullname" name="register" class=”optional" type="text"/>

</div>

To locate the email textbox:

id=email

name=register

**Handling Web Elements:**

It is quite straight forward to identify locators for static elements which are clearly defined with static IDs. But in some applications, we come across dynamic elements whose ids/xPaths change every time page reloads. It becomes quite challenging to interact with such dynamic elements.

*// Dynamic Element Locators*

*<button id="Submit-901" />*

*<button id="Submit-902" />*

*<button id="Submit-903" />*

Xpath Position or Absolute Xpath are most frequently used to resolve the dynamic element issues.

<div id="**register**">  
 <label>Email:</label>  
 <input type="text"/>

<label>**Full Name**:</label>  
 <input type="text"/>  
</div>

To locate the Full Name textbox in the above HTML snippet you could use the following:

xpath=id(‘register’)/input[2]

Absolute XPath: The easiest way of finding the xpath is to use the Browser Inspector tool to locate an element and get the xpath of it:

XPath Generated by the tool is : /html/body/div[2]/div/div/footer/section[3]/div/ul/li[3]/a

html/body/div[1]/div[2]/div[1]/div/div/div/div/div[2]/div[2]/div/div/div/div[1]/form/div[1]/div[2]

Relative XPath:

//\*[@id=’social-media’]/ul/li[3]/a

Absolute xpath: /html/body/div[2]/div/div/footer/section[3]/div/ul/li[3]/a

Relative xpath: //\*[@id=’social-media’]/ul/li[3]/a

Absolute xpath is using single slash at the start of the xpath and relative is using double slash.

**Cross Browsers Compatible Testing:**

**(Execution of Scripts on different browsers)**

Cross browser, testing refers to testing the application in multiple browsers like IE, Chrome, Firefox so that we can test our application effectively.IE, Chrome, Firefox so that we can test our application effectively. Cross browser, testing is very important concept in Automation because here the actual automation comes into the picture.



**Why Cross Browser Testing?**

* Font size mismatch in different browsers.
* JavaScript implementation can be different.
* CSS,HTML validation difference can be there.
* Some browser still not supporting HTML5.
* Page alignment and div size.
* Image orientation.
* Browser incompatibility with OS. Etc.

**Internet Explorer:**

Please uncheck all zones at below path.

Open IE, Internet Options --> Go to Security tab --> You can see 4 zones (Internet , Local Intranet, Trusted Sites, Unrestricted sites ) --> Please select each of them and uncheck "Enable Protected mode" check-box for all zones.

**Code Sample:**

package SampleTestcases;

*import org.openqa.selenium.WebDriver;*

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

*import org.openqa.selenium.ie.InternetExplorerDriver;*

*import org.testng.annotations.Parameters;*

*import org.testng.annotations.Test;*

*public class TestCase1 {*

*@Test*

*// Here this parameters we will take from testng.xml*

*@Parameters("Browser")*

*public  void test1(String browser) {*

*if(browser.equalsIgnoreCase("FF")){*

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

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

*driver.get("http://www.facebook.com");*

*driver.quit();*

*}*

*else if(browser.equalsIgnoreCase("IE")){*

*System.setProperty("webdriver.ie.driver", "./server/IEDriverServer.exe");*

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

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

*driver.get("http://www.facebook.com");*

*driver.quit();*

*}*

*}*

*}*

***XML File:***

*<?xml version="1.0" encoding="UTF-8"?>*

*<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">*

*Here parallel is an attribute which specify the mode of execution and thread-count specify how many browser should open*

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

*<test name="Test">*

*<parameter name="Browser" value="FF" />*

*<classes>*

*<class name="SampleTestcases.TestCase1"/>*

*</classes>*

*</test>*

*<test name="Test1">*

*<parameter name="Browser" value="IE" />*

*<classes>*

*<class name="SampleTestcases.TestCase1"/>*

*</classes>*

*</test>*

*</suite>3*

Note- To create testng.xml- Right, click on your testcase then go to TestNG then convert to TestNG> It will generate testng.xml then make changes as per above xml file and finish. You will get testng.xml file inside the project

**Exercise/Activity:**

1. Write Selenium script to launch Chrome browser.
2. What are the Selenium script differences in launching Chrome versus IE?
3. How do you test functional test cases if you don’t have any browsers?

**WebDriver API:**

– Get - Load a new web page in the current browser window.

– Close - Close the current window, quitting the browser if it's the last window currently open.

– findElement - Find the first WebElement using the given method.

– findElements - Find all elements within the current page using the given mechanism.

– getPageSource

– getTitle -

– getCurrentUrl - Get a string representing the current URL that the browser is looking at.

– getWindowHandle - Return an opaque handle to this window that uniquely identifies it within this driver instance.

– getWindowHandles -

– Navigate - An abstraction allowing the driver to access the browser's history and to navigate to a given URL.

– switchTo - Send future commands to a different frame or window.

– Quit - Quits this driver, closing every associated window.

Common Functionalities

**• Drag and Drop**

*WebElement element = driver.findElement(By.name("source"));*

*WebElement target = driver.findElement(By.name("target"));*

***(new Actions(driver)).dragAndDrop(element, target).perform();***

**• Windows & Frames**

*driver.switchTo().window("windowName");*

*for (String handle : driver.getWindowHandles()) { driver.switchTo().window(handle);*

*}*

***driver.switchTo().frame("frameName");***

**• History & Location**

*driver.navigate().to("http://www.example.com");*

***driver.navigate().foward();***

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

**Waits:**

**Implicit wait**

Implicit Waits are used to provide a default waiting time between each consecutive test step/command across the entire test script. Thus, subsequent test step would only execute when the specified amount of time have elapsed after executing the previous test step/command.

**Syntax**  
*drv.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);*

**Explicit Wait**

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.

*WebDriverWait wait = new WebDriverWait(drv,30); wait.until(ExpectedConditions.****visibilityOfElementLocated****(By.xpath("//div[contains(text(),'COMPOSE')]")));*

*// click on the compose button as soon as the "compose" button is visible*

*drv.findElement(By.xpath("//div[contains(text(),'COMPOSE')]")).click();*

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

driver.get("http://bing.com");

WebElement myDynamicElement = (**new** WebDriverWait(driver, **10**))

.until(ExpectedConditions.presenceOfElementLocated(By.id("myDynamicElement")));

**WARNING: Do not mix implicit and explicit waits. Doing so can cause unpredictable wait times.**

WebDriver introduces classes like **WebDriverWait** and **ExpectedConditions** to enforce Explicit waits.

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

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

**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.

**#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 and Exercise:**

1. Using the “Find By ID” - Search text “Dublin” in [www.yahoo.com](http://www.yahoo.com) in both FireFox and Chrome.
2. Using WebDriver code, launch site [www.seleniumhq.org](http://www.seleniumhq.org), click “Download” link using Find by LinkText.
3. In above exercise 2, simulate Navigate Back button through WebDriver code after clicking the Download button.
4. After exercise 3, Use Navigate Forward button to take it back to Download page.

Section – 10 TestNG

**TestNG Framework**

**What is TestNG?**

TestNG is an open source automated testing framework; where **NG** means **N**ext **G**eneration.

TestNG is inspired from JUnit and NUnit, but introducing some new functionalities that make it more powerful and easier to use.

It is designed to cover all categories of tests − unit, functional, end-to-end, integration etc.

It is written in Java and can be used with Java as well as with Java-related languages such as Groovy.

TestNG allows users to do test configuration through XML files and allows them to include (or exclude) respective packages, classes, and methods in their test suite.

**Why do we need testNG in Selenium?**

WebDriver has no native mechanism for generating reports.

TestNG can generate reports based on our Selenium test results.

TestNG can generate the report in a readable format.

There is no more need for a static **main method** in our tests. The sequence of actions is regulated by easy-to-understand **annotations** that do not require methods to be static.

**Uncaught exceptions** are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as **failed steps** in the report.

**TestNG Features**

Supports annotations which are easier to understand.

TestNG uses more Java and OO features.

Support for **Data Driven** Testing using Dataproviders.

Enables user to set execution **priorities** for the test methods.

**Multithreaded** execution.

Supports testing integrated classes (e.g., by default,

no need to create a new test class instance for every test method).

Supports **Group Tests**, **Dependent test** methods, **parallel testing**.

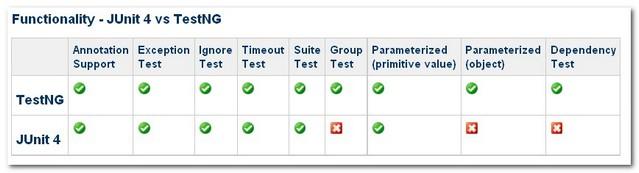
**TestNG Environment:**

**System Requirement:**

JDK 1.5 or above have to be installed in the system.

**TestNG Annotations:**

|  |  |
| --- | --- |
| **@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. |
| **@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. |
| **@BeforeTest** | The annotated method will be run before any test method belonging to the classes inside the <test> tag is run. |
| **@AfterTest** | The annotated method will be run after all the test methods belonging to the classes inside the <test> tag have run. |
| **@BeforeGroups** | This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked. |
| **@AfterGroups** | This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked. |
| **@BeforeMethod** | The annotated method will be run before each test method. |
| **@AfterMethod** | The annotated method will be run after each test method. |
| **@DataProvider** | Marks a method as supplying data for a test method. The annotated method must return an Object[ ][ ], where each Object[ ] can be assigned the parameter list of the test method |
| **@Factory** | Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[ ]. |
| **@Listeners** | Defines listeners on a test class. |
| **@Parameters** | Describes how to pass parameters to a @Test method. |
| **@Test** | Marks a class or a method as a part of the test. |



**TestNG Execution Procedure:**

**Example Program:**

**TestngAnnotation.java**

import org.testng.annotations.Test;

import org.testng.annotations.BeforeMethod;

import org.testng.annotations.AfterMethod;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.AfterTest;

import org.testng.annotations.BeforeSuite;

import org.testng.annotations.AfterSuite;

public class TestngAnnotation {

// test case 1

@Test

public void testCase1() {

System.out.println("in test case 1");

}

// test case 2

@Test

public void testCase2() {

System.out.println("in test case 2");

}

@BeforeMethod

public void beforeMethod() {

System.out.println("in beforeMethod");

}

@AfterMethod

public void afterMethod() {

System.out.println("in afterMethod");

}

@BeforeClass

public void beforeClass() {

System.out.println("in beforeClass");

}

@AfterClass

public void afterClass() {

System.out.println("in afterClass");

}

@BeforeTest

public void beforeTest() {

System.out.println("in beforeTest");

}

@AfterTest

public void afterTest() {

System.out.println("in afterTest");

}

@BeforeSuite

public void beforeSuite() {

System.out.println("in beforeSuite");

}

@AfterSuite

public void afterSuite() {

System.out.println("in afterSuite");

}

}

**Output:**

in beforeSuite

in beforeTest

in beforeClass

in beforeMethod

in test case 1

in afterMethod

in beforeMethod

in test case 2

in afterMethod

in afterClass

in afterTest

in afterSuite

First of all, beforeSuite() method is executed only once.

Lastly, the afterSuite() method executes only once.

Even the methods beforeTest(), beforeClass(), afterClass(), and afterTest() methods are executed only once.

beforeMethod() method executes for each test case but before executing the test case.

afterMethod() method executes for each test case but after executing the test case.

In between beforeMethod() and afterMethod(), each test case executes.

**Suite Test:**

**What is a test suite?**

A **test suite** is a collection of test cases intended to test a behavior or a set of behaviors of software program.

Running a set of test cases together is call executing a **Test Suite.**

In testNG, a suite is represented by an xml file.

A suite can contain one or more tests and is defined by the <suite> tag.

**Why do we need test suite?**

Test suite is to execute multiple test cases at the same time.

**Example program:**

**Test1.java**

**package** testngPackage;

**import** org.testng.annotations.\*;

**public** **class** Test1 {

@Test

**public** **void** PrintTest1(){

System.***out***.println(" Test 1 executed ");

}

}

**Test2.java**

**package** testngPackage;

**import** org.testng.annotations.\*;

**public** **class** Test2 {

@Test

**public** **void** PrintTest2(){

System.***out***.println(" Test 2 executed ");

}

}

**Testng.xml**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >

<suite name="Suite1">

<test name="exampletest1">

<classes>

<class name="testngPackage.Test1" />

</classes>

</test>

<test name="exampletest2">

<classes>

<class name="testngPackage.Test2" />

</classes>

</test>

</suite>

You can give any name to the suite and the test but you need to provide the correct name to the <classes> tag which is a combination of your Package name and Test Case name.

If we run the above Testng.xml, both the test cases will be executed.

In ***test-output* folder**, under the *Suite1* folder, you can see two html files created, exampletest1.html and exampletest2.html.

**Ignore Test**

**What is Ignore Test?**

Ignore test means bypassing a test method.

**Why do we need to ignore a test method and how to do that?**

Sometimes, it happens that our code is not ready and the test case written to test that method/code fails.

In such cases, annotation *@Test(enabled = false)* helps to disable this test case.

If a test method is annotated with *@Test(enabled = false)*, then the test case that is not ready to test is bypassed.

**Example program:**

**Test1.java**

***package*** *testngPackage;*

***import*** *org.testng.annotations.\*;*

***public******class*** *Test1 {*

*@Test*

***public******void*** *PrintTest1(){*

*System.****out****.println(" Test 1 executed ");*

*}*

*}*

*@Test (enabled=false)*

***public******void*** *PrintTest2(){*

*System.****out****.println(" Test 2 executed ");*

*}*

**Output:**

Test 1 executed

PASSED: PrintTest1

In the above example, Test2 method is not executed.

**Group Test:**

**What is Group test?**

Group test is a new innovative feature in TestNG, which doesn’t exist in JUnit framework.

In TestNG users can group **multiple test methods** into a named group.

**Why do we need to group test methods?**

It helps to segregate the tests based on the functionalities/features that the test method verifies.

This helps in executing only a particular set of tests as and when required.

**Example program:**

**groupExamples.java**

package testngPackage;

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");

}

@Test(groups="Smoke Test")

public void testCaseThree(){

System.out.println("Im in testCaseThree - And in Smoke Test Group");

}

@Test(groups="Regression")

public void testCaseFour(){

System.out.println("Im in testCaseFour - And in Regression Group");

}

}

**TestNG.xml**

<?xml version="1.0" encoding="UTF-8"?>

<suite name="Sample Suite">

  <test name="testing">

  <groups>

      <run>

        <include name="Regression"/>

      </run>

    </groups>

    <classes>

       <class name="testngPackage.groupExamples" />

    </classes>

  </test>

</suite>

**Output:**

Im in testCaseFour - And in Regression Group

Im in testCaseOne - And in Regression Group

Im in testCaseTwo - And in Regression Group

In the above example, only the test methods under Regression groups are executed and rest are bypassed.

Group of Groups:

Groups can also include other groups. These groups are called ***MetaGroups*.**

For example, you might want to define a group *all* that includes Regression and Smoke Test. Let's modify our testng.xml file as follows:

<?xml version="1.0" encoding="UTF-8"?>

<suite name="Sample Suit">

<test name="testing">

<groups>

<define name="all">

<include name="Regression"/>

<include name="Smoke Test"/>

</define>

<run>

<include name="all"/>

</run>

</groups>

<classes>

<class name=" testngPackage.groupExamples" />

</classes>

</test>

</suite>

**Output:**

Im in testCaseFour - And in Regression Group

Im in testCaseOne - And in Regression Group

Im in testCaseThree - And in Smoke Test Group

Im in testCaseTwo - And in Regression Group

In the above example, both the Regression group and Smoke test group are executed.

**Exclusion groups:**

You can ignore a group by using the <exclude> tag.

**Dependency Test**

**What is dependency Test?**

Dependency is a feature in TestNG that allows a test method to depend on a single or a group of test methods.

This will help in executing a set of tests to be executed before a test method.

**How it can be done in testNG?**

TestNG allows you to specify dependencies either with:

Using attribute ***dependsOnMethods*** in @Test annotations

Using attribute ***dependsOnGroups*** in @Test annotations.

**Example program:**

**Dependancy.java**

**package** testngpackage;

**import** org.testng.annotations.\*;

**public** **class** Dependancy {

@Test (dependsOnMethods={"method2"})

**public** **void** method1()

{

System.***out***.println("This is method 1");

}

@Test

**public** **void** method2()

{

System.***out***.println("This is method 2");

}

}

**Output:**

This is method 2

This is method 1

In the above example, method1 is executed only after the execution of method2.

If method2 is **failed**, method1 is **skipped.**

**Example program:**

**Dependancy.java**

***package*** *testngpackage;*

***import*** *org.testng.annotations.\*;*

***public******class*** *Dependancy {*

*@Test(dependsOnMethods={"method2"})*

***public******void*** *method1()*

*{*

*System.****out****.println("This is method 1");*

*}*

*@Test*

***public******void*** *method2()*

*{*

*System.****out****.println("This is method 2");*

*throw new RuntimeException();*

*}*

*}*

**Output:**

This is method 2

FAILED: method2

java.lang.RuntimeException

**Example program for dependsOnGroups:**

***package*** *testngpackage;*

***import*** *org.testng.annotations.\*;*

***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");*

*}*

*@Test(groups="Smoke Test")*

***public******void*** *testCaseThree(){*

*System.****out****.println("Im in testCaseThree - And in Smoke Test Group");*

*}*

*@Test(dependsOnGroups={"Smoke Test"})*

***public******void*** *testCaseFour(){*

*System.****out****.println("This is testCaseFour");*

*}*

*}*

**Output:**

Im in testCaseThree - And in Smoke Test Group

Im in testCaseOne - And in Regression Group

Im in testCaseTwo - And in Regression Group

This is testCaseFour

In the above example, testcaseFour will be executed only of testCaseThree is passed.

**Parameterized Test:**

**What is Parameterized testing and why we need?**

As we create software we always wish it should work differently with a different set of data.

We need to verify that our system is taking all set of combinations which it expected to support.

Here comes **Parameterization** in picture. To pass multiple data to the application at runtime, we need to parameterize our test scripts.

This concept which we achieve by parameterization is called **Data Driven Testing .**

***Parameterized tests*** allow developers to run the same test over and over again using different values.

**How it can be done?**

TestNG lets you pass parameters directly to your test methods in two different ways:

**testng.xml**

**Data Providers**

**Example program using testng.xml**

**Test Scenario**

Step 1) Launch browser & go to Google.com

Step 2) Enter a search keyword

Step 3) Verify the inputted value is same as that provided by our test data

Step 4) Repeat 2 & 3 until all values are inputted

**NoParameterWithTestNGXML.java**

***package*** *testngpackage;*

***import*** *java.util.concurrent.TimeUnit;*

***import*** *org.openqa.selenium.By;*

***import*** *org.openqa.selenium.WebDriver;*

***import*** *org.openqa.selenium.WebElement;*

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

***import*** *org.testng.Assert;*

***import*** *org.testng.annotations.AfterClass;*

***import*** *org.testng.annotations.BeforeClass;*

***import*** *org.testng.annotations.Test;*

***public******class*** *NoParameterWithTestNGXML {*

*WebDriver driver;*

*@Test*

***public******void*** *testNoParameter()* ***throws*** *InterruptedException{*

*String author = "John";*

*String searchKey = "Pleasanton";*

*WebElement searchText = driver.findElement(By.name("q"));*

*searchText.sendKeys(searchKey);*

*System.****out****.println("Welcome ->"+author+" Your search key is->"+searchKey);*

*Thread.sleep(3000);*

*System.****out****.println("Value in Google Search Box = "+searchText.getAttribute("value") +" ::: Value given by input = "+searchKey);*

*Assert.assertTrue(searchText.getAttribute("value").equalsIgnoreCase(searchKey));*

*}*

*@BeforeClass*

***public******void*** *beforeClass()*

*{*

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

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

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

*}*

*@AfterClass*

***public******void*** *afterClass() {*

*driver.quit();*

*}*

*}*

In the above example just imagine how complex the code will become when we do this for 3 input combinations.

So lets **parameterize** using testNG.

**ParameterWithTestNGXML.java**

***package*** *testngpackage;*

***import*** *java.util.concurrent.TimeUnit;*

***import*** *org.openqa.selenium.By;*

***import*** *org.openqa.selenium.WebDriver;*

***import*** *org.openqa.selenium.WebElement;*

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

***import*** *org.testng.Assert;*

***import*** *org.testng.annotations.AfterClass;*

***import*** *org.testng.annotations.BeforeClass;*

***import*** *org.testng.annotations.Optional;*

***import*** *org.testng.annotations.Parameters;*

***import*** *org.testng.annotations.Test;*

***public******class*** *ParameterWithTestNGXML {*

*WebDriver driver;*

*@BeforeClass*

***public******void*** *beforeClass()*

*{*

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

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

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

*}*

*@Test*

*@Parameters({"author","searchKey"})*

***public******void*** *testParameterWithXML( @Optional("Abc") String author,String searchKey)* ***throws*** *InterruptedException{*

*WebElement searchText = driver.findElement(By.name("q"));*

*//Searching text in google text box*

*searchText.sendKeys(searchKey);*

*System.****out****.println("Welcome ->"+author+" Your search key is->"+searchKey);*

*Thread.sleep(3000);*

*System.****out****.println("Value in Google Search Box = "+searchText.getAttribute("value") +" ::: Value given by input = "+searchKey);*

*Assert.assertTrue(searchText.getAttribute("value").equalsIgnoreCase(searchKey));*

*}*

*@AfterClass//called after every method*

***public******void*** *afterClass() {*

*driver.quit();*

*}*

*}*

**TestNG.xml**

*<suite name="TestSuite" thread-count="3" >*

*<test name="test1">*

*<parameter name="author" value=" John" />*

*<parameter name="searchKey" value=" San Ramon" />*

*<classes>*

*<class name="testngpackage.ParameterWithTestNGXML">*

*</class>*

*</classes>*

*</test>*

*<test name="test2">*

*<parameter name="author" value=" Mary" />*

*<parameter name="searchKey" value=" Pleasanton" />*

*<classes>*

*<class name="testngpackage.ParameterWithTestNGXML">*

*</class>*

*</classes>*

*</test>*

*</suite>*

**Output:**

Welcome -> John Your search key is-> San Ramon

Value in Google Search Box = San Ramon ::: Value given by input = San Ramon

Welcome -> Mary Your search key is-> Pleasanton

Value in Google Search Box = Pleasanton ::: Value given by input = Pleasanton

Parameter can be defined at two levels:

1. **Suite level**
2. **Test level**

In case if parameter name is same in suite level and test level then test level parameter will get preference over suite level.

**Drawbacks of parameters in xml**:

Parameter value in testng.xml cannot be **type casted** to the corresponding test method's parameter it will throw an error.

If you have **multiple values** for same parameter, it cannot be done.

You can have multiple different parameters but each parameter can only have a single value. Example is **config files** for your script,

Cross browser testing.

If you want to use multiple values for a parameter use **Data Providers.**

**Parameter using DataProviders**

**Why do we need Data Providers?**

To test with multiple sets of data we need to use DataProvider.

**How can you do this?**

This data driven concept is achieved by **@DataProvider**annotation in **TestNG.**

It has only one attribute 'name'. If you do not specify the name attribute then the DataProvider's name will be same as the corresponding method name.

Data provider returns a two dimensional java object to the test method and the test method will invoke M times in a M\*N type of object array.

**Example program:**

**ParameterByDataProvider.java**

**package** testngpackage;

**import** java.util.concurrent.TimeUnit;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.WebElement;

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

**import** org.testng.Assert;

**import** org.testng.annotations.AfterClass;

**import** org.testng.annotations.BeforeTest;

**import** org.testng.annotations.DataProvider;

**import** org.testng.annotations.Test;

**public** **class** ParameterByDataProvider {

WebDriver driver;

@BeforeTest

**public** **void** setup(){

driver = **new** FirefoxDriver();

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

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

}

@AfterClass

**public** **void** afterClass() {

driver.quit();

}

@Test(dataProvider="SearchProvider")

**public** **void** testMethod(String author,String searchKey) **throws** InterruptedException{

{

WebElement searchText = driver.findElement(By.*name*("q"));

searchText.sendKeys(searchKey);

System.***out***.println("Welcome ->"+author+" Your search key is->"+searchKey);

Thread.*sleep*(3000);

String testValue = searchText.getAttribute("value");

System.***out***.println(testValue +"::::"+searchKey);

searchText.clear();

Assert.*assertTrue*(testValue.equalsIgnoreCase(searchKey));

}

}

@DataProvider(name="SearchProvider")

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

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

{ "John", "Pleasanton" },

{ "Mary", "San Ramon" },

{ "Mark", "Dublin" }

{ "Mary1", "San Ramon" },

{ "Mark2", "Dublin" }

};

}

}

**Output:**

Welcome ->John Your search key is->Pleasanton

Pleasanton::::Pleasanton

Welcome ->Mary Your search key is->San Ramon

San Ramon::::San Ramon

Welcome ->Mark Your search key is->Dublin

Dublin::::Dublin

**Invoke Data Provider from other class:**

By default, DataProvider resides in the same class where test method is or its base class.

To put it in some other class we need to make data provider method as static and in test method we need to add an attribute dataProviderClass in @Test annotation.

**Types of Parameters in Dataprovider:**

There are two type of parameters supported by DataProvider method.

**Method**- If the SAME DataProvider should behave differently with different test method , use Method parameter.

**ITestContext**- It can used to create different parameters for test cases based on groups.

**TestNG Exceptions:**

In writing unit tests there can be certain scenarios where we need to verify that an exception is being thrown by the program during execution.

TestNG provides a feature to test such scenarios by allowing the user to specify the type of exceptions that are expected to be thrown by a test method during execution.

It supports multiple values being provided for verification.

If the exception thrown by the test is not part of the user entered list, the test method will be marked as failed.

Here the **expectedExceptions** parameter is used along with the @Test annotation.

**Example Program:**

**TestRuntime.java**

package testng.exception;

import org.testng.annotations.Test;

public class TestRuntime {

@Test(expectedExceptions = ArithmeticException.class)

public void divisionWithException() {

int i = 1 / 0;

}

}

**Assertions**

What is Assertion????

Asserts helps us to verify the conditions of the test and decide whether test has failed or passed. A test is considered successful ONLY if it is completed without throwing any exception.

Here we are verifying if the page title is equal to 'Google' or not. If the page title is not matching with the text / title that we provided, it will fail the test case.

To explain you what is assertion, lets us look into below code sample

*@Test*

*public void testCaseVerifyHomePage() {*

*driver= new FirefoxDriver();*

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

*Assert.assertEquals("Google", driver.getTitle());}*

Like wise there are many Assertions provided by the TestNG. The below are the few which are used commonly.

assertEqual(String actual,String expected) :- It takes two string arguments and checks whether both are equal, if not it will fail the test.

assertEqual(String actual,String expected, String message) :- It takes three string arguments and checks whether both are equal, if not it will fail the test and throws the message which we provide.

assertEquals(boolean actual,boolean expected) :- It takes two boolean arguments and checks whether both are equal, if not it will fail the test.

assertEquals(java.util.Collection actual, java.util.Collection expected, java.lang.String message) :- Takes two collection objects and verifies both collections contain the same elements and with the same order. if not it will fail the test with the given message.

Assert.assertTrue(condition) :- It takes one boolean arguments and checks that a condition is true, If it isn't, an AssertionError is thrown.

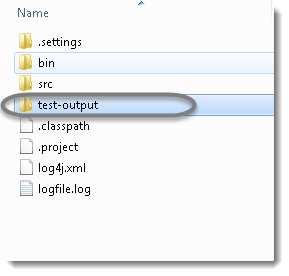
Assert.assertTrue(condition, message) :- It takes one boolean argument and String message. It Asserts that a condition is true. If it isn't, an AssertionError, with the given message, is thrown.

Assert.assertFalse(condition) :- It takes one boolean arguments and checks that a condition is false, If it isn't, an AssertionError is thrown.

Assert.assertFalse(condition, message) :- It takes one boolean argument and String message. It Asserts that a condition is false. If it isn't, an AssertionError, with the given message, is thrown.

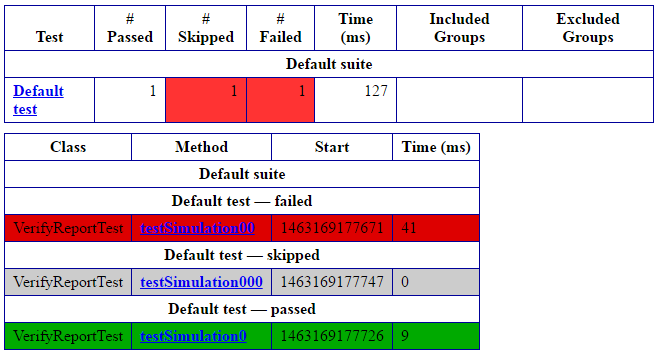
**TestNG Reports**

TestNG also produce HTML reports. To access those reports go to your **Project**folder and open **test-output** folder.



**SUMMARY REPORT**

It is the trimmed version and informs about the test pass/fail/skip count. You can see it from the <emailable-report.html> file. It’s an email friendly report which you can embed and share with the stakeholders.



**Building a BAT file**

Step 1: Open notepad  
Step 2: Paste the below lines of code - You may need to add your project location. In the example, project location is set as 'C:\Selenium\TestNGBatchExample'.  
Step 3: Save the file as 'testNGBatchFile.bat' in location that you want to save.

*set projectLocation=C:\Selenium\TestNGBatchExample*

*cd %projectLocation%*

*set classpath=%projectLocation%\bin;%projectLocation%\lib\\**

*java org.testng.TestNG %projectLocation%\testng.xml*

*pause*

**Example and Exercise:**

1. Create a TestNG project with 3 test cases – testCase300, testCase200, testCase100; Using DependOnMethods, run a test cases in below order:

testCase100

testCase200

testCase300

1. Using assertion, check google page title if it has ‘Google’.

Section – 11 WebDriver

**Topics:**

Concepts of Window ID

Protractor Elements

Handling Pop ups

Handling web tables

Logging and Debugging

Integration with Jenkins & Maven

## Window Handles

## Selenium WebDriver assigns an alphanumeric id to each window as soon as the WebDriver object is instantiated. This unique alphanumeric id is called window handle. Selenium uses this unique id to switch control among several windows. In simple terms, each unique window has a unique ID, so that Selenium can differentiate when it is switching controls from one window to the other.

## GetWindowHandle Command

## Purpose: To get the window handle of the current window.

## *String handle= driver.getWindowHandle();*

## GetWindowHandles Command

## Purpose: To get the window handle of all the current windows.

## *Set<String> handle= driver.getWindowHandles();//Return a set of window handle*

## SwitchTo Window Command

## Purpose: WebDriver supports moving between named windows using the “switchTo” method.

## *driver.switchTo().window("windowName");*

## Handling Pop ups

**Alert**

* ***accept()****To accept the alert*
* ***dismiss()****To dismiss the alert*
* ***getText()****To get the text of the alert*
* ***sendKeys()****To write some text to the alert*

**Simple alert**

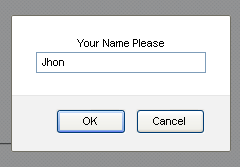
Simple alerts just have a***OK*** button on them.



**Confirmation Alert**



**Prompt Popup**



try

{

//Handle the alert pop-up using seithTO alert statement

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

//Print alert is present

System.out.println("Alert is present");

//get the message which is present on pop-up

String message = alert.getText();

//print the pop-up message

System.out.println(message);

alert.sendKeys("");

//Click on OK button on pop-up

alert.accept();

}

catch (NoAlertPresentException e)

{

//if alert is not present print message

System.out.println("alert is not present");

}

## Handling Web Table/HTML Table

## In this module we will learn about the web tables or html tables in a web page, tags available in html and how to handle web tables dynamically.

## Web tables are basically group of elements that are logically stored in a row and column format. It is used to organize similar information in a web page.

## Below is an example of Html table:

## Handling web tables in selenium

## Below is the snippet of html structure of an html table:

## Handling web tables in selenium 1

## Below tags are generally defined in an html tables:

## 1.’table’ tag defines html table.

## 2.’tbody’ tag defines container for rows and columns.

## 3.’tr’ defines rows in an html table.

## 4.’td’/’th’ define column of an html table.

## Find the details of a web table:

## There are many ways we can handle a web table.

## Approach #1:

## Below is the xpath of one of the cell in html table. Let’s say “firstname”

## //div[@id=’main’]/table[1]/tbody/tr[1]/th[1]

## tr[1] defines first row and th[1] defines first column.

## If number of rows and columns are always constant, let’s say our html table will always have 5 rows and 3 columns.

## *for(int numberOfRows=1; numberOfRows<=5; numberOfRows++)*

## *{*

## *for(int numberOfCol=1; numberOfCol <=3; numberOfCol++)*

## *{*

## *System.out.println(driver.findElement(By.xpath(“//div[@id='main']/table[1]/tbody/tr*

## *[“+numberOfRows+”]/th[“+numberOfCol+”]”)));*

## *}*

## *}*

## Yahoo Mail Test:

## pleasantontest

## pleasanton123

## Logging and Debugging

## Why use Log4j?

## It is an open source

## With Log4j, it is possible to store the flow details of our Selenium Automation in a file or databases

## Log4j is used for large as well as small projects

## In Log4j, we use log statements rather than SOPL statements in the code to know the status of a project while it is executing

## Loggers: It is responsible for logging information. To implement loggers into a project following steps need to be performed -

## Create an instance for logger class: Logger class is a Java-based utility that has got all the generic methods already implemented to use log4j

## Define the Log4j level: Primarily there are five kinds of log levels

## All - This level of logging will log everything ( it turns all the logs on )

## DEBUG – print the debugging information and is helpful in development stage

## INFO – print informational message that highlights the progress of the application

## WARN – print information regarding faulty and unexpected system behavior.

## ERROR – print error message that might allow system to continue

## FATAL – print system critical information which are causing the application to crash

## OFF – No logging

## Log4j Sample Script:

## import org.openqa.selenium.By;

## import org.openqa.selenium.WebDriver;

## import org.openqa.selenium.firefox.FirefoxDriver;

## import org.apache.log4j.Logger;

## public class LoggingDemo {

## /\*\*

## \* @param args

## \*/

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

## WebDriver driver = new FirefoxDriver();

## Logger log = Logger.getLogger("devLogger");

## 

## driver.get("http://healthunify.com/bmicalculator/"); log.debug("opening webiste");

## driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);

## log.debug("entring weight");

## driver.findElement(By.name("wg")).sendKeys("87");

## log.debug("selecting kilograms");

## driver.findElement(By.name("opt1")).sendKeys("kilograms");

## log.debug("selecting height in feet");

## driver.findElement(By.name("opt2")).sendKeys("5");

## log.debug("selecting height in inchs");

## driver.findElement(By.name("opt3")).sendKeys("10");

## log.debug("Clicking on calculate");

## driver.findElement(By.name("cc")).click();

## log.debug("Getting SIUnit value");

## String SIUnit = driver.findElement(By.name("si")).getAttribute("value");

## log.debug("Getting USUnit value");

## String USUnit = driver.findElement(By.name("us")).getAttribute("value");

## log.debug("Getting UKUnit value");

## String UKUnit = driver.findElement(By.name("uk")).getAttribute("value");

## log.debug("Getting overall description");

## String note = driver.findElement(By.name("desc")).getAttribute("value");

## 

## System.out.println("SIUnit = " + SIUnit);

## System.out.println("USUnit = " + USUnit);

## System.out.println("UKUnit = " + UKUnit);

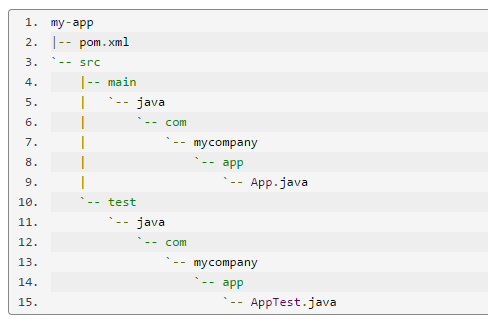
## System.out.println("note = " + note);

## driver.quit();

## }

## }

## Maven



**validate**: validate the project is correct and all necessary information is available

**compile**: compile the source code of the project

**test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

**package**: take the compiled code and package it in its distributable format, such as a JAR.

**integration-test**: process and deploy the package if necessary into an environment where integration tests can be run

**verify**: run any checks to verify the package is valid and meets quality criteria

**install**: install the package into the local repository, for use as a dependency in other projects locally

**deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

There are two other Maven lifecycles of note beyond the *default* list above. They are

**clean**: cleans up artifacts created by prior builds

**site**: generates site documentation for this project

mvn install

This command executes each default life cycle phase in order (validate, compile, package, etc.), before executing install. You only need to call the last build phase to be executed, in this case, install:

In a build environment, use the following call to cleanly build and deploy artifacts into the shared repository.

mvn clean deploy

The same command can be used in a multi-module scenario (i.e. a project with one or more subprojects). Maven traverses into every subproject and executes clean, then executes deploy (including all of the prior build phase steps).

**POM.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

**<groupId>testGrp</groupId>**

**<artifactId>TestArt</artifactId>**

**<version>0.0.1-SNAPSHOT</version>**

<packaging>jar</packaging>

<name>TestArt</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

**<dependencies>**

**<dependency>**

**<groupId>junit</groupId>**

**<artifactId>junit</artifactId>**

**<version>3.8.1</version>**

**<scope>test</scope>**

**</dependency>**

**</dependencies>**

</project>

## Jenkins:

**CI:**

Continuous Integration is a development practice that requires developers to integrate code into a shared repository at regular intervals. This concept was meant to remove the problem of finding later occurrence of issues in the build lifecycle. Continuous integration requires the developers to have frequent **builds**. The common practice is that whenever a code commit occurs, a build should be triggered.

Download URL

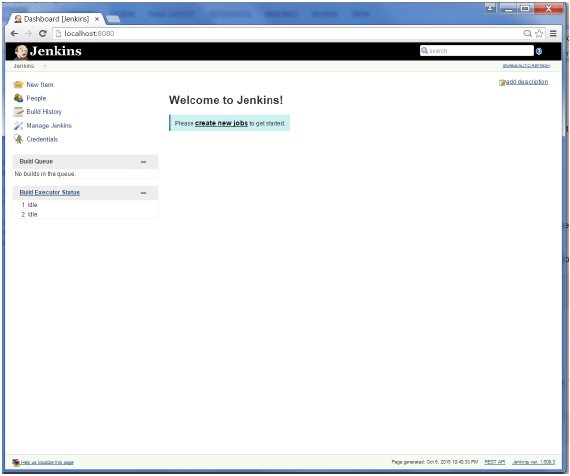
<https://jenkins-ci.org/>

D:\>Java –jar Jenkins.war

Accessing Jenkins

Once Jenkins is up and running, one can access Jenkins from the link

**http://localhost:8080**



Section – 12 Frameworks

**Topics:**

Why Framework

Introduction to Frameworks

Types of Frameworks

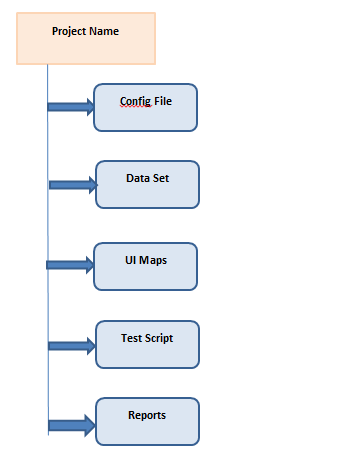
**Why Frameworks:**

**Selenium**will allow you to automate all your web (browsers) actions automations.

**Junit/TestNG** as the testing framework, including their default reports system

**Maven** for the project management and lifecycle

**Jenkins** is a good integration tool that will easily run the setup above



**Data Set:**

Data set stores the data files, Script reads test data from external data sources and executes test based on it.

**Test Script:**

A test is considered as a single action or a sequence of actions, that defines whether a specific feature meets functional requirements. It has multiple test files / packages / class files

**Test Report:**

Test report/results is a document which contains summary of test activities.

Advantages:

1. **Less Technical Expertise:** Once Framework is set up, manual testers or non technical testers can easily write test scripts for automation.
2. **Easy To Understand:** As it is maintained in Excel sheet and no coding is exposed, the test scripts are easy to read and understand. Keywords & actions resemble so closely manual test cases, which become easier to write and maintain.
3. **Early Start:** You can start building Keyword Driven test cases before the application is delivered, as Object Repository can be easily set up at the later stage. Using information gathered from Requirements or other documentation, keyword data tables can be created that mirror corresponding manual test procedures.
4. **Re-usability of component:** With implementing modularization in Keyword Driven, Re-usability can be further increased.
5. **Re-usability of code:** As there is just one Execution Engine in Keyword Driven Framework, it encourage extreme code re-usability.

**Types of FrameWorks :**

Page Object Model

Data Driven

TestNG Reporting

Hybrid Framework

User-Defined Functions (Key-word driven)

**Data-Driven:**

file1 = **new** FileInputStream(**new** File("TestIn\\LoanApplications.xls"));

HSSFWorkbook workbook = **new** HSSFWorkbook(file1);

HSSFSheet sheet = workbook.getSheet("ConfigInfo");

String Acct = sheet.getRow(1).getCell(1).getStringCellValue();

String UserId = sheet.getRow(1).getCell(2).getStringCellValue();

String Passwd = sheet.getRow(1).getCell(3).getStringCellValue();

file1.close();

workbook.close();

homePage1.login(Acct, UserId, Passwd) ;

**Example and Exercise:**

1. Using <> framework, launch the application in Chrome and login to the application using the given UserID/Password.
2. Launch the web application using FireFox and validate the Page Title for the given Title <>

**Additional Exercise:**

1. Execute failed Test cases in TestNG.

## Execute failed test cases using Selenium

## Steps

1-If your test cases are failing then once all test suite completed then you have to refresh your project . Right click on project > Click on refresh or Select project and press f5.

2-Check test-output folder, at last, you will get testng-failed.xml

3- Now simply run testng-failed.xml.

## Appendix:

**Data-Driven Testing:**

import java.io.FileReader;

import java.io.IOException;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

import com.opencsv.CSVReader;

public class CSVRead {

//Provide CSV file path. It Is In D: Drive.

String CSV\_PATH="D:\\Detail.csv";

WebDriver driver;

@BeforeTest

public void setup() throws Exception {

driver = new FirefoxDriver();

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

driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);

driver.get("http://only-testing-blog.blogspot.in/2014/05/form.html");

}

@Test

public void csvDataRead() throws IOException{

CSVReader reader = new CSVReader(new FileReader(CSV\_PATH));

String [] csvCell;

//while loop will be executed till the last line In CSV.

while ((csvCell = reader.readNext()) != null) {

String FName = csvCell[0];

String LName = csvCell[1];

String Email = csvCell[2];

String Mob = csvCell[3];

String company = csvCell[4];

driver.findElement(By.xpath("//input[@name='FirstName']")).sendKeys(FName); driver.findElement(By.xpath("//input[@name='LastName']")).sendKeys(LName);

driver.findElement(By.xpath("//input[@name='EmailID']")).sendKeys(Email);

driver.findElement(By.xpath("//input[@name='MobNo']")).sendKeys(Mob); driver.findElement(By.xpath("//input[@name='Company']")).sendKeys(company);

driver.findElement(By.xpath("//input[@value='Submit']")).click();

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

}

}

}

**Web Table**

@Test

public void Handle\_Dynamic\_Webtable() {

**//To locate table.**

WebElement mytable = driver.findElement(By.xpath(".//\*[@id='post-body-8228718889842861683']/div[1]/table/tbody"));

**//To locate rows of table.**

List<WebElement> rows\_table = mytable.findElements(By.tagName("tr"));

**//To calculate no of rows In table.**

int rows\_count = rows\_table.size();

**//Loop will execute till the last row of table.**

for (int row=0; row<rows\_count; row++){

**//To locate columns(cells) of that specific row.**

List<WebElement> Columns\_row = rows\_table.get(row).findElements(By.tagName("td"));

**//To calculate no of columns(cells) In that specific row.**

int columns\_count = Columns\_row.size();

System.out.println("Number of cells In Row "+row+" are "+columns\_count);

**//Loop will execute till the last cell of that specific row.**

for (int column=0; column<columns\_count; column++){

**//To retrieve text from that specific cell.**

String celtext = Columns\_row.get(column).getText();

System.out.println("Cell Value Of row number "+row+" and column number "+column+" Is "+celtext);

}

System.out.println("--------------------------------------------------");

}

}

}

**Scrolling**

public class Scrolling {

WebDriver driver;

@BeforeTest

public void setup() throws Exception {

driver =new FirefoxDriver();

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

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

driver.get("http://only-testing-blog.blogspot.in");

}

@Test

public void Scroll\_Page() throws IOException, InterruptedException {

//To scroll down web page by 600 pixels In x(vertical) direction.

//You can y parameter to scroll page In horizontal direction.

JavascriptExecutor javascript = (JavascriptExecutor) driver;

javascript.executeScript("window.scrollBy(0,600)", "");

Thread.sleep(3000);

//To scroll up web page by 300 pixels In x(vertical) direction.

javascript.executeScript("window.scrollBy(0,-300)", "");

}

}

Scroll down to bottom of page

If you wants to scroll down to bottom of the page, You can use bellow given @Test method In above example.

@Test

public void Scroll\_Page() throws IOException, InterruptedException {

//Scroll down to bottom of the page.

JavascriptExecutor javascript = (JavascriptExecutor) driver;

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

}

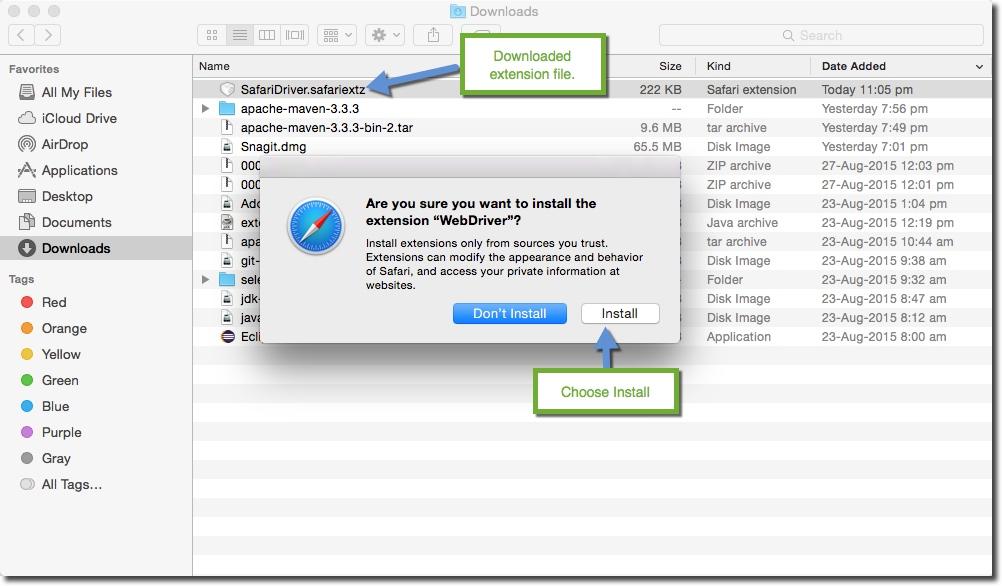
1. **Safari Browser**

## How to run Selenium tests in Safari browser?

### Step 1 – Set Up WebDriver Extension for *Safari* browser

1) *Download the Safari Browser Extension –* Latest version of *Safari browser extension* can be downloaded [*here*](https://github.com/SeleniumHQ/selenium/raw/master/javascript/safari-driver/prebuilt/SafariDriver.safariextz).

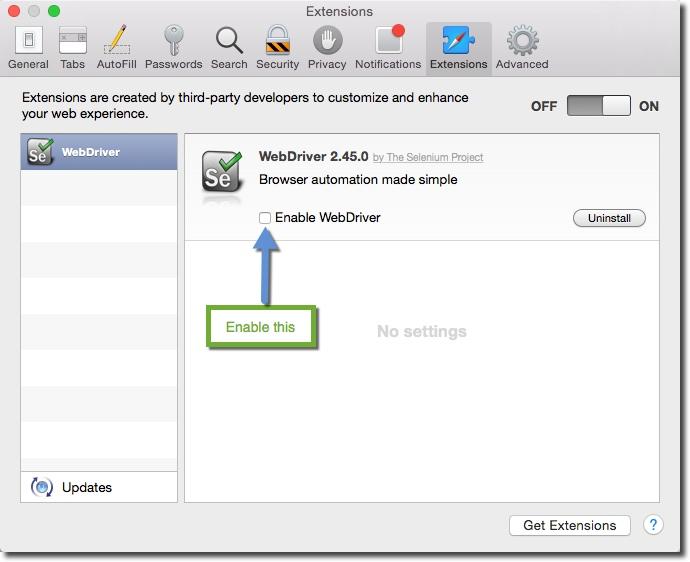
2) *Install the Safari Browser Extension –* Go to the folder where file has downloaded and double click on it. You will get a prompt, as shown in image below, there select “*Install*“



Download folder image

3) *Enable WebDriver browser Extension –* Now open the *preferences* pane on *Safari* browser. Go to *Safari >> Preferences*  and open the preferences window.

In the  preferences window select *Extension*. There you will find Selenium web driver listed in the extensions list, select the check box. As shown in the below image



Safari extension window

*Note : Make sure that you have “Enable WebDriver“ check-box enabled.*

4) *Restart your Browser –* All you have to do here is to restart your browser.

### Write Selenium WebDriver code to Launch Safari

As I said earlier running selenium tests in Safari is exactly similar to working with Firefox or IE. Safari browser is represented by a class called *SafariDriver* in the *org.openqa.selenium.safari package.* All we have to do is create an instance of the SafariDriver class. Below is a sample code to do that

How to run Selenium tests in Safari browser?  
  
  
Step 1 – Set Up WebDriver Extension for Safari browser  
  
1) Download the Safari Browser Extension – Latest version of Safari browser extension can be downloaded here.  
  
2) Install the Safari Browser Extension – Go to the folder where file has downloaded and double click on it. You will get a prompt, as shown in image below, there select “Install“

DownloadFolder  
Download folder image  
  
  
3) Enable WebDriver browser Extension – Now open the preferences pane on Safari browser. Go to Safari >> Preferences  and open the preferences window.  
  
In the preferences window select Extension. There you will find Selenium web driver listed in the extensions list, select the check box. As shown in the below image  
  
DownloadFolder  
Safari extension window  
Note : Make sure that you have “Enable WebDriver“ check-box enabled.  
  
4) Restart your Browser – All you have to do here is to restart your browser.  
  
  
  
Write Selenium WebDriver code to Launch Safari  
  
As I said earlier running selenium tests in Safari is exactly similar to working with Firefox or IE. Safari browser is represented by a class called SafariDriver in the org.openqa.selenium.safari package. All we have to do is create an instance of the SafariDriver class.

How to run Selenium tests in Safari browser?  
  
Step 1 – Set Up WebDriver Extension for Safari browser  
  
1) Download the Safari Browser Extension – Latest version of Safari browser extension can be downloaded here.  
  
2) Install the Safari Browser Extension – Go to the folder where file has downloaded and double click on it. You will get a prompt, as shown in image below, there select “Install“  
  
DownloadFolder  
Download folder image  
  
  
3) Enable WebDriver browser Extension – Now open the preferences pane on Safari browser. Go to Safari >> Preferences and open the preferences window.  
  
In the preferences window select Extension. There you will find Selenium web driver listed in the extensions list, select the check box. As shown in the below image  
  
DownloadFolder  
Safari extension window  
Note : Make sure that you have “Enable WebDriver“ check-box enabled.  
  
4) Re-start your Browser – All you have to do here is to restart your browser.  
  
  
  
Write Selenium WebDriver code to Launch Safari  
  
As we saw earlier running selenium tests in Safari is exactly similar to working with Firefox or IE. Safari browser is represented by a class called SafariDriver in the org.openqa.selenium.safari package. All we have to do is create an instance of the SafariDriver class. Below is a sample code to do that  
  
package Usage;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.safari.SafariDriver;

public class SafariUsage {

public static void main(String[] args)

{

WebDriver driver = new SafariDriver();

driver.get("http://store.test.com");

//Find some element

WebElement element = driver.findElement(By.id("login"));

element.click();

}

}

Here you can see that all we have to do is create an instance of *SafariDriver* and use it like a regular *WebDriver* that we have been using it like for other browsers.

*Issues:*

1. *Only http:// and https:// protocols are supported on Safari.*
2. *Safari is not able to handle alerts, so we have to suppress alerts in case of Safari. We will learn this in another chapter.*
3. ***MicroSoft Edge Browser:***
4. Correct version of MicrosoftWebDriver needs to be downloaded, based on the OS Build number
5. Go to Start > Settings > System > About and note down the OS Build number from Edge.
6. Download the proper version of the driver from this link - <https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/>
7. If the file that's downloaded is .msi, then install it to get the .exe driver. For one of the release, direct .exe can be downloaded.
8. Once the MicrosoftWebDriver.exe is downloaded, we can use it in our test script using either System.setProperty("webdriver.edge.driver", "driver location") or using environment variable

The sample script would be like this -

System.setProperty("webdriver.edge.driver","C:\\Driver\\MicrosoftWebDriver.exe"); //put actual location

WebDriver driver = new EdgeDriver();

driver.get("your link");

**How to take Screenshot?**

package Testing\_Pack;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.IOException;

import java.util.concurrent.TimeUnit;

import javax.imageio.ImageIO;

import org.apache.commons.io.FileUtils;

import org.openqa.selenium.By;

import org.openqa.selenium.OutputType;

import org.openqa.selenium.Point;

import org.openqa.selenium.TakesScreenshot;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.annotations.BeforeTest;

import org.testng.annotations.Test;

public class elementScreenshot {

WebDriver driver;

@BeforeTest

public void setup() throws Exception {

driver =new FirefoxDriver();

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

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

driver.get("http://only-testing-blog.blogspot.in/2014/09/selectable.html");

}

@Test

public void captureScreenshot() throws Exception {

//Locate Image element to capture screenshot.

WebElement Image = driver.findElement(By.xpath("//img[@border='0']"));

//Call captureElementScreenshot function to capture screenshot of element.

captureElementScreenshot(Image);

}

public void captureElementScreenshot(WebElement element) throws IOException{

//Capture entire page screenshot as buffer.

//Used TakesScreenshot, OutputType Interface of selenium and File class of java to capture screenshot of entire page.

File screen = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);

//Used selenium getSize() method to get height and width of element.

//Retrieve width of element.

int ImageWidth = element.getSize().getWidth();

//Retrieve height of element.

int ImageHeight = element.getSize().getHeight();

//Used selenium Point class to get x y coordinates of Image element.

//get location(x y coordinates) of the element.

Point point = element.getLocation();

int xcord = point.getX();

int ycord = point.getY();

//Reading full image screenshot.

BufferedImage img = ImageIO.read(screen);

//cut Image using height, width and x y coordinates parameters.

BufferedImage dest = img.getSubimage(xcord, ycord, ImageWidth, ImageHeight);

ImageIO.write(dest, "png", screen);

//Used FileUtils class of apache.commons.io.

//save Image screenshot In D: drive.

FileUtils.copyFile(screen, new File("D:\\screenshot.png"));

}

}

**How to?**

**public** **void** **getscreenshot**() **throws** Exception

{

File scrFile=((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);           

FileUtils.copyFile(scrFile, **new** File("D:\\screenshot.png"));

}

**try{**

**//the below statement will throw an exception as the element is not found, Catch block will get executed and takes the screenshot.**

**driver.findElement(By.id("testing")).sendKeys("test");**

**//if we remove the below comment, it will not return exception and screen shot method will not get executed.**

**//driver.findElement(By.id("gbqfq")).sendKeys("test");**

**}**

**catch (Exception e){**

**System.out.println("I'm in exception");**

**//calls the method to take the screenshot.**

**getscreenshot();**

**}**

<https://github.com/mozilla/geckodriver/releases>

gecko driver download – Search in Google.

Java Project

Package

Class

Methods

Java Statements

Development:

.JAVA - Source Code

.CLASS -

JRE -> JVM - > JIT

CLASS

OBJECT

**ClassName** newObjName = new **ClassName()** ;

Shapes Rectangle = new Shapes() ;

Rectangle.Draw() ;

BluePrint -CLass

Home - Object

Animals Tiger = new Animals() ;

Java1

Java2

Java3 {

Java1 J1 = new Java1() ;

Person Teacher = new Person() ; //Instantiation

}