TestNG

Test Next Generation

- TestNG is an automation testing framework in which NG stands for "Next Generation". TestNG is inspired from JUnit which uses the annotations (@).
- Using TestNG you can generate a proper report, and you can easily come to know how many test cases are passed, failed and skipped. You can execute failed test case separately.
- Suppose, you have five test cases, one method is written for each test case (Assume that the program is written using the main method without using testNG). When you run this program first, three methods are executed successfully, and the fourth method is failed. Then correct the errors present in the fourth method, now you want to run only fourth method because first three methods are anyway executed successfully. This is not possible without using TestNG.
- The TestNG provides an option, i.e., testng-failed.xml file in test-output folder. If you want to run only failed test cases means you run this XML file. It will execute only failed test cases.

Why Use TestNG with Selenium?

- Default Selenium tests do not generate a proper format for the test results. Using TestNG we can generate test results. Most Selenium users use this more than Junit because of its advantages. There are so many features of TestNG, but we will only focus on the most important ones that we can use in Selenium.

Key features of TestNG

- Generate the report in a proper format including a number of test cases runs, the number of test cases passed, the number of test cases failed, and the number of test cases skipped.
- Multiple test cases can be grouped more easily by converting them into testng.xml file. In which you can make priorities which test case should be executed first.
- The same test case can be executed multiple times without loops just by using keyword called 'invocation count.'
- Using testng, you can execute multiple test cases on multiple browsers, i.e., cross browser testing.
- The testing framework can be easily integrated with tools like Maven, Jenkins, etc.
- Annotations used in the testing are very easy to understand ex: @BeforeMethod, @AfterMethod, @BeforeTest, @AfterTest.
- WebDriver has no native mechanism for generating reports. TestNG can generate the report in a readable format.
- TestNG simplifies the way the tests are coded. There is no more need for a static main method in our tests. The sequence of actions is regulated by easy-to-understand annotations that do not require methods to be static.
- Uncaught exceptions are automatically handled by TestNG without terminating the test prematurely. These exceptions are reported as failed steps in the report.

Advantages of TestNG over JUnit

- Annotations are easier to understand.
- Test cases can be grouped more easily.
- Parallel testing is possible.
- 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.
- The ability to run tests in parallel is available in TestNG but not in JUnit, so it is the more preferred framework of testers using Selenium Grid.

Configuration of TestNG in Eclipse & Project

- Launch Eclipse IDE.
- Click on Help
- Click on Eclipse MarketPlace.
    - In search box, type testNG and Hit Enter.
    - Click on install
    - Click Next again on the succeeding dialog box until it prompts you to Restart the Eclipse.

Annotations In TestNG

- @BeforeSuite -> set system property
- @BeforeTest -> Launch Browser
- @Beforeclass -> Launch web Application
- @BeforeMethod -> Login to Application
- @Test -> Write your Testcase
- @AfterMethod -> Logout from Application
- @Afterclass -> Close Browser
- @AfterTest -> Delete All cookies
- @AfterSuite -> Generate Test report

Summary of TestNG Annotations

- **@BeforeSuite**: The annotated method will be run before all tests in this suite have run.
- **@AfterSuite**: The annotated method will be run after all tests in this suite 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.
- **@BeforeClass**: The annotated method will be run before the first test method in the current class is invoked.
- **@AfterClass**: The annotated method will be run after all the test methods in the current class have been run.
- **@BeforeMethod**: The annotated method will be run before each test method.
- **@AfterMethod**: The annotated method will be run after each test method.
- **@Test**: The annotated method is a part of a test case.

Note : By default, methods annotated by **@Test** are executed alphabetically.

Different Parameters of @Test annotation

- ***priority*** : If you want the methods to be executed in a different order, use the parameter "priority". Parameters are keywords that modify the annotation's function.
- ***dependsOnMethods*** : If two methods are depending on one another then at that time we will use “dependsOnMethods” parameter.
- ***alwaysRun*** : If set to true, this test method will always run even if it depends on a method that failed.
- ***invocationCount*** : The number of times this method should be invoked.
- ***enabled*** : Using this parameter we can skip any Test case during execution.
- ***groups*** : In a framework, you have different types of Test case scripts like smoke Test scripts and Regression Test Scripts. To group any Test Scripts, we use *@Test (groups= {“smoke”})* or *@Test (groups= {“regression”})*. If you want to run test scripts as both the smoke and the regression then we can use *@Test ({“smoke” , ”regression”})*.

Assert Class

Assert class is present under the package org.testng. This class contains static methods like :

- assertEquals(String actual,String expected, String message)
- assertNotEquals()
- assertTrue(Condition, Message)
- assertFalse(Condition, Message)
- assertSame()
- assertNotSame()
- assertNull()
- assertNotNull()
- fail()

These are the main methods which can be used for various comparisons that can be done while using TestNG framework for writing Test Methods in case of validations.

Note : *while checking for any validations, if the test is fail then any statement after assert will not be executed.*

SoftAssert Class

SoftAssert class is present in the package org.testng.asserts, and it works similar to Assert class. SoftAssert is also used to validate different conditions in our Test Scripts. It contains non-static methods which can be used by creating an object of this class.

Difference between Assert and SoftAssert Class

- While using Assert class if any validations fail, the later statements will not be executed whereas with SoftAssert class all the statements will be executed.
- In Assert class, all methods are static while in SoftAssert class all methods are non-static.
- To declare all assertions, we don't use ***assertAll()*** in Assert class, while in case of SoftAssert we have to use ***assertAll()*** method to assert all the assertions.

PageFactory Class

PageFactory class is used to make use of Page Objects simpler and easier.

public static void initElements(WebDriver driver, java.lang.Object page)

driver-The driver that will be used to look up the elements

page-The object with WebElement and List<WebElement> fields that should be proxied.

@FindBy Annotation

- Used to mark a field on a Page Object to indicate an alternative mechanism for locating the element or a list of elements. Used in conjunction with PageFactory this allows users to quickly and easily create PageObjects.
- It can be used on a types as well, but will not be processed by default. You can either use this annotation by specifying both "how" and "using" or by specifying one of the location strategies (eg: "id") with an appropriate value to use. Both options will delegate down to the matching By methods in By class.
- For example, below two annotations point to the same element:

```
@FindBy(id = "foobar")
WebElement foobar;
```

```
@FindBy(how = How.ID, using = "foobar")
WebElement foobar;
```

- and below two annotations point to the same list of elements:

```
@FindBy(tagName = "a")
List<WebElement> links;
```

```
@FindBy(how = How.TAG_NAME, using = "a")
List<WebElement> links;
```

@FindAll Annotation

- Used to mark a field on a Page Object to indicate that lookup should use a series of @FindBy annotation, It will then search for all elements that match any of the @FindBy criteria.
- Note that elements are not guaranteed to be in document order. It can be used on a types as well, but will not be processed by default.

```
@FindAll({@FindBy(how = How.ID, using = "foo"),@FindBy(className = "bar")})
List<WebElement> elements;
```

@FindBys Annotation

- Used to mark a field on a Page Object to indicate that lookup should use a series of @FindBy tags in a chain as described in ByChained. It can be used on a types as well, but will not be processed by default.

```
@FindBys({@FindBy(id = "foo"),@FindBy(className = "bar")})
WebElement element;
```