Topics

- Automation definition
- Disadvantages of manual
- Advantage of Automation
- Selenium java Architecture
- Upcasting in Selenium
- Steps to install/integrate selenium server to the java project
- Handling Browser
- What is an WebElement
- Locators
- Interview questions
- scroll down
- What is frame
- Actions class
- Handling pop-up
- what is findElements
- handle Auto Suggestion
- select List Box
- Page Object Model - Framework Design pattern
- TESTNG Framework
- SELENIUM GRID
- SELENIUM FRAMEWORK
- Assert and Verify
- List of Exceptions
- JENKINS
- Selenium Waits

Automation definition

Testing is an application feature with the help of automation tool is known as automation testing.

Important Automation tools

1) Selenium

2)QTP ------- (paid

3) Sahi

4)Sahi pro

5)protector

6)Applum

7)Selendroid

*MNC mostly use their developed tools.

Disadvantages of manual

1) Require more resources.

2) It is time consuming.

3) Compatibility testing (Cross browser testing) is very difficult in manual testing because if we have to check build on different different browser then if one browser take 10 min then 6 browser takes 60 minutes.

4) Test cycle duration will be increased.

5) more human efforts are required.

6) If we do manual regression testing then it is time consumming.

Advantage of Automation

1) Less resources are required.

2) compatibility testing is easy & less time consuming.

3) Regression testing is easy in Automation & less time consuming

4)Test cycle duration will be reduce

5) Reusability of scripts:

Test cases are converted into program that is called as test script .We can use test script for multiple times.

why we choose selenium

1) It is open source.

2) Supported by multiple languages 1)Java 2) Python 3) c# 4) pearl

3) compatibility Cross browser test is possible.

Disadvantage of selenium

1) we can automate only web base application.

2) we can not test captcha or barcode.

(3) we can not do file uploading and downloading

4) we can perform regression testing but can not perform Ad-hoc testing

Why Selenium is so popular and demanding?

Selenium is popular and demanding due to the following features.

1. it is an open source tool freely available on internet
2. No project cost involved
3. No licence requires
4. Can be easily customized to integrate with other Test Management tools like ALM, Bugzilla etc.
5. It supports almost 13 different software languages
   - Java
   - C#
   - Ruby
   - Pytho
   - Perl
   - Php
   - JavaScript
   - JavaScript (Node JS)
   - Haskel
   - R
   - Dar
   - TCL
   - Objective – C

6) It supports almost all the browsers.(Firefox, Chrome, Internet Explorer etc) and hence, cross browser testing/compatibility testing can be performed using selenium.

7) It supports almost all the Operating System (MAC, Windows, LINUX etc) and hence, cross platform testing can also be performed.

2.)What are the different flavours of Selenium ?

●Selenium Core (Developed by a company called Thought Works way back in 2004)

●Selenium IDE (supports only Mozilla Firefox - supports record and playback feature)

●Selenium RC (Remote Control - Version is 1.x) (Used for parallel execution of automation scripts on multiple remote systems)

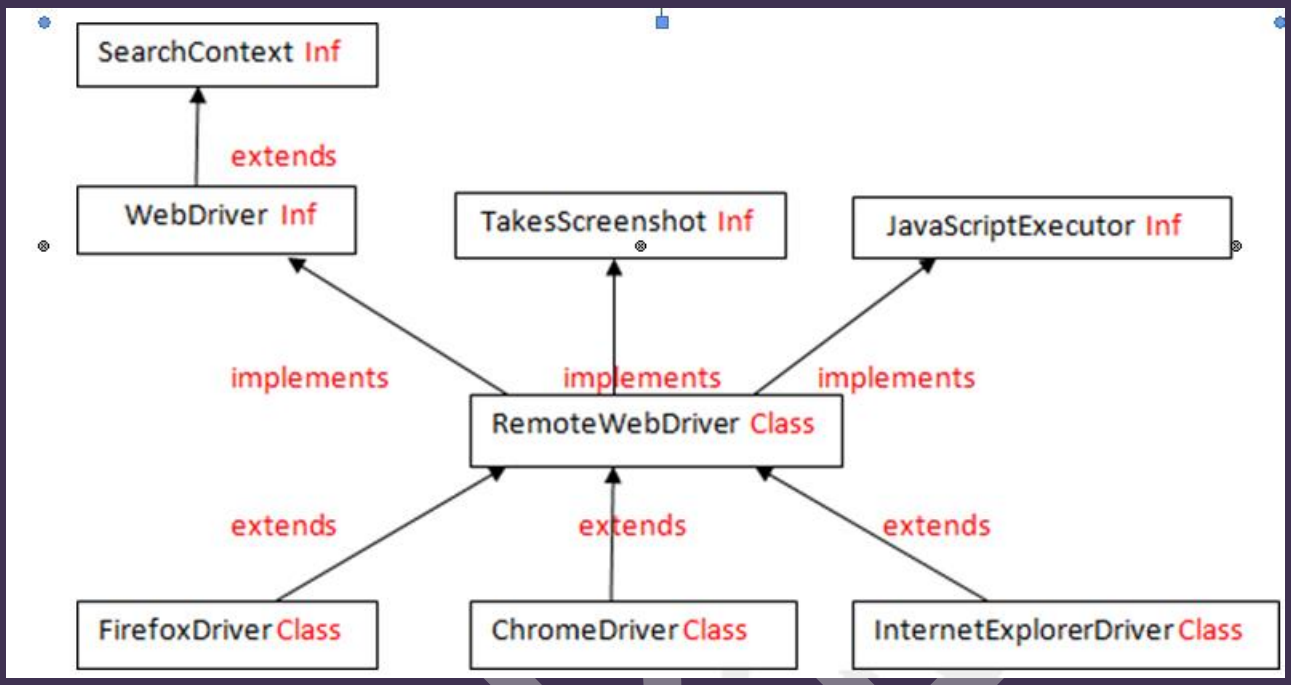
●Selenium WebDriver (Version is 2.x and 3.x)

Note:

Selenium WebDriver version 3.x is no longer capable of running Selenium RC directly, rather it does through emulation and via an interface called WebDriverBackedSelenium.But it does support Selenium Grid directly.

Selenium Grid:

4)Selenium java Architecture



1. **SearchContext is the supermost interface present in selenium webdriver.**
2. **An interface called WebDriver extends SearchContext interface.**
3. **A total of 13 interfaces are available in selenium, which is implemented by a super most class called RemoteWebDriver**
4. **Remote WebDriver is again extended by few browsers specific child classes such as,**
    - **FirefoxDriver class to automate on Firefox browser.**
    - **ChromeDriver class to automate on Chrome browser,**
    - **InternetExplorerDriver class to automate on IE and so on…….**

NOTE :

All the above mentioned **interfaces and classes** are present in a package called

"org.openqa.selenium".To view any information about Selenium interfaces, classes

and methods, navigate to the below page.

https://github.com/SeleniumHQ/selenium/tree/master/java/client/src/org/openqa/selenium **Highlighted below in red is the navigation path.**

2. List down all the methods present in below interfaces of Selenium WebDriver.

Methods of SearchContext interface :

1. findElement()
2. findElements()

Methods of WebDriver interface:

1. close()
2. get()
3. getTitle()
4. getPageSource()
5. getCurrentUrl()
6. getWindowHandle()
7. getWindowHandles()
8. manage()
9. navigate()
10. quit()
11. switchTo()

Methods of TakesScreenshot interface:

1. getScreenshotAs(args)

Methods of JavascriptExecutor interface:

1. executeScript()
2. executeAsyncScript() - we don't use this for automation

Methods of WebElement interface:

1. clear()
2. click()
3. getAttribute()
4. getCssValue()
5. getLocation()
6. getRect()
7. getSize()
8. getTagName()
9. getText()
10. isDisplayed()
11. isEnabled()
12. isSelected()
13. sendKeys()
14. submit()

6. Where did you use Upcasting in Selenium ?

WebDriver driver = new FirefoxDriver();

Explain the above statement.

1. WebDriver is an interface in Selenium that extends the supermost interface called SearchContext.

2. driver is the upcasted object or WebDriver interface reference variable.

3. " = " is an assignment operator.

4. new is a keyword using which object of the FirefoxDriver class is created.

5. FirefoxDriver() is the constructor of FirefoxDriver class which initialises the object and it will also launch the firefox browser.

7.Steps to install/integrate selenium server to the java project

1. Launch eclipse and go to package explorer [navigation path:- Window menu → Show View → Package Explorer]
2. Create a java project [File → New→ Java Project]
3. Right click on Java Project and add a new folder with name “driver” [File → New→ Folder]
4. copy geckodriver.exe file from your system and paste it into this driver folder
5. Similarly, create another folder with name “jar”and copy Selenium Standalone Server.jar file into this jar folder.
6. Expand the jar folder and right click on Selenium Standalone Server.jar file → select Build Path → select

Add to Build Path

7. As soon as you add any .jar files to build path, a new folder will be available called “Reference Libraries”

under the package explorer section and you can see the .jar file is added to this “Reference Libraries”

8. To remove the .jar file from the java build path, go to the Reference Libraries → select the .jar file → right click → select build path → Remove from build path.
9. Other way of adding .jar file to java build path is : right click on the project → build path → configure build path → Libraries tab → Add External jars → select the .jar file → Apply → ok

==========

8. This program demonstrates Upcasting concept (FirefoxDriver class object to WebDriver interface) and accessing various methods of WebDriver interface

```
package qspiders;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver


public class UpcastingToWebDriver_LaunchBrowser

{

public static void main(String[] args) throws InterruptedException

{
```

```
 //setting the path of the gecko driver executable
System.setProperty("webdriver.gecko.driver", ".\\driver\\geckodriver.exe");
 //Launch the firefox browser
WebDriver driver = new FirefoxDriver();
 //Enter the url
driver.get("http://www.google.com");
 //Get the title of the google page and print it on the console
 String title = driver.getTitle();
System.out.println("the title of the page is :"+ title);
 //Get the URL of the google page and print it on the console
String currentUrl = driver.getCurrentUrl();
System.out.println("the URL of the page is :"+ currentUrl);
 //Get the source code of the google page and print it on the console
String pageSource = driver.getPageSource();
System.out.println("the source code of the page is :"+ pageSource);
//Halt the program execution for 2 seconds
 Thread.sleep(2000);
// Close the browser
 driver.close();
}
}
```

Question : How to capture screenshots in Selenium ?

Answer :

We capture screenshots in Selenium using getScreenshotAs() method of TakesScreenshot interface.

Steps to take screenshot:

1. Create an object of specific browser related class (eg : FirefoxDriver) and then upcast it to WebDriver object (eg : driver)
2. Typecast the same upcasted driver object to TakesScreenshot interface type.
3. Using the typecasted object, we call getScreenshotAs(OutputType.FILE) which in turn returns the source file object.
4. Using the File IO operations (i.e FileUtils class), we store the screenshots to desired location in the project.

Selenium Code :

```
package pack1;
 import java.io.File;
import java.io.IOException;
 import java.util.Date;
import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
 public class CaptureScreenshot_ActiTIMEPage extends BaseClass{
        public static void main(String[] args) throws IOException {
//Creating an object of Date class
 Date d = new Date();
//Printing the actual date
 String date1 = d.toString(); System.out.println(date1);
//replacing the colon present in the date timestamp format to "_" using replaceAll()
//method of String class
String date2 = date1.replaceAll(":", "_");
 System.out.println(date2);
```

```
//Enter the url
driver.get("https://localhost:8080/login.do");
//Typecasting the driver object to TakesScreenshot interface type.
TakesScreenshot ts = (TakesScreenshot) driver;
//getting the source file using getScreenshotAs() method and storing in a file
 File srcFile = ts.getScreenshotAs(OutputType.FILE);
/*Created a folder called "screenshot" in the project directory
Created another file by concatenating the date value which has "_" in it (Underscore is the accepted character while creating a file in the project )*/
File destFile = new File(".\\screenshot\\"+date2+"  actiTIMELoginPage.png");
/*copyFile() method is a static method present in FileUtils class of JAVA storing the screenshot in the destination location*/
FileUtils.copyFile(srcFile, destFile);
//closing the browser
 driver.close();
}
}
```

Question : Why capturing screenshot of the web pages is important in the project?

Answer :

- We capture screenshots in order to debug the failed test scripts.
- It actually helps the automation test engineer to find the exact root cause of the issue in the application at the earliest.

Following are the possible scenarios after the script is failed:

- Whenever an automation script is failed, we first manually execute the steps to check whether there is any issue in the application or the issue is with the script.
- If the script fails due to an issue in the script itself, we fix the script and re-run it till it is passed.
- If there is an issue in the application due to which the script is failed, then we log defect against the same issue. In this way, automation team gets credibility in the project.

Handling Browser navigation

Question : How to navigate within the browser?

Answer :

Using navigate() methods.

```
public class BrowserNavigationExample {

public static void main(String[] args) throws InterruptedException {
System.setProperty("webdriver.gecko.driver", ".\\driver\\geckodriver.exe");

 WebDriver driver = new FirefoxDriver();

//Enter the url

 driver.get("http://localhost:8080/login.do");

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

 Thread.sleep(3000);

driver.navigate().back();

Thread.sleep(3000); .

driver.navigate().forward();

Thread.sleep(3000);

driver.navigate().refresh();

driver.close();

}

}
```

What is an WebElement ?

1. Any element present on a web page is called as web element
2. Developers use HTLM code to develop web pages.
3. For testing purpose, we can also create web page using HTML code
4. In order to create a web page, we need to write HTML code in any text pad (eg : notepad and save the file with .html extension)

Create a sample web page using HTML as mentioned below.

```
<html>
<body>
UN : <input type="text" id = "username" value = "admin"> PWD: <input type="text" id= "pass" value = "manager">
<a href="http://localhost:8080/login.do"> Click ActiTIME Link</a>
</body
</html>
```

In the above HTML tree, every element should have one of the 3 option

1. Tagname (this is mandatory for all the elements)
2. Attributes (Optional)
3. Text (Optional)

Example of Tagname in the above HTML Tree structure:

- html,
- body,
- input,
- a

Example of Attributes in the above HTML Tree structure:

- type = "text"
- id = “username”
- value = "admin"

Format : attributeName = “attributeValue”

Example of Text in the above HTML Tree structure:

- Click ActiTIME link

What are Locators ?

- Locators are used to identify the web elements on the web page.
- We have 8 types of locators in Selenium using which findElement() methods identifies elements on the web page:

1. id
2. name
3. tagName
4. className
5. linkText
6. partialLinkText
7. xpath
8. cssSelector

- findElement() method returns the address of the web elements on the web page and the return type is WebElement.
- If the specified locators returns multiple elements, then findElement() method returns the address of the first matching element.
- If the specified locators returns No element, then findElement() method throws an exception called "NoSuchElementException".

Note :

In below Selenium code snippet,

WebDriver driver = new FirefoxDriver();

1. driver.findElement(By.id(""));
2. driver.findElement(By.name(""));
3. driver.findElement(By.tagName(""));
4. driver.findElement(By.class(""))
5. driver.findElement(By.linkText(""))
6. driver.findElement(By.partialLinkText(""))
7. driver.findElement(By.xpath(""))
8. driver.findElement(By.cssSelector(""))

cssSelector locator:

1. It is one of the locators in Selenium using which we identify web elements on the web page.
2. It stands for Cascading Style Sheet.
3. The standard syntax for cssSelector expression is

// tagName[attributeName = 'attributeValue']

OR

tagName is not mandatory. [attributeName = "attributeValue"]

Important notes on LinkText and PartialLinkText locator

- Out of all the locators, linkText and PartialLinkText are used to identify only the links present on the webpage.(elements whose tagname is "a" -- a stands for anchor)
- LinkText locator should be used when the text of the link is constant
- PartialLinkText locator should be used when certain part of the link text is getting changed everytime the page is loaded. i.e for partially dynamically changing text, we use partialLinkText locator
- To handle those elements whose text changes completely, we can't use partialLInkText locator. It should be handled by another locator called "xpath"
- If we use try to use these 2 locators on other type of elements (except Links), then we get "NoSuchElementException"

XPATH :

1. xpath is one of the locators in selenium using which we identify objects or elements on the web page and perform specific actions to carry out automation testing.

2. xpath is the path of an element in the html tree.

3. xpath are of 2 types.

Absolute xpath

Relative xpath

Absolute xpath :

1. It refers to the path of the element right from the root node to the destination element.

2. While writing the absolute xpath, we use single forward slash (/) to traverse through each immediate child element in the html tree.

4. Using absolute xpath in selenium code as shown below.
driver.findElement(By.xpath("html/body/a")).click();

5.In xpath, if there are multiple siblings with same tagname, then the index starts from 1

6. In case of multiple siblings with same tagname, if we don't use index, then it considers ALL the siblings.

7. We can join multiple xpath using pipeline operator (|)

Relative xpath:

1. In Absolute xpath, we write the path of the element right from the root node and hence, the expression for absolute xpath is lengthy.

2. In order to reduce the length of the expression, we go for Relative xpath.

3. In Relative xpath, we use double forward slash (//), which represents any descendant.

Interview questions :

1. what is the difference between '/' and '//' ?

Answer : "/" refers to the immediate child element in the html tree.

“//” refers to any element in the html tree. It also represent any descendant.

2. What are the types of xpath?

Ans: Absolute and Relative xpath.

3. Derive an xpath which matches all the links present on a web page ?

Ans : //a

4. Derive an xpath which matches all the image present on a web page?

Ans : //img

5. Derive an xpath which matches all the links and images present on a web page?

Ans : //a | //img

6. Derive an xpath which matches all the 2nd links present on a web page?

//a[2]

7. Derive an xpath which matches all the links present inside a table present on a web page?

//table//a

8. Difference between “//a”and “//table//a “ ?

Ans : //a → refers to all the links present on the webpage.

//table//a → refers to all the links present within all the tables present on the webpage.

xpath by Attribute:

1. xpath expression can be written using attribute of the web element.

2. Based on the situation, we would use either single attribute or multiple attributes to find an element on a web page.

3. Using single attribute in xpath expression, if it returns one matching element, then we would use single attribute only.

4. In case, by using single attribute in xpath expression, if it returns multiple matching elements on the web page, then we would go for using multiple attributes in the xpath expression till we get one matching element.

xpath expression using Attribute:

1.using single attribute:

Syntax: //tagname [@attributeName = ‘attributeValue’]

//tagname [NOT(@attributeName = ‘attributeValue’)]

Sample application : actiTIME application url : https://demo.actitime.com/login.do

Write xpath for below few elements on above actiTIME login page :

Usage in selenium code :

driver.findElement(By.xpath(“paste any xpath here from above table”)) 2. Using multiple attribute :

xpath Syntax :

- //tagName[@AN1=’AV1’][@AN2=’AV2’]
- //tagName[@AN1=’AV1’] | //tagName[@AN2=’AV2’]

Element : View licence link

html code after inspecting the element using F12 key:

View License

xpath expression using “href” and “onclick” attributes :

//a[@href='javascript:void(0)' and @onclick='openLicensePopup();']

Usage in selenium code : driver.findElement(By.xpath(“//a[@href='javascript:void(0)']

[@onclick='openLicensePopup();']”))

Assignment :

Write xpath expression for below 7 elements present on actiTIME login page Elements :

1. UserName
2. Password
3. Login Button
4. Check box
5. Actitime Image
6. View Licence link
7. actiTIME Inc link

Use the below format (Sample example for actiTIME Inc Link): html code for <actiTIME Inc.> :

actiTIME Inc. xpath syntax

//tagname[@AN1 = 'AV1']

1. using href attribute:

//a[@href = 'http://www.actitime.com']

2. using target attribute

//a[@target = '_blank']

Note: Use all the attributes of an element to write xpath expression

xpath expression using text() function :

1. In the html code of an element, if attribute is duplicate or attribute itself is not present, then use text() function to identify the element.

2. In order to use text() function, the element should have text in the element's html code.

Syntax :

//tagName[text()='text value of the element']

Or

//tagName[.='text value of the element']

Note : Instead of text(), we can use dot (.) , the problem here with using dot (.) is sometimes, it returns the hidden element also present on the webpage. which might confuse the user. So the best practice is to use text() instead of using dot.

xpath expression using text() function for below elements present on actiTIME login page.

xpath expression using contains() function :

1. In the html code of an element, if the attribute value or the text is changing partially, then use contains() function to identify the element.

2. In order to use contains() function, the element should have either attribute value or text value.

Syntax :

- //tagName[contains(@attributeName,'attributeValue')]
- //tagName[contains(text(),'text value of the element')]

xpath expression using contains() function for below elements present on actiTIME login page.

3. We use contains() function when the text value is very lengthy or the attribute value is very lengthy.

eg: xpath to identify error message present on actitime login page (Click on login button without entering username and password to get the error message)

//span[contains(text(),'invalid')]

Program to illustrate xpath by attributes, text() function, contains() function and their usages with attributes and text values.

public class XpathUsingAttribute_Actitime extends BaseClass

{

```
public static void main(String[] args) throws InterruptedException {
System.setProperty("webdriver.gecko.driver", ".\\driver\\geckodriver.exe"); WebDriver driver = new FirefoxDriver();

//Enter the url of actiTIME application

driver.get("http://localhost:8080/login.do");

//xpath using multiple attributes

String xp = "//input[@class='textField'][ @id = 'username']";

Thread.sleep(2000);

//Enter admin into username text box driver.findElement(By.xpath(xp)).sendKeys("admin");
Thread.sleep(2000);

//find password element using xpath by attribute and enter manager in to password textbox.

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

Thread.sleep(2000);

//find an image on the web page whose attributes (src)contains a value called timer

WebElement clock = driver.findElement(By.xpath("//img[contains(@src,'timer')]"));

//store the width value of the clock image into a variable called widthValue

String widthValue = clock.getAttribute("width");

//Print the width of the clock image

System.out.println("the width is :"+widthValue);

//Print the height of the clock image

System.out.println("the height of the clock element is : "+ clock.getAttribute("height"));

//xpath using text() function

driver.findElement(By.xpath("//div[text()='Login ']")).click(); Thread.sleep(2000);

//xpath using contains() function and text() function
driver.findElement(By.xpath("//a[@id='loginButton']//div[contains(text(),'Login')]")).click();
Thread.sleep(2000);

driver.close();

}

}
```

xpath expression using starts-with() function :

1. We use starts-with() function to identify those elements whose text value starts with some specified value.

xpath using contains() function:

//[contains(text(),'actiTIME')] - this xpath will return 6 matching element on login page of actiTIME application.

xpath using starts-with() function

//[starts-with(text(),'actiTIME')] - this xpath will return only 3 matching element on login page of actiTIME application as the text value of these 3 elements starts with “actiTIME” text

Handling completely dynamic links :

1. When the text value of the elements is completely changing, then we can’t use functions like “contains()”, “starts-with()” etc to handle those elements.

2. In such cases, we identify the dynamically changing element using the nearby unique element. We call this concept as independent dependent xpath.

JavascriptExecutor

It is one of the interface in selenium which has below 2 methods.

1. executeScript()

2. executeAsyncScript()

We use JavascriptExecutor when we fail to perform some actions using selenium.

Write a script to enter a text in a textbox which is in disabled mode ?

Using sendKeys() of WebElement interface, if we try to enter, we get InvalidElementStateException

Using executeScript() of JavascriptExecutor interface, we can enter text in a disabled textbox.

In Selenium, we don’t have any method to scroll up or down on the webpage, in such case, we can use

JavascriptExecutor.

Steps to run javascript manually on browser webpage

1. Open the required page in the browser and press F12 from keyboard.

2. Navigate to Console tab, type the javascript statement and press Enter key

Write a script to scroll up and down on Selenium official website

Using executeScript() of JavascriptExecutor interface Selenium code below :

public class ScrollUpandDown {

```
public static void main(String[] args) throws InterruptedException {
System.setProperty("webdriver.gecko.driver", ".\\driver\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
driver.get("http://seleniumhq.org/download");
//typecasting driver object to JavascriptExecutor interface type
 JavascriptExecutor js = (JavascriptExecutor) driver;
for (int i = 1; i < 10; i++) {
//scroll down on the webpage
js.executeScript("window.scrollBy(0, 1000)");
Thread.sleep(3000);
}
for (int i = 1; i < 10; i++) {
//scroll up on the webpage
 js.executeScript("window.scrollBy(0, -1000)");
 Thread.sleep(3000);
}}}
```

Write a script to scroll down to a specific element (Applitool webelement) on Selenium official website

Using executeScript() of JavascriptExecutor interface Selenium code below :

```
public class ScrollUpandDowntospecificElementonWebpage {
public static void main(String[] args) throws InterruptedException {
System.setProperty("webdriver.gecko.driver", ".\\driver\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
 driver.get("http://seleniumhq.org/download");
//click on the close icon of the yellow color background pop up
driver.findElement(By.id("close")).click();
// find the Applitools element on the webpage
```

```
WebElement ele = driver.findElement(By.xpath("//img[contains(@src,'applitools')]"));


// get the X-coordinate and store in a variable int x = ele.getLocation().getX();

// get the Y-coordinate and store in a variable int y = ele.getLocation().getY();

JavascriptExecutor js = (JavascriptExecutor) driver;

//Scroll to Applitools element's x and y coordinate js.executeScript("window.scrollBy("+x+",
"+y+")");

 Thread.sleep(3000);

}}
```

Assignment :

Write a script to scroll down to the bottom of the page ?

Using executeScript() of JavascriptExecutor interface Selenium code below :

```
public class NavigatetoBottomofthePage {


public static void main(String[] args) throws InterruptedException {
System.setProperty("webdriver.gecko.driver", ".\\driver\\geckodriver.exe");

 WebDriver driver = new FirefoxDriver();
driver.get("http://www.seleniumhq.org/download/");
driver.findElement(By.id("close")).click();

//select an element which is present at the bottom of the page

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


int x = element.getLocation().getX();

int y = element.getLocation().getY();

System.out.println("X coordinate is :"+x + " and Y coordinate is :"+ y);

 JavascriptExecutor js = (JavascriptExecutor) driver;
js.executeScript("window.scrollBy("+x+","+y+")"); Thread.sleep(3000);

element.click();


}}
```

What is Frame

- Webpage present inside another webpage is called embedded webpage.
- In order to create frame or embedded webpage, developer uses a tag called iframe.
- In order to perform any operation on any element present inside a frame, we first have to switch the control to frame.
- We switch to frame using the below statement
- driver.switchTo().frame(arg);
- frame() is an overloaded method which accepts the following arguments.
    - frame(index)
    - frame(id)
    - frame(name)
    - frame(WebElement)
- If the specified frame is not present, we get an exception called "NoSuchFrameException"

- In order to exit from the frame, we use the following statements.
  driver.switchTo().defaultContent(); → it will take you to the main page
  driver.switchTo().parentFrame(); → it will take you to the immediate parent frame
- Easiest way to verify that an element is present within a frame is to right click on the element and verify that
- this frame option is displayed in the context menu.

==========

Write a script to enter a text into an element which is present inside a frame ?

Selenium code:

```
public class Frame_Demo{

public static void main(String[] args) {

System.setProperty("webdriver.gecko.driver", ".\\driver\\geckodriver.exe");

WebDriver driver = new FirefoxDriver();

driver.get("file:///D:/Ajit/Selenium/SeleniumBtm_7thSep17/webpages/Frame_Page2.html")
;

//using index of the frame - [ int value] [ index of frames starts with zero]

driver.switchTo().frame(0);

driver.findElement(By.id("t1")).sendKeys("a"); driver.switchTo().defaultContent();
```

```
driver.findElement(By.id("t2")).sendKeys("a");
//using id attribute of the frame -string
 driver.switchTo().frame("f1");
driver.findElement(By.id("t1")).sendKeys("b");
driver.switchTo().defaultContent();
 driver.findElement(By.id("t2")).sendKeys("b");
//using name attribute of the frame -string
 driver.switchTo().frame("n1");
driver.findElement(By.id("t1")).sendKeys("c");
driver.switchTo().defaultContent();
driver.findElement(By.id("t2")).sendKeys("c");
//using address of the frame -webelement
WebElement f = driver.findElement(By.className("c1"));
driver.switchTo().frame(f);
 driver.findElement(By.id("t1")).sendKeys("d");
driver.switchTo().defaultContent();
driver.findElement(By.id("t2")).sendKeys("d");
 driver.close();
}}
```

Actions class:

How do you handle Context Menu in Selenium?

OR

Write a script to right click on “ActiTIME Inc.” link on actitime login page and then open it in new window ? Using contextClick() method of Actions class

Selenium code:

```
public class ContextClickusingActionsClass {
//ContextClick does not work on firefox browser - pls do it on chromebrowser
public static void main(String[] args) throws AWTException, InterruptedException {
System.setProperty("webdriver.chrome.driver", ".\\driver\\chromedriver.exe")
//open the browser
WebDriver driver = new ChromeDriver();
//enter the url
driver.get("http://localhost:8080/login.do");
//find the ActiTIME Inc. link
WebElement link = driver.findElement(By.linkText("actiTIME Inc."));
//right click (context click) on actitime link
Actions actions = new Actions(driver);
actions.contextClick(link).perform();
Thread.sleep(3000);
//press 'w' from the keyboard for opening in a new window
Robot r = new Robot();
r.keyPress(KeyEvent.VK_W);
 r.keyRelease(KeyEvent.VK_W);
//quit() method closes all the browsers opened by Selenium
 driver.quit();
}}
```

Imp Note :

Whenever we call any method of Actions class, we have to explicitly call perform() method of Actions class. Otherwise, it will not perform any action on the browser.

Assignment :

Automate the following scenario using contextClick() method of Actions Class.

Scenario Steps :

1. Login in to gmail.
2. Based on the subject of a mail, Right click on the mail
3. Select Archive option

Selenium Code:

```
public class gmail_contextClickDemo_mailArchive {

 public static void main(String[] args) throws InterruptedException {
System.setProperty("webdriver.chrome.driver", ".\\driver\\chromedriver.exe");

 WebDriver driver = new ChromeDriver();

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

//enter email id

driver.findElement(By.xpath("//input[@type='email']")).sendKeys("enter your username");

//click on Next button driver.findElement(By.xpath("//span[.='Next']")).click();
Thread.sleep(3000);

//enter password id
driver.findElement(By.xpath("//input[@type='password']")).sendKeys("enter ");

//click on Next button driver.findElement(By.xpath("//span[.='Next']")).click();
Thread.sleep(10000);

//Write xpath expression for the mail item based on a subject

String xp = "(//b[contains(.,'Following Openings (for Bangalore)')])[2]";

//get the address of the mail item which you want to archive

 WebElement mail = driver.findElement(By.xpath(xp));

//print the subject of the mail

 System.out.println(mail.getText());

//Creating an object of Actions class

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

```
//using Actions class object and contextClick() method, right click on the mail item
actions.contextClick(mail).perform();
Thread.sleep(6000);
//click on Archive to archive the mail
driver.findElement(By.xpath("(//div[@class='J-N-JX aDE aDD'])[1]")).click();
}}
```

Program :

How do you mouse hover on any element on a web page ?

Using moveToElement() of Actions class

Automate the following scenario using moveToElement() method of Actions Class.

Scenario Steps :

Login in to http://www.actimind.com

1. Mouse hover on “About Company” menu 3. Click on Sub Menu - “Basic Facts”

Selenium Code:

```
public class DropdownMenu {
public static void main(String[] args) {
System.setProperty("webdriver.chrome.driver", ".\\driver\\chromedriver.exe");
//open the browser
WebDriver driver = new ChromeDriver();
 driver.get("http://www.actimind.com/");
//find the menu "About Company"
 String xp = "//span[.='About Company']";
WebElement menu = driver.findElement(By.xpath(xp));
//mouse hover on "About Company" menu
Actions actions = new Actions(driver);
actions.moveToElement(menu).perform();
//click on submenu "Basic Facts"
WebElement submenu = driver.findElement(By.linkText("Basic Facts"));
 submenu.click();}}
```

Handling pop-up

In selenium, pop up are categorized into following types.

1. Hidden Division popup
2. File Upload popup
3. File download popup
4. Child browser popup
5. Window popup

1. Alert Pop up : (only this pop up and child browser pop up is usefull for us)

Characteristics features :

- We can't inspect this pop up.
- We can't move this kind pop up.
- This pop up will have white color background with black color font.
- This pop up will have only one "OK" button

How to handle Alert pop up

In order to handle the alert pop up, we first have to switch to alert window using the below statement.

driver.switchTo().alert();

After transferring the control to alert window, we can use the following methods of "Alert" interface.

getText() → to get the text present on the alert window. accept() / dismiss() → to click on OK button on the alert window.

In order to handle the alert pop up, we first have to switch to alert window using the below statement.

driver.switchTo().alert();

After transferring the control to alert window, we can use the following methods of "Alert" interface.

getText() → to get the text present on the alert

window. accept() / dismiss() → to click on OK button on the alert window.

Child Browser Pop up:

Characteristic features :

- We can move this pop up.
- We can also inspect it.
- this pop up is very colorful and will have both minimise and maximise buttons.

How to handle Child browser pop up ?

- We handle child browser popup by using getWindowHandle() and getWindowHandles() methods of WebDriver interface.
- In Selenium, every browser will have an unique window handle id.
- In firefox browser, window handle is an integer value, whereas in Chrome browser, it is an unique alpha numeric string.

Difference between getWindowHandle() and getWindowHandles() ?

- getWindowHandle() returns the window handle id of the current browser window.
- getWindowHandles() returns the window handle id of all the browser windows.

.

Program to print the window handle of a browser window ?

Selenium Code :

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
 public class Print_windowHandle
 {
public static void main(String[] args) throws InterruptedException
 {
 System.setProperty("webdriver.gecko.driver", ".\\driver\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
driver.get("http://localhost:8080/login.do");
//get the window handle id of the browser
String windowHandle = driver.getWindowHandle();
System.out.println(windowHandle);
```

```
}
}
```

Program to print the window handle id of browser?

Selenium Code:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
public class Print_windowHandle {
public static void main(String[] args) throws InterruptedException {
System.setProperty("webdriver.gecko.driver", ".\\driver\\geckodriver.exe");
WebDriver driver = new FirefoxDriver();
 driver.get("http://localhost:8080/login.do");
//get the window handle id of the browser
String windowHandle = driver.getWindowHandle();
 System.out.println(windowHandle);
}}
```

Program :

Scenario :

Write a script to automate the following scenarios:

1. Count the number of browser windows opened by selenium
2. Print the window handle of all the browser windows
3. Print the title of all the browser windows ?
4. Close all the browser windows.

Selenium Code :

```
public class ChildBrowserPopUp extends BaseClass{
 public static void main(String[] args) {

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

//using getWindowHandles(),
 get a set of window handle IDs
 Set<String> allWindowHandles = driver.getWindowHandles();
//using size(),
 get the count of total number of browser windows
 int count = allWindowHandles.size();
System.out.println("Number of browser windows opened on the system is : "+ count);
 for (String windowHandle : allWindowHandles) {
//switch to each browser window
 driver.switchTo().window(windowHandle);
 String title = driver.getTitle();
//print the window handle id of each browser window
System.out.println("Window handle id of page -->"+ title +" --> is : "+windowHandle);
//close all the browsers one by one
 driver.close();
}
```

```
/*Instead of using driver.close(), we can use driver.quit() to close all the browsers at once*/
//driver.quit();
}}
```

Program :

Write a script to close only the main browser window and not the child browser windows.
Selenium Code :

```
public class CloseMainBrowserOnly extends BaseClass{
public static void main(String[] args) {
driver.get("https://www.naukri.com/");
//get the window handle id of the parent browser window

String parentWindowhandleID = driver.getWindowHandle();
 Set<String> allWindowHandles = driver.getWindowHandles();
 int count = allWindowHandles.size();
System.out.println("Number of browser windows opened on the system is : "+ count);
 for (String windowHandle : allWindowHandles) {
//switch to each browser window
driver.switchTo().window(windowHandle);
/* compare the window id with the Parent browser window id, if both are equal, then only close the main browser window.*/

if (windowHandle.equals(parentWindowhandleID)) {
driver.close();
System.out.println("Main Browser window with title -->"+ title +" --> is closed");

}}}}
```

Program :

Write a script to close all the child browser windows except the main browser.

Selenium Code :

```
public class CloseALLChildbrowsersONLY extends BaseClass{ public static void main(String[] args) {

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

//get the window handle id of the parent browser window

String parentWindowhandleID = driver.getWindowHandle();

Set<String> allWindowHandles = driver.getWindowHandles();

int count = allWindowHandles.size();

System.out.println("Number of browser windows opened on the system is : "+ count);

for (String windowHandle : allWindowHandles) {

//switch to each browser window

driver.switchTo().window(windowHandle);

String title = driver.getTitle();

/* compare the window id of all the browsers with the Parent browser window id, if it is not equal, then only close the browser windows.*/

if (!windowHandle.equals(parentWindowhandleID)) {

 driver.close();

System.out.println("Child Browser window with title -->"+ title +" --> is closed");

}}}}
```

what is findElements ()?

● findElements() method is present in SearchContext interface, the super most interface in Selenium.

● findElements () identifies the elements on the webpage based on the locators used.

● It returns a list of webElements if it find the matching element.

● If it does not find any matching web element on the web page, it returns an empty list.

Automate the following scenario

● Login in to actitime

● click on Tasks

● Count the total number of checkbox present on the page

● Select all the checkbox

● Deselect all the checkboxes in reverse orde

● Select first and last checkbox

In the below webtable,

find the following scenarios :

● print the total number of ROWS present

● print the total number of COLUMNS present

● print the total number of CELLS present

● print ONLY the NUMERIC values present

● Count the TOTAL number of NUMERIC values present

● print the SUM of all the numeric values in the table Selenium Code:

```
public class WebTable_Example extends BaseClass{
 public static void main(String[] args) {
driver.get("D:\Ajit\Selenium\SeleniumBtm_7thSep17\webpages\WebTable.html");
//Count Total number of rows present in the table List<WebElement> allRows =
driver.findElements(By.xpath("//tr"));
int totalRows = allRows.size();
System.out.println("total number of rows present in the table is :"+ totalRows);
```

```
//count total number of columns
List<WebElement> allColumns = driver.findElements(By.xpath("//th"));
 int totalColumns = allColumns.size();
System.out.println("Total number of columns in the table is :" + totalColumns);
//Count number of cells present in the table
List<WebElement> allCells = driver.findElements(By.xpath("//th|//td"));
int totalCells = allCells.size();
System.out.println("Total number of cells present in the table is :" + totalCells);

//Print ONLY the numbers int countNumberValue = 0; int sum=0;
for (WebElement cell : allCells) {
String cellValue = cell.getText();
try{
int number = Integer.parseInt(cellValue);
System.out.print(“ ”+number);
countNumberValue++;
sum = sum+number;
}
catch (Exception e) {
}
}
System.out.println("Total count of numeric values is :"+countNumberValue);
System.out.println("Total sum of all the numeric values is :"+sum);
//close the browser
driver.close();
}
}
```

How to handle Auto Suggestion list box?

Answer :

Using findElements() method Program:

Automate the following scenario:

- Navigate to google page
- Enter Selenium in google search text box
- Print the list of auto suggestion values
- Click on a specified link (Selenium Interview Questions) displayed in the dropdown

Selenium Code :

```
public class Autosuggestion Ex_GoogleSearch extends BaseClass{


public static void main (String [] args) throws InterruptedException {
driver.get("http://www.google.com");

//Enter Selenium in google search text box driver. findElement(By.id("lst-
ib")).sendKeys("selenium");

 Thread.sleep(2000);

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

int count = allOptions.size();

System.out.println(“Number of values present in the dropdown is : ” + count);

String expectedValue="selenium interview questions";

//Print all the auto suggestion values for

 (WebElement option : allOptions) {

String text = option.getText(); System.out.println(" "+text);

//Click on Java Interview Questions

if (text.equalsIgnoreCase(expectedValue)) {

option.click();

 break;

}

}
```

How to select List Box ?

- In Selenium, we handle listbox using Select class.
- Select class is present in org.openqa.selenium.support.ui package.
- Select class has a parameterized constructor which accepts an argument of WebElement object (List box element)
- Following are the available methods of Select class

➔ selectByIndex()

➔ selectByValue()

➔ selectByVisibleText()

➔ deSelectByIndex()

➔ deSelectByValue()

➔ deSelectByVisibleText()

➔ isMultiple()

➔ getOptions()

➔ getAllSelectedOptions()

➔ getFirstSelectedOption()

➔ deSelectAll()

- We can use the following deSelect() methods only on multi select listbox. If we try to use it on single select list box, then it throws UnsupportedOperationException

➔ deSelectByIndex()

➔ deSelectByValue()

➔ deSelectByVisibleText()

➔ deSelectAll()

Page Object Model - Framework Design pattern

Definition:

Page object model is a page factory design pattern. We implement this model in our automation framework because of the following advantages listed below.

Advantages of POM:

- Easy to Maintain
- Easy readability of scripts
- Reduce or eliminate duplicacy
- Re-usability of code
- Reliability
- It is the object repository
- we achieve encapsulation using this pom framework.

Pom class for Actitime Login page.

```
 package pages;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;
 public class LoginPage {
//Declaration
 @FindBy(id="username")
 private WebElement unTB;
 @FindBy(name="pwd")
 private WebElement pwTB;
@FindBy(xpath="//div[.='Login ']")
 private WebElement loginBtn;
//Initialisation
```

```
public LoginPage(WebDriver driver){
 PageFactory.initElements(driver, this);
}
//Utilisation
public void setUsername(String un){
unTB.sendKeys(un);
}

public void setPassword(String pw){
 pwTB.sendKeys(pw);
}
public void clickLogin(){
loginBtn.click();
}
```

TESTNG Framework

Testng is a framework that we implement in our Selenium automation framework for following advantages.

- Data Driven testing can be achieved (Data parameterisation is possible using testng)
- we can execute the test scripts in batch (i.e multiple test scripts at one go)
- we can also execute selected test scripts based on priority.
- we can execute the test case group wise or module wise
- generation of Automatic HTML reports
- We can integrate testNG with Maven as well
- Due to certain annotations that testNG provides, it has become so powerful

Write a program to check the sequence in which the annotations of Testng class gets executed.

Selenium code below :

```
package testngpackage;

import org.testng.annotations.BeforeMethod;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.Reporter;
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 BaseTestNg {

@BeforeMethod
```

```
public void beforeMethod() {
Reporter.log("beforeMethod", true);
}
@AfterMethod
public void afterMethod() {
 Reporter.log("afterMethod", true);
}
@BeforeClass
public void beforeClass() {
Reporter.log("beforeClass", true);
}
@AfterClass
public void afterClass() {
Reporter.log("afterClass", true);
}
@BeforeTest
public void beforeTest() {
Reporter.log("beforeTest", true);
}
@AfterTest
public void afterTest() {
Reporter.log("afterTest", true);
}
@BeforeSuite
public void beforeSuite() {
Reporter.log("beforeSuite", true);
}
@AfterSuite
```

```
public void afterSuite() {
Reporter.log("afterSuite", true);
}}
```

Demonstration on few parameters of @Test annotation as shown below.

```
package demotest;
import org.testng.Reporter;
import org.testng.annotations.Test;
 public class DemoA {
@Test(priority=1, groups={"user", "smoke"})
public void CreateUser(){
Reporter.log("CreateUser", true);
}
@Test(priority=2, invocationCount=1, enabled=true, groups={"user"})
 public void editUser(){
Reporter.log("editUser", true);
}
@Test(priority=3, groups={"user"})
public void deleteUser(){
Reporter.log("deleteUser", true);
}
@Test(priority=1, groups={"product", "smoke"})
public void createProduct(){
Reporter.log("createProduct", true);
}
@Test(priority=2, invocationCount=1, enabled=true, groups={"product"})
public void editProduct(){
Reporter.log("editProduct", true);


}
```

```
@Test(priority=3, groups={"product"})
public void deleteProduct(){
Reporter.log("deleteProduct", true)
}
}
```

Convert the above testng class to create testng.xml suite file as shown below How to convert a testng class to testng.xml suite file ?

Right click on the testng class → TestNg ---> Convert to testng Below xml file is created with the following data.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite">
<test name="Test"
<groups
<run>
<include name="user"></include>
<exclude name="user"></exclude>
</run>
</groups>
<classes>
<class name="testngpackage.DemoA"/>
</classes>
</test> <!-- Test -->
</suite> <!-- Suite →
```

Launch Multiple browser using Testng.xml suite file parameters

We can create multiple test blocks to work on multiple browsers in automation project. Example is shown below.

```
package  testngpackage;

import java.io.FileInputStream;

 import java.io.FileNotFoundException;

import java.io.IOException;

import java.util.Properties;

 import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.testng.Reporter;

import org.testng.annotations.Parameters;

 import org.testng.annotations.Test;

public class LaunchFirefoxAndChromeTogether {

 static{

System.setProperty("webdriver.gecko.driver", "./driver/geckodriver.exe");
System.setProperty("webdriver.chrome.driver", "./driver/chromedriver.exe");

}

WebDriver driver;

@Test

@Parameters({"browser"})

public void loginFFandCHROME(String browser) throws InterruptedException, IOException{

//Reporter.log(browser,  true);

if (browser.equals("firefox")) {

driver = new FirefoxDriver();

}

else {
```

```
driver = new ChromeDriver();
}
FileInputStream configPath = new FileInputStream(".\\config.properties");
 Properties prop = new Properties();
prop.load(configPath);
String url = prop.getProperty("URL");
driver.get(url);
WebElement un = driver.findElement(By.id("username"));
 for (int i = 0; i < 10; i++) {
un.sendKeys("admin" + i); Thread.sleep(2000);
 un.clear();
}
driver.close();
}
}
```

Use the below testng.xml suite file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Suite" parallel="tests">
<test name="TestFirefox">
<parameter name="browser" value="firefox"></parameter>
<classes>
<class name="testngpackage.LaunchFirefoxAndChromeTogether"/>
</classes>
</test> <!-- Test -->
<test name="TestChrome">
<parameter name="browser" value="chrome"></parameter>
<classes>
<class name="testngpackage.LaunchFirefoxAndChromeTogether"/>
```

</classes>

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

</suite> <!-- Suite →

Parameterisation using

@DataProviders

1. DataProvider is an annotation in testng using which we can create data bank.

2. THis databank can used across any testng class , thus achieiving data parameterisation.

Executing the same script with multiple sets of data is called data parameterisation. From Testng Class by creating our own data bank using DataProvider annotation

From Testng class

1. We use DataProvider annotation to create the data bank,

2. We utilise this data from any testng methods by using "dataProvider" parameter as an argument to "Test" annotation.

code :

```
package scripts;

import org.testng.Reporter;

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class DataProviderExample{

 @DataProvider

public Object[][] dataBank(){

Object[][] data = new Object[2][2]; data[0][0] = "admin1";

data[0][1] = "manager1";

data[1][0] = "admin2";

data[1][1] = "manager2";

 return data;

}

@Test(dataProvider="dataBank")

public void useDataBank(String un, String pwd){ Reporter.log(un + " --> "+ pwd, true);}}
```

SELENIUM GRID

To run the same scripts on multiple browsers and multiple systems parallelly, we use Selenium Grid. Here, there will be 2 types of system.

1. HUB
2. NODE.

Node is the remote system on which you run the automation scripts.

In node system, JDK and Browser should be installed and we should also know the ip address

.==========
==========

SELENIUM FRAMEWORK

Framework is set of rules and guidelines which should be followed by all the automation engineers in the team while automating an application.

There are 3 major statuses as mentioned below.

1. Automation Framework Design
2. Automation Framework Implementation
3. Automation Framework Execution

Questions:

Which framework have you developed/implemented in your project?

We have implemented Hybrid driven framework, which is a combination of POM driven framework, testng framework, data driven framework, method driven framework and modular driven framework.

POM Driven Framework:

1. In POM driven framework, we have created POM pages for all the page of our application under test.

2. In our application, we have 20 pages in total and for all these 20 pages, we have created 20 pom classes. All these pom classes, we have created in a single package called scripts.

3. In each POM class, we declared the elements present on that particular page using @FindBy annotation. As an argument to FindBy annotation, we can use any one of the locator using which, element can be uniquely identified on the web page.

4. Once elements are declared, we initialise all the elements declared above using PageFactory.initElements() method, which accepts 2 arguments - both are of type object. First argument is WebDriver driver object and second argument is this (which represent the current class object), we write this statement inside the constructor, so that when the object of this pom class is created from another class, it will invoke the constructor and initialise all the elements which are declared in the pom class.

5. Once elements are declared and initialized, we utilise all the elements by creating respective setter methods. This is what we have done on a high level inside a pom class.

TESTNG FRAMEWORK :

1. Based on the number of test cases, we will create that many number of Testng class. In our project, we had close to 678 regression test cases and we have developed 678 testng class with one test method in each class.

2. In test method, we create object of respective POM class and using this reference variable, we keep calling the relevant method of pom class based on the manual test steps. This is how, we have automated our scripts using testng framework.

Data Driven Framework :

1. Executing the same scripts with multiple set of data is called data parameterisation. We used Excel file to get data from external source and utilised it in the scripts.

2. Using apache poi related jar, we implemented this data driven technique to achieve data parameterisation in our framework. Hence, our framework is also a data driven framework.

Modular Driven Framework:

1. Module wise execution of test scripts is known as modular driven framework.

2. In our project, when ever we develop a test method while automating a test case, we tag it to some group based on the module name.

3. Now, if any test script fails during normal execution cycle, we log defects and once developer fix the issue, we ensure that all the related test scripts of this particular module are executed and passed. This process of execution of module wise test scripts is known as modular driven framework.

Method Driven Framework :

1. We created few generic methods to access data from external sources like Excel file, config file, etc.

2. And furthermore, based on the manual regression test case steps, we call the relevant method of pom class from the testng class, In this way , our framework is a kind of method driven framework as well.

In our way, the framework that we have implemented is a **HYBRID framework**.

List of Exceptions

1. IllegalStateException [Java - Unchecked] (driver exe path not set)
2. InterruptedException [Java - Checked] (Thread.sleep)
3. IOException[Java-Checked][File handling scenario]
4. AWTException [Abstract Window Toolkit] [java - checked] [While handling Robot object]

5. NoSuchElementException[Selenium - unchecked][unable to locate the element]
6. JavascriptException[Selenium-Unchecked][on clicking on a button using submit() and the button don't have an attribute called type='submit']
7. NoSuchFrameException[Unchecked - selenium] [when specified frame is not present on the webpage]
8. NoSuchWindowException[Unchecked - selenium] no such window: target window already closed
9. NoAlertPresentException [Selenium - unchecked] [When no alert is present]
10. NoSuchSessionException: Session ID is null [Unchecked - Selenium] when driver.quit is called and then you are trying to access any browser

//URL for the automation framework
https://github.com/ajitbiswas/AutomationProjectWeekendBatch.git

Assert and Verify

The differences between Assert and Verify are listed below −

Assert	Verify
Verifies if the specified condition is true and false. If the result is true, the next test step will be executed. In case of false condition, the execution would terminate.	Verifies if the specified condition is true and false. If the result is true, the next test step will be executed. In case of false condition, the execution would still continue.
In case of false condition, the next text case of the suite will be executed.	In case of false condition, the next test step of the same text case will continue.
There are two types of assets namely hard and soft asserts.	There are no categories for verification.

Thus we see there two types of Asserts.

The difference between Soft Assert vs Hard Assert are listed below −

	Hard Assert	Soft Assert
	Throws an exception immediately after the assert fails and carries out with the next test case of the suite.	Does not throw an exception immediately when the assertion fails, collects them and carries out with the next validation.
	This throws an AssertException instantly so handled with a catch block. After suite execution is completed the test is made as PASS.	

Selenium Waits

You might have come across wait commands while writing your first Selenium program. In this article,

you would be learning about what exactly Selenium Waits is.

You would be covering various types and other necessary factors one needs to understand to get started with Selenium Waits

What is Selenium Waits?

Waits in Selenium is one of the important pieces of code that executes a test case. It runs on certain commands called scripts that make a page load through it. Selenium Waits makes the pages less vigorous and reliable. It provides various types of wait options adequate and suitable under favorable conditions.

This ensures you don't mess up and get ended into failed scripts while performing automation testing with it.

Implicit Waits

The main function of implicit Wait is to tell the web driver to wait for some time before throwing a "No Such Element Exception". Its default setting is knocked at zero.

Once the time is set, the driver automatically will wait for the amount of time defined by you before throwing the above-given exception.

Explicit Waits

Explicit Waits also known as Dynamic Waits because it is highly specific conditioned. It is implemented by WebDriverWait class.

To understand why you need Explicit Wait in Selenium, you must go through the basic knowledge of the wait statements in a program.

In simple terms, you must know some conditions. Such conditions have been created to give you a gist of the Explicit Waits and why they are important.

Condition 1:

Suppose you have a web page consisting of a login form that takes input and loads the Home or Main page content.

This page is dynamic because of the time constraints and network frequency, sometimes taking 10 seconds or maybe 15 seconds to load completely.

Explicit Wait comes in handy in such cases and allows you to wait until the page is not present to display.

Condition 2:

Consider that you are working on an application that is travel themed and users fill the web form and submit it using submit button.

Now, you might need to wait until and unless the specific data is not displayed. In such a case,

Explicit Wait becomes helpful by waiting until a specific period for the set of elements that are not displayed yet.

Difference between Implicit and Explicit

1. Implicit Wait applies to all the elements in the script, while Explicit Wait is applicable

only for those values which are to be defined by the user.

2. Implicit Wait needs specifying "ExpectedConditions" on the located element,

while Explicit Wait doesn't need to be specified with this condition.

3. Implicit Wait needs time frame specification in terms of methods like element visibility, clickable element, and the elements that are to be selected. In contrast,

Explicit Wait is dynamic and needs no such specifications.

Fluent Wait

Fluent Wait is quite similar to explicit Wait. It is similar in terms of management and functioning.

In Fluent Wait, you can perform wait for action for an element only when you are unaware of the time it might take to be clickable or visible.

JENKINS

It is a continuous integration tool widely used by software project. Advantages of Jenkins :

1. We can schedule the time for automation framework suite execution.

2. Automatic kick off of automation suite execution as soon as the build gets deployed from DEV environment to QA environment.

How to download Jenkins URL :

https://jenkins.io/download/ click on the link highlighted below.


SHREE