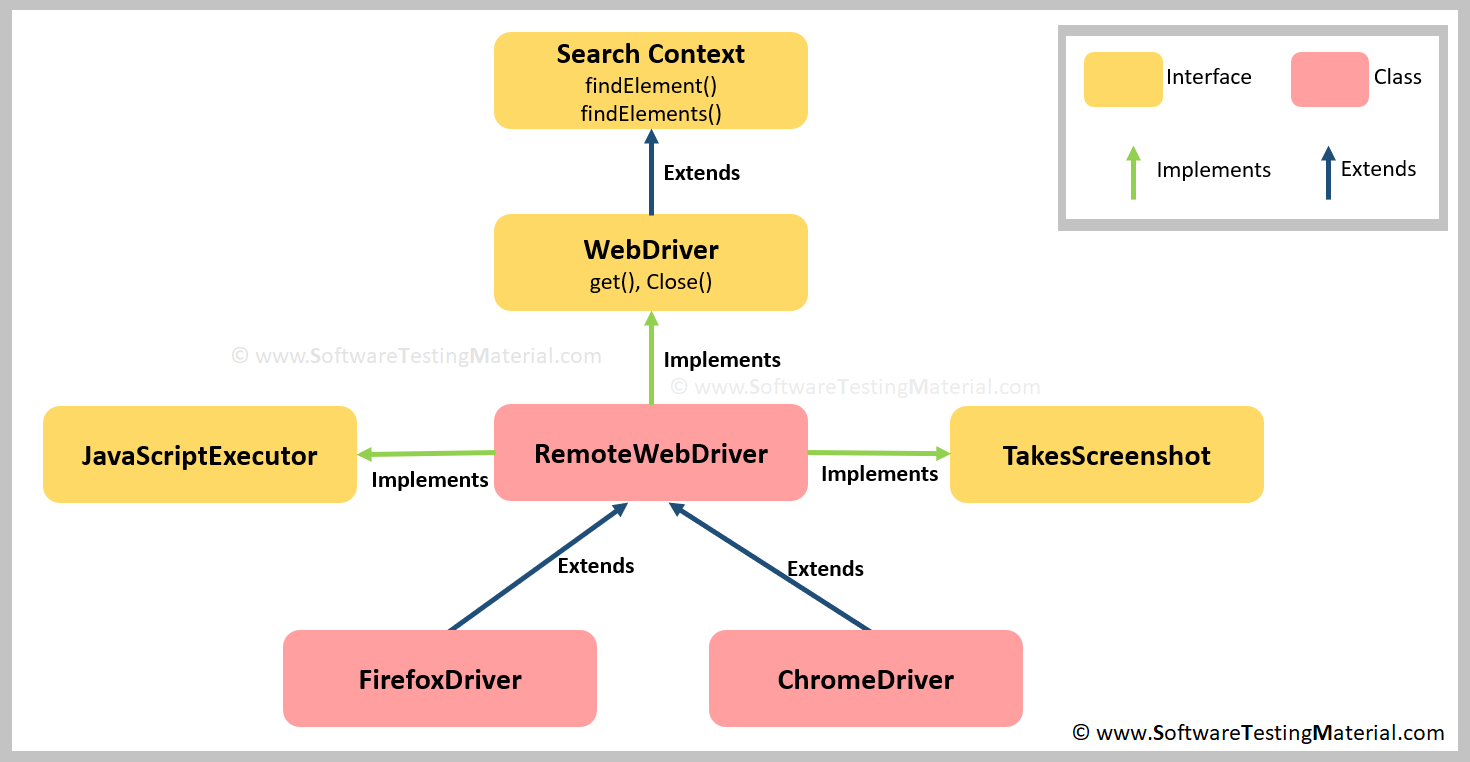
# Selenium Structure



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

We cannot write our code like this because we cannot create Object of an Interface. WebDriver is an interface.

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

The FirefoxDriver instance which gets created based on above statement will be only able to invoke and act on the methods implemented by FirefoxDriver and supported by Firefox Browser only. We know that FirefoxDriver is a class and it implements all the methods of WebDriver interface. Using this statement, we can run our scripts only on Firefox Browser.

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

We can create Object of a class FirefoxDriver by taking reference of an interface (WebDriver). In this case, we can call implemented methods of WebDriver interface.

## Pre-requisites

Rule:

Object repository: should capture driver by constructor

Public class page

{

Public void page(WebDriver driver)

{

This.driver=driver;

Pagefactory.initElements(driver, this)

}

}

Step def: Should pass driver during object creation

Public class stepdef

{

Public void definition()

{

Page p=new page(driver);

}

}

# Selenium Introduction

Selenium – Web and mobile applications

For desktop – SIKULI &AutoIT (add-on automation tool)

1. Selenium IDE

Record and playback

Add-on for firefox (only)

It can be recorded only in firefox, but can be played back in other browsers, through RC and WebDriver.

If app only compatible with IDE or Chrome no other options.

1. Selenium RC

Remote control

Act as Server/API

1. WebDriver

To overcome the disadvantages of RC

1. Grid

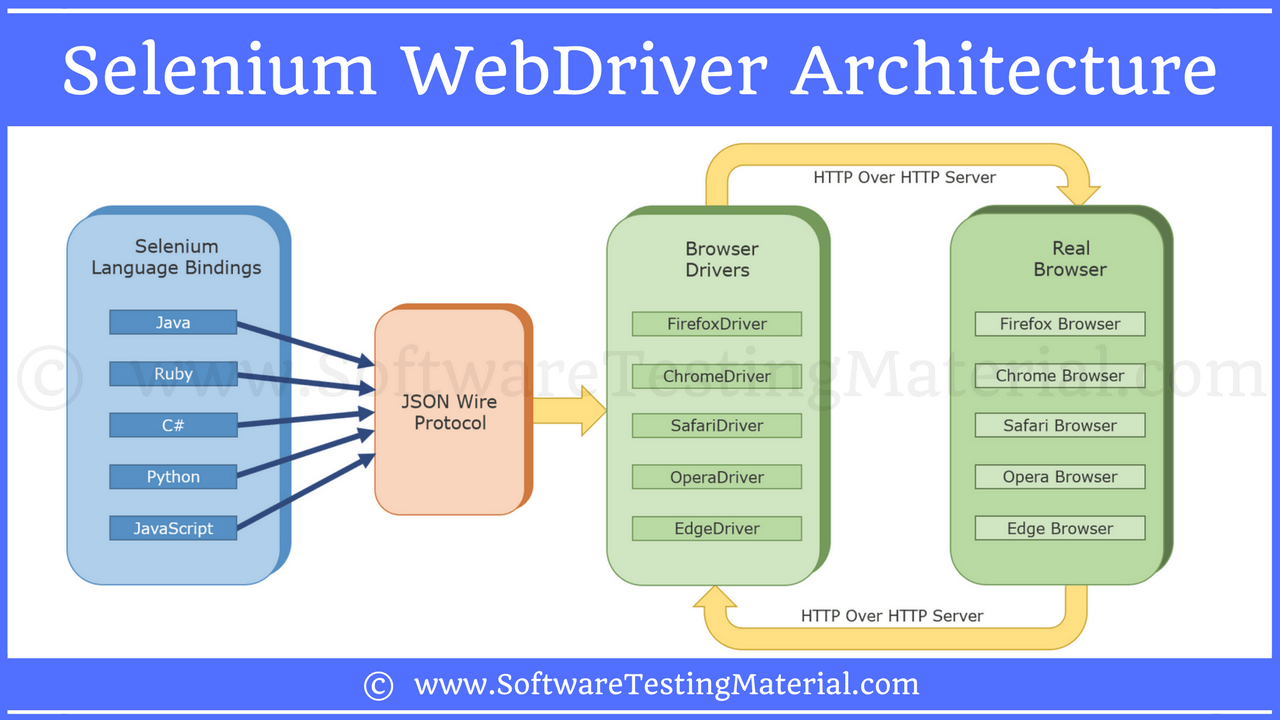
For parallel execution

100 Test cases (Simultaneously in Chrome, Safari, Firefox, IE, Opera)

**Selenium configuration**

1. Eclipse
2. JDK/JRE 8
3. Selenium standalone server 2.44.0 jar

# WebDriverArchitecture



## **Selenium Client Libraries/Language Bindings:**

Language bindings allow Selenium to support multiple languages like Java, Ruby, Python, etc.

## **JSON Wire Protocol over HTTP Client:**

JSON (JavaScript Object Notation) is a **lightweight data-interchange format**.

It is used to transfer data between a server and a client on the web over HTTP server.

## **Browser Drivers:**

Each browser contains separate browser driver. When a browser driver is received any command then that command will be executed on the respective browser and the response will go back in the form of HTTP response.

## **Browsers:**

Selenium supports multiple browsers such as Firefox, Chrome, IE, Safari etc.

In real time, you write a code in your UI (say Eclipse IDE) using any one of the supported Selenium client libraries (say Java).

### Function

Once you click on Run, every statement in your script will be converted as a URL with the help of JSON Wire Protocol over HTTP. The URL’s will be passed to the Browser Drivers.

Every Browser Driver uses a HTTP server to receive HTTP requests.  Once the URL reaches the Browser Driver, then the Browser Driver will pass that request to the real browser over HTTP. Then the commands in your selenium script will be executed on the browser.

If the request is *POST* request then there will be an action on browser

If the request is a *GET* request then the corresponding response will be generated at the browser end and it will be sent over HTTP to the browser driver and the Browser Driver over JSON Wire Protocol and sends it to the UI (Eclipse IDE).

This is all about Selenium WebDriver Architecture.

# Selenium Locators

Locatorsprovide a way to access the HTML elements from a web page.

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

1. Locating by id

driver.findElement(By.id(“id name”))

1. Locating by name

driver.findElement(By.name(“name”))

1. Locating by tagname

driver.findElement(By.tagName(“tagname”))

1. Locating by LinkText

driver.findEleme nt(By.linkText(“name”))

1. Locating by Partial LinkText

driver.findElement(By.partialLinkText(“name”))

1. Locating by class name

driver.findElement(By.className(“name”))

1. Locating by CSS Locators

driver.findElement(By.cssSelector(“name”))

e.g.,

By.cssSelector("#cheese li")

Used to find webelements inside frames

1. Locating by Xpath

driver.findElement(By.xpath("//div[@class='name']"));

# Handlingdynamicobjects

## Pre requisites

1. Absolute xpath = Starts with single slash / - html/body/div[1]/div[2]
2. Relative xpath = starts with double slash // - //form/div[1]/div/
3. Multiple attribute - //\*[@id='Email'][@name='Email']
4. AND - //\*[@id='Email' and @name='Email']
5. OR - //\*[@id='Email' or @name='Email']

## If partially changes

1. Contains()-//div[contains(@id,’username’)]
2. Starts-with() - //div[starts-with(@id,’user’)]
3. Starts-with in cssSelector() - By.cssSelector("td[class^=buttonTitle]")
4. Ends-with in cssSelector() - By.cssSelector("td[class$=buttonTitle]")
5. Contains in cssSelector() - By.cssSelector("td[class\*=buttonTitle]")
6. Text() - //\*[text()='New look for sign-in']
7. Last() – **(**//\*[@id='Email']**)[last()]**
8. Last()-n – **(**//\*[@id='Email']**)[last()** **-** **n]**
9. Position() - **(**//\*[@id='Email']**)[position()** **= n]** or **(**//\*[@id='Email']**)[n]**
10. Index- //input[2]
11. Following [all following tags] - //td[3]//**following::td**
12. Following-sibling [sibling only] - //td[3]//**following-sibling::td**
13. Preceding [all preceding tags] - //td[3]//**preceding::td**
14. Preceding-sibling [sibling only] - //td[3]//**preceding-sibling::td**
15. Parent - //td[3]//**parent::tr** [one level of parent]
16. Child - //tr[3]//**child::td**
17. Ancestor - //input[@id='email']//**ancestor::\*** [parent, grandparent……..]
18. Descendant - //div[@class='\_4bl9 \_ihf']//descendant::\* [child, grandchild……..]
19. -or-self - including current tag

**Precedings:** all the nodes before current node except Ancestors

**Ancestors:** all parents, grand parents…

<a>

<b>

<c/>

</b>

<d>

<e/>

</d>

</a>

For tag <e>

1. Precedings – b, c
2. Ancestors – a, d

## If changes completely

Selenium provides different API to use function keys. For example tab key, enter keys, F5 etc.

driver.findElement(By.id(“password”)).sendKeys(Keys.ENTER));

or

driver.findElement(By.*id*(“password”)).sendKeys(Keys.TAB));

# WebDriver Commands

driver.get(“url”); – Load page in the current window

driver.findElement(WebElement); – find an element

driver.getTitle();–Fetches the current page title

driver.getClass();– retrieve the Class object (returns xpath of an element)

driver.getAttribute("class"); - retrieve the value of the specified attribute

driver.getCurrentUrl();–current page URL

driver.getPageSource();– Source code of the page

driver.close();– To close the current window

driver.quit();– To close all the windows

driver.getText(); – get inner text of the specified web element

driver.getWindowHandle();- get current window’s unique id (String)

driver.getWindowHandles(); - Get all windows’ unique IDs (Store in a Set as string)

driver.isEnabled(); - Check Whether the Element is Enabled Or Disabled

driver.navigate().to("URL"); - Launch URL

driver.navigate().forward(); - Move forward

driver.navigate().back(); - Move backward

driver.navigate().refresh(); - Refresh

**WebElementCommands**

**driver**

element.clear(); - To clear the populated value

element.sendKeys(“text”); - To type the input

element.click(); - Clicking of an element

element.isDisplayed();- Check whether the element is displayed

element.isEnabled(); - Check whether the element is Enabled

element.isSelected(); - Check whether the element is Selected (Checkbox/Radio button)

 element.submit(); - This method works well/better than the click()

element.getText(); - This method will fetch the visible (i.e. not hidden by CSS) innerText of the element

element.getTagName(); - This method gets the tag name of this element

element.getCssValue(); - Fetch CSS property value of the give element

element.getAttribute(); - gets the value of the given attribute of the element

element.getSize();- fetch the width and height of the rendered element

element.getLocation(); -locate the location of the element on the page

# Select Class/Drop down handling

1. Single select dropdown
2. Multi select dropdown

Select select = new Select(driver.findElement(By.id("SelectID\_One")));

select.selectByValue("greenvalue");

select.selectByVisibleText("Red");

select.selectByIndex(2);

Types of Select Methods:

1. selectByVisibleText()
2. selectByIndex()
3. selectByValue()
4. public List<WebElements> getOptions()

Types of DeSelect Methods:

1. deselectByVisibleText
2. deselectByIndex
3. deselectByValue
4. deselectAll

# Action Class

To perform mouse actions

Actions actions=new Actions(driver);

actions.moveToElement(Element).build().perform();

click (): Simply click on element

doubleClick (): Double clicks onElement

contextClick(): Performs a right clickon an element

clickAndHold(): Clicks at the present mouse location (without releasing)

dragAndDrop(source, target): Clicks in source location and moves to the location of the target element [Two arguements]

dragAndDropBy(Sourcelocator, x-axis pixel of Destinationlocator, y-axis pixel of Destinationlocator)[three arguments by pixels]

moveToElement(toElement): It shifts the mouse to the center of the element

release(): Releases the depressed left mouse button at the existing mouse location

# JavascriptExecutor (Scrolling)

Scenario 1: To scroll down the web page by pixel.

((JavascriptExecutor)driver).executeScript("window.scrollBy(0,1000)");

Scenario 2: To scroll down the web page at the bottom of the page.

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

Scenario 3: To scroll down the web page by the visibility of the element.

((JavascriptExecutor)driver).executeScript("arguments[0].scrollIntoView();", driver.findElement(By.linkText("VBScript")));

Scenario 4: Horizontal scroll on the web page.

((JavascriptExecutor)driver).executeScript("arguments[0].scrollIntoView();", driver.findElement(By.linkText("VBScript")));

# Screenshot

FileUtils.*copyFile*(((TakesScreenshot)driver).getScreenshotAs(OutputType.*FILE*), new File("C:/selenium/error.png"));

TakesScreenshot scrshot=((TakesScreenshot)*driver*);

File source=scrshot.getScreenshotAs(OutputType.***FILE***);

File target=**new** File(".//Screenshot"+".//"+name+".png");

FileHandler.*copy*(source, target);

# Alert& Popup

1. driver.switchTo().alert().dismiss();
2. driver.switchTo().alert().accept();
3. driver.switchTo().alert().getText();
4. driver.switchTo().alert().sendKeys("Text");

# Excel Read/Write

Excel Read/Write

**Read**

**Abce.xls**

**Abcd.xlsx**

FileInputStream fis=new FileInputStream(“path”);

XSSFWorkbook workbook=new XSSFWorkbook(fis);

XSSFSheet sheet=workbook.getSheetAt(0);

Row row=sheet.getRow(0);

Cell cell=row.getCell(0);

cell.getCellValue();

**Write**

FileOuputStream fos=new FileOuputStream(“path”);

XSSFWorkbook workbook=new XSSFWorkbook(fis);

XSSFSheet sheet=workbook.createSheetAt(0);

Row row=sheet.createRow(0);

Cell cell=row.createCell(0);

Cell.setCellValue();

workbook.write(fos)

***For different form use this link***

<https://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/>

Example read:

package com.howtodoinjava.demo.poi;

//import statements

public class ReadExcelDemo

{

    public static void main(String[] args)

    {

        try

{

            while (rowIterator.hasNext())

            {

                Row row = rowIterator.next();

                //For each row, iterate through all the columns

                Iterator<Cell>cellIterator = row.cellIterator();

                while (cellIterator.hasNext())

                {

                    Cell cell = cellIterator.next();

                    //Check the cell type and format accordingly

                    switch (cell.getCellType())

                    {

                        case Cell.CELL\_TYPE\_NUMERIC:

                            System.out.print(cell.getNumericCellValue() + "t");

                            break;

                        case Cell.CELL\_TYPE\_STRING:

                            System.out.print(cell.getStringCellValue() + "t");

                            break;

                    }

                }

                System.out.println("");

            }

            file.close();

        }

        catch (Exception e)

        {

            e.printStackTrace();

        }

    }

}

Output:

ID      NAME        LASTNAME

1.0     Amit        Shukla

2.0     Lokesh      Gupta

3.0     John        Adwards

4.0     Brian       Schultz

Example write:

static void basedOnValue(Sheet sheet)

{

    //Creating some random values

    sheet.createRow(0).createCell(0).setCellValue(84);

    sheet.createRow(1).createCell(0).setCellValue(74);

    sheet.createRow(2).createCell(0).setCellValue(50);

    sheet.createRow(3).createCell(0).setCellValue(51);

    sheet.createRow(4).createCell(0).setCellValue(49);

    sheet.createRow(5).createCell(0).setCellValue(41);

    SheetConditionalFormattingsheetCF = sheet.getSheetConditionalFormatting();

    //Condition 1: Cell Value Is   greater than  70   (Blue Fill)

    ConditionalFormattingRule rule1 = sheetCF.createConditionalFormattingRule(ComparisonOperator.GT, "70");

    PatternFormatting fill1 = rule1.createPatternFormatting();

    fill1.setFillBackgroundColor(IndexedColors.BLUE.index);

    fill1.setFillPattern(PatternFormatting.SOLID\_FOREGROUND);

    //Condition 2: Cell Value Is  less than      50   (Green Fill)

    ConditionalFormattingRule rule2 = sheetCF.createConditionalFormattingRule(ComparisonOperator.LT, "50");

    PatternFormatting fill2 = rule2.createPatternFormatting();

    fill2.setFillBackgroundColor(IndexedColors.GREEN.index);

    fill2.setFillPattern(PatternFormatting.SOLID\_FOREGROUND);

    CellRangeAddress[] regions = {

            CellRangeAddress.valueOf("A1:A6")

    };

    sheetCF.addConditionalFormatting(regions, rule1, rule2);

}

# Window Handling

String mainWindow=driver.getWindowHandle();

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

Iterator<String>itr= set.iterator();

while(itr.hasNext())

{

if(!mainWindow.equals(itr.next()))

{

driver.switchTo().window(itr.next());

driver.close();

}

}

driver.switchTo().window(mainWindow);

# WebTables

Public class dynamic\_table

{

PSVM (string [] args) throws interruptException

{

WebDriver driver = new firefoxDriver();

Driver.get(“url”);

WebElementdynamictable=driver.findElement(By.xpath(“path”));

List<WebElement>rows\_table=dynamictable.findElements (by.tagname(“tr”));

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

For(int row=0; row)

# Synchronization

Two or more components involved to perform any action, we expect these components should be worked together with same pace

Exceptions are

* ElementNotVisibleException
* NoSuchElementException

In such scenarios, we use Synchronization/wait to overcome these kinds of exceptions.

Automation Testing is commonly comprising of following two segments:

1. Unconditional Synchronization – Thread.*sleep*(times) – not advisable
2. Conditional Synchronization

**Conditional**

**Implicit wait**

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

Normally, it is not recommended to use implicit waits, when we can use explicit waits or fluent waits.

**Explicit wait**

Explicit Wait tells the WebDriver to Wait until the specified condition is met or maximum time elapses before throwing NoSuchElement (or) ElementNotVisible Exceptions.

We must first create instance for “WebDriverWait” class.

WebDriverWait wait = new WebDriverWait(driver, 10);

wait.until(ExpectedConditions.*visibilityOfElementLocated*(By.id("someid"))).click();

We would normally use explicit wait if an element takes a long time to load. We also used explicit wait to check CSS property of an element (presence, clickability. etc) which can change in Ajax applications.

Following are the ExpectedConditions:

* alertIsPresent()
* elementSelectionStateToBe()
* visibilityOfAllElementsLocatedBy()
* elementToBeSelected()
* frameToBeAvaliableAndSwitchToIt()
* invisibilityOfTheElementLocated()
* invisibilityOfElementWithText()
* elementToBeClickable()
* presenceOfAllElementsLocatedBy()
* presenceOfElementLocated()
* textToBePresentInElement()
* textToBePresentInElementLocated()
* textToBePresentInElementValue()
* titleIs()
* titleContains()
* visibilityOf()
* visibilityOfAllElements()
* visibilityOfElementLocated()

**Fluent wait**

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

**Above code is deprecated in Selenium v3.11 and above. You need to use**

Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(Duration.ofSeconds(30))

.pollingEvery(Duration.ofSeconds(5))

.ignoring(StaleElementReferenceException.class);

# Accessing properties file

1. Create file (xxx.properties)
2. Give hardcode data as key value pairs (e.g., LoginId = Kalaiselvamk)
3. Calling from base class
   * Properties prop=new Properties();
   * FileInputStream fin = new FileInputStream (“properties file path”);
   * Prop.load(fin);
   * prop.getProperty(“LoginId”);

Commenting in properties file is “**#”**

If we call commented data the output will be “**null**”

# Frame handling

R-Click on frame 🡪 if frame option is available the frame exists.

All the frames are having the tag name as “iframe”.

To find the number of frames in the webpage

List<WebElement> frameList=driver.findElements(By.tagName(“iframe”));

Here we will not be able to find/locate elements inside frames.

For that we have to switch into frame.

driver.switchTo().frame(1); //pass frame number as parameter.

or

driver.switchTo().frame(“frame Name”); //pass frame name as parameter.

or

driver.switchTo().frame(“xpath of the frame”);

After that we can be able to handle elements

driver.findElement(//\*[@id=’username’]).sendKeys(“username”);

# Cucumber [HTML Report/ Extent Reports]

1. Extent-config.xml [Paste it in resources package]
2. Copy paste some configurations in @AfterClass method [runner file]
3. Add destination path in (CucumberOptions {plugins})
4. Add vimalselvam dependency in pom.xml

# Retry/Retest

Two Levels

1. Test Level
2. Run time Level

*Create First class, that extends****IRetryAnalyser****:*

Public class **RetryAnalyser**implements **IRetryAnalyser**

{

int counter=0;

int limit=3;

public Boolean **retry**(**ITestResult** result)

{

If(counter<limit)

{

return true;

}

return false;

}

## Test Level

@Test (retryAnalyser = Package.RetryAnalyser.class)

## RunTime Level

Create another class that extends ***IAnnotationTransformer***

Public classMyTransformer implements IAnnotationTransformer

{

public void transformer (ITestAnnotationannotation, Class testClass, Constructor testConstructor, Method testMethod)

{

Annotation.setRetryAnalyser(RetryAnalyser.class)

}

}

<listeners>

<listener class-name = “package.MyTransformer”/>

<listeners>

Two Ways

1. Create RetryAnalyser class → implements ***IRetryAnalyser*** (From TestNG) → override **retry( )** method

Note: We can directly call this inside @Test annotations

1. Create MyTransformer class → implements ***IAnnotationTransformer*** (From TestNG) → override **transformer( )**method

→ call RetryAnalyser class (e.g., **Annotation**.setRetryAnalyser(**RetryAnalyser**.class))

Add <Listeners> code inside TestNG – It will retry the failed test cases by default

# Screenshot for failed TestCases

Create Listener class →implements ***ITestListener*** (From TestNG) → override **onTestFailure()** method

TestNG coding for Screenshot

<**listeners**>

    <**listener**class-name="com.pack.listeners.TestListener"/>

</**listeners**>

# Rerun Failed TestCases

@RunWith(ExtendedCucumber.**class**)

@ExtendedCucumberOptions(jsonReport = "target/cucumber.json", retryCount = 3)

Just add the above @ExtendedCucumberOptions above the @CucumberOptions and changethe @RunWith(Cucumber.class) as @RunWith(ExtendedCucumber.class)

# Screenshot in Cucumber

@After → Screenshot code with the condition of Scenario.isFailed

# ExtendReport in cucumber

One plugin for com.vimalselvam dependency. Connect in runner file in cucumber options, in format

# Schedule Execution in Jenkins

Build periodically → so many options with regular expressions, by using those codes we can schedule.

# AutoIT code

class AUTOIT implements Runnable{

public void run()

{

Runtime.getRuntime().exec("path where AUTOIT cod exe file exists");

}

driver.get(“application URL”);

AUTOIT autoit=new autoit ();

Thread t1 =new Thread(autoit);

t1.start();

# Automation Testing

Test cases to be automated can be selected using the following criterion to increase the automation ROI

* High Risk - Business Critical test cases
* Test cases that are repeatedly executed
* Test Cases that are very tedious or difficult to perform manually
* Test Cases which are time-consuming

The following category of test cases are not suitable for automation:

* Test Cases that are newly designed and not executed manually at least once
* Test Cases for which the requirements are frequently changing
* Test cases which are executed on an ad-hoc basis.

# Questions

## Difference between driver.get() and driver.navigate().to()

Driver.get()

* Doesn’t maintain history & cookies
* Hence we can't move forward and backward
* Refreshes the page

Driver.navigate().to()

* It will maintain history & cookies
* Hence we can move forward and backward
* Wont refresh the page

## Driver.navigate().back() explanation

**Driver**– is instance

i.e., WebDriver driver=new firefoxDriver();

**Navigate**() – Method in firefoxDriver class (in driver)

**Back()**– Return type (class) of Navigate() method

public class ReturinigClass

{

public static void main(String[] args)

{

Class cl=new Class();

cl.add().aaa().add().aaa().add().aaa();

}

Class aaa()

{

Class cl=new Class();

System.out.println("aaa method in RerurinigClass");

return cl;

}

}

class Class

{

ReturinigClassrt=new ReturinigClass();

ReturinigClassadd() {

System.out.println("I am add() method");

return rt;

}

}

# Singleton Class

Scenario:

1. Keep track of same driver instance throughout execution.
2. DBMS connectivity.
3. Loading external files like properties, excel etc once rather than loading again and again.
4. Logger.
5. Declare constructor of class as private so that no one instantiate class outside of it.
6. Declare a static reference variable of class. Static is needed to make it available globally.
7. Declare a static method with return type as object of class which should check if class is already instantiated once.

package com.softwaretestingblog.singleton;

public class SingletonDesignDemo1

{

private static SingletonDesignDemo1 obj=new SingletonDesignDemo1();

//We Make Constructor as Private so other class will not create object

private SingletonDesignDemo1()

{

}

public static SingletonDesignDemo1 getObject()

{

return obj;

}

//Other Method protected by Singleton-ness

public static void demomethod()

{

System.out.println("Singleton Design Pattern Method 1");

}

}

Why should we return this and newLandingPage????????

Consider their example

public LoginPagetypeUsername(String username) {

driver.findElement(usernameLocator).sendKeys(username);

return this;

}

public LoginPagetypePassword(String password) {

driver.findElement(passwordLocator).sendKeys(password);

return this;

}

public HomePagesubmitLogin() {

driver.findElement(loginButtonLocator).submit();

return new HomePage(driver);

}

now assume we have create the page object create as page. If you didn't return anything your code would like

page.typeUsername("tarun");

page.typePassword("lalwani");

HomePagenewPage = page.submitLogin()

But when you return this, it allows you to do method chaining. So I can use it like below

HomePagenewPage = page.typeUsername("tarun").typePassword("lalwani").submitLogin()

As you can see it will save you some coding effort and much more elegant with IDE intellisense

Online Exam answers

# BlockUI Handling

**try** {newButton.click();}

**catch**(ElementClickInterceptedExceptione)

{

e.printStackTrace();

*wait*.until(ExpectedConditions.*invisibilityOf*(*driver*.findElement(By.*xpath*("//div[contains(@class,'blockUI blockOverlay)]"))));

newButton.click();

}

# Executing/Calling exe file inside code

Runtime.*getRuntime*().exec("C:\\Users\\ext.kkanagasabai\\Downloads\\ZoomInstaller.exe");

# Get Latest File

**publicstatic** File getLatestFile() {

File file=**null**;

File dir = **new** File("C:\\Users\\ext.kkanagasabai\\Downloads");

FileFilterfileFilter = **new**WildcardFileFilter("\*.csv");

File[] files = dir.listFiles(fileFilter);

**if** (files.length> 0) {

Arrays.*sort*(files, LastModifiedFileComparator.***LASTMODIFIED\_REVERSE***);

file=files[0];

}

**return**file;

}

# SQL connection/ DB Connection

**class TestUtil{**

**publicstatic** Connection getConnection() **throws**ClassNotFoundException, SQLException, InstantiationException, IllegalAccessException {

Class.*forName*("com.microsoft.sqlserver.jdbc.SQLServerDriver");

Connection connection = (Connection) DriverManager.*getConnection*("jdbc:sqlserver://10.13.44.170:1433;databaseName=WTT\_CMA\_UAT;","WTT\_TestUser","wtt@test123");

**return**connection;

}

}

Public void execution(){

Statement statement =TestUtil.*getConnection*().createStatement();

String queryString = "Select Container\_Type From tblRateAgreementMaster where Container\_size ='"+row.getCell(21).getStringCellValue()+"'";

ResultSetrs = statement.executeQuery(queryString);

**while** (rs.next()) {

containerType.add(rs.getString(1));

}

}

# Default things to be tested/Interface/Static contents

* Header
* Footer
* Logo
* Forward/Backward

# Headless Browser

WebDriver driver = new HtmlUnitDriver();

Sometimes we need to add proxy details

# TestNG Framework (Testing New Generation Framework)

TDD – Test Driven Development Framework

It’s a unit testing framework, for developers. Testers also can use to design test cases in systematic way.

Features available:

* 1. HTML reports
  2. Different annotations
  3. Priorities/sequences
  4. Dependencies
  5. Grouping
  6. Paremeters
  7. Data provider
  8. Testng.xml
  9. Can generate proper format of test results
  10. Testcases can be grouped and execute by testing.xml
  11. Same testcase can be execute multiple times (invocation count)
  12. Make it run in certain order by prioritizing them
  13. Skip particular test
  14. Annotations available
  15. No static main method necessary
  16. Uncaught exceptions are automatically handled by TestNG, by considering that test case as failed.
  17. Parallel testing is possible

Annotations

|  |  |  |
| --- | --- | --- |
| @BeforeSuite | Set system property | Pre conditions |
| @BeforeTest | Launch chrome browser |
| @BeforeClass | Login to app |
| @BeforeMethod | Enter URL (repeat b4 all @Test) |
|  |  |  |
| @Test | Google title test | Test cases |
|  |  |  |
| @AfterMethod | Logout (repeat after all @Test) | Post conditions |
| @AfterClass | Close browser |
| @AfterTest | Delete all cookies |
| @AfterSuite | Generate test report |

1. @BeforeMethod and @AfterMethod will execute before and after each and every @Test methods
2. @Test => Execute in alphabetical manner
3. @Test(enabled = false) => Skip the testcase
4. @Test(priority = 1) => Make them execute in particular order/sequence
5. @Test(priority = 1, groups = “cars”) => make test cases grouping
6. @Test(dependsOnMethods= “loginTest”) => if loginTest got failed, it’s all dependent methods will get skipped
7. @Test(dependsOnMethods= “loginTest”, alwaysRun = true)=> this method will always run even its dependent method fails
8. @Test(invocationCount= 5) => Same test case will be executed 5 times
9. @Test(timeOut = 2000) => Entire test method should be terminated in 2000milli seconds
10. @Test(expectedExceptions = ArithmeticException.class) => instruct JVM not to consider failed, if this exception arises.
11. @DataProvider and @Test(dataprovider = “**getTestData**”) => provide data from external source
12. @Parameters({“url”}, {“username”})

## TestNG html report

Test-Output folder -> Index.html

* 1. Info
     + Testing.xml
     + 1 Tests
     + 4 Groups
     + Times
     + Reporter output
     + Ignored methods
     + Chronological view
  2. Results
     + Passed
     + failed

## Data Driven Framework with TestNG – Data Provider (@dataprovider)

**[Test case will execute multiple times for each and every set of data]**

1. public class TestUtil

{

Public static ArrayList<Object[]> getDataFromExcel()

{

//assume 4 rows of data

//read all data from excel by apache POI or XLs\_Reader

//Store every row in Object[] array

//add that array in an ArrayList<Object[]> in every iteration

//finally return the ArrayList<Object[]>

}

}

1. @Dataprovider

Public Iterator<Object[]> **getTestData**()

{

ArrayList<Object[]> testData = TestUtil.getDataFromExcel();

return testData.iterator();

}

1. @Test(dataprovider = “**getTestData**”)

Public void myTestPage(String firstName, String lastName, String email, String state)

{

//use the parameters

}

Result:

PASSED: myTestPage(String firstName1, String lastName1, String email1, String state1)

PASSED: myTestPage(String firstName2, String lastName2, String email2, String state2)

PASSED: myTestPage(String firstName3, String lastName3, String email3, String state3)

PASSED: myTestPage(String firstName4, String lastName4, String email4, String state4)

**Parameters (keep environment variables – generic variables throughout the program)**

Passing parameters from testing.xml file

**Testmg.xml**

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

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

<suite name="Suite">

<test name="Test">

<parameter name=“url” value=“https://www.gmail.com”>

<parameter name=“username” value=“abcd.gmail.com”>

<classes>

<class name="com.paramenter.ParameterTest" />

</classes>

</test>

</suite>

**TestCase**

@test

@parameters({“url”, “username”})

Public void gmailLoginTest(String url, String userName)

{

//code

}

**Parallel run of tests/classes/suites in TestNG**

**Parallel – methods**

<suite name="Suite">

<test name="Test" parallel= “methods”>

<classes>

<class name="com.paramenter.ParameterTest" />

</classes>

</test>

</suite>

**Parallel – Classes**

<suite name="Suite">

<test name="Test" parallel= “classes”>

<classes>

<class name="com.paramenter.ChromeTest" />

<class name="com.paramenter.FirefoxTest" />

</classes>

</test>

</suite>

**Parallel – Tests**

<suite name="Suite" parallel="tests" thread-count="1">

<test name="Test">

<classes>

<class name="com.paramenter.ChromeTest" />

<class name="com.paramenter.FirefoxTest" />

</classes>

</test>

<test name="Test">

<classes>

<class name="com.paramenter.ChromeTest" />

<class name="com.paramenter.FirefoxTest" />

</classes>

</test>

</suite>

# POM (Page Object Model)

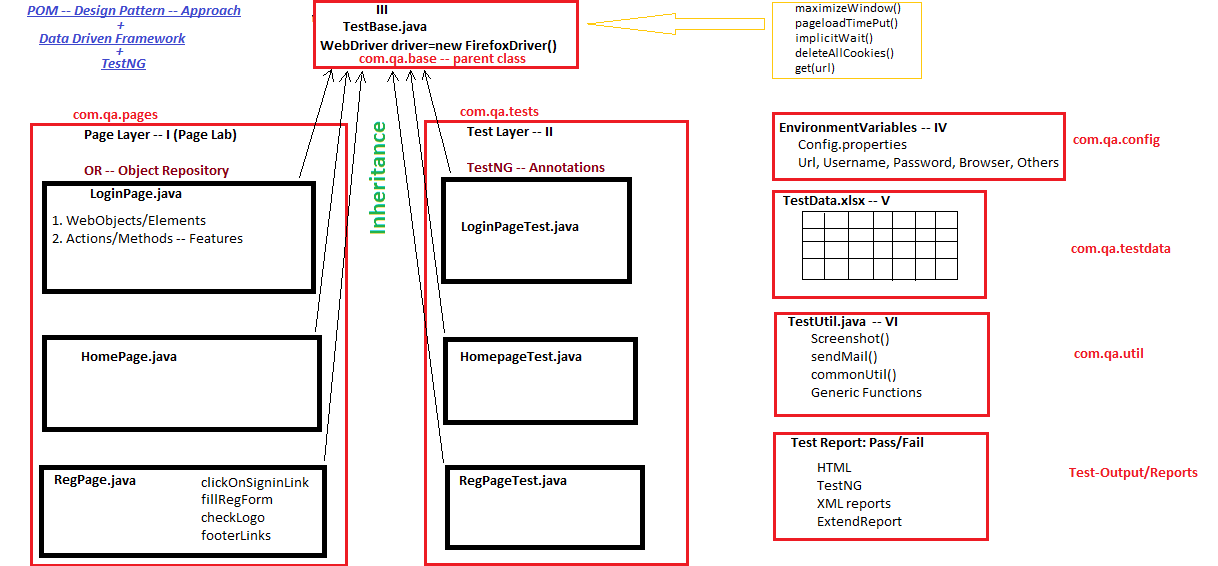
Design pattern to maintain object repository for WebElements

Easier to maintain the code

Objects can be reused, avoid duplicates

Layers:

* 1. Page layer – WebElements and their corresponding methods
  2. Test Layer – Test cases and its verifications



### POM without PageFactory

**In page layer:**

public class HomePage {

    By emailAddress = By.xpath("//div[contains(@id,'Emaild')]");

    public void typeEmailId(String id){

        driver.findElement(emailAddress).sendKeys(id)

    }

}

### POM with PageFactory

**In Page Layer:**

public class HomePage {

public HomePage()

{

PageFactory.initElements(driver, this);

}

@FindBy(xpath="//button[text()='email']")

WebElement emailAddress;

Public void typeEmailId(String id)

{

emailAddress.sendKeys(id);

}

}

# Listeners

**public** **class** Listener **implements** ITestListener

{

**public** **void** onFinish(ITestContext arg0)

{

*driver*.quit();

logger.info("Test is Finished for GreenDMS");

}

**public** **void** onStart(ITestContext arg0)

{

logger.info("Test is starting for GreenDMS");

}

**public** **void** onTestFailedButWithinSuccessPercentage(ITestResult arg0)

{

logger.info("On Test Failed But Within Having Success Percentage");

}

**public** **void** onTestFailure(ITestResult arg0)

{

logger.info("Test is Failed");

//logic for Screenshot

}

**public** **void** onTestSkipped(ITestResult arg0)

{

logger.info("Test is skipped");

}

**public** **void** onTestStart(ITestResult arg0)

{

logger.info("Test is going to begin ");

}

**public** **void** onTestSuccess(ITestResult arg0)

{

logger.info("Test is successfully completed");

}

}

# Selection of testcases for automation

High Risk - Business Critical test cases

Test cases that are repeatedly executed

Difficult to perform manually

Which are time-consuming in manual

Not suitable for automation:

Newly designed and not executed manually at least once

Requirements are frequently changing

Which are executed on an ad-hoc basis.