What is Framework?

- Framework is systematic and sequential way to write the test scripts for automation testing of the application.
- It set some rules and regulation to write test script to achieve test results.

Why to use Framework?

Framework is a structural way for building/automating application which provides

- Easy code maintenance
- Increase code reusability
- Higher code readability
- Reduced script maintenance cost
- Reduced tests time execution
- Reduced human resources
- Easy reporting

Components of the Framework?

1. Programming languages: Java + Selenium
2. IDE: Eclipse tool
3. Testing Framework: TestNG
4. WebDriver Manager
5. Apache POI
6. Extent Report
7. Log4j
8. Version Control: GIT, and Github

JENKINS:

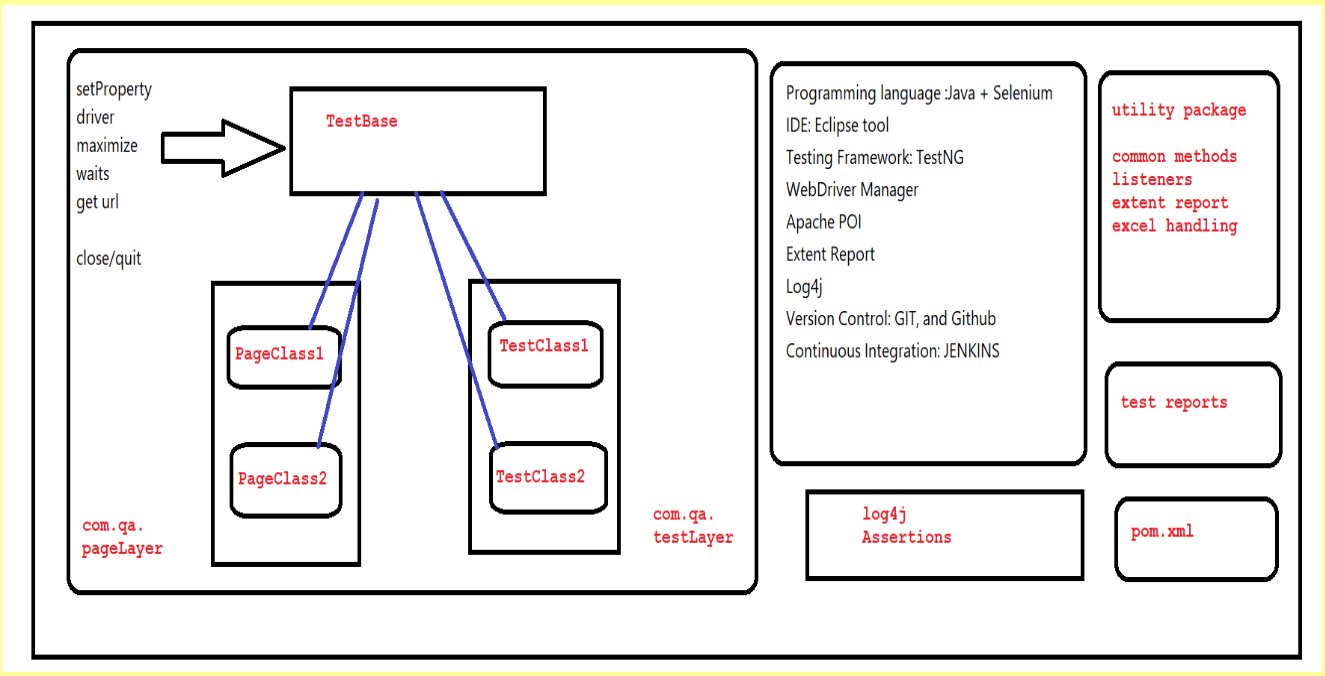
- Jenkins is an open-source automation server.
- It is used for the continuous integration/continuous delivery and deployment (CI/CD) of the automation software, which is written in the Java programming language.

Types of Automation Frameworks:

1. Data-Driven framework
2. Keyword-driven framework
3. Hybrid-driven testing framework

1. **Data-driven Framework:**
    - ❖ As name suggest data driven means executing testcases with the help of test data which is stored in some specific format like table or in excel sheet
2. **Keyword-driven Framework:**
    - ❖ It is also known as table-driven testing.
    - ❖ In Keyword-driven testing, we use a table format to define keywords or action words for each function or method that we would execute.
3. **Hybrid-driven Framework:**
    - ❖ hybrid driven framework which is Combination of data+ keyword driven framework.

- ❖ I was working on the Data Driven framework.
- ❖ Data driven allows to automate the application using test script which can executed by using all the test data available in table. (Excel)
- ❖ Also, data driven covering high priority testcases those may be positive or negative test cases.
- ❖ For scripting we are using maven project,



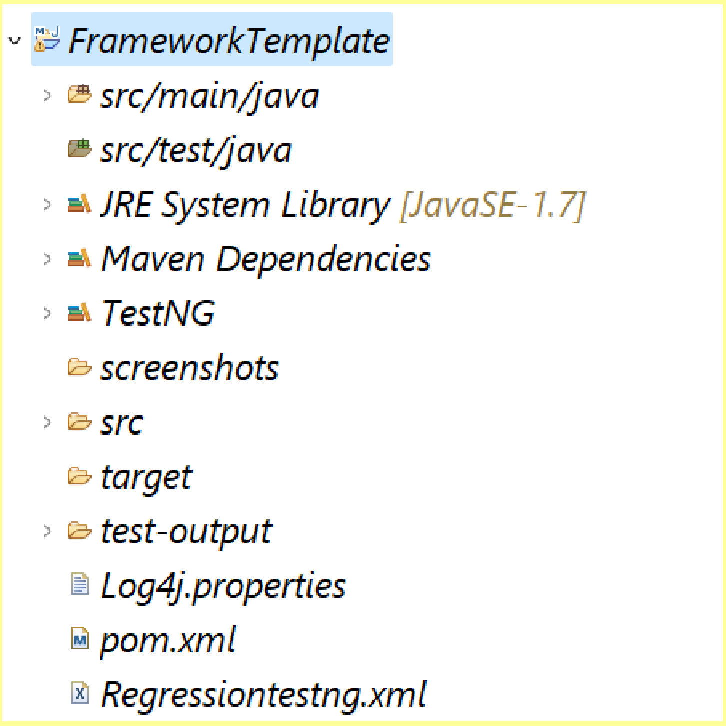
Maven Project -

- Maven is software project management tool and build automation tool.
- It is hosted by the Apache Software Foundation in 13 July 2004. It was begin with the Jakarta Project.
- Maven projects are configured using the POM (Project Object Model), which is stored in pom.xml file.
- Maven dynamically downloads the Java libraries, maven plugins from one or more repositories and store them in local cache.

Maven is used:

- To build the project
- To manage the project
- To define the project structure
- Test management
- Dependency build
- It is used to check the compilation issues between the framework components whenever the multiple test engineers integrate their files into same framework.

Maven Project structure:



```
FrameworkTemplate
  src/main/java
  src/test/java
  JRE System Library [JavaSE-1.7]
  Maven Dependencies
  TestNG
  screenshots
  src
  target
  test-output
  Log4j.properties
  pom.xml
  Regressiontestng.xml
```

So, there are two main folder that is main java (to write test script logic) and test java (to write test cases) and other are supporting plug-ins or files.

pom.xml file:

- A Project Object Model (pom) an XML file which have information about the project.
- It has different tabs like overview, dependencies, dependencies hierarchy, effective pom, pom.xml
  - In overview tab, there are artifact details and project details.
  - Details like group id, artifact id, version, project name, URL.
  - In dependencies tab, there are dependencies which added in our project. We can add it and remove it as per our requirement.
  - Dependencies hierarchy tab, where as its name says hierarchy of dependencies are there.
  - Effective pom tab, it is detailed xml file, all information about project you will find here.

- pom.xml tab, it has group id, artifact id, version details, Where we can add dependencies as
    - first we have to go on to the maven repository website and then search the dependency which we want then select the version of it just click on code it will automatically copy then paste it in pom.xml file.

Here's an example:

```
<project>
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.mycompany.app</groupId>
        <artifactId>my-app</artifactId>
        <version>1</version>
</project>
```

- <project> –> it is the root
- <modelVersion> -> should be set to 4.0.0
- <groupId> -> The id of the project's group.
- <artifactId> -> The id of the artifact (project)
- <version> -> The version of the artifact under the specified group

src/main/java

- Here we are going to write code by using Page Object Model design pattern with Page Factory.
- POM: As per the Page Object Model, we have to maintain classes for every web page. Each web page has a separate class, and that class holds the functionality and members of that web page. Separate classes for every individual test.
- POM uses encapsulation feature of OOPs, (Explain Encapsulation in detail). where variables are private and methods are public and we use getter and setter methods but there is disadvantage of POM i.e. can't find the hidden elements so we will use page factory class to overcome disadvantage of POM

- In page factory we use @findby annotation above your web element and we have to use static method like init element to initialize the data member in page factory. which contains the locators, different methods to perform actions on web elements like sendKeys(), click(), etc.
- It also contains the Utility class, Base class, Property class

src/test/java

- As per the Maven project, all tests classes are kept in the 'src/test/java' folder.
- So it contains the execution for all the POM classes which are stored in src/main/java. Because in POM classes we can't write the main method.

TestNG

- TestNG is an Automation framework in which NG stands for Next Generation.
- TestNG is unit testing framework, inspired from junit which have new functionality added.
- TestNG uses the annotations (@). Using the TestNG we can perform the end-to-end testing.
- Using testNG we can generate the proper report, we can easily come to know how many test cases are failed, how many test cases are passed, which test cases are passed and which are failed and also which test cases are skipped.
- One of the best feature of the testNG is we can only execute the Failed test cases.

Why we use testNG?

- Generate detailed HTML test reports
- Run test cases in groups
- Retesting is easy.
- Regression testing using testNG is easy
- Parallel test execution

- Control test execution
- Annotations made tester life easy
- Without main method we can execute the classes

Annotations:

- Annotations are lines of code that can control, how the method below it will executed.
- Annotations are always preceded by @ symbol.
- The test classes contains the annotations not the POM classes. So it will seen in the src/test/java classes.

There are following annotations used in testing:

1. @BeforeSuit:
   a. The @BeforeSuite annotated method will run before the execution of all the test methods in the suite.
   b. To initiate the data base connection.
2. @BeforeTest:
   a. The @BeforeTest annotated method will be executed before the execution of all the test methods of available classes belonging to that folder.
3. @BeforeClass:
   a. The @BeforeClass annotated method will be executed before the first method of the current class is invoked.
4. @BeforeMethod:
   a. The @BeforeMethod annotated method will be executed before each test method will run.
   b. In this method we write the code for Browser launching, delete cookies, maximize the browser, hit the application URL that means the common properties.

    c. Here we can execute the login functionality methods are which are written in Login POM class.

5. @AfterMethod:
    a. The @AfterMethod annotated method will run after the execution of each test method.
    b. Here we can execute the logout functionality methods are which are written in Logout POM class. Using the @BeforeMethod and @AfterMethod we can do the end-to-end testing.
    c. of all the test methods of the current class.
    d. Here we can close/quit the browser.
6. @AfterClass:
    a. The @AfterClass annotated method will be invoked after the execution
7. @AfterTest:
    a. The @AfterTest annotated method will be executed after the execution of all the test methods of available classes belonging to that folder.
8. @AfterSuite:
    a. The @AfterSuite annotated method will run after the execution of all the test methods in the suite.
9. @BeforeGroups
    a. The @BeforeGroups annotated method run only once for a group before the execution of all test cases belonging to that group.
10. @AfterGroups
    a. The @AfterGroups annotated method run only once for a group after the execution of all test cases belonging to that group.
11. @Test Annotation:
    a. The method below these annotations contains the test cases. And @Test annotations we can write as many times we want in single class, but above annotations are written once per class.

b. If three 3 @Test (@Test, @Test, @Test) are there, then for each @Test @BeforeMethod & @AfterMethod will execute.

c. i.e. @Test is surrounded by @BeforeMethod & @AfterMethod.

d. Without main method we can executes the methods using @Test and other annotations.

Keywords:

- Parameters are keywords or keywords are parameters to @Test annotation, that modify the annotation's function.

Total 5 keywords are there.

1. Priority
   a. If there are multiple test case are there then we use priority.
   b. If we don't provide any priority then it will execute in alphabetical order, which is set by the TestNG.
   c. 0 priority means no priority.

      e.g.

      @Test (priority = 1)

      public void homepage () {}

      @Test (priority = 2)

      public void searchPage () {}
2. invocationCount:
   a. If number times you want to execute same method, then Invocation Count keyword is used.
   b. e.g. @Test (invocationCount = 2) public void homePage () {} } // homePage () will run for two times but its test count will be 1. We can use multiple keywords at a time separated by, (comma).

   c. @Test (priority = 2, invocationCount = 3)

3. Enabled
   a. If you don't want to execute the particular test then you can use the comments line but if there are too many scenarios which is not in sequence that you don’t want to execute then it will be time consuming and also there will be chance of errors.
   b. So, to avoid that TestNG provides "enabled" keyword with value "false" i.e. "enabled=false" then that test scenario and method will take part in execution.
   c. *e.g.*
      @Test(enabled = false)
      public void homePage () {}
4. dependsOnMethods:
   a. If you want to execute particular scenario after execution of its previous scenario, then it is depending on the previous scenario in such case we us dependsOnMethods.
   b. *Syntax:*
      @Test (dependsOnMethods = {"previousMethodName"})
      public void currentMethodName () {}
5. timeOut
   a. When a test method is taking longer time than the specified time (in milliseconds) to execute, then that test method will be terminated and marked as failed, this feature is available in TestNG.
   b. While running test methods there can be cases where certain test methods get struck or may take longer time than to complete the execution than the expected. We need to handle these types of cases by specifying Timeout and proceed to execute further test cases / methods.
   c. Syntax:
      Test(timeOut=timeInMillisecond)

Regression Testing using TestNG through Automation:

testing.xml file syntax:

```
<?xml version="1.0">
<suit>
    <test>
        classes>
            <class name>
        </classes>
    </test>
</suit>
```

Grouping:

- TestNG Groups allow you to perform groupings of different test methods. Grouping of test methods is required when you want to access the test methods of different classes.

Meta Group:

- We can also specify a group within another group. The groups which are defined in other groups are known as Meta Groups. when we create test suit then we don't need to run class, we can run test suit directly

failed.xml:

- whenever your test cases get failed at that time one folder will be created in that failed.xml file will be created.
- So, we can do retesting means only failed test cases we can reexecute.
- If suppose after multiple test data it is showing error then we log it as bug and that bug is resolved then again, we have to perform the testing on that functionality that we called retesting. So, we don't have to check all test cases we have to test only failed test cases which we can perform through TestNG very easily, for that only execute the failed.xml file.

Assertion:

- In Selenium, Asserts are validations or checkpoints for an application.
- Assertions will state confidently that application behaviour is working as expected.
- That means actual result produced by application is matching to expected result or not. So, Asserts in Selenium are used to validate the test cases.
- It helps to testers to understand if tests cases/scenarios are passed or failed.

Types of Assertion:

1. Hard Assertions
2. Soft Assertion

1. Hard Assertions
    a. Hard assertions are which if the test does not meet assert conditions, then the test executions will get terminated, that means if actual and expected not matched in assertion then that Test case will mark as failed and further execution is aborted or stopped. And it will start to execute next test case/scenario.
    b. If both results matched then only the test case will marked as passed and execution of that test case continues.
    c. In case of an assertion error, it will throw the “java.lang.AssertionError” exception.

There are different types of hard assertion or different methods are present:

assertEquals(), assertNotEquals(), assertTrue(), assertFalse(), assertNull(), assertNotNull().

2. Soft Assertions:
    a. In hard assertion, if an assertion fails then it aborts the test case otherwise it continues the execution. Sometimes we want to execute the whole script even if the assertion fails. This is not possible in Hard Assertion. To overcome this problem, we need to use a soft assertion in testing.
    b. So, to perform we require SoftAssert class and need to create the object of that class and using object refVar we can execute the assertions methods.
    c. Also it is compulsion that after all assertion used, we have to
    d. use assertAll() method otherwise it will produce weird output.

Listeners:

- TestNG provides the @Listeners annotation which listens to every event that occurs in a selenium code.
- Listeners are activated either before the test or after the test case. It is an interface that modifies the TestNG behaviour.
- For example, when you are running a test case either through selenium and suddenly a test case fails. We need a screenshot of the test case that has been failed, to achieve such scenario, TestNG provides a mechanism, i.e., Listeners. When the test case failure occurs, then it is redirected to the new block written for the screenshot.
- Also, when test script pass then we have to show the success or pass message in the console box i.e., logs we can and that logs we can modify and set using Listeners.

- Listeners are implemented by the ITestListener interface. An ITestListener interface has the following methods:
    - ***onTestStart(),***
    - ***onTestSuccess(),***
    - ***onTestFailure(),***
    - ***onTestSkipped(),***
    - ***onStart(),***
    - ***onFinish().***
- We can create the TestNG Listeners in two ways. First we can use the @Listeners annotation within the class and second way to use the within the suite.
- Within the class: @Listeners (packageName.ClassName.class)
- In the suit, we have to add <listeners> tag after <suit> and before <test> tag.
    - <listener class-name=" packageName.ClassName "/> </listeners>

Extendable report:

- This is third party tool installed into our project.
- Extended report which is used to make a report of executed script and test suit (regression suit).
- After compilation of execution of test cases, email will be triggered and report is directly sent to the email id.