**Selenium-**

**Explain how you can login into any site if it’s showing any authentication popup for password and username?**

Pass the username and password with url

* Syntax-http://username:password@url
* ex- <http://creyate:tom@www.gmail.com>

**Explain the difference between single and double slash in X-path?**

Single slash ‘/ ’

* Single slash ( / ) start selection from the document node
* It allows you to create ‘absolute’ path expressions

Double Slash ‘// ’

* Double slash ( // ) start selection matching anywhere in the document
* It enables to create ‘relative’ path expressions

**Absolute** xpath: **single slash** : **root node**.

**Relative** xpath: **double slash** : **anywhere**

**How do you handle alert pop-up ?**

Ans- To handle alert pop-ups, we need to 1st switch control to alert pop-ups then click on ok or cancle then move control back to main page.

driver.switchTo().alert()

**Give the example for method overload in WebDriver.**

Ans- frame(string), frame(int), frame(WebElement).

**What are the different exceptions you got when working with WebDriver ?**

Ans- ElementNotVisibleException, ElementNotSelectableException, NoAlertPresentException, NoSuchAttributeException, NoSuchWindowException, TimeoutException, WebDriverException etc.

Difference between Selenium IDE, RC and Webdriver

| **Feature** | **Selenium IDE** | **Selenium RC** | **WebDriver** |
| --- | --- | --- | --- |
|  |  |  |  | |
| Browser Compatibility | Selenium IDE comes as a Firefox plugin, thus it supports only Firefox | Selenium RC supports a varied range of versions of Mozilla Firefox, Google Chrome, Internet Explorer and Opera | WebDriver supports a varied range of versions of Mozilla Firefox, Google Chrome, Internet Explorer and Opera. Also supports HtmlUnitDriver which is a GUI less or headless browser. | |
| Record and Playback | Selenium IDE supports record and playback feature | Selenium RC doesn't supports record and playback feature | WebDriver doesn't support record and playback feature | |
| Server Requirement | Selenium IDE doesn't require any server to be started before executing the test scripts | Selenium RC requires server to be started before executing the test scripts | WebDriver doesn't require any server to be started before executing the test scripts | |
| Architecture | Selenium IDE is a Javascript based framework | Selenium RC is a JavaScript based Framework | WebDriver uses the browser's native compatibility to automation | |
| Object Oriented | Selenium IDE is not an object oriented tool | Selenium RC is semi object oriented tool | WebDriver is a purely object oriented tool | |
| Dynamic Finders (for locating web elements on a webpage) | Selenium IDE doesn't support dynamic finders | Selenium RC doesn't support dynamic finders | WebDriver supports dynamic finders | |
| Handling Alerts, Navigations, Dropdowns | Selenium IDE doesn't explicitly provides aids to handle alerts, navigations, dropdowns | Selenium RC doesn't explicitly provides aids to handle alerts, navigations, dropdowns | WebDriver offers a wide range of utilities and classes that helps in handling alerts, navigations, and dropdowns efficiently and effectively. | |
| WAP (iPhone/Android) Testing | Selenium IDE doesn't support testing of iPhone/Andriod applications | Selenium RC doesn't support testing of iPhone/Andriod applications | WebDriver is designed in a way to efficiently support testing of iPhone/Android applications. The tool comes with a large range of drivers for WAP based testing. For example, AndroidDriver, iPhoneDriver | |
| Listener Support | Selenium IDE doesn't support listeners | Selenium RC doesn't support listeners | WebDriver supports the implementation of Listeners | |
| Speed | Selenium IDE is fast as it is plugged in with the web-browser that launches the test. Thus, the IDE and browser communicates directly | Selenium RC is slower than WebDriver as it doesn't communicates directly with the browser; rather it sends selenese commands over to Selenium Core which in turn communicates with the browser. | WebDriver communicates directly with the web browsers. Thus making it much faster. | |

**What is Selenese?**

Selenese is the language which is used to write test scripts in Selenium IDE.

**What do we mean by Selenium 1 and Selenium 2?**

Selenium RC and WebDriver, in a combination are popularly known as Selenium 2. Selenium RC alone is also referred as Selenium 1.

* + Explicit, implicit wait and Fluent wait
  + Annotations in TestNG
  + Selenium Grid
  + Maven, Jenkins
  + Github
  + Why not use Junit
  + Verbose in testing.xml
  + Logging
  + Browser capabilities
  + POI- read excel file
  + POM and page factory
  + Screenshot
  + Assert vs verify
  + Extracting all broken links on page
  + what is Datadriven framework and Keyword driven?
  + How to handle window and web alerts ?
  + creating object repository using properties File
  + @Dataprovider, @Factory
  + Handling dynamic Ids

<http://www.software-testing-tutorials-automation.com/2014/09/selenium-webdriver-interview-questions_16.html>

**Object repository using property file in Selenium**

To create new properties file, Right click on your project package -> New -> Other .> File- give file name (object.properties).

In object.properties file, give key and value combination.

username.id=ctl00\_PlaceHolderMain\_signInControl\_UserName

password.id=ctl00\_PlaceHolderMain\_signInControl\_password

login.id=ctl00\_PlaceHolderMain\_signInControl\_login

Using key, use values from property file

File src=**new** File("C:\\Test\_workspace\\Practice\\src\\practice\_pck\\object.properties");

**//Create Object of FileInputStream Class. Pass file path.**

FileInputStream fs=**new** FileInputStream(src);

//Create Object of Properties Class.

Properties prp=**new** Properties();

**//Pass object reference fs to load method of Properties object.**

prp.load(fs);

**//Accessing element locators of all web elements using obj.getProperty(key)**

driver.findElement(By.*id*(prp.getProperty("username.id"))).sendKeys("Teammanager.demo2@evalueserve.com");

driver.findElement(By.*id*(prp.getProperty("password.id"))).sendKeys("Evs1234$");

**Why Jenkins and Selenium?**

Jenkins is the leading open-source continuous integration tool. It is used to build and test software projects continuously making it easier to integrate changes to the project.  it is used to schedule jobs for regression testing without manual intervention and hence monitor infrastructural and functional health of application. It can be used like a scheduler for integration testing and also can be used to validate new deployments/environments on a single click on a Build now button .

* Running Selenium tests in Jenkins allows you to run your tests every time your software changes and deploy the software to a new environment when the tests pass.
* Jenkins can schedule your tests to run at specific time.
* You can save the execution history and Test Reports.
* Jenkins supports Maven for building and testing a project in continuous integration.
* Early issue finding – Bug can be detected in early phase of the software development
* Automatic integration – no separate effort required to integrate all changes

Run Jenkin server from cmd by going to location (cd desktop) and giving command : java –jar Jenkins.war.

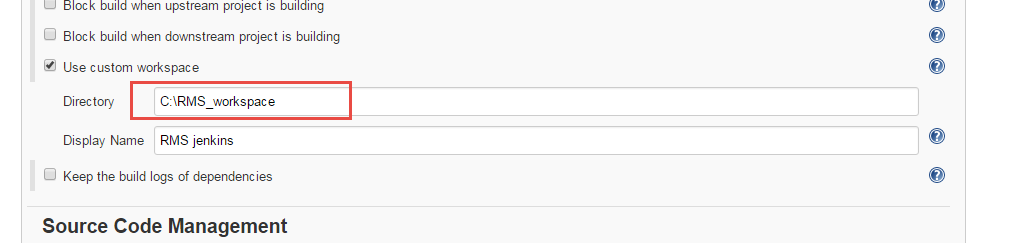
Go to browser and enter URL- <http://localhost:8080>.

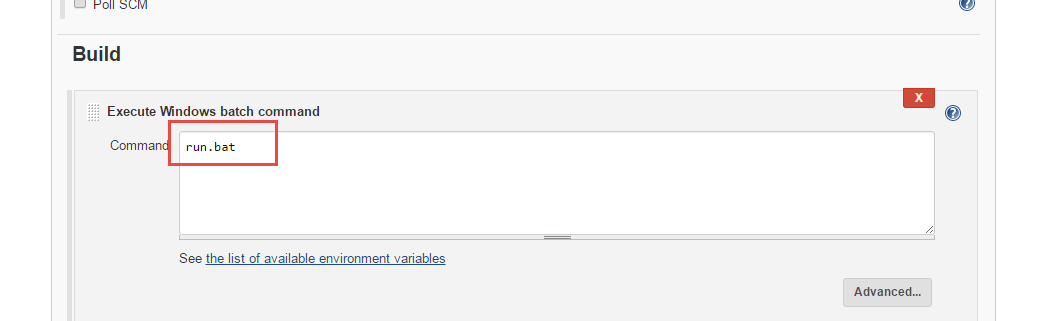
Create new freestyle project in Jenkins.

Configure> Create a batch file (run.bat) and give path of testing.xml file in it.-- java -cp bin;libs/\* org.testng.TestNG testing.xml. Place run.bat in project workspace.

Before it, bin and libs folder (with all jar files of libraries) should be available in workspace.

Give this run.bat file path in Jenkins. Also, give workspace location in Configure.





Click Build now or schedule it.

**Maven – a build automation tool** which is distributed under Apache Software Foundation. It is mainly used for java projects. It makes build consistent with other project. Maven is also used to manage the dependencies. It works very effectively when there is huge number of Jar files with different versions. if any new dependency added in pom file then maven downloads from central repository to local repository. It cuts down the good number of steps in build process and it makes it one step process to do a build.  It helps in bypassing my steps like adding jars to the project library, building reports, executing Junits test cases, creating Jar, War Ear files for the project deployment and many more

**Build tool** is used to setup everything which is required to run your java code independently.

**Dependencies** are the libraries, which are required by the project. For example Log4j jars, Apache Poi jars, Selenium Jars etc. Dependencies are mentioned in the Maven pom.xml like this:

<dependency>

   <groupId>org.seleniumhq.selenium</groupId>

   <artifactId>selenium-java</artifactId>

   <version>2.43.1</version>

   </dependency>

**POM** is Project Object Model XML file that contains information about the project and configuration details used by Maven to build the project. . It contains default values for most projects. Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on.

 If libraries are not available in central repository then maven looks for remote repository. User has to configure the remote repository in pom.xml to download from remote repository.

### ****Build Life Cycle:****

Basic maven phases are used as below.

* **clean**: deletes all artifacts and targets which are created already.
* **compile**: used to compile the source code of the project.
* **test**: test the compiled code and these tests do not require to be packaged or deployed.
* **package**: package is used to convert your project into a jar or war etc.
* **install**: install the package into local repository for use of other project.
* **Github-** Git Hub is a Collaboration and for version control platform. It allows you to keep both local and remote copies of your project. A project which you can publish it among your team members as they can use it and update it from there itself. Multiple people when they work on the same project they can update project details and inform other team members simultaneously.

Steps after installation-

sign up for git hub and create new repository

Create a maven project- New> Other> Maven project.

 Push the code/local repository to Git. On project right click> Team>Share- select local repository. On clicking finish, we can see the change in the project structure- ? icon.

1. Push our code to Git Hub Repository. Right-click on the project and team then click on commit. Commit and push and see icon changed in project structure.

http://www.guru99.com/selenium-github.html

<https://guides.github.com/activities/hello-world/--> Creating repository, creating branches, committing changes to github and open, merge pull request.(and finally delete branch)

**Junit**- do not perform dependency test which TestNG does.

Also Junit do not have annotations like @BeforeTest, @Beforesuite, @Beforegroups which TestNg have.

Junit has limited annotations- @Before (method), @After, @Beforeclass.

Dependency test in TestNG-

@Test(dependsOnMethods = { "initEnvironmentTest" })

@Test(groups = { "init" })

@Test(dependsOnGroups = { "init.\*" })

**Logging**- Log4j is a fast, flexible and reliable logging framework.

It has 3 principal components- Loggers, Appenders and Layout.

**Loggers** - It is responsible for logging information. Logger class is a Java-based utility.

static Logger APP\_LOGS = Logger.getLogger("devpinoyLogger");

**static** Logger *log* = Logger.*getLogger*(actionclass.**class**.getName());

*log*.info("Logging requestor");

there are five kinds of log levels- All, Debug, Info, Warn, Fatal, Error. With level as-

ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF.

**Appenders-** it is used to write the logs in file.  (select where to write- in console, file)

**Layout**- It is responsible for formatting logging information in different styles.

Step 1- Add jar files in build path

Step2- Create Log4j.property file and add properties

Step3- Create 2 files- manual.log(manually logged statements) and selenium.log (system created log)

Step 4- Import org.apache.log4j.Logger;

Step 5- Add static Logger APP\_LOGS = Logger.getLogger("devpinoyLogger") in class

Step 6- Use APP\_LOGS.debug or info or…

**Assert :--**

Assert.*assertEquals*(detailtitle, "Request detail");

*Assert.assertTrue(condition, message)*

*Assert.assertFalse(condition, message)*

**POM- Page object model.**

* **Page Object Model** is a design pattern to create **Object Repository** for web UI elements.
* Under this model, for each web page in the application there should be corresponding page class.
* This Page class will find the WebElements of that web page and also contains Page methods which perform operations on those WebElements.
* Name of these methods should be given as per the task they are performing
* **object repository is independent of testcases**, so we can use the same object repository for a different purpose with different tools.

all Web Elements of the **AUT** and the method that operate on these Web Elements are maintained inside a class file. Task like **verification** should be **separate** as part of Test methods.

**Page factory-** Page Factory is an inbuilt page object model concept for Selenium WebDriver. Page Factory is an extension to Page Object. Classes same as POM, Additionally with the help of PageFactory class we use annotations **@FindBy** to find WebElement. We use initElements method to initialize web elements. Web page classes or Page Objects containing web elements need to be initialized using Page Factory before the web element variables can be used. This can be done simply through the use of initElements function on PageFactory:

@FindBy(className="barone")

WebElement titleText;

public Guru99Login(WebDriver driver){

this.driver = driver;

//This initElements method will create all WebElements

PageFactory.initElements(driver, this);

LoginPage page = PageFactory.intElements(driver,LoginPage.class)

@DataProvider Is [**TestNG annotation**](http://www.software-testing-tutorials-automation.com/2014/04/testng-annotations-with-selenium.html). @DataProvider Annotation of testng framework provides us a facility of storing and preparing data set In method. Task of @DataProvider annotated method Is supplying data for a test method. @DataProvider annotated method must return an Object[][] with data. (2D Object array)

dataprovider is really good when you want to import data from a source(excel/db) and treat each row as a separate testcase by running on the same @test method .Result is u get testng report stating result of each row as a unique testcase

Example- We can retrieve userid and password one by one from @DataProvider annotated method –

**@DataProvider(name="LoginCredentials")**

public **Object[][]** **LoginCredentials()**

and will feed them In LogIn\_Test(String Usedid, String Pass) one by one using annotation-

@Test**(dataProvider="LoginCredentials")**

public void LogIn\_Test(**String Usedid, String Pass**){

*The Test method will run as many times as the*

*//rows in the excel –if using excel to get data (with Userid, password having different value everytime)*

**Taking screenshot-** For taking  screenshots Selenium has provided TakesScreenShot interface in this interface you can use getScreenshotAs method which will capture the entire screenshot in form of file then using FileUtils we can copy screenshots from one location to another location

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

FileUtils.*copyFile*(scrFile, **new** File("C://Users//chaman.preet//Desktop//screenshot.png"));

**how to take screen shot ONLY for failed tests.**

we need to create a class and then implement [*TestNG 'ITestListener'*](http://testng.org/javadoc/org/testng/ITestListener.html). We will have a method called *'onTestFailure'.* We will use ITestResult Interface which will provide us the test case execution status and test case name.

**Reading values and then Storing values in arraylist to use it in your test cases.**

// You got the the first rows values in the inputDataArray and you can assign this values to any variables and you can use for testing .

Like inputDataArray[3] The loop will iterate to the second row after processing the first row.

//Else you can store the entire datas in the excel sheet in an arraylist and you can process for each test cases. This will conserve CPU utilization as everytime the script won't read the excel sheet.

**Store rows and column values in Array** of size –

String cellvalue[][]=**new** String[Total\_rowcount+1][Col\_count];

Where **int** Total\_rowcount = sheet.getLastRowNum()-sheet.getFirstRowNum();

**int** Col\_count = sheet.getRow(1).getPhysicalNumberOfCells();

Or **Use Iterator** object which enable you to iterate through a list of collection until next is found.

Iterator<Row> rowIterator=sheet.rowIterator();

**while**(rowIterator.hasNext())

{

Row row=rowIterator.next();

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

**while**(cellIterator.hasNext())

{

Cell cell=cellIterator.next();

System.***out***.println("Cell content is " +cell.getStringCellValue());

}

}

Or **Using Arraylist**

ArrayList<String> list=**new** ArrayList<String>();

Iterator<Row> rowIterator=sheet.rowIterator();

**while**(rowIterator.hasNext())

{

Row row=rowIterator.next();

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

**while**(cellIterator.hasNext())

{

Cell cell=cellIterator.next();

String c=cell.getStringCellValue();

**list.add(c);**

}

}

**for**(**int** i=0;i<list.size();i++)

{

System.***out***.println("arraylist values are " +**list.get(i));**

}

}

}

**Listeners in TestNG**

**Listeners** "listen" to the event defined in the selenium script and behave accordingly. These are interfaces used in selenium to generate logs or customize the testing reports. Examples- ITestListener, ISuiteListener, IExecutionListener, IReporter.

ITestListener has following methods- **OnStart, onTestSuccess, onTestFailure, onTestSkipped, onTestFailedButWithinSuccessPercentage, onFinish.**

**Implement** ITestListener in your listener class.

package Listener\_Demo;

import org.testng.ITestContext ;

import org.testng.ITestListener ;

import org.testng.ITestResult ;

public class ListenerTest implements ITestListener

{

public void onTestFailure(ITestResult Result)

{

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

}

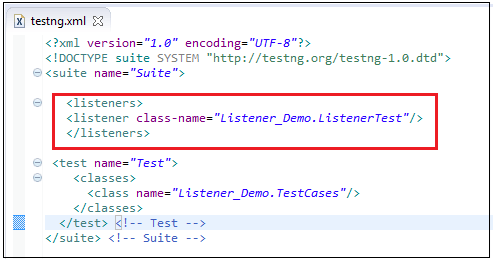
We can use this listener in our regular project class by 2 ways-

1) use Listeners annotation by- @Listeners(PackageName.ListenerClassName)

Eg- @Listeners(Listener\_Demo.ListenerTest.class)

Methods in class "ListenerTest " are called automatically according to the behavior of methods annotated as @Test.

2)In TestNG xml- If project has multiple classes, we can create a testng.xml and add listeners tag in XML. When you run this XML file, listeners will work throughout the test suite irrespective of the number of classes mentioned.



**To get parameters also in screenshotname-** result.getName()+ "-" + Arrays.*toString*(result.getParameters());

Explicit, implicit wait and Fluent wait

**Fluent wait-** FluentWait defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition.

FluentWait wait=**new** FluentWait(driver);

wait.pollingEvery(1, TimeUnit.***SECONDS***);

wait.withTimeout(60, TimeUnit.***SECONDS***);

wait.ignoring(NoSuchElementException.**class**);

Object find= wait.until(**new** Function<WebDriver, WebElement>(){

**public** WebElement apply(WebDriver driver){

**return** driver.findElement(By.*id*("dnn\_ctr474\_Login\_Login\_DNN\_plUsername\_lblLabel"));

**Implicit wait-** The implicit wait will tell to the web driver to wait for a certain amount of time before throwing an **exception** that it cannot find the element on the page. The default setting is 0. Once we set a time, the Web Driver waits for the period of the WebDriver object instance. (for the entire time the browser is open)

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

**Explicit wait-** The explicit wait is used to tell the Web Driver to wait for certain conditions (**Expected Conditions**) or the maximum time exceeded before throwing an "**ElementNotVisibleException**" exception. An explicit wait can only be implemented in cases where synchronization is needed and the rest of the script is working fine.

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

Wait.until(ExpectedConditions.*presenceOfElementLocated*(By.*id*("dnn\_ctr474\_Login\_Login\_DNN\_plUsername\_lblLabel")));

**Selenium Grid**

* Selenium Grid is used to run multiple tests simultaneously in different browsers and platforms.
* Grid uses the hub-node concept.
* The hub is the central point wherein you load your tests.
* Nodes are the Selenium instances that will execute the tests that you loaded on the hub.
* To install Selenium Grid, you only need to download the Selenium Server jar file.

**Launch hub-** Using the command prompt, navigate to the root of Machine A's - C drive, because that is the directory where we placed the Selenium Server.

On the command prompt, type **java -jar selenium-server-standalone-2.30.0.jar -role hub**

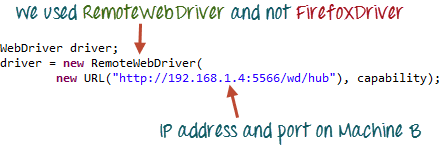
To Verify whether the hub is running—Go to browser and open <http://localhost:4444/grid/console>

To Verify if node can access hub- Go to browser and type [http://192.168.1.3:4444/grid/console where 192.168.1.3:4444](http://192.168.1.3:4444/grid/console%20where%20192.168.1.3:4444) is IP address of hub.

**Launch Node-** Using the command prompt, navigate to the root of Machine A's - C drive, and type a command.

* To run test scripts on the Grid, you should use the DesiredCapabilities and the RemoteWebDriver objects.
* DesiredCapabilites is used to set the type of browser and OS that we will automate
* RemoteWebDriver is used to set which node (or machine) that our test will run against.

[Introduction to Selenium Grid](http://cdn.guru99.com/images/DesiredCapabilities_code.png)

[](http://cdn.guru99.com/images/RemoteWebDriver_code.png)

<https://www.linkedin.com/pulse/selenium-real-time-interview-questions-answers-technologies-13600-?trk=v-feed&lipi=urn%3Ali%3Apage%3Ad_flagship3_feed%3Bg665p7Awol4wM0Bk8jNbjQ%3D%3D>

Types of assert-

Soft assert (same as verify) – will continue even if condition fails.

**SoftAssert.assertAll()method is to be used at the end to execute all statements.**

Hard assert – Will exit method if condition fails

**Changes in Selenium 3 :-**

Minimum java version is now 8+  
\* The original RC APIs are only available via the leg-rc package.  
\* To run exported IDE tests, ensure that the leg-rc package is on the classpath.  
\* Support for Firefox is via Mozilla's geckodriver.  
\* Support for Edge is provided by MS:  
\* Official support for IE requires version 9 or above  
\* New html-table runner backed by WebDriver.  
\* Unused command line arguments are now no longer parsed.

**Selenium 3.0 vs selenium2.0 :-**

* [**Selenium**](https://www.youtube.com/playlist?list=PLyGqUe6Oa_5Elc-Dv9jPzHKDx-m2GvMOd) 1.0 (IDE) + RC = Selenium 2.0

But Selenium 3.0 has removed  Selenium 1.0 but supports Selenium RC indirectly through back-end Webdriver.

* More than 9+ versions of IE are supported in Selenium 3.0.
* [**Selenium**](https://www.youtube.com/playlist?list=PLyGqUe6Oa_5EUyOz-jBZ7hiYUyQ64JHVp)3.0 has became a W3C (World wide Web consortium) standard.
* Firefox Browser version is supported by 47+. If you are working with more than 47+ either in Selenium 2.0 or Selenium 3.0 requires Gecko driver installed and configured.
* Selenium Grid bug fixes and webdriver bug fixes are done are Selenium 3.0
* 3.0 not have HeadLess Driver (HtmlWebDriver) but 2.0 have
* support for Microsoft’s Edge Browser

TestNG-

Before Suite- executed first (executed once)

Before Test- executed before class (executed once)

Before class- executed before test methods (executed once)

Before method- executed before each test method (executed multiple times)

**Hybrid framework-**

4 Packages- export excel, objects, Operation, Test cases

Export Excel>> POIexcel.java –not used

Objects>> object.properties- property file for storing objects.

url=http://google.com/

username=UserName

password=Passwod

form\_click=//div[@id='Login']

login=pinkLogBtn

Operation>>

* browser\_Selection.java—to choose browser in which we want to run scripts. (set property for each browser)
* Getobjectclass.java— to get objects to pass into find element based on input object name and type.

**public** By getObject(Properties p,String objectname,String objectType) **throws** Exception

{

//Find by xpath

**if**(objectType.equalsIgnoreCase("XPATH")){

**return** By.*xpath*(p.getProperty(objectname));

}

//find by class

**else** **if**(objectType.equalsIgnoreCase("CLASSNAME")){

**return** By.*className*(p.getProperty(objectname));

}

* Readfile.java—to read excel file –not used.
* Readobject.java—to load property file and set key pair value using set property.
* UIoperations.java—to define methods for common keywords in switch case.

public void perform(Properties p,String operation,String objectname,String objectType,String value)

* Utility.java—to capture screenshot- not used.

Testcases>>

* Screenshot.java- using ITestListener for capturing screenshot- not used.
* Module wise java classes—for defining page specific keywords in switch case.
* Hybridexecutetest.java—Data provider get the content from excel and pass it to @Test method which runs when Runmode=Y and testcasename!=null&&testcasename.length()!=0.

**public** **class** HybridExecuteTest **extends** readfile {

**static** WebDriver *webdriver*;

readfile rf=**new** readfile();

@Test(dataProvider="hybridData")

**public** **void** testlogin(String testcasename,String keyword,String objectname,String objectType,String value,String runmode) **throws** Exception

{

**if**(runmode.equals("Y"))

{

**if**(testcasename!=**null**&&testcasename.length()!=0)

{

FirefoxProfile fp = **new** FirefoxProfile();

fp.setPreference("network.proxy.type", ProxyType.***AUTODETECT***.ordinal());

System.*setProperty*("webdriver.gecko.driver", "C://Users//chaman.preet//Downloads//geckodriver.exe");

*webdriver*=**new** FirefoxDriver(fp);

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

}

Readobject robject=**new** Readobject();

Properties allobjects=robject.getobjectrepository();

UIoperations Uoperation=**new** UIoperations(*webdriver*);

Uoperation.perform(allobjects, keyword, objectname, objectType, value);

Download\_comps objdowm=**new** Download\_comps(*webdriver*);

objdowm.download(allobjects, keyword, objectname, objectType, value);

Trading\_comps objtrading=**new** Trading\_comps(*webdriver*);

objtrading.trading(allobjects, keyword, objectname, objectType, value);

UploadComps objupload=new UploadComps(webdriver);

objupload.upload(allobjects, keyword, objectname, objectType, value);

}

**else** **if**(runmode.equals("N"))

{

**throw** **new** SkipException("Test case is skipped as Run mode is N");

}

}

@DataProvider(name="hybridData")

**public** Object[][] getDatafromDataprovider() **throws** IOException

{

Object[][] object=**null**;

POIexcel file=**new** POIexcel();

XSSFSheet sheet1=file.readexcel("C://Users//chaman.preet//Documents//C data//git//compsnew//Comps\_project", "TestSteps.xlsx", "Test\_steps");

**int** rowcount=sheet1.getLastRowNum()-sheet1.getFirstRowNum();

System.***out***.println("row count is " +rowcount);

**int** col\_count=sheet1.getRow(1).getPhysicalNumberOfCells();

object=**new** Object[rowcount][col\_count];

**for**(**int** i=0;i<rowcount;i++)

{

XSSFRow row=sheet1.getRow(i+1);

**for** (**int** j = 0; j < row.getLastCellNum(); j++) {

//Print excel data in console

XSSFCell cell=row.getCell(j);

object[i][j] = cell.toString();

// System.out.println("values are " +object[i][j]);

}

}

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

**return** object;

}

**TestNG reports-**

Emailable-report.html

Index.html

**JDBC connectivity-**

1) Download sqljdbc Jar and then manually install it into your local maven repository using cmd as admin in project folder when pom.xml is present-

Command:-

mvn install:install-file -Dfile=<Jar file path> -DgroupId=com.microsoft.sqlserver -DartifactId=sqljdbc4 -Dversion=4.0 -Dpackaging=jar

In our system-

mvn install:install-file -Dfile=C:\sqljdbc\_6.0\enu\jre8\sqljdbc42.jar -DgroupId=com.microsoft.sqlserver -DartifactId=sqljdbc4 -Dversion=4.0 -Dpackaging=jar

2) Then Add dependencies in Maven-

<dependency>

<groupId>com.microsoft.sqlserver</groupId>

<artifactId>mssql-jdbc</artifactId>

<version>6.1.0.jre8</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.microsoft.sqlserver</groupId>

<artifactId>sqljdbc4</artifactId>

<version>4.0</version>

<scope>runtime</scope>

</dependency>

3) Write following commands in your script for establishing connection

//Create Connection to DB

Connection con = DriverManager.getConnection(dbUrl,username,password);

Where

dbUrl= “jdbc:sqlserver://ipaddress:portnumber/db\_name”

//Query to Execute

String query = "select \* from employee;";

//Load mssql jdbc driver

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

//Create Statement Object

Statement stmt = con.createStatement();

// Execute the SQL Query. Store results in ResultSet

ResultSet rs= stmt.executeQuery(query);

// While Loop to iterate through all data and print results

while (rs.next()){

String myName = rs.getString(1);

String myAge = rs.getString(2);

System. out.println(myName+" "+myAge);

}

// closing DB Connection

con.close();

4) Before running you should check port and ipaddress from PC>Manage > computer management>

services and applications

Go to SQL server configuration manager> SQL server network configuration> Protocols for

MSSQLSERVER.

Go to TCP/IP in right side and check properties.

In IP addresses section, enable and active all IPs and check port.

Ipaddress is localhost.

So, String dbUrl= url+servername+dbName i.e

String dbUrl = "jdbc:sqlserver://localhost:1433;serverName=EVS01TST06;databaseName=PI\_HealthEconomics";

Connection conn = DriverManager.getConnection(url+servername+dbName,username,password);

5) Now, start SQL services- Go to computer management> services and applications> Services> SQL server (MSSQLSERVER) and right click and start.

6) Run script.

**TestNG concepts-**

<http://testng.org/doc/documentation-main.html>

**When TestNG annotations are used in inheritance-**

TestNG guarantees that the "@Before" methods are executed in inheritance order (highest superclass first, then going down the inheritance chain), and the "@After" methods in reverse order (going up the inheritance chain).

MainTest which will be used to initiate the WebDriver as we need to use the same webdriver instance across our framework. So, whenever there is a need of using webdriver instance then that class will extend this MainTest. Then we can maintain a single webdriver instance across our project.