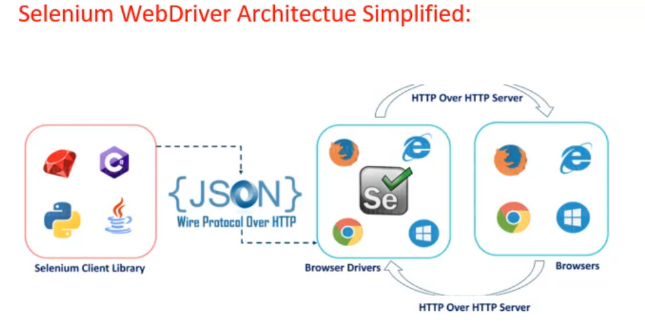
Guru99.com / tutorialsPoint.com



* Selenium code gets converted into JSON format and transferred to the browser driver (which we have downloaded and explicitly mentioned it’s path in our code; e.g. chromedriver.exe).
* That browser driver reads the JSON file and perform the action on corresponding browsers (like click, enter, etc.), gets the output/response from the browser and send it to back your selenium code (or client i.e. output result) in JSON format.
* (*JSON: JavaScript Object Notation.* ***JSON****is one of the most used Data-Exchange format. It is a light weight format represented as pure text. Due to the ability of almost all the languages to parse text, Json becomes independent of the programming language. JSON is a human and machine-readable format to represent data as****Structured Data****. JSON is used primarily to transfer data from one computer to another or even between different programs on the same computer.****Key-Value****pairs in JSON are used to represent a property of the Object* )
* We write Code in Java language in Eclipse Editor (Consider Java Project is nothing but a Test Suite and each Java class in that project is nothing but a Test Case)
* Selenium is not a tool, It’s just a jar file. In order to create a script that interacts with Selenium Server or to create local Selenium WebDriver script, we need to make a use of language-specific client driver (download corresponding JAR file from Selenium official website).
* How will Selenium code (which is written in Java language in Eclipse editor) run? For that, we to download that JAR files and add them into Java Project. That’s it.
* To do that, right click on Java project name in eclipse editor > Properties > Java Build Path > Libraries > Add External Jars. Then add the downloaded JARs (all) in your project.

**TestNG Annotations:**

@Test

* Mark the class or method as a part of Test/test case
* If priority not set, priority will be set to 0 by default to all test cases
* Methods will be executed by alphabetical orders

@Test(priority=1)

* Defined priority of test cases
* Lower number in priority, will execute test cases first

@Test(description = “this is a test method”)

* It is a description of the method

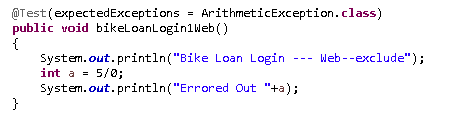
@Test(alwaysRun = true)

* This is used when we want to make sure a method always runs even if the parameters on which the method depends, fails.
* If set to true, this test method will always run.

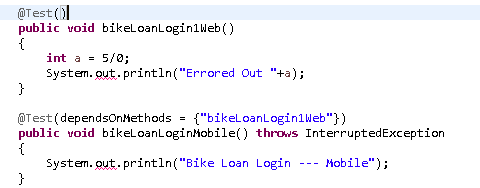
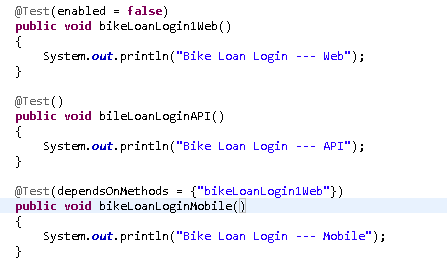
@Test(enabled = false)

* Following test case will be ignored in execution

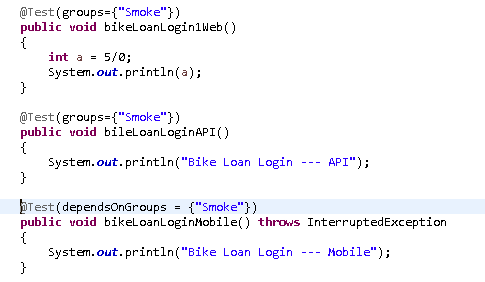
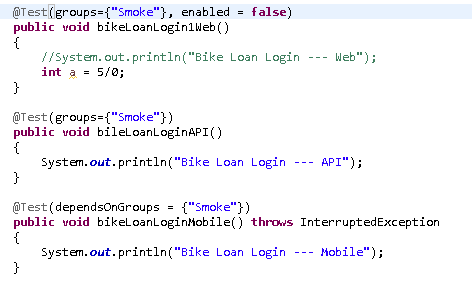
@Test(expectedExceptions = ArithmeticException.class)

* It can be used when you are expecting an error and you still want to mark the test case as Passed.
* Below test case will be executed without marked as failed. First statement will be printed but second statement will not be printed since exception has been occurred in the calculation.
* If ‘expectedException’ attribute is not added, then first statement will be printed out and this test case will be marked as Failed.
* 

@Test(dependsOnMethods = {“OpenBrowser”, “loginPage”})

* Following Method will be executed only if the methods given in {} have been executed successfully
* In below example, bikeLoanLoginMobile() will be skipped since bikeLoanLogin1Web() is Failed.
* bikeLoanLoginMobile() will be executed only if bikeLoanLogin1Web() is successfully executed
* 
* In below example, program will not be completed. It will be errored out with the error “*bikeLoanLoginMobile() is depending on method public void bikeLoanLogin1Web(), which is not annotated with @Test or not included*”
* 

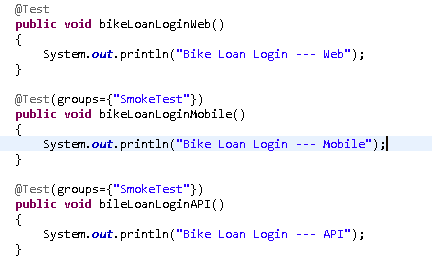
@Test(dependsOnGroups = {“HomePage”, “DepositPage”})

* Following method will be executed only if all the methods in the given groups in {} have been executed successfully
* In below example, bikeLoanLogin1Web() will be marked as Failed, bikeLoanLoginAPI() will be Passed, and bikeLoanLoginMobile() will be skipped. Third method will be skipped since it depends on group of methods and all methods did not finished successfully.
* 
* In following example, bikeLoanLoginAPI() and bikeLoanLoginMobile() both will be executed successfully and marked as Passed.
* bikeLoanLogin1Web() will be ignored in test execution
* 

@Test(groups={“SmokeTest”})

* It declares the method that belongs to same group. For all those method, give same group name
* it helps when you have to include or exclude specific test cases only in your execution
* Groups are specified in TestNG.xml using <groups> tag.
* We can add it under <suite> or <test> tag in TestNG.xml. If it defines under <suite>, it applies to all <test> tag under it. If it defines under <test>, it applies to all <classes> tag under it.





**Another example of include/exclude in TestNG:**

<suite name=*"Bank ABC"*>

<test thread-count=*"5"*name=*"Loan Department"*>

<classes>

<class name=*"com.TestNG.BikeLoan"*>

**<methods>**

**<include name=*"bikeLoanLoginMobile"*/>**

**</methods>**

</class>

<class name=*"com.TestNG.HomeLoan"*>

**<methods>**

**<exclude name=*"homeLoanLoginAPI"*/>**

**</methods>**

</class>

<class name=*"com.TestNG.CarLoan"*/>

</classes>

</test>

</suite>

@Test(invocationCount = 5)

* It will refer that the how many times method will be invoked. it’s kind of loop.
* Here, method will be executed 5 times

@Test(invocationCount = 5, invocationTimeOut=3000)

* It shows the maximum milliseconds method should take for allinvocationCountto complete
* invocationTimeOut attribute will be ignored if invocationCount is not given
* if execution of all invocationCount is not finished in the specified invocationTimeOut, then test case/method will be marked as Failed.

@Test(dataProvider = “dataProviderName”)

* dataProvider is used to provide any data for parameterization
* generally, DataProvider resides in the same class where test method is or its base class

@DataProvider

* Marks a method as supplying data for a test method.
* The annotated method must return an Object[ ][ ], where each Object[ ] can be assigned the parameter list of the test method.
* The @Test method that wants to receive data from this DataProvider needs to use a dataProvider name equals to the name of this annotation.
* It is explained in separate topic

@BeforeSuite / @AfterSuite

* following method will be executed only once before/after all test execution in the suite
* e.g. WebDriver can be starts/initialize in BeforeSuite method and stops in AfterSuite method
* add more e.g. what we can do here???

@BeforeClass / @AfterClass

* @BeforeClass: following methods will be executed only once before a first test method in the current class
* @AfterClass: the annotated method will be run only once after all the test methods in the current class
* e.g. set up webdriver capabilities , driver timeouts in beforeclass and close driver in after class

@BeforeMethod / @AfterMethod

* method will be executed before / after of every test method.

@BeforeTest / @AfterTest

* method will be executed before / after a test(module)
* note: here ‘Test’ (in ‘BeforeTest’/ ‘AfterTest’) is not a test method. It’s a module / Test which is defined in TestNG.xml under <Test>…</Test> tag.

@BeforeGroups

* The list of groups that this configuration method will run before.
* This method is guaranteed to run shortly before the first test method that belongs to any of these groups is invoked

@AfterGroups

* The list of groups that this configuration method will run after.
* This method is guaranteed to run shortly after the last test method that belongs to any of these groups is invoked.

@Parameters

* Describes how to pass parameters to a @Test method.
* It is explained in separate topic

@Listeners

* Defines listeners on test class

@Factory

* Marks a method as a factory that returns objects that will be used by TestNG as Test classes. The method must return Object[ ].

**Sequence of @annotation execution in TestNG:**

@BeforeSuite – executes method before invoking/execution of testing suite

@BeforeTest – executes method before invoking all classes methods mentioned under <test> tag in testing.xml

@BeforeClass - executes method before invoking all classes methods mentioned under <class> tag in testing.xml

@BeforeGroups – executes method before invoking a specific (mentioned) group

@BeforeMethod - executes method before invoking all methods in the class

@Test

@AfterMethod

@AfterGroups – Once all methods are executed for a mentioned group, this method gets executed

@After Class

@AfterTest

@AfterSuite

**Parameterization**:

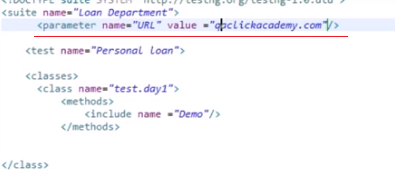
How to handle parameterization in Selenium? Or how to build data-driven framework?

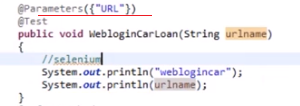
There are two methods. 1) using parameterization in TestNG.xml and 2) the dataProvider annotation.

(more methods like 3) using separate file .config 4) store parameters and values in .xls file)

1) using parameterization in TestNG.xml

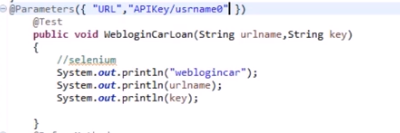
* Parameter can be defined in TestNG.xml as shown on below and that can be called in your scripts.
* If parameter is defined in the <Suite> tag, that parameter can be accessed by any method of the suite
* If the parameter is defined in the <test> tag in TestNG.xml, then it can be accessed by any class method mentioned in the <test> tag
* If The Same parameter name is declared in both places; test level parameter will get preference over suit level parameter.





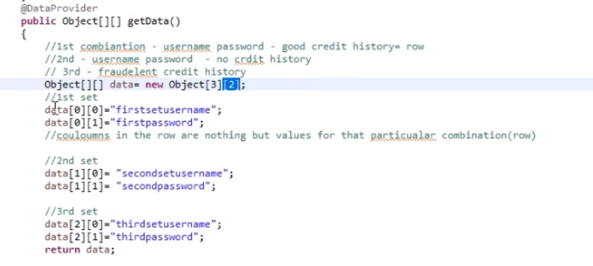
* Here, you have to create the variable into the method declaration to catch the parameter value coming from testng.xml
* You can define & access multiple parameters

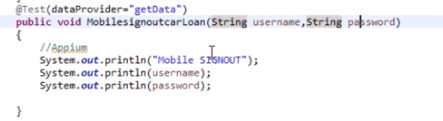


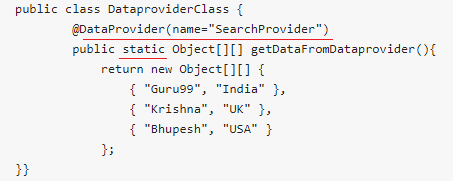
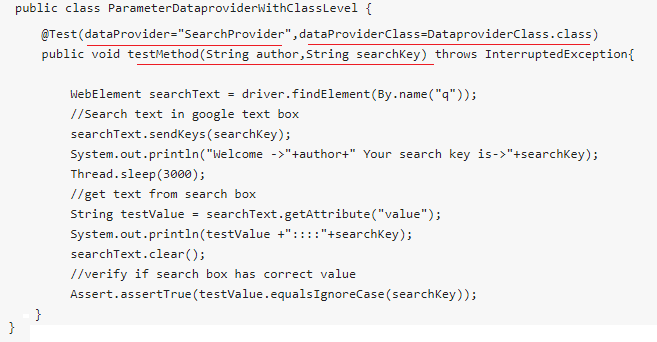
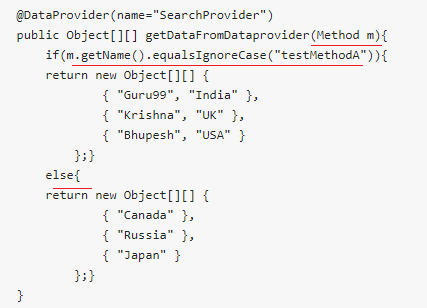
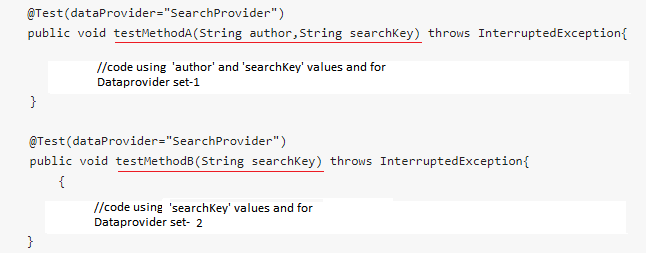
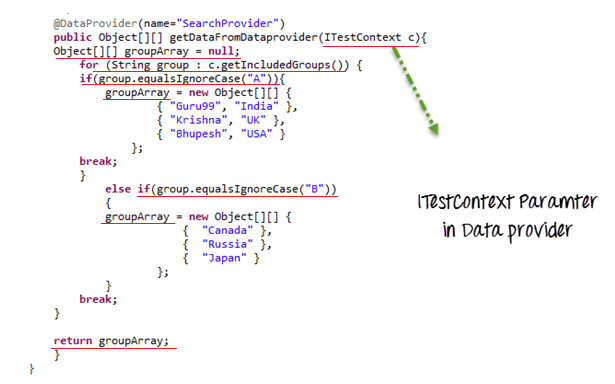
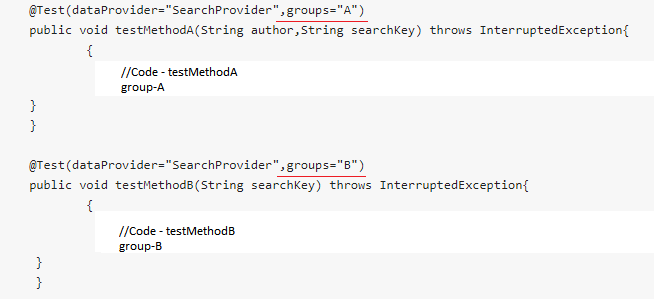


2) using DataProvider annotation

* If you have set of data and you have to feed it to test cases multiple times (loop into same test case), we can use @DataProvider annotation
* Method followed by @DataProvider contains set of data which can be invoked/used in another method
* The annotated method must return an Object[ ][ ], where each Object[ ] can be assigned the parameter list of the test method.
* using @Parameter+TestNG.xml only one value can be set at a time, but @DataProvider return an 2d array of Object.

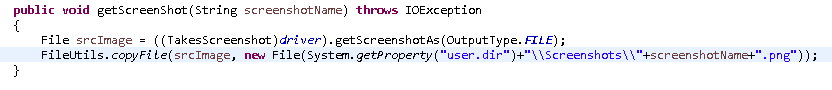


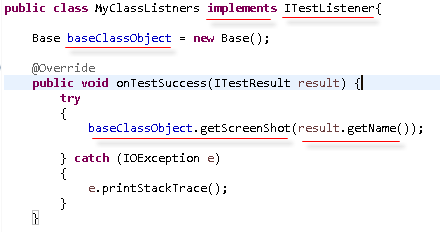
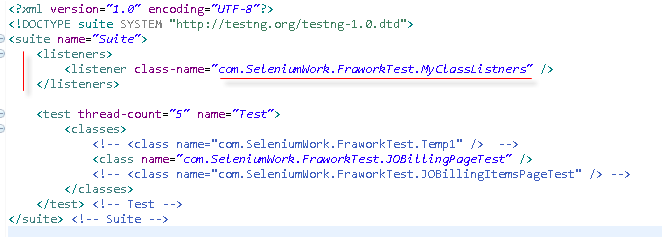
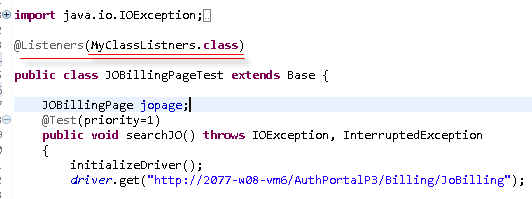


* It has only one attribute 'name'. If you do not specify the name attribute then the DataProvider's name will be same as the corresponding method name.
* if the DataProvider returns an array of 2\*3 objects, the corresponding testcase will be invoked 2 times with 3 parameters each time.
* We can invoke DataProvider from different Class also; as below:
* To put it in some other class we need to make **data provider method as static** and in test method we need to add an attribute **dataProviderClass** in**@Test** annotation
* 
* 
* Types of parameters in DataProvider methods:
* There are 2 types of parameters supported by DataProvider methods
* **Method**: if same dataProvider behaves differently with different test methods, use Method parameter
* 
* 
* **ITestContext**: It can be used to create different parameters for test cases based on groups
* In real life, you can use ITestContext to vary parameter values based on Test Methods, hosts, configurations of the test
* In the following code example, We have 2 groups A & B
* Each test method is assigned to a group
* If value of group is A, a particular data set is returned
* If value of group is B, another data set is returned
* 
* 

**Listeners:**

* TestNG manages everything thru Suite, Test & Methods.
* And Listeners gives us the ability to act before and after of every Suite, Test & Methods
* There are many Listeners in TestNG like IMethodInterceptor, IReporter, ISuiteListener, ITestListener, IExecutionListener, IHookable, IAnnotationTransformeretc
* Listener is defined as interface that modifies the default TestNG's behavior.
* It used to create customizedTestNG reports or logs
* ISuiteListener has two methods:
  + OnStart() – Before TestNG picks up your suite for execution, it first makes a call to onStart() method
  + OnFinish() -  it again makes a call to onFinish() method after a suite has been run.
* IInvokedMethodListener has two methods:
  + beforeInvocation()- Invoke before each method
  + afterInvocation() - Invoke after each method
* ITestListener has following methods
  + **OnStart**- OnStart method is called when any Test starts.
  + **onTestSuccess**- onTestSuccess method is called on the success of any Test.
  + **onTestFailure**- onTestFailure method is called on the failure of any Test.
  + **onTestSkipped**- onTestSkipped method is called on skipped of any Test.
  + **onTestFailedButWithinSuccessPercentage**- method is called each time Test fails but is within success percentage.
  + **onFinish**- onFinish method is called after all Tests are executed.
* This listener is implemented throughout the test suite irrespective of the number of classes you have. When you run this XML file, listeners will work on all classes mentioned.
* You can also declare any number of listener class.
* Create a getScreenShot() method in Base class



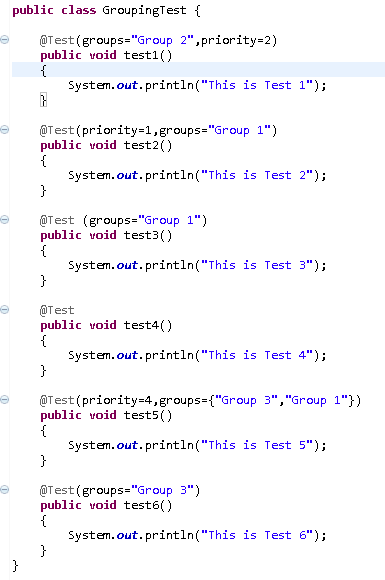
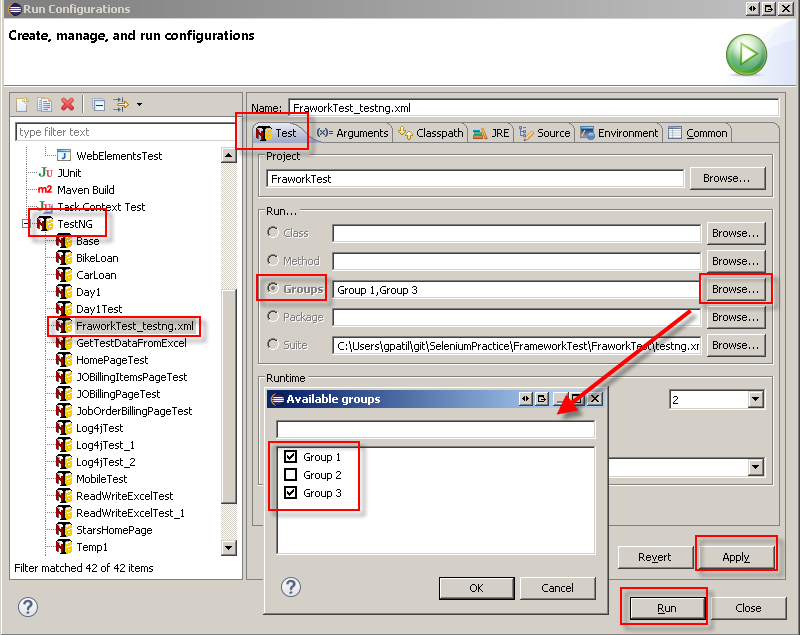
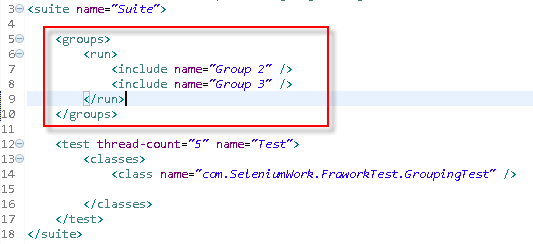
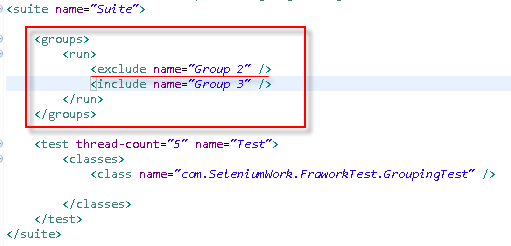
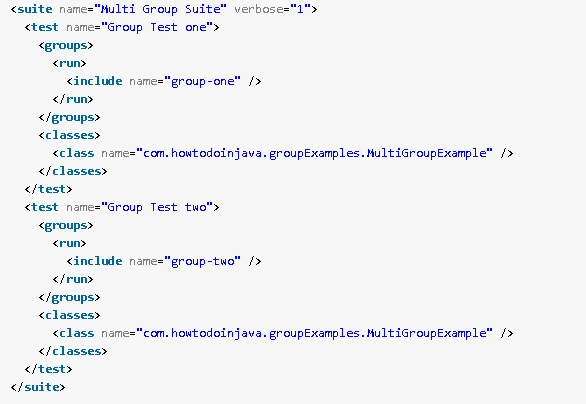
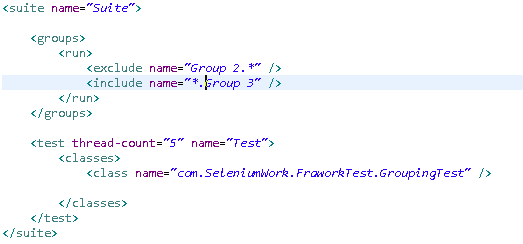
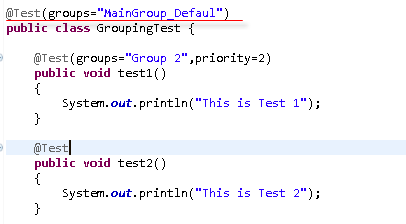
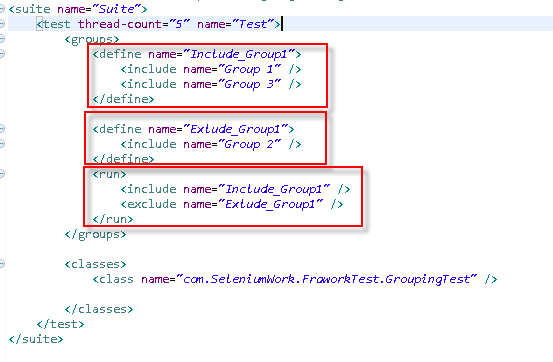
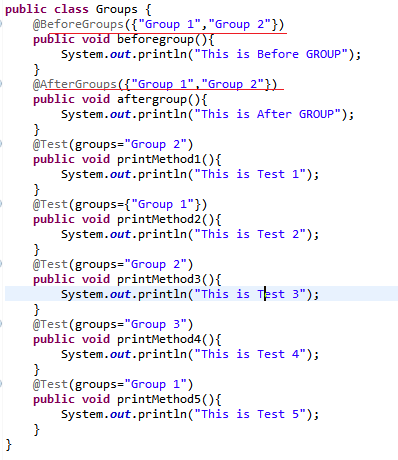
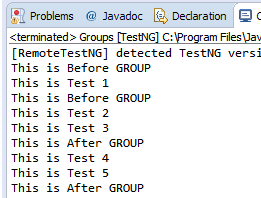
* Create a yourListener class in main package which implements ITestListener interface
* 
* Create Base Class object and invoke getScreenShot() wherever you want to do. Here I have invoked in onTestSuccess method.
* result.getName() gives you the @Test method name where this listener is getting called.
* We are passing @Test method name to getScreenshot()so that we will know for which test cases that screen shot has taken.
* Now we have to apply the Listeners for required test cases/classes; for that we have to add it in TestNG.xml file
* 
* Define the <listener> tag under the suite tag; meaning it will applies to all the classes and test methods under that suite.
* Note that screenshot’s extension should be .PNG.
* If you have to apply Listener for a class without testng.xml (i.e. you can do it using @Listeners annotation)
* Wherever you want to call Listener, give @Listeners annotation just above your Test Class.
* And provide Listener class file (with .class)
* 

Note: You can also take screenshot using “aShot” WebDriver screenshot utility.

For that you have to download corresponding JAR file and add it into Project.



**TestNG Groupings:**

* It is used to segregate different test methods into certain groups (based on functionalities/features etc.);
* so that you can include or exclude certain groups in/from the execution
* 
* Define the groups in @Test annotation
* You can run TestNG groups through Eclipse settings (this just FYI)
* Goto eClipse menu > Run > Run Configuration > Select ‘TestNG’ from left side window > Select your Project > Select ‘Test’ tab > Select ‘Groups’ radio button > Browse the groups and select the groups which you wanted to execute > click Apply and Run.
* 
* You can run TestNG groups through testng.xml too
* You can define groups under **<suite>**tag **OR <test>**tag in testng.xml
* 
* When you run this testng.xml, it will execute ‘Group 2’ and ‘Group 3’ test methods only
* 
* Here ‘Group 2’ will be excluded from the execution and ‘Group 3’ will be included in the run
* You create <groups> tag under <test> and also can be added as under multiple classes as below
* 
* We can also use regular expression for groups in testing.xml as below:
* 
* Here include the groups whose name ending with ‘Group 3’ and exclude the group whose name is starting with ‘Group 2’. (if the group names are defined as e.g. ‘Group 2 – logoutPage’ and ‘Imp Group 3’)
* We can also use as ‘\*.Group.\*’ expression
* Assigning Default Group:
* we can assign a Default group to all public methods of a class
* for that we have to define group above the class name as shown below so automatically all the public methods of the class will become part of the group
* 
* Group of Groups (MetaGroups):
* We can create a group out of existing groups and can be used in <test>
* 
* You can use @BeforeGroups and @AfterGroups annotation to execute set of code before invoking certain groups methods. See below:
* 
* Output:
* 

**TestNG.xml – Include/exclude methods (ADD MORE DETAILS HERE):**

<suitename=*"Bank ABC"*>

<testthread-count=*"5"*name=*"Loan Department"*>

<classes>

<classname=*"com.TestNG.BikeLoan"*>

**<methods>**

**<includename=*"bikeLoanLoginMobile"*/>**

**</methods>**

</class>

<classname=*"com.TestNG.HomeLoan"*>

**<methods>**

**<excludename=*"homeLoanLoginAPI"*/>**

**</methods>**

</class>

<classname=*"com.TestNG.CarLoan"*/>

</classes>

</test>

</suite>

**Maven:**

* Apache Maven is Software Project Management and Build Management tool for JAVA Framework (so, it can be Java Projects, Selenium automation code in Java, Rest Assured automation using Java or Mobile automation code in Appium-Java)
* Why Maven:
  + Central Repository to get Dependencies:

Maven maintain all dependency files / jars which are required in Java framework building. Using just few line of xml code in pom.xml, required dependencies can be auto-downloaded in a project folder. You don’t need to explicitly download and maintain those jar files.

* + Maintain common structure across the organization:

Maven has many project folder templates. So while creating a project, you can choose template according to your project or organization so that appropriate folder structure will get created. We don’t need to create any different folder structure our own; because of these consistency can be maintain across the bigger projects or organization

* + Flexibility in integration with CI tool:

It supports and easy to integrate with CI tool, Jenkin.

* + Plugins for Test Framework execution:

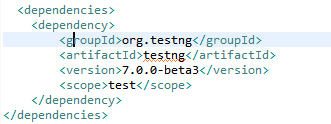
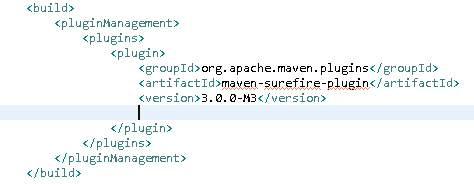
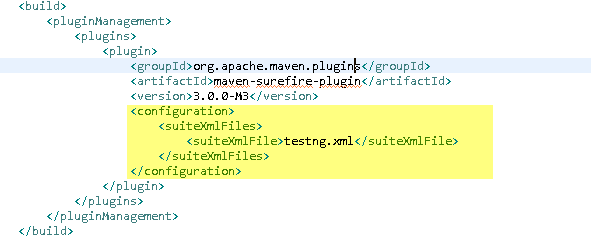
It supports different Test frameworks like TestNG, JUnit. And it supports different plugins accordingly.

* Generally Maven gets installed with newer version of eClipse; otherwise we can add Maven into eClispe thru eClipse> Help >eClipseMarketPlace feature
* Same, you can add TestNG into eClipse
* Also, you have to download a ready-made binary distribution archival file from <https://maven.apache.org/download.cgi> and follow installation instructions given <https://maven.apache.org/install.html>
* Unzip the downloaded file in your dedicated folder
* Ensure JAVA\_HOME environment variable is set and points to your JDK installation
* Add the bin directory of the created directory apache-maven-3.6.0 to the PATH environment variable

C:\Gopal P\eClipse\eclipse\apache-maven-3.5.4\bin

* Create new MAVEN\_HOME environment variable and set the path before bin

C:\Gopal P\eClipse\eclipse\apache-maven-3.5.4

* Just to check you can run the command c:\>mvn --version
* You will get the maven version
* You can also see ‘Maven’ option in eClipse> Windows > Preferences
* You can find Local repositories path at eClipse>Windows >Preferences > Maven > User Settings > Local Repository
* When Maven program runs, it checks all required dependency files under local repository and use them for execution; otherwise, it will download from Maven Repositories to this local folder.
* We need to add dependencies in pom.xml which will pull required jars from maven repository
* <groupId>:groupId will identify project uniquely across all the projects. It is the project folder or main path where maven look for that dependency file at it’s server to get downloaded. Also, it’s a Package Name
* <artifactId>: it is a file, usually a JAR, under the main folder/path where maven look for that dependency file at it’s server to get downloaded. Also, it’s Project Name.
* When you go in Eclipse Java Project and menu ‘Windows’ > Preferences; you should see ‘Maven’ in the list; meaning this Eclipse version supports Maven.
* When you goto‘Windows’ > Preferences> Maven > User Settings, you can see Local Repository path. It is the path/place of your local machine where JARs will get downloaded by default.
* Add dependencies under <dependencies> tag
* 
* Test Class name (under src/test/java) should include “Test” keyword at the end of their name
* Surefire plugin identify the executable tests by the keyword “Test” and execute them
* We have to add Surefire plugin in pom.xml along with other dependencies.
* 
* When you run the command from Project Folder :/>mvn test , it will execute all the test classes which are in src/test/java folder
* If we have to execute TestNG.xml thru maven, then we need to add such configuration in Pom.xml <https://maven.apache.org/surefire/maven-surefire-plugin/examples/testng.html>
* 
* When :/>mvn test command will run, it will execute the testing.xml
* If there is any version error of compiler source of compiler target, we can add following two lines in pom.xml under <Properties>

<properties>

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

<maven.compiler.source>1.6</maven.compiler.source>

<maven.compiler.target>1.6</maven.compiler.target>

</properties>

* If there are multiple testng.xml created; let’s say one for Regression and other for Smoke Test and based on requirement, we have to execute specific testng.xml then we can create different **Profiles** in pom.xml and execute it accordingly.
* 
* So if you want to run only Smoke test, you can execute this command :/>mvn test –PSmoke

Where ‘–P’ stands for profile and ‘Smoke’ is profile name

* If you have to run single test class, then use following command

:/>mvn –Dtest=MobileTest test

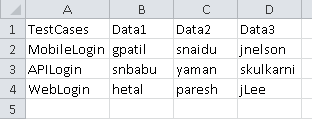
Where, ‘MobileTest’ is a class name which will be executed.

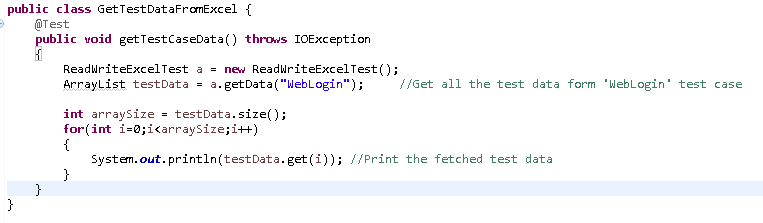
* Note that, ‘Test’ word should be there is classname so that maven willpickup that class for execution. E.g. MyClassTest.java
* :\>mvn clean 🡺 to clean the previous builds/references/temporary files
* :\>mvn compile => to compile the mvn project & check for syntax errors if any
* :\>mvn test => to execute the mvn project/class. Program gets compiled automatically when you run the program
* When you run these above commands, you have to run it from project folder. E.g. C:\my test\myproject:\>mvn test
* GroupID: GroupId uniquely identifies your project across all projects. A group ID should follow Java's package name rules. This means it starts with a reversed domain name you control.if the current project is a multiple module project, it should append a new identifier to the parent's groupId. E.g. org.apache.maven, org.apache.maven.plugins, org.apache.maven.reporting
* ArtifactID:ArtifactID is the name of JAR.
* ArcheTypeArtifact: Archetype is Maven project templating toolkit. Archetype helps to create a maven project template with certain project folder structures, parameters etc. e.g. ‘maven-archetype-quickstart’ archetype creates sample maven project where it creates project folder including ‘src’ subfolder and pom.xml with certain parameters. ‘src’ folder contains ‘main’ and ‘test’ subfolders.
* Surefire plugin: It used during test phase. It executes tests which are present in src/test/java folder. Maven Surefire plugin is used to follow sequence of the test in testng.xml file. If we don’t include it in pom.xml, then it will execute all the test cases which has prefix or suffix as ‘test’ and these test will execute without any sequence.

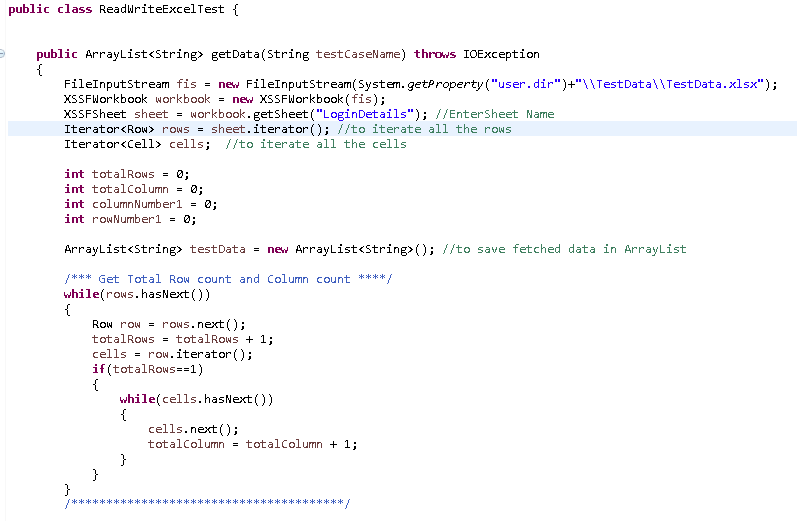
**Excel File (Data Driven – Read Data from Excel):**

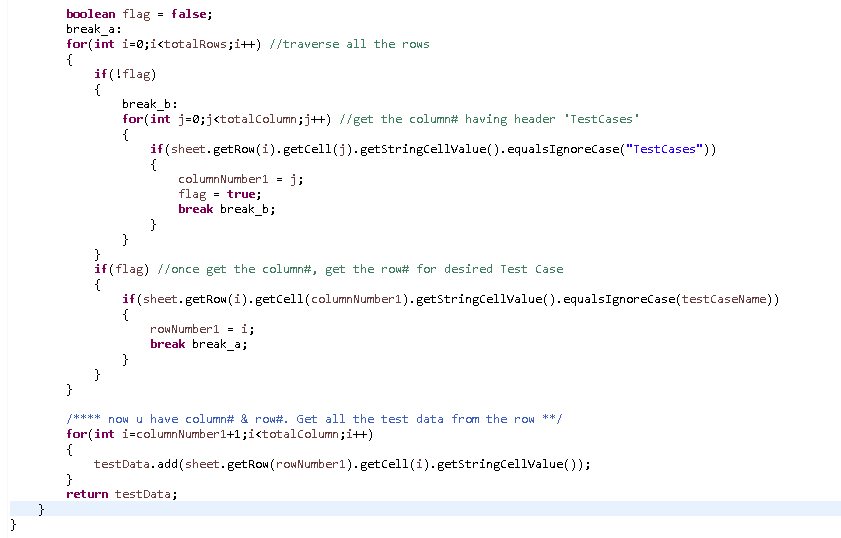
Write a general method which will fetch all test data for required Test Cases.

Excel File

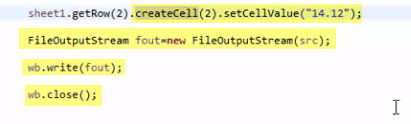




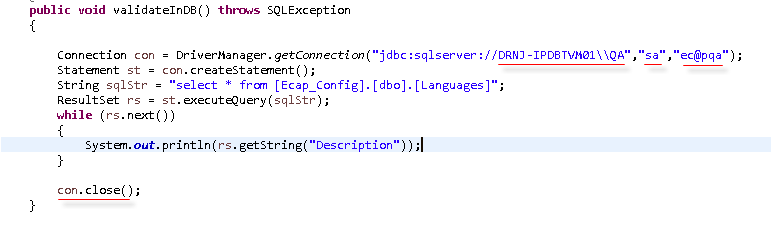
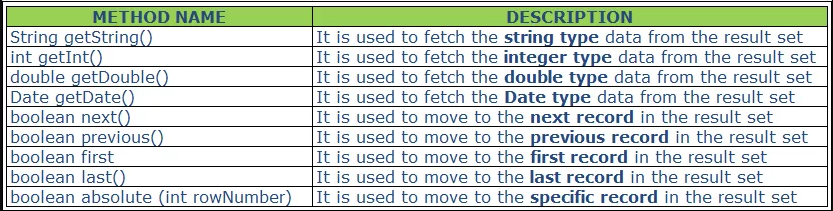




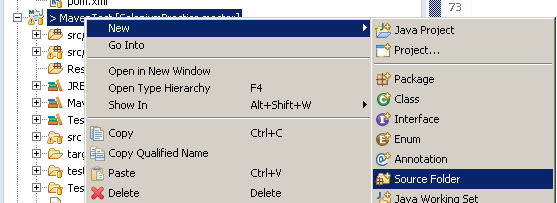
To Write the data into Excel, “setCellValue()” can be used as below. Also, you need create FileOutputStream object.

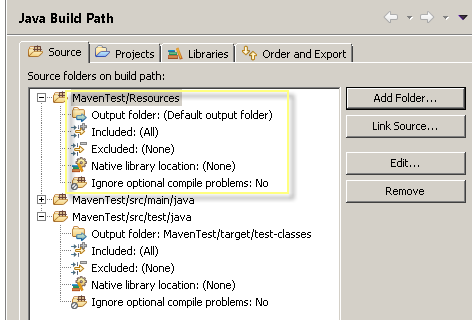


**Database Testing:**

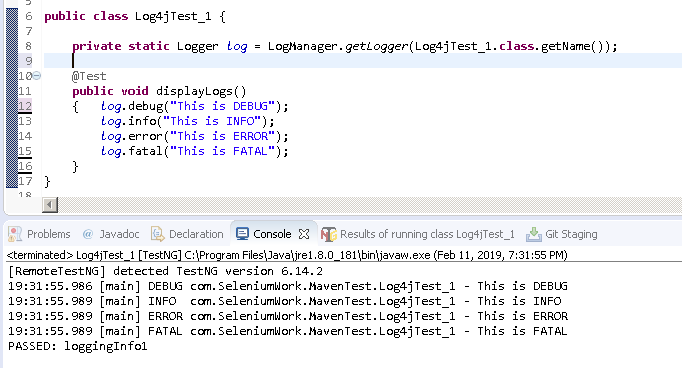
* JDBC (Java Database Connectivity) is SQL level API that provides connection between Java programming language and the database
* Using JDBC API, we can access virtually any data sources like any relational databases, spreadsheet or flat files
* It allows us to connect DB, send query to DB and execute it, fetch the data and process the result
* JDBC API provides below classes and interfaces:
  + DriverManager
  + Driver
  + Connection
  + Statement
  + ResultSet
  + SQLException
* For that, we need to add maven dependency ‘Microsoft JDBC Driver for SQL Server’ in pom.xml
* Or download the corresponding jar and add that jar in the project
* 
* Above example will display all the values from ‘Description’ column of [Languages] table.
* List of common methods to process Result:
* 

**Apache Log4j:**

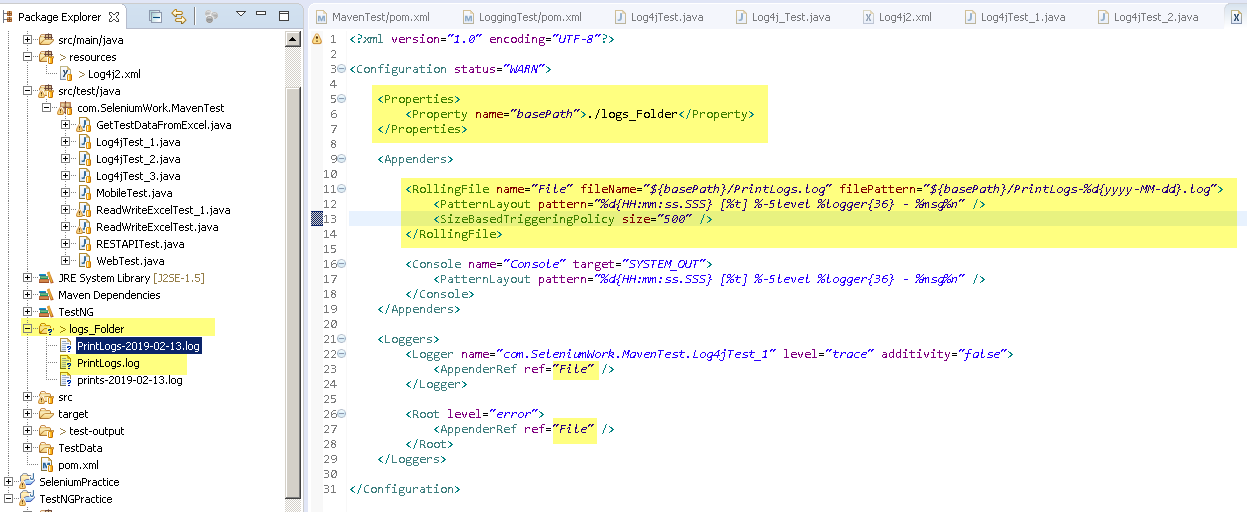
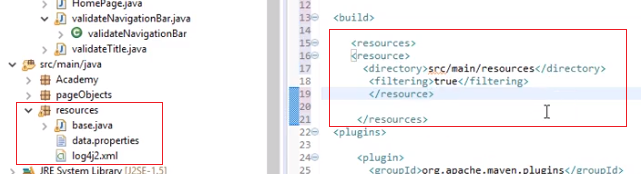
* It’s logging framework API which is used to log the output/activity/message in a log file
* It’s reliable, fast & flexible. We can generate error only logs, success logs with timestamp. Also you can save previous days log
* For that, you need to download Log4j Jars from Apache website or add corresponding dependencies in pom.xml. “log4j-api” and “log4j-core”.
* To configure the logging style/method, we have to define a configuration file. For that –
* Create a Source Folder ‘resources’ under the project
* 
* Create XML file under it. Right click on the folder > New > Other > XML > XML File
* NOTE: xml file name should be “**log4j2.xml**” or “**Log4j2.xml**”
* Copy the XML tags from <https://logging.apache.org/log4j/2.x/manual/configuration.html> and paste it into the created xml. i.e. the basic log4j configuration.
* Now understand the XML, there are 2 sections. 1) Appenders 2) Loggers
* Appenders section define the “where to keep the logs” and “how to keep the logs (format/pattern)”. i.e. it is used to write the logs in file/console.
* <Loggers> section defines, “what to log”
* Into the Loggers section, there is <Root> tag which defines what to log for all the classes into the project. E.g. either ERROR, DEBUG, FATAL or INFO. TRACE means log everything.
* There is one more tag <Logger> which tells you what to log for a particular class.
* Into the below snapshot, logs will be displayed at “Console”. Errors will be logged for all classes; however ‘trace’ (mean all type of logs) will be logged for ‘com.SeleniumWork.MavenTest.Log4jTest\_1’ class. ‘additivity’ attribute tells that don’t repeat the logs for this class. So it will not consider the logs from <Root> tag.
* 
* NOTE: after adding the log4j2.xml, you have to add it into the Build Path of the Project (if it’s not added into it). Right click on the Project > Build Path > Configure Build Path > Source > Add Folder > select Resources folder and add



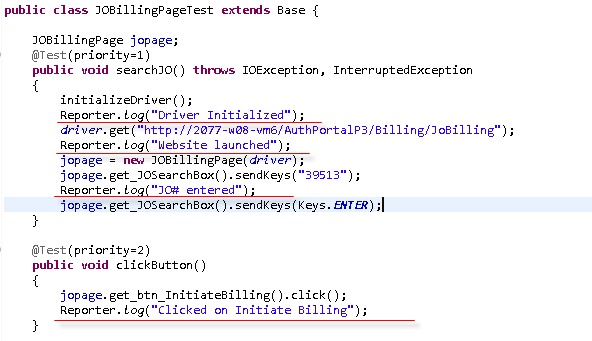
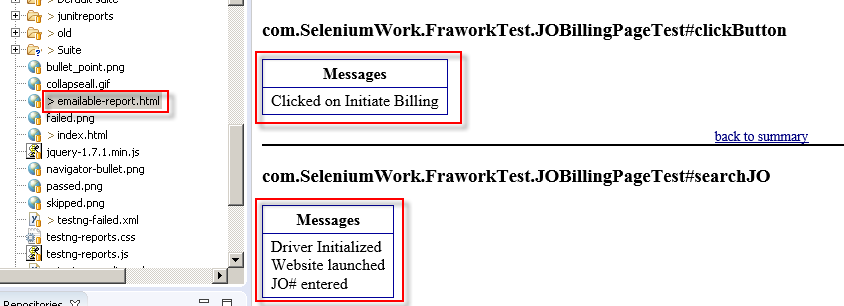
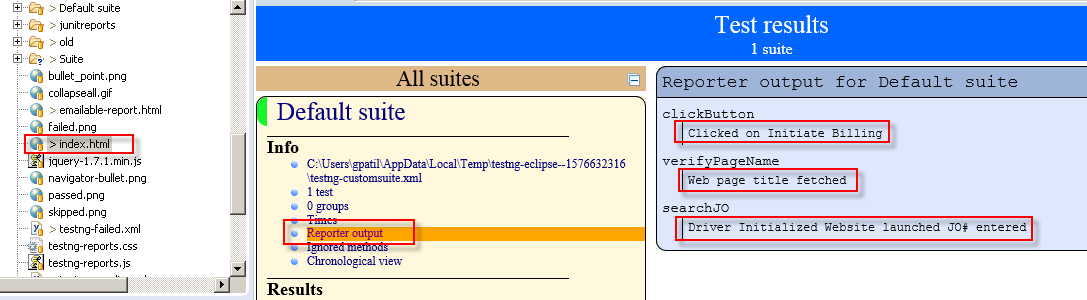
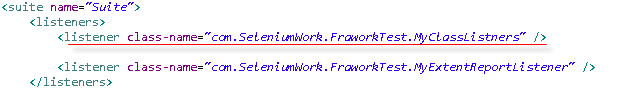
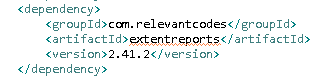
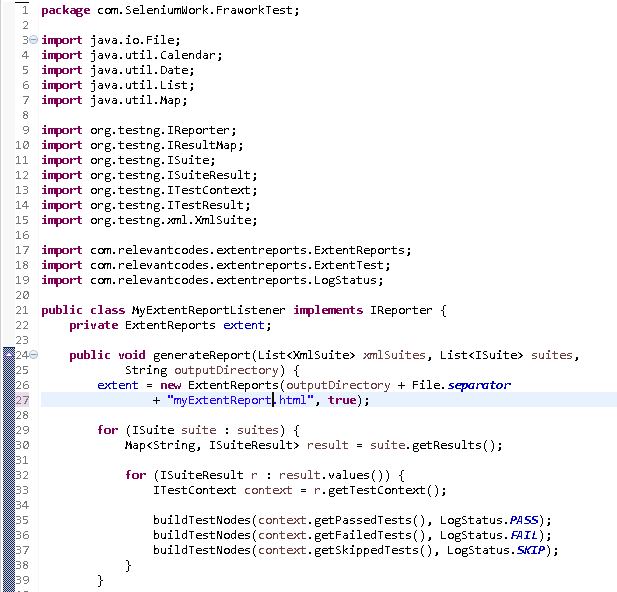
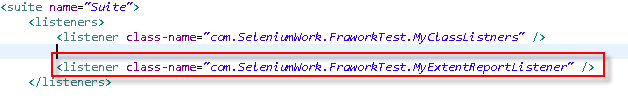
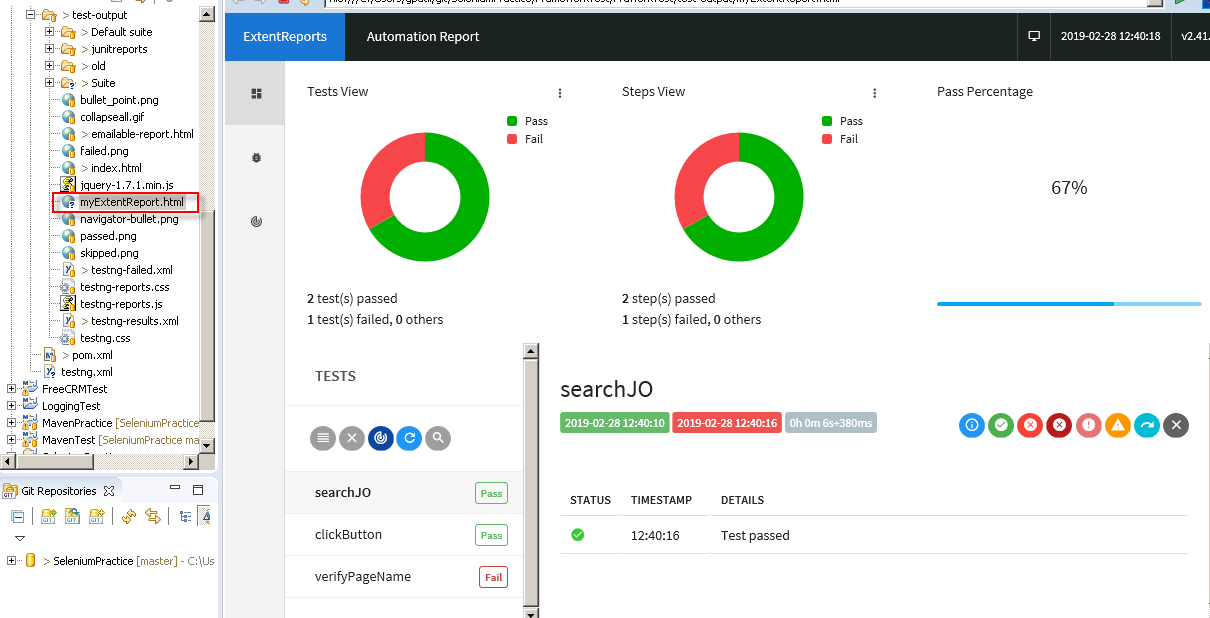
* We have to create instance of Logger class.
* Logger class is java based utility implemented for log4j; which has all generic corresponding methods



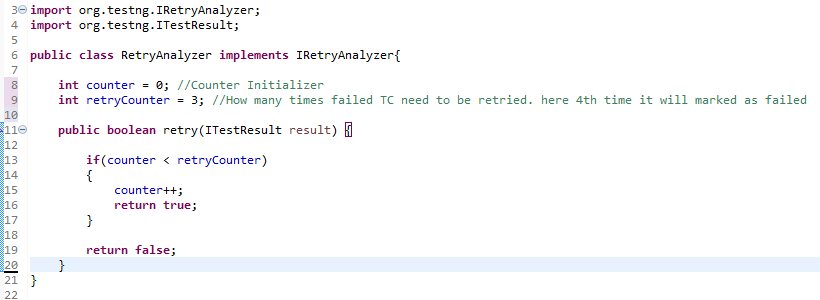
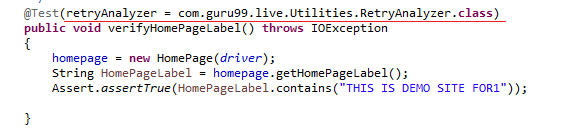
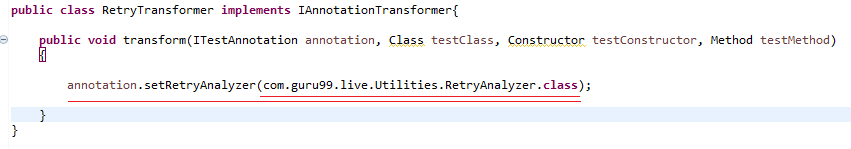
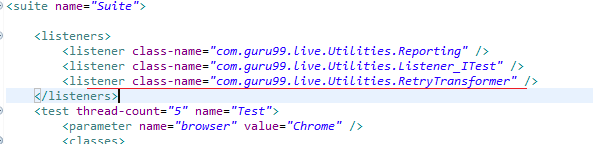
NOW save the logs in a file (instead of console):

* For that, you have to add <RollingFile> tag section into log4j2.xml file (under Appenders)
* Also, change AppenderRef to ‘File’ in <Loggers> section
* 
* Create a Folder in the Project and create a log file. As I created here ‘logs\_Folder’ folder and ‘PrintLogs.log’ file
* We define log file path into <Property> tag in log4j2.xml
* ‘filePattern’ attribute is for defining file name format
* ‘SizeBasedTriggeringPolicy’ attribute defines the file size i.e. here new log file automatically be generated once file size become 500 bytes. New file name will be given based on ‘filePattern’ attribute.
* When to Use log. Error, debug and info methods in Selenium test cases:
  + Log.Error() : to log when elements are not displayed in the page or if any validations fail
  + Log. Debug() : When each Selenium action is performed like click, SendKeys, getText()
  + Log.info() : When operation is successfully completed ex: After loading page, or after any successful validations. It’s just counterpart to log. Error()
* Now, when you run the project thru maven, POM.xml should know about this log4j2.xml otherwise logging will not be happened. We need to configure log4j2.xml info into pom.xml
* 
* To configure it in pom.xml, add <resources> section and add the directory path in the project where log4j2.xml is located.
* Highlighted code in pom.xml tells that consider the files which are located into the directory Projectfolder/src/main/resources while execution.
* Since log4j2.xml is located there, it will be executed and logs will be saved.
* **NOTE:**
  + You can generate the logs in console with just Logger interface (i.e. without defining log4j2.xml file); However, Only ‘Error’ and ‘Fatal’ logs will be displayed. Even though you have added statements to log ‘Info’ and ‘warn’ logs, it will not be displayed.
  + That’s why we have to created ‘log4j2.xml’ file.
  + Also, there are multiple settings can be done there like format of the loggings, log type, log location (i.e. where to log, how to log and what to log)
  + Above mentioned <resources> directory need to be mentioned in POM.xml because when you run the project from Jenkins or as a suite (testng.xml), then ‘log4j2.xml’ file which is saved under ‘resources’ folder will be considered.
  + Otherwise, Only ‘Error’ and ‘Fatal’ logs will be displayed.

**TestNG Reporting:**

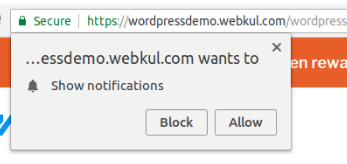
* When you execute TestNG suite (using testng.xml), TestNG generates ‘test-output’ folder at the root of the project. It contains two types of reports
* Index.html: this is the complete report of current execution which contains information like an error, groups, time, reporter logs, testing XML files
* Emailable-report.html: this is the summarize report of the current test execution which contains Test Case message in green (for passed TCs) and red (for failed TCs)
* Note that, if you run the project from maven commands, maven creates ‘target’ folder at the root of the project and it contains reports in ‘surefire-reports’ sub folder.
* What is Reporter class?
  + You can save system generated logs or user generated log in an external file; but there is one more simple way to generate log information i.e. using Reporter class
  + It’s present in TestNG and provided 4 methods to store log info:
    - Reporter.log(String s);
    - Reporter.log(String s, Boolean logToStandardOut);
    - Reporter.log(String s, int level);
    - Reporter.log(String s, int level, Boolean logToStandardOut);
  + When you use Report.log into the Test cases, log messages will be displayed into TestNG generated reports i.e. Index.htm and Emailable-report.html
  + 
  + 
  + 
* There are three ways to generate the report
* Using ITestListener interface: It will generate the report in console or a file. Here we are creating report in console.
* 
* And hook that listener in testing.xml
* 
* Using ReportNG:
* **Using Extent report**
* For that we need extent report jars. Add extentreports dependency in pom.xml
* 
* Create a Listener class in src/main/java for extent report
* 
* 
* Or get this class from <https://github.com/naveenanimation20/PageObjectModel>
* Now, add the Listener in testing.xml so that report will be generated at the end
* 
* That’s it.
* When you run the testng.xml, report will be generated at the project root in ‘test-output’ folder.
* Here ‘myExtentReport.html’ is generated, this name we have given in above code.
* 
* Also, you can add the failed test cases screenshots into the extent report.
* For this, do following changes
* You can generate PDF report out of extent report which is created in ‘test-output’ root folder
* <https://www.guru99.com/pdf-emails-and-screenshot-of-test-reports-in-selenium.html>
* You can send the report thru email as well
* FYI: You can also get another logic at <https://www.swtestacademy.com/extentreports-testng/>

**Retry / Re-run Failed Test Cases:**

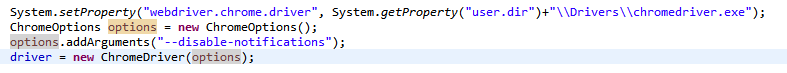
* Use followings ‘IRetryAnalyzer’ interface, ‘IAnnotationTransformer’ interface
* There are three ways to rerun failed test cases:
* **Using testng-failed.xml:**
* When we run test cases using testng.xml, it auto-creates ‘testng-failed.xml’ in “test-output” folder for Failed Test Cases. This xml contains only failed test cases.
* We can right click on the ‘testng-failed.xml’ and Run As >TestNG Suite as usual. It will execute only failed test cases.
* **Using ‘IRetryAnalyzer’ interface:**
* 
* Step1: Create your RetryAnalyzer class which implements “IRetryAnalyzer” interface
* IRetryAnalyzer interface has only one method i.e. “retry()” which returns true or false. If method is failed and needs to re-execute, it will return true. And if method is failed and do not need to re-execute, it will return false.
* Step2: If you want to execute specific test cases multiple time if it is failed,provide below parameter in @Test annotation to the particular test case
* retryAnalyzer = <packageName>.<RetryAnalyzerClassName.class>
* Here, if ‘verifyHomePageLabel()’ fails, it will re-executed 3 times and at 4th time it will be marked as failed.
* 
* **Using ‘IRetryAnalyzer’ interface and ‘IAnnotationTransformer’ interface:**
* Step1: Same as above Step1. i.e. Create your RetryAnalyzer class which implements “IRetryAnalyzer” interface. Add the same code as above step1 screenshot.
* Step2: Create your Listener class as below which implements ‘IAnnotationTransformer’ interface
* ‘IAnnotationTransformer’ interface has only one method ‘transform()’ where we tell the class name in which retry logic is added.
* 
* Step3: Add above Listener class in TextNG.xml so that whichever test cases will fail, that will be re-executed multiple times before marking it as Failed.
* 

**Others:**

* How to handle Browser’s (chrome) notifications??



🡺



More about ChormeOptions:

* It is a Class introduced in Selenium after 3.6.0 version (earlier DesireCapabilities class being used)
* Used to change certain setting of Chrome browser
* By Default, selenium starts chrome browser with fresh profile having no settings, cookies, history.
* Frequently use methods:

To add new extensions/plugins to browser:

**options.addExtensions(new File(“/path/to/plugin1.crx”));**

To accept untrusted certificate

**options.setAcceptInsecureCerts(true);**

To disable info-bar

**options.addArguments(“--disable-infobars”);**

For Headless mode:

**options.addArguments(“--headless”);**

**HeadLess Browser:**

/\*\*\*\* HeadLessHtmlUnitDriver \*\*\*\*\*/

**publicstaticvoid** main(String[] args) {

**WebDriver driver = newHtmlUnitDriver();**

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

System.***out***.println("Title : "+driver.getTitle());

}

/\*\*\*\* HeadLessChromeDriver \*\*\*\*\*/

**publicstaticvoid** main(String[] args) {

System.*setProperty*("webdriver.chrome.driver", System.*getProperty*("user.dir")+"\\Drivers\\chromedriver.exe");

**ChromeOptionsoptions = newChromeOptions();**

**options.addArguments("--headless");**

**WebDriver driver = newChromeDriver(options);**

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

System.***out***.println("Title : "+driver.getTitle());

}

More chromeOptions arguments:

<https://peter.sh/experiments/chromium-command-line-switches/>

**Static Drop-down:**

WebElementdd\_View = driver.findElement(By.*cssSelector*("#ctl00\_ContentPlace1"));

**Select s = newSelect(dd\_View);**

/\*To Select dropdown value \*/

s.selectByIndex(1); //If Index has been given in HTML code

s.selectByValue("9999"); //if 'Value' attribute is given with some values in the code

s.selectByVisibleText("100"); //if text is given with some values in the code

/\*To get first selected value \*/

System.***out***.println(s.getFirstSelectedOption().getText()); //to get selected text

/\*To display all dropdown values \*/

List<WebElement>list = s.getOptions();

System.***out***.println("List Size = "+list.size());

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

{

System.***out***.println(i+" "+list.get(i).getText());

}

/\* Other useful methods in Select Class

s.deselectAll();

s.deselectByIndex(arg0);

s.deselectByValue(arg0);

\*/

**List of Interfaces:**

* WebDriver:

WebDriver driver = new ChromeDriver(); // ‘SearchContext’ is super the Interface of WebDriver

* TakesScreenshot:

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

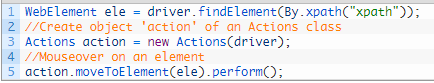
FileUtils.copyFile(screenshotFile,newFile("filename\_with\_path.png"));

**List of Classes:**

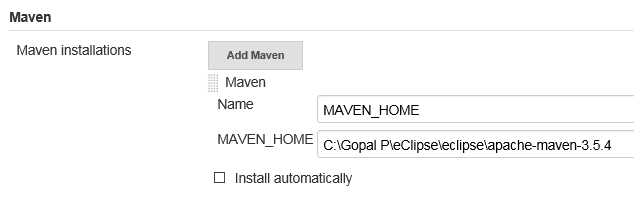
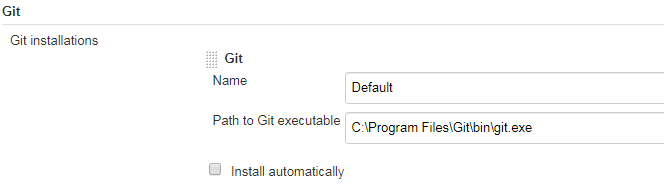
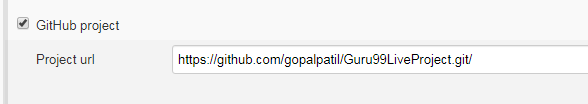
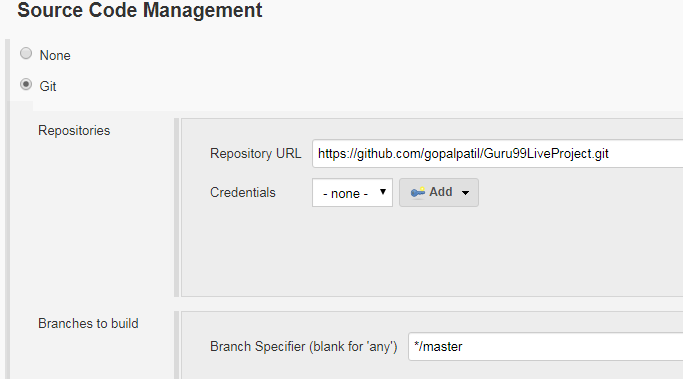
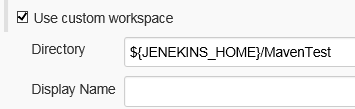
* ChromeDriver, FirefoxDriver, InternetExplorerDriver

WebDriver driver = new ChromeDriver();

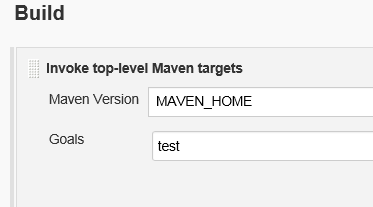
* Actions



**Jenkins:**

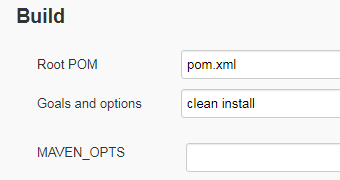
* Continuous Integration tool
* Using Jenkin, We can trigger test execution automatically, schedule test, set test execution on occurring particular event, send the test report automatically after test complete etc.
* Jenkin run on it’s own server
* Jenkin uses and maintains it’s own TestNG and Java versions. If your test cases build in older version, Jenkin uses its own version of required applications and run the program.
* To Install Jenkin, you have to download “Generic Java Package.WAR” file from Jenkin’s web
* Command prompt and goto the path where .war file is saved
* Type the command
* C:\>MyPathWhereWarFileSaved > java –jar jenkins.war –httpPort=8080
* It will show the message that, Jenkins is fully loaded and running
* You can access the jenkin in browser <http://localhost:8080>
* In the company that jenkinurl can be different (not a localhost)
* Now, you have to set up Maven and Java Locations which are installed . For that, Jenkins > Manage Jenkins > Global Tool Configuration
* Set up Java. Give Name and Path same as given in system environmental variable.
* 
* Set up Maven Give Name and Path same as in system environmental variable and Save.
* 
* IF we want to run the test from GITHub, give followings settings in GIT
* 
* Now, Create new Jenkin Item as Free Style Project or Maven Project
* Then Goto General > Advance option
* Select ‘Use Custom Workspace’ (IF we don’t have GitHub or SCM )
* FOR GIT, we assume that you have already set up local GIT repository and linked it up to GitHub for Selenium project
* .jenkins folder gets created under your users folder e.g. C:\Users\gpatil\.jenkins
* IF Don’t have GIT, Copy you Maven Project folder and paste it in C:\Users\gpatil\.jenkins folder. (I am not sure why we need to perform this step)
* If have GIT, don’t need to perform above steps.
* If have GIT, do following settings in Jenkin Project configuration > General tab
* 
* If have GIT, do following settings in Jenkins Project configuration > Source Code Management tab
* 
* IF NOT have GIT, Set Project Folder path in Directory field
* ${JENEKINS\_HOME}/MavenTest (where project folder is ‘Maven Test’ and {JENKINS\_HOME} gives the .jenkins folder path)
* 
* Now, Jenkins knows where is the project to be executed
* Goto Build Environments > Build > Add Build Step > Invoke top-level Maven Targets
* Select MAVEN\_HOME from the ‘Maven Version’ drop down since it is a system environment variable which is set up for Maven
* Enter Maven Command in ‘Goals’ box. Note that here we don’t have to write ‘mvn’; so instead of :\>mvn test command, just write ‘test’.
* If you want to execute profiler command i.e. :\>mvn test –Pregression Then just write

‘test –Pregression’ in Goals.

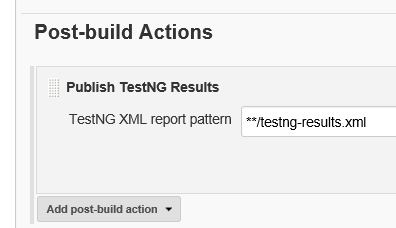
* 
* IF you have created MAVEN Project in Jenkins, then we don’t have to above setting.
* Just enter appropriate mvn command in Goals box as below:
* E.g.

test

Clean install

* 
* Save the changes, Goto created ProjectLevel and click on ‘Build Now’ option. Maven project will be executed.

**Reports**

* In Eclipse, after you refresh the project, you can see auto-generated report at eClipseProejct> target > surefire-reports > index.html
* Surefire generates this report.
* Also testing generates the report in xml format at the same location eClipseProejct> target > surefire-reports > testing-results.xml
* We can use this testing-results.xml to get the report into Jenkins.
* For that we need to install TestNG results plugin in Jenkins to read the testing-results.xml
* Goto Jenkins > Manage Jenkins > Manage plugins > Available tab > Search for “TestNG Results PlugIn” > Install it
* Now you want to generate the report from Jenkins. For that, goto your Project created in Jenkins > Configure > Post-Build Action > Publish TestNG Result. Text box will be populated automatically with \*\*/testing-results.xml. Save
* 
* Run the build. You can see the TestNG Results on Jenkins > you Project page. (refresh it to get reflect)

Run Build Periodically:-

Also, you can schedule the test Periodically in Jenkins > Project > Configure > Build Triggers > Build Periodically

By setting the schedule period to 15 13 \* \* \* you tell Jenkins to schedule the build every day of every month of every year at the 15th minute of the 13th hour of the day.

Jenkins used a [cron expression](https://en.wikipedia.org/wiki/Cron" \l "CRON_expression), and the different fields are:

1. MINUTES Minutes in one hour (0-59)
2. HOURS Hours in one day (0-23)
3. DAYMONTH Day in a month (1-31)
4. MONTH Month in a year (1-12)
5. DAYWEEK Day of the week (0-7) where 0 and 7 are sunday

If you want to schedule your build every 5 minutes, this will do the job : \*/5 \* \* \* \*

If you want to schedule your build every day at 8h00, this will do the job : 0 8 \* \* \*

For the past few versions (2014), Jenkins have a new parameter, H (extract from the [Jenkins code documentation](https://github.com/jenkinsci/jenkins/blob/master/core/src/main/resources/hudson/triggers/TimerTrigger/help-spec.jelly)):

To allow periodically scheduled tasks to produce even load on the system, the symbol H (for “hash”) should be used wherever possible.

For example, using 0 0 \* \* \* for a dozen daily jobs will cause a large spike at midnight. In contrast, using H H \* \* \* would still execute each job once a day, but not all at the same time, better using limited resources.

Note also that:

The H symbol can be thought of as a random value over a range, but it actually is a hash of the job name, not a random function, so that the value remains stable for any given project.

 If you want to run job every 15 mins on Jenkins use this:

H/15 \* \* \* \*

To allow periodically scheduled tasks to produce even load on the system, the symbol H (for “hash”) should be used wherever possible. For example, using 0 0 \* \* \* for a dozen daily jobs will cause a large spike at midnight. In contrast, using H H \* \* \* would still execute each job once a day, but not all at the same time, better using limited resources.

Examples:

* H/15 \* \* \* \* - every fifteen minutes (perhaps at :07, :22, :37, :52):
* H(0-29)/10 \* \* \* \* - every ten minutes in the first half of every hour (three times, perhaps at :04, :14, :24)
* H 9-16/2 \* \* 1-5 - once every two hours every weekday (perhaps at 10:38 AM, 12:38 PM, 2:38 PM, 4:38 PM)
* H H 1,15 1-11 \* - once a day on the 1st and 15th of every month except December

**Others:**

* Limiting the scope of driver
* Click all the links using Ctrl+Click

**publicvoid**getFooterLink()

{

/\*initiateDriver();

driver.get("https://firstsiteguide.com/what-is-blog/"));

HomePagehomepage = new HomePage(driver);\*/

//get Total links on the web page

**int**totalLinks = driver.findElements(By.*tagName*("a")).size();

System.***out***.println("Total links in webpage "+totalLinks);

//WebElement of Footer section

WebElementfooterSection = driver.findElement(By.*id*("footer"));

//get total links in Footer section

//For that limit the scope of driver using WebElement 'footerSection' as driver

**int**totalLinksInFooter = **footerSection**.findElements(By.*tagName*("a")).size();

System.***out***.println("Total links in Footer section: "+totalLinksInFooter);

//get total links in Footer section only in first column

//For that limit the scope of driver using WebElement 'footer1stColumn' as driver

WebElementfooter1stColumn = **footerSection**.findElement

(By.*xpath*("//div[1]/div[@class='footer-menu footer-menu-first']"));

**int**totalLinksInFooter1stCol = **footer1stColumn**.findElements(By.*tagName*("a")).size();

System.***out***.println("Total links in Footer section 1st Column: "+totalLinksInFooter1stCol);

//click and Open the all the links in 1st column of footer

**for**(**int**i=0;i<totalLinksInFooter1stCol;i++)

{

footer1stColumn.findElements(By.*tagName*("a")).get(i).click(); //X

//Above stmtwon’t work since it’ll give error after clicking 1st link; because

//you will move to next page &2nd link won’t available to click on; so in this

//case, you will have to navigate back and perform click operation on next link

//Instead of above, you can write code as below which will Click all the //links (Ctrl+click)in 1st column of footer >which will open in new browser tab

String clickOnLnk = **Keys.*chord*(Keys.*CONTROL*,Keys.*ENTER*);**

footer1stColumn.findElements(By.*tagName*("a")).get(i).sendKeys(clickOnLnk);

}

//Now, get all the titles of the open browser windows/pages

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

Iterator<String>it = windowHandles.iterator();

**while**(it.hasNext())

{

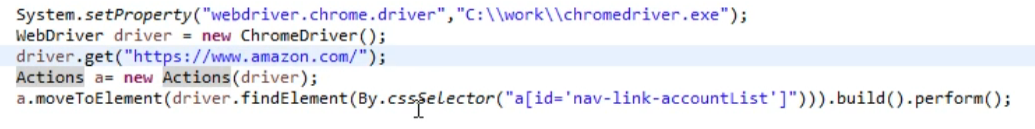
driver.switchTo().window(it.next());

System.***out***.println(driver.getTitle());

}

}

**Mouse Move Actions:**



This way, you can perform multiple actions:

* a.moveToElement(myWebElement**).click().keyDown(Keys.SHIFT).sendKeys(“Hello”).doubleClick().build().perform();**

here, you are performing multiple actions on an element. (‘a’ is a object of Actions class based on above screenshot).

* For Right click, we can use **“.contextClick()”**
* Question: What special you have done in the coding which is appreciated by your team or manager?
  + We had to open multiple links and check whether they are opening successfully or not. Earlier someone was opening the link one by one and navigating back to home page to open next link. It was taking 5-10 mins to run the code. I used Key.chord(Keys.CONTROL, Keys.ENTER) concept and execution time reduced to 1 min.

**Challenges faced using selenium automation testing, and how to solve them**

1. Image or text overlapping issue

2. No facility to deal with Captcha, Bar Code

3. Doesn’t support any non web based (Like Win 32, Java Applet, Java Swing, .Net Client Server etc) applications

4. When you compare selenium with QTP, Silk Test, Test Partner and RFT, there are many challenges in terms of maintainability of the test cases

5. Since Selenium is a freeware tool, there is no direct support if one is in trouble with the support of applications

6. Bitmap comparison is not supported by Selenium

7. Any reporting related capabilities, you need to depend on third party tools

8. You need to learn any one of the native language like (.Net, Java, Perl, Python, PHP, Ruby) to work efficiently

9. Difficult to identify dynamic objects

10. Working with frames

11. Selenium test playback is slow (IDE)

12. JavaScript sandbox, Flash, Applets, Silverlight, and HTML 5’s Canvas all present problems in Selenium

13. Dealing with pop-up windows: Selenium can sometimes fail to record common popups in web apps. The Alert interface brings with it the following commands: void dismiss(), void accept (), getText(), void sendKeys(String stringToSend). The first two basically click on the “cancel” and “OK” buttons respectively on a popup window.

14. Timeout resulting from synchronization problems: One should ideally use selenium.IsElementPresent(locator) to verify that the object is in a loop with Thread.Sleep

15. Testing Flash apps: To automate flash apps with Selenium, one can use Flex Monkium. The application source code must be compiled with the swc files generated by Flex Monkium. Then the app and the Selenium IDE are connected, and the tests can be recorded with IDE.

16. Unexpected error launching Internet Explorer. Browser zoom level should be set to 100% by default for the IE browser to overcome this error

17. Protected Mode must be set to the same valueerror occurs when trying to run Selenium WebDriver on a fresh Windows machine. This issue can be fixed by using capabilities when launching IE

18. Cross Browser Testing Issues

19. Ajax Components

**GIT/GITHub:**

* GIT: it is a version control software which needs to install locally.  It is a tool to manage different versions of edits, made to files in a git repository. We can commit, branch, tag all the changes on our local machine. Git also is known as distributed version control system which means using Git you can push and pull yours as well as others changes to other people’s machines. Now, using Git you can work on the same copy as your team member. But the copy on which you will be working, won’t reflect any of the changes in the main directory unless and until you pull your collaborator’s changes and push back your own changes.
* Now, using Git you can work on the same copy as your team member. But the copy on which you will be working, won’t reflect any of the changes in the main directory unless and until you pull your collaborator’s changes and push back your own changes. Here GitHub comes into the picture
* GITHUB: *GitHub* is a Web-based Git version control repository hosting service. It is a space to upload a copy of the Git repository
* Go thru GIT terms here <https://linuxacademy.com/blog/linux/git-terms-explained/>
* Install GIT on local machine
* Create Repository on GitHub
* Use following commands in command prompt:

<https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html>

* To Tell GIT who you are. You need to run these commands very first time

C:/MyFolder> git config --global user.name “patilsureshc54”

C:/MyFolder>gitconfig --global user.email “patilsureshc54@gmail.com”

* Create new local repository on your machine. Goto the folder where you wanted to create local repository. Here folder is ‘MyFolder’.

C:/MyFolder> git init

It will create hidden .git file into the folder and this folder will be referred as local repository

* Now you have to set your local repository to Remote server (GitHub Repository)

C:/MyFolder> git remote add origin <GitHub Repository URL>

* If you wanted to commit the files to upload to gitHub:

C:/MyFolder> git add \*.\*

C:/MyFolder> git commit –m “Commit Message”

* Now, it’s ready to Push the files in GitHub.

C:/MyFolder> git push origin <branch name>

* If you want to pull files from GitHub.

C:/MyFolder> git pull origin <branch name>

If you are pulling very first time in fresh local repository then u can use below command only first time

C:/MyFolder> git clone <GitHub Repository URL>

* To check status before committing files:

C:/MyFolder> git status

* To create and switch to new Branch from existing branch

C:/MyFolder> git checkout –b <newBranchName>

* To switch to another Branch from one branch

C:/MyFolder> git checkout <BranchName>

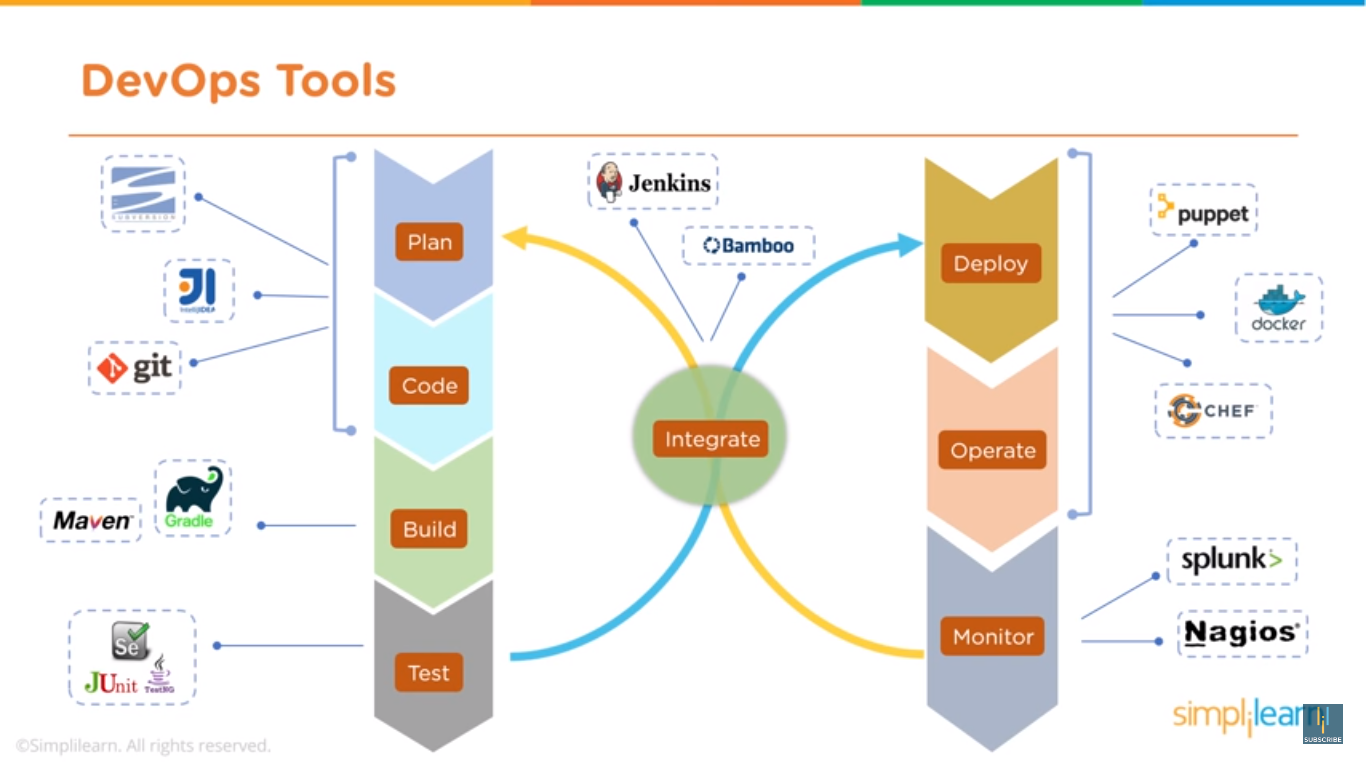
* To merge branch1 to master branch, then switch to master branch and then do merge. Note that if there are changes in branch1, you have to perform commit first before merging.

C:/MyFolder> git checkout <masterbranch>

C:/MyFolder> git merge <branch1>

**DevOps:**

We would say, it is an extension of Agile model of continuous delivery. Where Development team (Dev +Tester) and Operations team work closely from Planning phase to deployment and monitor phase. Here everything is mostly automated with different tools. Dev and Prod environment would be similar here for successfully devops stages.



Advantages:

* Time to create and release software releases are reduced
* Complexity of maintaining an application is reduced
* Improved collaboration between Dev and Operations team

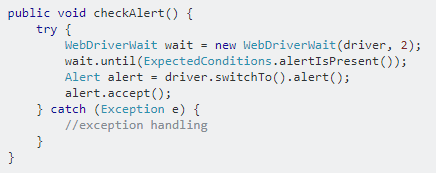
**Test Driven Development (TDD):**

* TDD is similar to Unit Testing; but not Unit Testing
* In Unit Testing, piece of code is developed first then it could be tested
* In TDD, Test Case is created first based on the business logic and then the piece of code is developed to satisfy the test case
* The Steps are as: Write Failing Tests (because the code is not exits), > Write minimal of code to pass the test, > Re-factor the logic/code to meet coding standardization

**QUESTIONS**:

* How to check if Element is present in webpage?
* How to check if Element is visible or invisible in webpage?
* How would you test that if the web session is expired or got deleted, user should be landed on homepage on performing any action on current page?
* 🡺 Delete the cookie using driver.manage().deleteCookieNamed(“sessionID/cookiename”) and then perform any action like click or submit on current page. Login page should be displayed.
* There are 100 test cases and you have to run only 10 test cases as Smoke TCs, how will you run it?
* How to perform an action on Web alert popup only if alert it present?

Use Try /Catch block and use wait to check alertIsPresent()

* 
* How to navigate to multiple popup windows

StringhMyWindow=driver.getWindowHandle();

Iterator<String>hWindows=driver.getWindowHandles();

while(hWindows.hasNext())

{

StringhWindow=hWindows.next();

driver.switchTo().window(hWindow);

driver.switchTo().window(hWindow).getTitle();

}

* How to handle AJAX control
* What is Page Object Model (POM) and PageFactory

**Convert an Array to ArrayList:**

String[] arry = {“Name1”,”Name2”,”Name3”};

String myName = “Name2”;

List names = Arrays.asList(arry);

If(names.contains(myName)) 🡺 returns true; since “Name2” is present into array List ‘names’.

Interview Questions reference links:

<https://www.techbeamers.com/latest-selenium-interview-questions-and-answers/>

<https://www.edureka.co/blog/interview-questions/selenium-interview-questions-answers/>

<https://artoftesting.com/interviewSection/selenium-interview-questions.html>