TestNG-Framework:

- TestNG stands for testing new generation.
- It is a framework which allow the user to decide the **conditions and configuration** for a specific or multiple test cases and generate a report after the completion of execution.
- TestNG is **an open source automated testing framework**.
- TestNG is used to design test cases in the systematic way.
- It is available in the form of **jar files.**
- Using TestNG we can see proper output and we will come to know how many test cases got **passed, failed or skipped**.
- TestNG does not require a main method to run and methods written need not be static.
- It provides different **annotations** support which makes testers life easy.
- TestNG framework allows us to create multiple test cases in a single class.
- We can create multiple test cases with the help of **@Test annotations.**
- Because **@Test annotation** is used to tell that method below it is a test case.
- We can give **priorities** to our test cases.
- It provides use of different keywords like **priority, invocation count, DependsOnMethods** etc.
- TestNG is a testing framework inspired from JUnit and NUnit, but introducing some new functionalities that make it more powerful and easier to use.

Advantages of TestNG:

- We can decide whether test case is **passed or fail or skipped** on a particular condition.
- After the execution **emailable report** is available.
- Description can be added for a test case.
- We can **group the test case** according to the requirement.
- We can decide the **priority** i.e. which test case will execute first and which one at last.
- We can create **multiple test cases by using multiple data.**
- We can execute the **test cases in parallel**.
- It has different assertion that help to **check expected and actual result.**
- TestNG generates **XML and HTML report** and allow to open in web browser.

Disadvantages of TestNG:

- Setup of TestNG is **time-consuming.**
- If your **project does not require test case prioritization** that case TestNG is of no use.
- In the olden day, it was less commonly used compared to JUnit, so fewer people was having experience with it.

Test cases Execution order in TestNG

- **Without any keyword:**
  - If we have not use any keyword which decides the execution order, in this case the execution will depends ***on dictionary order*** of the test case name.

Keywords Used in TestNG:

In TestNG we have 5 keywords:

1. priority
2. enabled
3. invocation count
4. timeout
5. dependsOnMethod

1. **priority** keyword **:**
   - If we set the priority to the test case then test case execution will happen according to it.
   - Default value of priority would be 0.
   - **Priority of test case can be:**
     - Positive value
     - Negative value
     - Zero Value
     - Duplicate
   - **Priority of test case cannot be:**
     - Decimal value
     - Variable

   - **For example:**

```
@Test (priority = 1)
        public void testCase4 () {

                Reporter.log ("Test Case 4 has executed", true);
```

2. **enabled** keyword **:**
   - If we want to ignore or skip a particular test case from an execution process then we have to use enabled keyword.
   - with enabled keyword we can write false or true either test case considered or we want to skip:

```
    @Test (enabled = false)
        public void testCase3 () {

                Reporter.log("TC3 Got Executed" , true);

    }
```

3. **invocationCount** keyword **:**
   - We can execute a particular test method multiple times with the help of invocation count keyword.

```
@Test (invocationCount = 6)
    public void testCase3 () {

        Reporter.log ("TC3 Got Executed" , true);
    }
```

4. **timeOut** keyword **:**
   - If one of the test case is taking a long time due to which other test case are failed.
   - The timeout is a time period provided to the test case to completely execute its test case.
   - This keyword is use to when we have to wait for execution of test cases for some time and if the @Test method does not get executed in provided timeOut then it will throw the timeOut error

```
@Test (timeOut =8000)
    public void testcase () {

        Reporter.log ("Login success");
    }
```

5. **dependsOnMethod** keyword :
   - This keyword allow the user to make a dependency over a method and execute the dependent method only if the method on which it get depends get execute.
   - If the method got failed then the dependent method will get skip.

```
@Test (priority = 2 , dependsOnMethods = "testcase")
public void testCase1 () {

    Reporter.log ("Login Failed");

    }
```

Reporter class in TestNG:

- Reporter class is a class in TestNG which has a log method that gives functionality to user for mentioning the messages inside the html report as well as in the console (Standard input output media).

```
@Test (priority = 1)
public void testCase4 () {

    Reporter.log ("Test Case 4 has executed", true);
```

Annotations used in TestNG:

- 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 see in the **src/test/java** classes.
- Annotations are used **to *control the flow of methods***, and they make selenium coded ***manageable and effective***.
- We have **pre-condition annotations i.e., @Before** and **post-condition annotation i.e. @After**
- After wiring annotation compulsory it should have a method.
- **Different annotations and its sequences are :**
    1. @BeforeSuite
    2. @BeforeTest
    3. @BeforeClass
    4. @BeforeMethod
    5. @Test
    6. @AfterMethod
    7. @AfterClass
    8. @AfterTest
    9. @AfterSuite
- **In this these 3 annotations will run together in a cycle**
    1. @BeforeMethod
    2. @Test
    3. @AfterMethod.

1. **Before Suite :**
    - The **@BeforeSuite** annotated method will run before the execution of all the test methods in the suite.
    - **To initiate the data base connection.**
    - The annotated method will be run only once before all tests in this suite have run.

2. **Before Test :**
    - The **@BeforeTest** annotated method will be executed before the execution of all the test methods of available classes belonging to that folder.
    - The annotated method will be run before any test method belonging to the classes inside the tag is run.

3. **Before Class :**
    - The **@BeforeClass** annotated method will be executed before the first method of the current class is invoked.

4. **BeforeMethod:**
   - The **@BeforeMethod** annotated method will be executed before each test method will run.
   - In this method we write the code for ***Browser launching, delete cookies, maximize the browser, and hit the application URL*** that means the **common properties**.
   - Here we can execute the login functionality methods are which are written in Login POM class.

5. **AfterMethod:**
   - The **@AfterMethod** annotated method will run after the execution of each test method.
   - 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.
   - **Here we can close/quit the browser.**

6. **AfterClass:**
   - The **@AfterClass** annotated method will be invoked after the execution of all test methods in a class.
   - The annotated method will be run only once after all the test methods in the current class have run.

7. **AfterTest:**
   - The **@AfterTest** annotated method will be executed after the execution of all the test methods of available classes belonging to that folder.
   - The annotated method will be run after all the test methods belonging to the classes inside the tag have run.

8. **AfterSuite:**
   - The **@AfterSuite** annotated method will run after the execution of all the test methods in the suite.

9. **BeforeGroups:**
   - The **@BeforeGroups** annotated method run only once for a group before the execution of all test cases belonging to that group.

10. **AfterGroups:**
    - The **@AfterGroups** annotated method run only once for a group after the execution of all test cases belonging to that group.

11. Test Annotation:

- The method below these annotations contains the test cases.
- If three 3 **@Test (@Test, @Test, @Test**) are there, then for each **@Test @BeforeMethod & @AfterMethod will execute.**
- **@Test is surrounded by @BeforeMethod & @AfterMethod.**
- Without main method we can executes the methods using @Test and other annotations.

What is test suite how it will be created?

Test-Suite:

- It is xml file which contains all the test classes' name which need to be executed.
- It is use to execute all/multiple test classes.
- To add classes in the throughout the project
- **syntax of test suite:**

```
<suite name="Suite name">
 <test name="Test name">
 <classes>
 <class name="packageName.className"/>
 </classes>
 </test>
</suite>
```

- **How to create test suite?**
  - First to right click on any one class of out of multiple running class >>click on TestNG>> convert to TestNG>>change name of xml file or same>>finish or create one xml file to your project >> double click on that >> select file source and give class name
  - To run three class/multiple classes run at a time
  - Test suite follow sequential order means we give class name that manner these are to be execute.

```
package annotations;
import org.testng.Reporter;
import org.testng.annotations.Test;
public class DemoTest_1 {

    @Test
    public void case1 () {
        Reporter.log("Test Case 1" , true);
    }
    @Test
    public void case2 () {
        Reporter.log("Test Case 2" , true);
```

```
    }

}
```

```
package annotations;
import org.testng.Reporter;
import org.testng.annotations.Test;
public class DemoTest_2 {

    @Test
    public void case3 () {

        Reporter.log("Test Case 1" , true);
    }
    @Test
    public void case4 () {

        Reporter.log("Test Case 2" , true);
    }
        }
```

```
<suite name="Suite">
  <test thread-count="5" name="Test">
    <classes>
      <class name="annotations.DemoTest_1"/>
      <class name="annotations.DemoTest_2"/>
    </classes>
  </test>
</suite>
```

Assertion in TestNG:

- Assertion determines the state of the application whether it is the same what we are expecting or not.
- If the assertion fails, then the test case is failed and stops the execution.
- If we use if else verification process is to verify expected result of a test case and the length of the test case is increase and also test script will take more time for execution.
- To reduce length of test script we need to use Assert class for verification which contains static method.
- Important static methods present in the Assert class and all the static method should be imported for **org.testNG**.
- Assert class contains static method like
- **Hard Assert Methods:**
    1. assertEquals()

2. assertNotEquals()
3. assertTrue()
4. assertFalse()
5. assertNull()
6. assertNotNull()
7. fail()

- By using this assert class static method we compare or matching the actual or expected result and to print test case pass or fail.

1. **assertEquals():**
   - **assertEquals ()** method used to ***verify expected and actual result***, if both results *are same* then output is pass otherwise fail.

```
package assertionInTestNG;
import org.testng.Assert;
import org.testng.annotations.Test;

public class DemoClass1 {

	@Test
	public void TC1()
	{
	String exp="hi";
	String act="hi";
	Assert.assertEquals(act, exp);
	}
	@Test
	public void TC2()
	{
	String exp="hi";
	String act="hello";
	Assert.assertEquals(true, false, "Actual and expected result are not match");
	}
}
```

Output on Console:

```
[RemoteTestNG] detected TestNG version 7.4.0
PASSED: TC1
FAILED: TC2
java.lang.AssertionError: expected [hi] but found [hello]
```

2. **assertNotEquals():**
   - **assertNotEquals ()** method used to verify expected and actual result, if both results are not same then output is pass otherwise fail.

```
package assertionInTestNG;
import org.testng.Assert;
import org.testng.annotations.Test;
public class DemoClass1 {

    @Test
    public void TC1()
    {
    String exp="hi";
    String act="hi";
    Assert.assertNotEquals(act, exp, "actual and expected results are  match");
    }
    @Test
    public void TC2()
    {
    String exp="hi";
    String act="hello";
    Assert.assertNotEquals(act, exp, "actual and expected results are not match");
    }
            }
```

Output on Console:

```
[RemoteTestNG] detected TestNG version 7.4.0
PASSED: TC2
FAILED: TC1
java.lang.AssertionError: actual and expected results are  match did not expect [hi] but found [hi]
```

3. **assertTrue()**
   - assertTrue() method use to verify *condition are true or false*, if condition is true output is pass otherwise fail.

```
package assertionInTestNG;
import org.testng.Assert;
import org.testng.annotations.Test;
public class DemoClass2 {
```

```
	@Test
	public void TC1()
	{
	boolean result = true;
	Assert.assertTrue(result);
	}
	@Test
	public void TC2()
	{
	boolean result = false;
	Assert.assertTrue(result);
	}
}
```

Output on Console :

```
[RemoteTestNG] detected TestNG version 7.4.0
PASSED: TC1
FAILED: TC2
		java.lang.AssertionError: expected [true] but found [false]
```

4. **assertFalse()**
   - assertTrue() method use to verify condition are true or false, if condition is false output is pass otherwise fail.

```
package assertionInTestNG;
import org.testng.Assert;
import org.testng.annotations.Test;
public class DemoClass2 {

		@Test
		public void TC1()
		{
		boolean result = false;
		Assert.assertFalse(result);;
		}
		@Test
		public void TC2()
		{
		boolean result = true;
		Assert.assertFalse(result);
		}

	}
```

Output on Console:

[RemoteTestNG] detected TestNG version 7.4.0
PASSED: TC1
FAILED: TC2
java.lang.AssertionError: expected [false] but found [true]

5. **assertNotNull()**
   - assertNotNull() method is use to verify *component or text fields empty or not*, if it is not empty output is pass otherwise fail.

```
package assertionInTestNG;
import org.testng.Assert;
import org.testng.annotations.Test;
public class DemoClass3 {

    @Test
    public void TC1()
    {
    String str = "Mangesh";
    Assert.assertNotNull(str);
    }
    @Test
    public void TC2()
    {
    String str = null;
    Assert.assertNotNull(str);
    }
}
```

Output on Console:

[RemoteTestNG] detected TestNG version 7.4.0
PASSED: TC1
FAILED: TC2
java.lang.AssertionError: expected object to not be null

6. **assertNull():**
   - **assertNotNull()** method is use to verify component or text fields empty or not, if it is empty output is pass otherwise fail.

```
package assertionInTestNG;
import org.testng.Assert;
import org.testng.annotations.Test;
public class DemoClass4 {
```

```
@Test
public void TC1()
{
String str = null;
Assert.assertNull(str);
}
@Test
public void TC2()
{
String str = "Mangesh";
Assert.assertNull(str);
}
}
```

Output on Console:

```
[RemoteTestNG] detected TestNG version 7.4.0
PASSED: TC1
FAILED: TC2
java.lang.AssertionError: expected [null] but found [Mangesh]
```

7. **fail():**
   - This method is use to intentionally fail test method
   - In a test class if one of the method is fail then then testNG will stop execution of failed test method and other test method execution is continue.
   - In a test class in one of the test method is failed and that test method execution required for other test method execution then other test methods will be skipped.

```
@Test
public void TC2()
{
String exp="hi";
String act="hello";

Assert.fail();

}
```

Output ON Console:

```
[RemoteTestNG] detected TestNG version 7.4.0
FAILED: TC2
java.lang.AssertionError: null
```

Disadvantages of assert class:

- If a test class containing multiple test method, in one of the test method, multiple verification are present, while executing if one verification is failed then rest of the verification will not be verified & testNG will execute next method by failing verification field method.

```
package assertionInTestNG;
import org.testng.Assert;
import org.testng.annotations.Test;
public class DemoClass1 {
    @Test
    public void TC1()
    {
    String exp = "hi";
    String act = "hi";
    Assert.assertNotEquals(act, exp);
    Assert.assertEquals(act, exp);
    Assert.assertNotEquals(act, exp);
    }
```

Output on Console:

```
FAILED: TC1
 java.lang.AssertionError: did not expect [hi] but found [hi]
```

There are two types of assert:

1. Hard Assert
2. Soft Assert

1. Hard Assert:

- In this assert if the test case got failed at any of the line then further it won't execute that particular test case, directly it will mark it as failed and move on to the next test case.
- Hard Assert throws an **AssertException** immediately when an assert statement fails and test suite continues with next @Test.
- *For example:*

```
Assert. Fail ();
Assert.assetEquals (actual, Expected, message);
```

2. Soft Assert:

- This assert will complete all the test steps inside the @Test method but at the last it will mark that particular test case as pass or fail.
- To evaluate we have to call **assertAll ()** on that basis the result get evaluated.
- **To use soft assert first we have to create an object of Soft Assert Class.**

```
SoftAssert sa = new SoftAssert() ;
```

sa.assertEquals(loginerrorMSG, "Invalid credentials", "Text not got matched ");

- to decide the final status of the test case

sa.assertAll ();

Soft Assertion Vs Hard Assertion in TestNG

- In test automation, when we want to validate the results we use **Assertion class.**
- If the condition fails, subsequent steps will be aborted and test will be marked as failed. This type of assertion is *called as* ***Hard Assertion.***
- The disadvantage of Hard Assertion is when we have more assertions in a test and if the assertion at the beginning fails all the subsequent assertions will be not executed/validated.
- And if we create one test case for each assertion that is not good practice.

The differences between Assert and Verify are listed below:

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

The difference between Soft Assert and Hard Assert:

Hard Assert	**Soft Assert**
Throws an error immediately after the assert statement fails and carries out with the next test case of the suite.	Does not throw an error immediately when the assertion fails, collects them and carries out with the next validation.
This throws an **AssertError** instantly so handled with a catch block. After suite execution is completed the test is made as PASS.	This accumulates the errors in each @Test execution.
It does not allow further execution of test if the line containing hard assert gets failed.	Next steps would be executed even if the line containing soft assertion gets failed.
Whole test case gets failed if at least 1 hard assert fails.	When used in raw format, failed test step would also yield in Pass test script.

How to execute only failed test Cases:

- **failed.xml**
  - While executing the automation scripts, test cases may fail for several reasons.
  - To optimize our next runs, we need to re-run only failed test cases.
  - **Steps to execute failed.xml file**
    - Create *testng. xml* file under project folder.
    - execute testng.xml file
    - In the test-output folder >> testng-*failed. xml* file will be created.
    - execute "testng-failed. xml"
  - In this way we can execute fail test cases in TestNG class.
  - ***Reasons for fail TC***
    - environment issue
    - script error
    - bug

```
 package assertionInTestNG;
import org.testng.Assert;
import org.testng.annotations.Test;
public class DemoClass1 {

    //disadvantages of Assert class
    @Test
    public void TC1 ()   {
    String exp = "hi";
    String act = "hi";
    Assert.assertNotEquals(act, exp);
    Assert.assertEquals(act, exp);
    Assert.assertNotEquals(act, exp);
    }
}
```

```
<suite name="Failed suite [Default suite]" guice-stage="DEVELOPMENT">
  <test thread-count="5" name="Default test(failed)">
    <classes>
      <class name="assertionInTestNG.DemoClass1">
        <methods>
          <include name="TC1"/>
        </methods>
      </class> <!-- assertionInTestNG.DemoClass1 -->
    </classes>
  </test> <!-- Default test(failed) -->
</suite> <!-- Failed suite [Default suite] -->
```

TestNG Groups:

- If we want to execute the test cases in groups then we create the groups.
- By making groups we can execute the particular groups of @Test methods.
- In groups we can use include or exclude keyword to execute only specific group of a method and avoid the execution of unnecessary methods.
- Groups starts with ***tagname <groups>*** this tag is specified inside the **<test> tagname.**

```
package groupofTestCaseExecute;
import org.testng.Reporter;
import org.testng.annotations.Test;

public class Groups {

   @Test (groups = "Regression")
    public void testCase1 () {
            Reporter.log("TC1 Got Executed" , true);
    }

   @Test (groups = "Sanity")
    public void testCase2 () {
            Reporter.log("TC2 Got Executed" , true);
    }

   @Test (groups = "Sanity")
    public void testCase4 () {
            Reporter.log("TC4 Got Executed" , true);
    }

   @Test (groups = {"Regression" ,"Sanity" })
    public void testCase5 () {
            Reporter.log("TC5 Got Executed" , true);
    }
}
```

Output on Console:

```
TC1 Got Executed
TC2 Got Executed
TC4 Got Executed
TC5 Got Executed

===============================================
Suite
                    Total tests run: 4, Passes: 4, Failures: 0, Skips: 0
```

```
<suite name="Suite">
  <test thread-count="5" name="Test">
  <groups>
  <run>
       <include name="Sanity"></include>
       <include name="Regression"></include>
  </run>
  </groups>
    <classes>
                  <class name="groupofTestCaseExecute.Groups"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

- **INCLUDE & EXCLUDE:**
  - **Include keyword** is used to execute the group of specific name.

    ```
    <include name="Sanity"></include>
    ```

    In this case only Sanity Group will executed.

  - **Exclude keyword** is used to exclude the execution of specific group.

    ```
    <exclude name="Regression"></exclude>
    ```

    In this case Regression group not considered for execution.

    The tag for creating group is groups.
    It is in curly braces using double quote in front of annotation @Test.

    Syntax: `@Test (groups = "Sanity")`

    ```
    @Test (priority = 3, groups = {"Regression","Sanity" })
    ```

NOTE: If we use include & exclude for the same group or method then execution will give first preference to "exclude".

META GROUP - (GROUP OF GROUPS):

- Metagroup is a group of groups.
- We can say Meta group's includes the multiple groups.
- Meta Group is declare inside the <group> by using tag <define></define> </group>

```
<suite name="Suite">
  <test thread-count="5" name="Test">
  <groups>
  <define name="Core_Functionality">
```

```
        <include name="Regression"></include>
        <include name="Smoke"></include>
      </define>
      </groups>
        <classes>
          <class name="metaGroups.GroupingOfTestCases"/>
        </classes>
      </test> <!-- Test -->
    </suite> <!-- Suite -->
```

```
import org.testng.Reporter;
import org.testng.annotations.Test;
public class GroupingOfTestCases {

   @Test (groups = "Regression")
  public void testCase1 () {
                        Reporter.log("TC1 Got Executed" , true);
   }

   @Test (groups = "Sanity")
        public void testCase2 () {
                Reporter.log("TC2 Got Executed" , true);
        }

   @Test (groups = "Regression")
        public void testCase3 () {
                Reporter.log("TC3 Got Executed" , true);
        }
   @Test (groups = "Smoke")
        public void testCase4 () {
                Reporter.log("TC4 Got Executed" , true);
        }

   @Test (groups = {"Regression" ,"Smoke" })
        public void testCase5 () {
                Reporter.log("TC5 Got Executed" , true);
        }
}
```

Output on Console:

```
TC1 Got Executed
TC2 Got Executed
TC4 Got Executed
```

TC5 Got Executed

===
Suite
Total tests run: 4, Passes: 4, Failures: 0, Skips: 0

Parallel testing or parallel execution:

- Parallel testing or parallel execution, as the name suggests, is a process of running the test case parallel rather than one after the other.
- In parallel testing, the program's multiple parts (or modules) execute together, saving the testers a lot of time and effort.
- Use in test suite, after suite name immediate to write parallel=”tests”.
- To create 3 test cases and run parallel and to create multiple test cases.
- 3 chrome browser open for 3 test cases, not 1 chrome browser multiple tab not open. To open new chrome browser each test case in parallel testing.

```
<suite name="Suite" parallel="classes">
  <test thread-count="13" name="Parallel_Execution">
    <classes>
      <class name="parallelExecution.OpenChrome"/>
      <class name="parallelExecution.OpenBrowserEdge"/>
    </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

```
package parallelExecution;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.annotations.Test;

public class OpenChrome  {
        @Test
        public void openChrome() {
                System.setProperty("webdriver.chrome.driver",
"E:\\D_ChromeDriver\\chromedriver.exe");
            WebDriver driver = new ChromeDriver () ;

        driver.get("https://signup.zerodha.com/");
        }
}
```

```
package parallelExecution;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.edge.EdgeDriver;
import org.testng.annotations.Test;

public class OpenBrowserEdge {
    @Test
       public void openEdge() {

                //Set Property Key And Path

       System.setProperty("webdriver.edge.driver","E:\\D_EdgeDriver\\msedgedriver.exe
");
                // Object Creation
                WebDriver driver = new EdgeDriver () ;
                //Get Method
                driver.get("https://www.facebook.com/");
       }
}
```

Output on Console:

```
Only local connections are allowed.
Please see https://aka.ms/WebDriverSecurity for suggestions on keeping Microsoft Edge WebDriver
safe.

ChromeDriver was started successfully.
Dec 01, 2022 10:59:22 AM org.openqa.selenium.remote.ProtocolHandshake createSession
Dec 01, 2022 10:59:22 AM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
Microsoft Edge WebDriver was started successfully.
[1669872563.737][WARNING]: This version of Microsoft Edge WebDriver has not been tested with
Microsoft Edge version 105.
===============================================
Suite
Total tests run: 2, Passes: 2, Failures: 0, Skips: 0
```

MultiBrowser or Cross Browser Testing:

- To use @Parameters (“variableName / browserName”) before taking @Test annotation in the test class.
- In the test suite declare parameter name=”browserName / variableName” after test name.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="Suite">
  <test thread-count="5" name="Cross_Browser_Testing">
  <parameter name="browserName" value="edge"></parameter>
  <classes>
  <class name="parallelExecution.MultipleBrowserTest"></class>
  </classes>
  </test> <!-- Test -->
   </suite> <!-- Suite -->
```

```
package parallelExecution;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.edge.EdgeDriver;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;

public class MultipleBrowserTest {

	@Parameters("browserName")
	@Test
	public void launchBrowser (String browserName) {
		WebDriver driver = null;
		if(browserName.equals("chrome"))
		{
		System.setProperty("webdriver.chrome.driver",
"E:\\D_ChromeDriver\\chromedriver.exe");
		driver = new ChromeDriver();
		driver.get("https://signup.zerodha.com/");
		}
		else if (browserName.equals("edge"))
		{

	System.setProperty("webdriver.edge.driver","E:\\D_EdgeDriver\\msedgedriver.exe");
		driver = new EdgeDriver();
		driver.get("https://www.facebook.com/");
		}
		else {

			System.out.println("Choose Correct Browser");
		}
	}
		}
```

Output on Console:

```
[RemoteTestNG] detected TestNG version 7.4.0
Starting Microsoft Edge WebDriver 104.0.1293.63 (7dd22b4a7f122e6953a166b0388edf494d716c7d) on port 58802
To submit feedback, report a bug, or suggest new features, please visit https://github.com/MicrosoftEdge/EdgeWebDriver
Only local connections are allowed.
Please see https://aka.ms/WebDriverSecurity for suggestions on keeping Microsoft Edge WebDriver safe.
Microsoft Edge WebDriver was started successfully.
[1669873768.547][WARNING]: This version of Microsoft Edge WebDriver has not been tested with Microsoft Edge version 105.

===============================================
Suite
    Total tests run: 1, Passes: 1, Failures: 0, Skips: 0
```

- If we want to execute the same test with different browser at the same time then we have to add one more test tag in Test suite.
- Test tag name must be different.
- Also we have to change parameter value accordingly depends upon the execution.

- **Refer below XML file.**

```
<suite name="Suite" parallel="tests">
        <test thread-count="5" name="Cross_Browser_Test_Chrome">
         <parameter name="browserName" value="chrome"></parameter>
  <classes>
        <class name="parallelExecution.MultipleBrowserTest"></class>
  </classes>
  </test> <!-- Test -->
        <test thread-count="5" name="Cross_Browser_Test_Edge">
         <parameter name="browserName" value="edge"></parameter>
  <classes>
        <class name="parallelExecution.MultipleBrowserTest"></class>
  </classes>
  </test> <!-- Test -->
</suite> <!-- Suite -->
```

Difference between Junit and TestNG-Junit Old version of TestNG:

Parameter	Junit	TestNG
Supports Annotation	It does not support advanced annotation.	It supports advanced annotation.
Parallel test execution	JUnit does not support to run parallel tests.	TestNG can run parallel tests.
Dependency tests	The dependency tests are missing in JUnit.	Dependency tests are present in TestNG.
Grouping tests	Grouping tests together is not possible in JUnit.	Tests can be grouped together and run parallel.

Listeners:

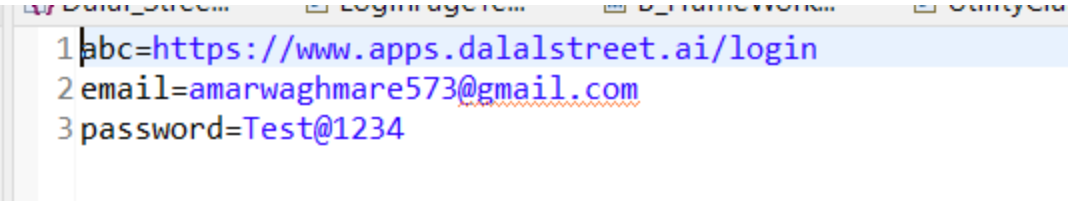
- TestNG provides the *@Listeners annotation* which listens to every event that occurs in a selenium code.
- Listeners are activated either before the test or after the test case.
- It is an interface that modifies the TestNG behavior.
- *For example, when you are running a test case either through selenium 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.*
- Listeners are implemented by the ITestListener interface.
- *An ITestListener interface has the following methods:*
    1.onTestStart(),
    2. onTestSuccess(),
    3. onTestFailure(),
    4. onTestSkipped(),
    5.onStart(),
    6. 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 tag after and before tag.
    <listener class-name=" packageName.ClassName "/> </listeners>

Property file Selenium (Object Repository):

- An object repository is a common storage location for all objects.
- In Selenium WebDriver context, objects would typically be the locators used to uniquely identify web elements.
- The major advantage of using object repository is the segregation of objects from test cases.

- If the locator value of one WebElement changes, only the object repository needs to be changed rather than making changes in all test cases in which the locator has been used.
- Maintaining an object repository increases the modularity of framework implementation.
- Selenium WebDriver does not offer an in-built object repository by default.
- However, object repositories can be built using the key-value pair approach wherein the key refers to the name given to the object and value refers to the properties used to uniquely identify an object within the web page.
- Property file in selenium is used to store the URL, UN and PWD and then call & this important credential can't store in excel sheet.
- Excel sheet store only the test case data.
- Store the data in the property file in terms of keys and values.
- To create a common function in the utility class to read data.
- How to create property file in selenium.
- Creating a properties file in eclipse:
    - Right-click on the main project folder and Select New->> Other->select General -> >File and click on 'Next' button->
    - Provide a valid file name with the extension '.properties' on the new file resource window and click on 'Finish' button.
- Storing data on to properties file:
    - Data is stored in properties file in the form of key-value pairs, with the key being unique across the file.
    - Open file in Eclipse and store some data
    - e.g.- URL= https://kite.zerodha.com/
- **Reading data from properties file:**

```
package com.testData;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
public class PropertiesFileUse {

 public static String propertiesfile (String value) throws IOException {

                Properties prop = new Properties();

 FileInputStream  fis = new FileInputStream(System.getProperty("user.dir") +
"//dalalStreet.properties");

        prop.load(fis);

        return prop.getProperty (value);

       } }
```



Parameterization:

- The process of fetching information from the external source and using it into selenium test script is called as *'Parameterization'*.
- To perform the action of parameterization we need to import the third party tool that *"**Apache POI**"*.
- APECHE POI- Apache provides a very famous library POI, by using this library. We can read & write the **XLSX** file format **Excel**.
- **CLASSES TO HANDLE EXCEL-SHEET**
    - To fetch the data from excel sheet we have to use the class.

      FileInputStream f = new FileInputStream (path);
    - To export the data to the excel-sheet we have another class.

      FileOutputStream j = new FileOutputStream (path);

- **Methods:**
    - ***getStringCellValue();*** - it will fetch the string data from the excel sheet.
    - ***getNumericCellValue();*** - it will fetch only numeric value from the excel sheet.
    - ***getCellType();*** - it will help us to check which type of data stored in particular cell.
    - ***getLastCellNumber();*** - it will shows the last cell number in excel sheet.
    - ***getRow();***
    - ***getCell();***

- **DIFFERENT INTERFACES**
    - **Workbook**- This interface has two implementation classes
      HSSFWorkbook- it represents .xls file
      XSSFWorkbook- it represents .xlsx file

    - **Sheet**- It is also an interface; it has to two implementation classes.
      HSSFSheet- it represents .xls file
      XSSFSheet- it represents .xlsx file

    - **Row**- It is also an interface; it has to implementation classes.
      HSSFRow- it represents .xls file
      XSSFRow- it represents .xlsx file

    - **Cell**- It is also an interface; it has to implementation classes.
      HSSFCell- it represents .xls file
      XSSFCell- it represents .xlsx file.

To fetch the data from excel sheet:

```
public static  String readData (int row , int column)throws IOException  {

        String path =".\\TestData\\MavenProjectTestData.xlsx";

FileInputStream excelfile = new FileInputStream(path);

        //to load the workbook

        XSSFWorkbook workbook = new XSSFWorkbook(excelfile);

        // to get the Sheet from the workbook

        XSSFSheet sh1 = workbook.getSheetAt(0);

        String value = sh1.getRow(row).getCell(column).getStringCellValue();

        System.out.println(value);
        return value;

}
```

To Write the data into the excel sheet:

```
        public static void writeData ( int row , int column , String value)throws IOException {

                String path =".\\TestData\\MavenProjectTestData.xlsx";

    XSSFWorkbook workbook = new XSSFWorkbook();

   XSSFSheet sheet = workbook.createSheet(value);

    sheet.createRow(row).createCell(column).setCellValue(value);

      FileOutputStream fout = new FileOutputStream(path);

      workbook.write(fout);
        }

}
```

How to generate email able report in TestNG:

- **Email able Report:**
  - Report generation is very important when we are doing the automation testing as well as for manual Testing.
  - By looking at the result, we can easily identify how many test cases are passed, failed and skipped.
  - By looking at the report, we will come to know what the status of the project is.
  - Selenium web driver is used for automating the web-application, but it won't generate any reports.
  - The TestNG will generate the default report.

- ***Steps to generate Emailable report:***
  - Execute Test class and refresh the project.
  - You will get test-output folder.
  - In That folder Right click on the *"emailable-report.html"* and select the option Open with the web browser or double click on it.

- If we use sop () to display text as an output then result will be displayed in console not in email able report.
- To display text in email able report we need to use static method log present in Reporter class.
- For example : Reporter.log("String msg", true)

POM (Page object module) with page factory:

- Page object model (POM) is a design pattern, popularly used in test automation that creates object repository for web UI elements.
- The advantage of the model is that it reduces ***code duplication and improves test maintenance.***
- It is a java design pattern use for design of classes in test script.
- Page object model is an object design pattern in selenium, where
  - **Web pages are represented as classes & the various elements on the page are defined as variables on the class.**
- In this case we will use *page factory* to initialize web elements that are defined in web page classes.
- **POM Strictly follows encapsulation concept where**
  - **Declaration:** Data member should be declared globally with access level modifier as private.
  - **Initialization:** Initialize within a constructor with access level public using page factory.
  - **Utilization:** Utilize within a method with access level modifier as public.

- **Important Note:**
  - No of data member that need to be created under a **POM** class will depends on no of element that need to be handle in a webpage.
  - **POM** class will not contain a main method, to run a **POM** class we require another class with main () i.e**. Test class**.
  - Test class will contain all the navigation steps to test an application.

- **POM class:**
  - **POM** classes depends on webpages present in an application.
  - For each webpage **POM** class will be created, no of **POM** class depends on no of webpages present in an application.
  - In each **POM** class data member/variable are created in encapsulation concept by using page factory.

- No of data member created in **POM** class will depend on no of elements present in a webpage.
- Each declared data member should initialized & utilized in **POM** class.

- **Pagefactory:**
  - It is a class which contains static method like **initElements.**
  - To initialize data member /variable in Page Factory we need to use initElements method within the constructor.

Syntax:

PageFactory.initElements (driver, this);

- **InitElements** will initialize data member by identifying each component present in a webpage by using **@findBy** annotation, which takes locator type as an input.

- **Syntax:** @FindBy(locator Type ="locator value/expression")
  private WebElement data_member;

- **Working of Page Factory:**
  - While executing test script **initElement** method will convert all the data members **@findBy** *annotation* **to findElement (),** this process is known as ***basic/early initialization***.--after creating object of **POM** class.

```
//DECLARATION
@FindBy(xpath = "//*[@name='email']")
private WebElement email_txt_Box;

/UTILIZATION
public void verifyLogin () throws IOException, InterruptedException {
email_txt_Box.sendKeys("Shinde@@@@@");

}
```

OR

```
//DECLARATION
private By select_for_company= By.xpath("//a[contains(text(),'HDFC.NS')]");


//UTILIZATION
public void selectofCompany () {

driver.findElement (select_for_company).click ();
}
```

- To perform action on component we need to *call a methods.*
- Before performing each action initElement method will identifies component present or not, then it will do complete initialization this process is known **as late/lazy initialization.**

- **Disadvantage of POM :(why use POM with page factory?**
  - **POM** will initialize all the component before performing actions, but sometimes application may contains few components which will be hidden & displayed once we perform action on components, that hidden component wil**l** not be displayed while **POM** initializing, so it throws **"No such element" exception.**
  - To overcome drawback of **POM**, we need to use "**PageFactory**" which is an extension of **POM**.

- **Advantages:**
  - Reduces code duplication
  - Improve test maintenance
  - Makes the code reusable
  - It makes ease in maintaining the code
  - Makes code readable
  - The code becomes less and optimized

Difference between POM and Page Factory:

POM	**Page Factory**
It will initialize all the data member present in class completely before performing action on components.	It will initialize all the data member present in class performing each action.
It will use if webpage is not containing hidden element.	Hidden element It will use if webpage is containing hidden element.

@FindBy:

- An annotation used in page factory to locate and declare web elements using different locators.

InitElements ():

- InitElements is a static method in **Page Factory** *class.*
- This method which accept the two parameter such as driver and this keyword.
- Using the initElements method, we can initialize all the web elements located by @FindBy annotation.

How to capture screenshot of only failed test cases?

- To get test case status as pass, failed or skip, we need to use listener (interface) in TestNG.
- To check status of test case pass, failed and skip we need to use if else to compare the status.
- If status is pass no need to capture screenshot, if test case results failed so that time to use *if else* and *call utility class method capture screenshot.*
- Each test case is execute that time *it goes to @After Method* and to apply the condition so those test cases is run to *get status of that run method* and compare *status pass or fail*.
- If it is pass no problem but it is fail so that time to pass the code of *capture screenshot.*
- To add this code in *the @AfterMethod,* before that we have to declare which *listener* which we have to use here because multiple listener are present in TestNG.
- To get the status of the test case for that purpose we need to use **ITestResult** *inside* the *constructor of @AfterMethod it* is an interface or listener.
- This is give the status and store purpose we need to take one object immediate **ITestResult.**
- In this object to store the result pass, fail and skip.
- Result store in this object so by using this result we need to use if else statement.

```
@AfterMethod
    public void tearDown (ITestResult result) throws InterruptedException, Exception {

        Thread. Sleep(4000);

        ///Screenshots code placed in utility package. Call here
        ScreenShot.capturescreenshot (driver, "failure");

        driver. Quit ();
    }
```

Where to use encapsulation in your project?

- At the time of POM design we declare private before declaring the variable that time we use encapsulation concept in our project.

NullPointerException:

- To get the data from excel sheet and to enter some index (0, 5) but that index data is not present so that time get NullPointerException.

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:**
    1. To build the project.
    2. To manage the project.
    3. To define the project structure.
    4. Test management.
    5. Dependency build.
    6. 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:**

- 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.
        1. In overview tab, there are artifact details and project details.
        2. Details like group id, artifact id, version, project name, URL.
        3. In dependencies tab, there are dependencies which added in our project.
        4. We can add it and remove it as per our requirement.
        5. Dependencies hierarchy tab, where as its name says hierarchy of dependencies are there.

6. 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>FrameWork002727</groupId>
  <artifactId>Maven0027</artifactId>
  <version>0.0.1-SNAPSHOT</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 to declare the web element and we have to use static method like **initelement** to initialize the data member in page factory.
  - 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.

Maven Build Life Cycle:

- **Maven Build Life Cycle :**
  - Maven is based around the **central concept of a build lifecycle.**
  - The process for building and **distributing artifacts (projects)** is clearly defined.
  - **There are 3 *built in* build lifecycle.**
  - The **default lifecycle** handles your *project* **deployment**.
  - The **clean lifecycle** handles *project* **cleaning**.
  - While the **site lifecycle** handles the **creations of projects site documentation.**

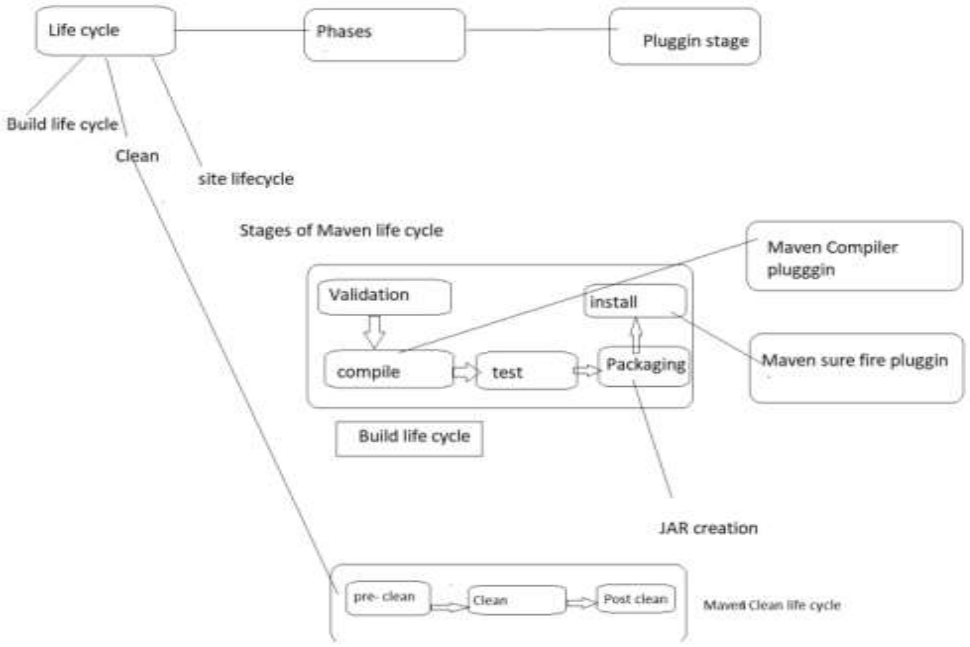
- **A Build life cycle is made up of phases:**
  - **The default life cycle comprises of the following phases:**
  - **Validate***:* Validate the project is correct and all the necessary information is available.
  - **Compile***:* Its compiles the source code of the project. It's usage maven usage compiler plugin.
  - **Test***:* test the compiled source code using suitable unit testing framework. These tests should not require the code be packaged or deployed.
  - **Package***:* Take the compiled code and package in its distributable format, such as JAR etc.
  - **Verify***:* run any checks on results of integration test to ensure quality criteria are met.
  - **Install***:* install the package into the local repository, for use as a dependency in other projects locally.
  - **Deploy***:* done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

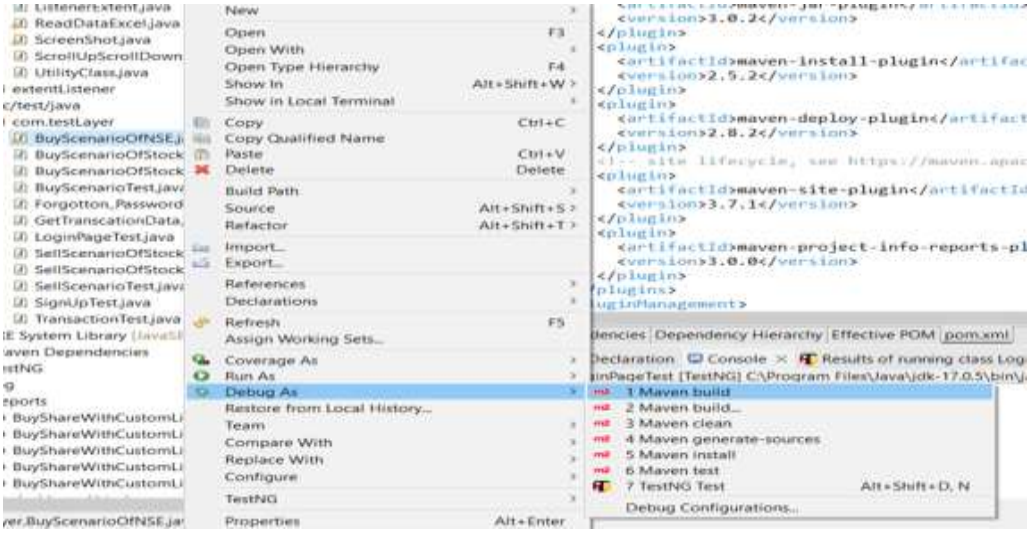
- **A build phase is made up of plugin goals:**
  - A plugin goals represents a specific task which contributes to the building and managing the project.
  - It may be bound to zero or more build phases.
  - A goal not bound to any build phase could be executed outside of the build lifecycle by direct invocation.
  - The order of executions is depends on the order in which the goal (s) and the build phase are invoked.

- ## Maven Commands:

1. **mvn clean:**

   - This command cleans the maven project by deleting the target folder or directory.
   - $ mvn clean
   - [INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ maven-example-jar ---





2. **mvn compiler: compile**

   - This command compiles the java source classes of the maven project.
   - $ mvn compiler:compile
   - [INFO] --- maven-compiler-plugin: 3.8.1:compile (default-cli) @ maven-example-jar ---

- [INFO] Changes detected - recompiling the module!

3. **mvn compiler: testCompile**

   - This command compiles the test classes of the maven project.
   - $ mvn compiler:testCompile
   - [INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-cli) @ maven-example-jar ---
   - [INFO] Changes detected - recompiling the module!

4. **mvn package:**

   - This command builds the maven project and packages them into a JAR, WAR, etc.
   - **mvn package**

5. **mvn install:**

   - This command builds the maven project and installs the project files (JAR, WAR, pom.xml, etc.) to the local repository.
   - **mvn install**

6. **mvn deploy**

   - This command is used to deploy the artifact to the remote repository.
   - The remote repository should be configured properly in the project pom.xml file distribution Management tag.
   - The server entries in the maven settings.xml file is used to provide authentication details.

7. **mvn validate**

   - This command validates the maven project that everything is correct and all the necessary information is available.

Chrome Options commands in Selenium Web driver:

Chrome Options commands:

1. **start-maximized:** Opens Chrome in maximize mode.
2. **incognito***:* Opens Chrome in incognito mode.
3. **headless***:* Opens Chrome in headless mode.
4. **disable-extensions:** Disables existing extensions on Chrome browser.
5. **disable-popup-blocking:** Disables pop-ups displayed on Chrome browser.
6. **make-default-browser:** Makes Chrome default browser.

7. **version***:* Prints chrome browser version.
8. **disable-infobars***:* Prevents Chrome from displaying the notification 'Chrome is being controlled by automated software.

```
//Alternative against System and set property..
Add dependency of webdriverManager

        WebDriverManager.chromedriver().setup ();
        ChromeOptions options = new ChromeOptions();

        options.addArguments("--start-maximized");
        options.addArguments("--headless");
        options.addArguments("--disable-notifications ");

        options.addArguments("--disable-popup-blocking");
        options.addArguments("--disable-extension");
        options.addArguments("--product-version");
        driver = new ChromeDriver (options) ;
```

Framework:

- Framework is nothing but systematic and sequential way to write the test scripts which are written for the automation testing of the application, software.
- It defines some set of rules and while writing the script we have to follow those rules for our beneficial & to achieve the particular results.
- Framework is structural way for building and automating the application.
- **Why to use Selenium Framework**
- Framework is a structural way for building/automating application which provides
    1. Easy code maintenance.
    2. Increase code reusability.
    3. Higher code readability.
    4. Reduced script maintenance cost.
    5. Reduced tests time execution.
    6. Reduced human resources.
    7. 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
9. Continuous Integration: JENKINS

- **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 Selenium 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 test cases 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 and keyword driven framework.
  - I am 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 both positive and negative test cases.
  - For scripting we are using maven project.

- **Log Generation**
  - To generate the log in selenium framework we need to make use of log4j jar file.
  - Log4j is an open source logging framework(API)
  - By using log4j we can store selenium automation flow logs in file or database.
  - **Log4j has 3 principle components:**
    1. Loggers
    2. Appenders
    3. Layout

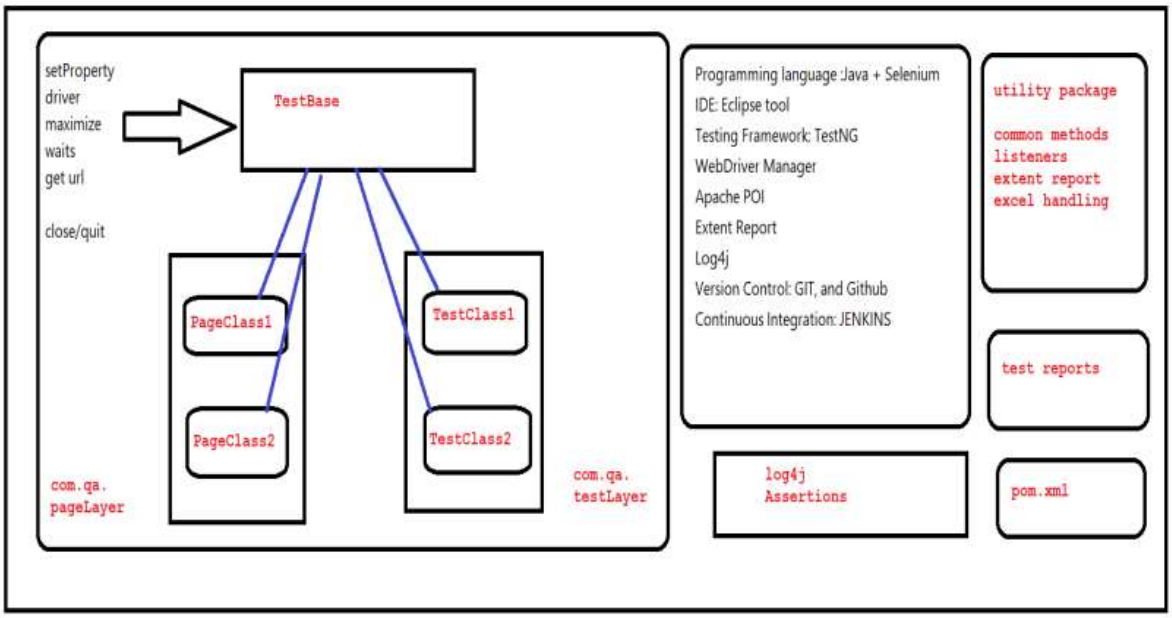
1. **Loggers**
    - It is responsible for logging information.
    - We use Logger's methods to generate log statements.

2. **Appenders**
    - Apache log4j provides Appender objects which are primarily responsible for printing logging messages to different destinations such as consoles, files, sockets, NT event logs, etc.
    - Use to deliver log events to their destination.
    - Use to write logs in a file.

3. **Layout**
    - Layout class and overrides the format () method to structure the logging information according to a supplied pattern.
    - It is responsible for logging information in different styles.

- **Extent Reports:**
    - Extent Reports is an open-source reporting library useful for test automation.
    - It can be easily integrated with major testing frameworks like JUnit, NUnit, TestNG, etc.
    - These reports are HTML documents that display results as pie charts.
    - They also allow the generation of custom logs, snapshots, and other customized details.
    - Extent reporter is a customized HTML report which can be integrated into selenium web driver using TestNG framework.

- **Advantages**
    - Customizable HTML report with stepwise & pie chart representation.
    - Display the time taken for test case execution in report.
    - Each test step can be associated with a screenshot.
    - Easily integrated with TestNG.
    - Multiple test cases runs can be within suite can be tracked easily.



Oops Concept used in framework:

1. **INTERFACE:**

   WebDriver driver = new ChromeDriver ( );

   In this statement WebDriver is nothing but the interface in selenium.

2. **UPCASTING:**

   WebDriver driver = new ChromeDriver ( );

   In the above Statement is nothing but UPCASTING in Selenium.

3. **INHERITANCE:**
   - We create a base class in the framework to initialize WebDriver interface, WebDriver waits, Property files etc. in the base class.
   - We extends the Base class in tests class. That is nothing but inheritance in Selenium Framework.

4. **Polymorphism:**
   - Combination of overloading and overriding is known as Polymorphism.

   1. **METHOD OVERLOADING:**
      - We use implicit wait in selenium. Implicit wait is an examples of overloading.
      - In implicit wait we use different time stamps such as **SECONDS, MINUTES,** and **HOURS** etc.

2. **METHOD OVERRIDING:**

   - Declaring a method in child class which is already present in the present class is called METHOD OVERRIDING.
   - Examples are get and navigate methods of different drivers in Selenium.

5. **ENCAPSULATION:**

   - All the POM classes in a framework are an example of Encapsulation. In POM classes,
   - We declare the data members using @Find By and initialization of data members will be done using constructor to utilize those in methods.
   - Encapsulation is a mechanism of binding code and data together in a single unit.
   - Encapsulation is the process of wrapping up code and data together in a single unit. It is used to hide the data of a class from another class.
   - Encapsulation can be achieved when we declare all variables as private and use that variable into public method in a class to get the values of the variable.

6. **ABSTRACTION:**

   - In Page Object Model design pattern, we write locator (such as id, name, Xpath etc.) in page class.
   - We utilize these locators in POM class but we can't see these locators in the tests. Literally we hide the locators from the tests.
   - Abstraction is a methodology of hiding the implementation of internal detail and showing the functionality to the users.

Framework Explanation:

- Basically framework is nothing but the **systematic** way to write test scripts to **automate the web based application.**

- So currently I am a working on the **hybrid driven framework** in which I am involved in data driven framework.

- As name suggest data driven means executing test cases with **the help of test data** which is stored in some specific format like table or in excel sheet.

- In data driven framework we have several components such as we are using java as a programming language, for test script writing we are **using *Eclipse IDE***, Also for the test data we have separate folder for ***excel sheet*** because *test data* is in the table format.

- Also we are using **properties file** that stores the information that remain throughout the framework such as ***browser specific information, application URL , security questions*** and ***answers , screenshot path , password.***

- Each parameter in the properties file is stored as *pair of strings*, in key-value pair format *(key=value)* where each key is on one line.

- We have another component as **reports or log**. We are using ***extend report*** for reporting.

- For maintaining the log we **have *log4j API*** *implementation* which maintains logs for test execution.

- For scripting we are using **the *maven project*** which is used to ***build and manage the projects.***

- If we want to build the java projects ***into executable jar files*** then we can use maven project. Able to work with multiple project at same time so maven is basically ***build automation tool.***

- In maven we can add *dependencies* **from *remotes repository*** of maven and stored into local cache and use according to our requirements. The dependencies are download automatically over the internet.

- In maven project, here **we have separate packages for tests and pages**. All the web page related classes are comes under the page packages and all the test classes are comes under the test packages.

- Maven is based on page object model. We are using page object model with page factory to avoid ***stale_element_reference_Exception.***

- AS we know POM is design pattern to ***create object repository*** for web UI elements.

- It has advantage of to **reduce *code duplication and improves test maintenance.***

- There is a clean separation between ***test code and page specific code.***

- In page factory, we are using ***@FindBy annotation*** to declare the web element. We initialize that web element in the constructor.

- In POM, we have maintained a class for every webpage. **Each web page has a separate class and that class holds the functionality and members of that web page**. Also we have separate classes for every individual test classes.

- As per the structure of maven project all the test are kept in ***src/test/java*** and remaining files such as properties file, POM classes, test data, utility files are comes under the ***src/main/java.***

- POM supports multiple *concepts of OOPs* in java such as:

- **We are declaring the *web element as* variable and we declare the variable as *private* and use that variable in *public* method so here we achieve *encapsulation.***

- **Also we have *base class* for *initialize the browser , waits, properties files etc.* and we extends the *base* class so here we achieve *inheritance.***

- **Also in *waits* we have time configurations in *seconds, minutes,* and *hours* so here we achieve *polymorphism.***

- Also we using the ***web driver* which is *an interface***. Also we perform *up casting* by creating the *reference variable* of the *web driver* interface as driver and making the object the browser class.

- For test script we are using *testNG* which is inspired by Junit by adding new functionalities which made testNG more powerful than other framework.

- TestNG have some features like we can run the *test cases in groups* according to the requirements **also we can *control the execution of test cases*.**

- We can executes the test classes or *test cases parallel* by doing modifications *in the xml file.*

- In testNG we have ***annotations and keyword***. Annotations *control the line of code* that we have *written in the method.*

- So we are using ***@symbol*** before the method.

- By using keywords we are executing the **test cases *on priority***, we can add ***dependencies*** over the test cases, and also if we *don't want to consider the test case* to execute here we are using *enabled keyword.*

- **If any of test case got *failed due to time issue***, if it requires *more time* then we use ***timeout*** *keyword*, also if we want to execute the particular test case multiple time then we use ***invocation count***.

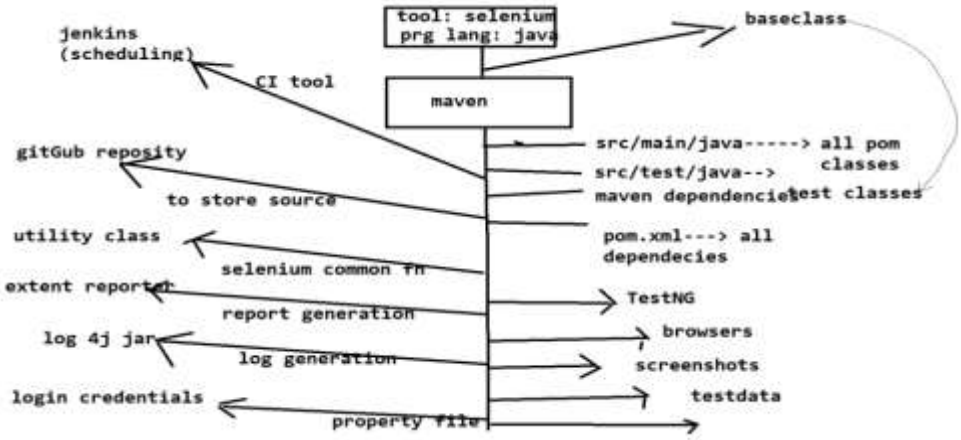
- TestNG does not require a main method to run and methods written need not be static.

- Also we are using ***Jenkins tool*** for run the test *script or project over remote server*.

- It is basically an open source ***continues integration/continuous delivery* and *deployment (CI/CD)*** automations *software DevOps tool* written in the Java programming language.

- And for maintaining the source code we are using version control tool such as *Git remote repository.*
- That's All About my framework on which I have worked. Thank You…

Framework Explanation another example:



How to explain your selenium framework?

- In our framework we use the automation tool as a selenium and programming language java.
- In our project we use the maven project.
- Inside maven project we store all the POM processing, **src/main/java**- in that we store the all the POM classes**, src/test/java**- in that we store the all the test classes, **maven dependencies**- to automatically create ajar file when we add the dependency in the pom.xml, **pom.xml**- to add/contains all the POM selenium, testNG, apache poi etc.
- To design a test class **we use our project java unit supporting framework like testNG.**
- So the main advantage to use testNG in our framework like by using testNG we can **run/check multiple test classes** at the same time by using test suite, it can **generate email able report,** by using testNG we can perform the **parallel execution** also compatibility testing so for that reason we use the testNG in our framework.
- In our maven project there are different folders **like browser, screenshot, and test data** etc. so in the browser folder we store the chromedriver.exe file, geckodriver.exe file etc.

- Then in the **screenshot folder** to store a capture screenshot when our test script is fail and in the test data folder we store the excel sheet file in that excel sheet file we store test data.
- In our project one super most class is present that is a base class.
- This base class is inherited means extends to each and every test class.
- To store the important login credential of application we use property file in our framework. Important login credential of applications like **URL, UN, and pwd etc**. also to generate the logs we use the log4j jar files.
- When we run the test class so every time logs are store in the separate folder.
- To generate the report we use the **Extent reporter** in our selenium framework.
- In our framework there is one important class that is utility class.
- When we create the common system related function **likes fetching data from excel sheet, capture screenshot also handling iframe and web table.**
- Create a generic function in that class. To create selenium related common function in that utility class.
- In our project we use the **GitHub as a version control system.**
- By using this GitHub we store the all the daily source code in to central repository.
- In project or framework we use the continuous integration tool like Jenkins the main use of this for scheduling purpose.
- It is also configure to GitHub repository to continuous monitor.

Questions on TestNG:

1. **What are the advantages of using TestNG?**
   - We can decide whether testcase is passed or fail or skipped on a particular condition.
   - After the execution email able report is available.
   - Description can be added for a test case.
   - We can group the testcase according to the requirement.
   - We can decide the priority i.e. which testcase will execute first and which one at last.
   - We can create multiple test cases by using multiple data.
   - We can execute the test cases in parallel.
   - It has different assertion that help to check expected and actual result.
   - TestNG generates XML and HTML report and allow to open in web browser

2. **What is the importance of testng.xml file?**
   - TestNG.xml file is an XML file which contains all the Test configuration and this XML file can be used to run and organize our test.

3. **What is TestNG Assert and list out common TestNG Assertions?**
   - Asserts helps us to verify the conditions of the test and decide whether test has failed or passed. Some of the common assertions are:

```
AssertEqual (String actual, String expected)
AssertTrue (condition)
AssertFalse (condition)
```

4. **How to disable a test case in TestNG? How to ignore a test case in TestNG?**

```
@Test (enabled = false)
public void test ()
{
 System.out.println ("This test is disabled");
}
```

5. **What are the different ways to produce reports for TestNG results?**
   - There are two ways to generate a report with TestNG − Listeners − for implementing a listener class, the class has to implement the *org.testng.ITestListener* interface.
   - These classes are notified at runtime by TestNG when the test starts, finishes, fails, skips, or passes.
   - Reporters − for implementing a reporting class, the class has to implement an *org.testng.IReporter* interface.
   - These classes are called when the whole suite run ends.

6. **List out various ways in which TestNG can be invoked?**
   - Command Line
   - Maven
   - IDE

7. **What is the use of @Test (invocationCount=x)?**
   - The invocation count attribute tells how many times TestNG should run a test method.
   - In this example, the method test Case will be invoked ten times:

```
@Test (invocationCount = 10)
 public void testCase ()
{
System.out.println ("Invocation method");
}
```

8. **What does the test timeout mean in TestNG?**
   - If one of the test case is taking a long time due to which other test case are fail.
   - To overcome such situation we have to use timeout keyword with duration in seconds.
   - The timeout is a time period provided to test case to completely execute the test case.

```
@Test (timeout=5000)
public void executeTimeOut () throws InterruptedException
{Thread.sleep (3000) ;}
```

9. How is TestNG better than JUnit?

- TestNG supports more annotations than Junit.
- TestNG supports ordering of tests but Junit doesn't.
- TestNG supports various types of listeners using annotations but Junit doesn't.
- TestNG reports are better than Junit.

10. What is the difference between Assert and Verify commands?

- In case of the "Assert" command, as soon as the validation fails the execution of that particular test method is stopped and the test method is marked as failed.
- Whereas, in case of "Verify", the test method continues execution even after the failure of an assertion statement.

11. What is the use of property file in Selenium?

- Property file can be used to store the different web elements of an application or to store all the different application, framework configurations.

12. How can we run test cases in parallel using TestNG?

- By using the parallel attribute in testng.xml.
- *The parallel attribute of suite tag can accept four values: –*
  - **tests***:* All the test cases inside tag of testing xml file will run parallel.
  - **classes***:* All the test cases inside a Java class will run parallel
  - **methods***:* All the methods with @Test annotation will execute parallel.
  - **instances***:* Test cases in same instance will execute parallel.

13. What could be the cause for Selenium WebDriver test to fail?

There could be many reasons for test failure. Some of them are listed below: –

- Driver is null or not found.
- Element is not found on the page.
- Element is present but not intractable.
- Page synchronization issues.

14. What are Annotations and what are the different annotations available in TestNG?

- Annotations in TestNG are lines of code that can control how the method below them will be executed. They are always preceded by ***the @ symbol.***
- Here is the list of annotations that TestNG supports.
- **@BeforeSuite:**
  The annotated method will be run only once before all tests in this suite have run.
- **@AfterSuite:**
  The annotated method will be run only once after all tests in this suite have run.
- **@BeforeClass:**
  The annotated method will be run only once before the first test method in the current class is invoked.
- **@AfterClass:**
  The annotated method will be run only once after all the test methods in the current class have run.

- **@BeforeTest:**
  The annotated method will be run before any test method belonging to the classes inside the tag is run.
- **@AfterTest:**
  The annotated method will be run after all the test methods belonging to the classes inside the tag have run.
- **@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.
- **@BeforeMethod:**
  The annotated method will be run before each test method.
- **@AfterMethod:**
  The annotated method will be run after each test method.
- **@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 data provider needs to use a dataProvider name equals to the name of this annotation.
- **@Factory:**
  Marks a method as a factory that returns objects that will be used by TestNG as Test classes.
  The method must return Object [ ].
- **@Listeners:**
  Defines listeners on a test class.
- **@Parameters:**
  Describes how to pass parameters to a @Test method.
- **@Test:**
  Marks a class or a method as a part of the test.

15. What is the use of @Listener annotation in TestNG?

- Listener is defined as interface that modifies the default TestNG's behavior.
- As the name suggests Listeners **"listen"** to the event defined in the selenium script and behave accordingly.
- It is used in selenium by implementing Listeners Interface.
- It allows customizing TestNG reports or logs.

16. There are many types of TestNG listeners available: -

1. IAnnotationTransformer
2. IAnnotationTransformer2
3. IConfigurable

4. IConfigurationListener
5. IExecutionListener
6. IHookable
7. IInvokedMethodListener
8. IInvokedMethodListener2
9. IMethodInterceptor
10. IReporter
11. ISuiteListener
12. ITestListener

17. What are the different types of keyword you have used in TestNG framework?

In TestNG we have 5 keywords:

1. **priority**
2. **enabled**
3. **invocation count**
4. **timeout**
5. **dependsOnMethod**

18. What is TestNG Assert and list out some common assertions supported by TestNG?

- Asserts helps us to verify the conditions of the test and decide whether test has failed or passed. A test is considered successful ONLY if it is completed without throwing any exception. Some of the common assertions are: -
    1. **AssertEqual**
    2. **AssertTrue**
    3. **AssertFalse**

19. What is the use of @Factory annotation in TestNG?

- A factory will execute all the test methods present inside a test class using separate instances of the class.
- It is used to create instances of test classes dynamically.
- This is useful if you want to run the test class any number of times.

Questions on TestNG Framework:

20. How do you decide a particular framework for a project?

- To help determine which framework is right for your organization, ask yourself the following helpful questions:
    1. Consider the application and the technology involved.
    2. Think about testing requirements.
    3. Determine license cost of the tool.
    4. Evaluate the skill sets available within your organization.

21. What are the main traits of a good Software Test Automation framework?

- Below are some of the key parameters that a software tester needs to keep in mind, while developing a test automation framework.
    1. Handle scripts and data separately.
    2. Create libraries.
    3. Follow coding standards.
    4. Offer high extensibility.
    5. Less maintenance
    6. Script/Framework version control

- **Tips:**
    1. Proper knowledge of Programming language concept
    2. Write page classes for all the pages
    3. Test Classes may be differ based on the requirements
    4. Write validations (Assertions) in the test class only
    5. Write the user actions In Page class only
    6. Use proper waiting mechanism in page class (user actions) only
    7. Usage of proper Naming convention
    8. Always add the comments when you create new class
    9. All test cases should be independent as possible

22. What is Page Factory?

- Page Factory class in selenium is an extension to **the Page Object Design pattern.**
- It is used to initialize the elements of the page object or instantiate the page objects itself.
- **Annotations in Page Factory are like this:**

```
@FindBy (id = "username")
WebElement txt_UserName;
```

- ***We need to initialize the page object like this:***
    - PageFactory.initElements (driver, Login. Class);

23. What is the difference between Page Object Model and Page Factory?

- Page Object Model is a design pattern to create an Object Repository for web UI elements.
- However, Page Factory is a built-in class in Selenium for maintaining object repository.

24. How do you generate reports using testing?

- Report generation is very important when you are doing the Automation Testing as well as for Manual Testing.
- By looking at the result, you can easily identify how many test cases are passed, failed and skipped.
- By looking at the report, you will come to know what the status of the project is.

- Selenium web driver is used for automating the web-application, but it won't generate any reports.
- There are two ways we can generate reports in *testNG*:
- *Using emailable-report.html* - The TestNG will generate the default report.
- When you execute testng.xml file, and refresh the project. You will get test-output folder in that folder.
- Right click on the emailable-report.html and select the option. Open with the web browser.
- *Using index.html* - When you execute testng.xml file, and refresh the project. You will get test-output folder in that folder.
- Right click on the index.html and select the option. Open with the web browser.

25. What is TestNG Assert and list out some common assertions supported by TestNG?

- Assertions in TestNG are a way to verify that the expected result and the actual result matched or not.
- If we could decide the outcome on different small methods using assertions in our test case, we can determine whether our test failed or passed overall.
- An example of assertion can be logging into the website, checking the title of the webpage, verifying the functionality of an input box that takes only integers, etc.
- We should remember that an assertion in TestNG is successful only if there are no exceptions thrown during the test case execution.
- TestNG asserts (or assertions) popularly validate the results in TestNG using selenium.

26. How to create and run TestNG.xml?

- **testng.xml** file is a configuration file in TestNG.
- It is used to define test suites and tests.
- It provides different options to include packages, classes and independent test methods in our test suite.
- It also allows us to configure multiple tests in a single test suite and run them in multi-threaded environment.

27. What is parameterized testing in TestNG?

- Parameterization [Data driven test] is an execution strategy, which allows us to run a test case automatically, multiple times with different input values.
- To pass multiple data to the application at runtime, we need to parameterize our test scripts.
- **There are two ways by which we can achieve parameterization in TestNG**
- With the help of Parameters annotation and TestNG XML file.
- With the help of Data Provider annotation.

28. What is the time unit we specify in test suites and test cases?

- We specify the time unit in test suites and test cases is in milliseconds.

29. Can I call a single data provider method for multiple functions and classes?

- Yes, the same DataProvider can be used in multiple functions and classes by declaring DataProvider in separate class and then reusing it in multiple classes.

30. List out various ways in which TestNG can be invoked?

- **TestNG can be invoked in the following ways**
    1. Using Eclipse IDE
    2. Using ant build tool
    3. From the command line
    4. Using IntelliJ's IDEA

31. How to run TestNG using command prompt?

- ***Step 1*:** Open notepad
- ***Step 2*:** Paste the below lines of code - You may need to add your project location. In the example, project location is set as 'F: \Selenium\TestNGBatchExample'.
- ***Step 3**:* Save the file as 'testNGBatchFile.bat' in location that you want to save.

```
set projectLocation=F: \Selenium\TestNGBatchExample
cd %projectLocation%
set classpath=%projectLocation%\bin; %projectLocation%\lib\*
java org.testng.TestNG %projectLocation%\testng.xml
pause
```

32. What is the use of @Test (invocationCount=x)?

- The invocation count attribute tells how many times TestNG should run a test method

```
@Test (invocationCount = 5)
public void testCase1 ()   {
}
```

33. What does the test timeout mean in TestNG?

- The maximum number of milliseconds a test case should take.

```
@Test (threadPoolSize = 3, invocationCount = 10, timeout = 10000)
public void testCase1 () { }
```

34. Is it possible to pass test data through testng.xml file, if yes how?

- TestNG allows the user to pass values to test methods as arguments by using parameter annotations through **testng.xml file**.
- Sometimes it may be required for us to pass values to test methods during run time.
- The **@Parameters** annotation can be placed on any method that has a **@Test**, **@Before/After** or **@Factory** annotation.

35. Do you run test cases in parallel with TestNG? If yes how many threads and does it cause any problem?

- TestNG provides multiple ways to execute tests in separate threads.

- In testng.xml, if we set **'parallel'** attribute on the tag to 'tests', testNG will run all the **'@Test'** methods in tag in the same thread, but each tag will be in a separate thread.
- If we want to run methods/classes in separate threads, we need to set **'parallel'** attribute on the **tag to 'methods' / 'classes'**.
- This helps us to run test **methods / classes / tests** in parallel.
- By using parallel execution, we can reduce the 'execution time' as tests are executed simultaneously in different threads.
    1. In testNG we can achieve parallel execution by two ways.
    2. One with testng.xml file and we can configure an independent test method to run in multiple threads.

36. What's TestNG Listener Class & why do we use it?

- TestNG Listeners also allows you to customize the **tests logs or report** according to your project requirements.
- TestNG Listeners in Selenium WebDriver are modules that listens to certain events and keep track of test execution while performing some action at every stage of test execution.
- TestNG Listeners in Selenium WebDriver can be implemented at two levels:
    1. **Class level:** In this, you implement listeners for each particular class no matter how much test cases it includes.
    2. **Suite level:** In this, you implement listeners for a particular suite which includes several classes as test cases.
- There are numerous TestNG listeners in Selenium WebDriver, some of them are used very frequently by the testing community & some are almost forgotten.
- In this TestNG tutorial, I will demonstrate the most popular TestNG listeners with examples but before that, let me enlist the various TestNG listeners in Selenium WebDriver.
    1. **ITestListener**
    2. **IAnnotationTransformer**
    3. **IInvokedMethodListener**
    4. **ISuiteListener**
    5. **IReporter**
    6. **IConfigurable**
    7. **IExecutionListener**
    8. **IHookable**
    9. **IMethodInterceptor**
    10. **IConfigurationListener**

Questions on Maven:

37. What is Maven and explain its life cycle phases?

- Maven is a build / project management tool, based on the concept of a project object model (POM) contains information of project and configuration information for the maven to build the project such as dependencies, build directory, source directory, test source directory, plugin, goals etc.

- **The default life cycle comprises of the following phases:**
    1. **Validate** - validate the project is correct and all necessary information is available
    2. **Compile** - compile the source code of the project
    3. **Test** - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
    4. **Package** - take the compiled code and package it in its distributed format, such as a JAR.
    5. **Verify** - run any checks on results of integration tests to ensure quality criteria are met
    6. **Install** - install the package into the local repository, for use as a dependency in other projects locally
    7. **Deploy** - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.
- Maven provides a common platform to perform these activities which makes programmer's life easier while handling the huge project.

- **Maven Build Life Cycle:**

  Basic maven phases are used as below.
    1. **clean***:* deletes all artifacts and targets which are created already.
    2. **compile***:* used to compile the source code of the project.
    3. **test***:* test the compiled code and these tests do not require to be packaged or deployed.
    4. **package***:* package is used to convert your project into a jar or war etc.
    5. **install***:* install the package into the local repository for use of another project.

38. What is Maven and its advantages of using it in your Selenium Project?

- **Advantages:**
    1. Better dependency management
    2. More powerful builds
    3. Better debugging
    4. Better collaboration
    5. More componentized builds
    6. Reduced duplication
    7. More consistent project structure

- **Uses in Selenium:**
    - We can create Maven project for writing script and create dependency-using POM.xml once dependency is set Maven will download all the dependent jar files automatically and in future if any update comes from Selenium or TestNG side it will simply update all the required changes.

39. What plug-in you used in maven?

- What are Maven Plugins?
- Maven is actually a plugin execution framework where every task is actually done by plugins.

- **Maven Plugins are generally used to −**
    1. create jar file
    2. create war file
    3. compile code files
    4. unit testing of code
    5. create project documentation
    6. create project reports

- A plugin generally provides a set of goals, which can be executed using the following syntax
    mvn [plugin-name]:[goal-name]

- For example, a Java project can be compiled with the maven-compiler-plugin's compile-goal by running the following command.
    mvn compiler: compile

- **Plugin Types**
    - **Maven provided the following two types of Plugins −**
        1. **Build plugins**- They execute during the build process and should be configured in the <build/> element of pom.xml.
        2. **Reporting plugins-**They execute during the site generation process and they should be configured in the <reporting/> element of the pom.xml.
            - **Clean**-*clean* up after build.
            - **Compiler**-*compiles* java source code.
            - **deploy**-deploys the artifact to the remote repository.
            - **Failsafe**-*runs* the JUnit integration tests in an isolated class loader.
            - **Install**-*installs* the built artifact into the local repository.
            - **resources**- copies the resources to the output directory for including in the JAR.
            - **site**- generates a site for the current project.
            - **Sure-fire**-*runs* the JUnit unit tests in an isolated class loader.
            - **Verifier**-verifies the existence of certain conditions. It is useful for integration tests.

40. What is the name of maven folder which contains all the libraries?

- The local repository of Maven is a folder location on the developer's machine, where all the project artifacts are stored locally.
- When maven build is executed, Maven automatically downloads all the dependency jars into the local repository.
- Usually this folder is named .m2.
- Here's based on OS:
    1. Windows: C:\Users\<User_Name>\.m2
    2. Linux: /home/<User_Name>/.m2
    3. Mac: /Users/<user_name>/.m2

4. And of course, for both on Linux or Mac:
5. Linux/Mac: ~/.m2

41. Without internet can we work with maven repository?

- You need an internet connection.
- Maven isn't initially self-sufficient.
- It needs to download a bunch of plugins along with their dependencies and the dependencies of your own project.

42. What is the difference between Maven and TestNG?

- TestNG is a testing framework. It also generates testing reports.
- Maven is a software project management and comprehension tool.
- It manages all dependencies and different flows for building a project.

43. In Maven, from where the jar files will get downloaded?

- When you add a dependency ("Dependency" here means what your project depends on e.g., if you want to use Log4j in your project, then you depend on that library) to the pom.xml, your IDE (Eclipse) will use Maven to download the jars and store them in your local repository (%HOME%/.m2 folder) so that you can compile your project and run it.
- Maven also allows manages transitive dependencies for you (libraries that your dependencies rely on).

44. What's the difference between a Maven project and a Java project?

- In **Normal Java Project**, if you want to work on any third party / API applications then you have to associate those jar files and associate/configure those jar files to your project manually,
- whereas in **Maven project** provide the third party/API applications dependency in **POM** file and then click on Maven install then automatically those respective libraries automatically to your project.

45. In Maven what are the two setting files called and what are their location?

- In Maven, the setting files are called settings.xml, and the two setting files are located at
- Maven installation directory: $M2_Home/conf/settings.xml
- User's home directory: $ { user.home }/ .m2 / settings.xml

46. In Maven, do we have to manually download and configure/update the required jar files?

- **Answer:** No

47. What is the command to build your Maven site?

- Type the command – mvn site

48. What is POM?

- POM stands for Project Object Model.
- In Maven, it is a fundamental Unit of Work and it is an XML file.
- You can find it in the base directory of the project.
- It consists of information about the project and various configuration details used by Maven to build the project(s).

49. What would the command mvn clean do?

- This command deletes the target directory with all the build data before starting the build process.

50. Tell me the command to install JAR file in local repository.

- **mvn install**

51. What is Archetype?

- An archetype is a Maven plugin whose task is to create a project structure as per its template.

52. What is the command to create a new project based on an archetype?

- Type the following command – ***mvn archetype:generate***

53. What is SNAPSHOT in Maven?

- SNAPSHOT can be defined as a special version that indicates a current development copy.

Scenario Based and Some Tricky Questions

54. **What will you do if there are failures in your suite execution and what is your approach?**

- While executing the automation scripts, test cases may fail for several reasons.
  To optimize our next runs, we need to re-run only failed test cases.

55. How to execute failed Test cases? What is the best approach?

- In TestNG class, we can easily re-run the test cases using two methods as explained below:
  Method 1: By using testng-failed.xml file in test-output folder.
  Method 2: By implementing TestNG IRetryAnalyzer.

56. Suppose there is one method in interface as private and a class is implementing. Can implemented method be public or vice versa?

- In interface only public, abstract, default, static modifiers are permitted.

57. Tell me any difficulties you faced in developing automation scripts? Can you give any examples for any complex scenarios handled?

- **Answer:**
  1. **Sync issue or Timeout**
     - Example: After clicking on some button one alert should present and we have to handle via code but due to many issues alert might come after a few seconds, in that case, the script will fail.

- We need to handle this kind of scenario using an explicit wait. This is just one example like this we have many examples which show without a smart locator we cannot build stable scripts.

2. **Smart locators**
   - As we all know that locators are the core part of any scripting and we need to keep on enhancing our XPath and CSS for script stability, because if XPath and CSS are not proper then chances are very high that script might fail in upcoming releases.
   - We should always write dynamic or custom XPath or class, which can make our script more stable.

3. **Cross browser testing**
   - While designing script we always focus on one browser and we design our script for that browser only, but when it comes to real execution of the script then we have to make sure that our script should run in all browser which is known as Cross Browser Testing (Chrome, FF, IE at least).
   - In order to avoid failure once the script is developed, we need to run them on the different browsers and analyse the result. If it is failing on another browser then we need to change locator strategy.

4. **Pop up handling**
   - In many applications, you will find random pop that keeps coming and their behaviour is not persistent, so we also have to take care of these unwanted pop up which stops our execution.

5. **When Code Review is not done**
   - Many companies they do not follow proper code review which can create many issues in the future.
   - Example- Code review includes code formatting, proper validation or assertion, effective usage of the framework, the design pattern used and so on.
     1. Checkstyle
     2. PMD
     3. FindBugs

58. If you execute the scenarios in your project multiple times, will the reports override? If they override, how will you take backup of previous test report?

- **Answer:** You can override reports.

59. What will you do when you have more number of lines of code having repeated code?

- We can avoid the duplicate code by using custom methods, Inheritance etc.

60. In Selenium project, we use hierarchy like interface, followed by abstract class, followed by a class. Can't we directly use interface followed by a class?

- **Answer:** Yes we can do that; in that case we need to implement all the methods from interface in the class which implements the interface.

61. You can create your file name to be the current timestamp.

- **This way, it will be easy to have a unique name for your report file -**

String timestamp = new SimpleDateFormat ("yyyy.MM.dd.HH.mm.ss").format (new Date ());

HtmlReporter = new ExtentHtmlReporter (System.getProperty ("user.dir") +"\\test-output\\" + timestamp + ".html");

62. What is the problem with Thread.Sleep in code?

- *It is a* **Static wait***:* If given a wait of 5000 Milliseconds (5 seconds) and an element just take just 1-2 seconds to load, script will still wait for another 3 seconds which is bad as it is unnecessarily increasing the execution time. So **Thread.sleep(),** increases the execution time in cases where elements are loaded in no due time.
- When using **Thread.sleep(),** we have to mention wait time in advance, there is no guarantee that the element will be displayed in that specific wait time, there may be case when it will takes may be more than 5 seconds to load and again the script would fail.
- ♦ You need to write sleep () method whenever we need to make web driver wait. So if you want to wait for two web elements, you need to write **Thread.sleep ()** twice just before you locate web elements.
- ♦ It is not good programming practice. Instead you can use implicit or explicit waits.

63. Manually you opened a Firefox browser window with Gmail login, now with selenium you opened a Firefox browser window with Facebook login, what happens when we use quit method? Will it closes all windows including Gmail one?

- **Answer:** No it will not close manually opened windows.

64. We have 2 interfaces and both have print methods, in my class I have implemented the print method, how you will get to know that I have implemented the first interface and how you will use it, if you want to use it?

- If a type implements two interfaces and each interface define a method that has identical signature, then in effect there is only one method, and they are not distinguishable.
- If, say, the two methods have conflicting return types, then it will be a compilation error.
- This is the general rule of inheritance, method overriding, hiding, and declarations, and applies also to possible conflicts not only between 2 inherited interface methods, but also an interface and a super class method, or even just conflicts due to type erasure of generics.

65. If an explicit wait is 10 sec and the condition is met in 5 sec, will driver move to execute next statement after 5 sec or it will wait for complete 10 sec then move?

- Driver will move to next statement after 5 seconds.

66. If element is loaded by taking much time, how to handle this situation in selenium?

- Apply fluent wait.