**TEST NG – Selenium Framework**

TestNG is a testing framework inspired **from Junit(Java) and NUnit(C#)** but introducing some new functionality that make it more powerful and easier to use. It is an open source tool where NG denotes next generation.

It is like add-on in Eclipse

**Benefits of TestNG**

There are number of benefits of TestNG but from Selenium perspective, major advantages of TestNG are

* It gives the ability to produce HTML Reports of execution
* Annotations made life easy(a lot of annotaions)
* Test cases can be Grouped & Prioritized/Seqencing more easily
* Parallel testing is possible – GRID(Cross Browser Testing) using testNG.xml
* Generates Logs(Reports) – Log4j,Listeners
* Data Parameterization is possible -- Parameterisation(xml) & Data Providers(arrays & Excel)

**Difference between annotations of JUnit and TestNG?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **JUnit** | **TestNG** |
| test annotation | @Test | @Test |
| run before all tests in this suite have run | — | @BeforeSuite |
| run after all tests in this suite have run | — | @AfterSuite |
| run before the test | — | @BeforeTest |
| run after the test | — | @AfterTest |
| run before the first test method in the current class is invoked | @BeforeClass | @BeforeClass |
| run after all the test methods in the current class have been run | @AfterClass | @AfterClass |
| run before each test method | @Before | @BeforeMethod |
| run after each test method | @After | @AfterMethod |
| ignore test | @ignore | @Test(enabled=false) |
| expected exception | @Test(expected = ArithmeticException.class) | @Test(expectedExceptions = ArithmeticException.class) |
| Timeout | @Test(timeout = 1000) | @Test(timeout = 1000) |

**How to Install TestNG in Eclispse?**

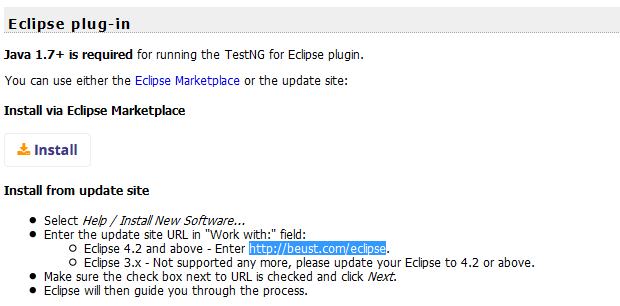
**Navigate to the below page and click download tab**

<http://testng.org/doc/eclipse.html>

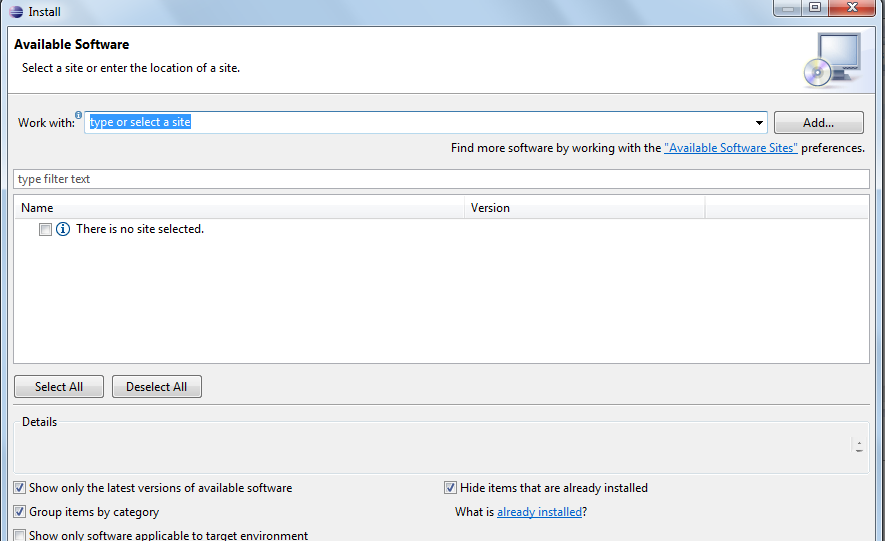
or

Open the webpage [*http://testng.org/doc/download.html*](http://testng.org/doc/download.html)

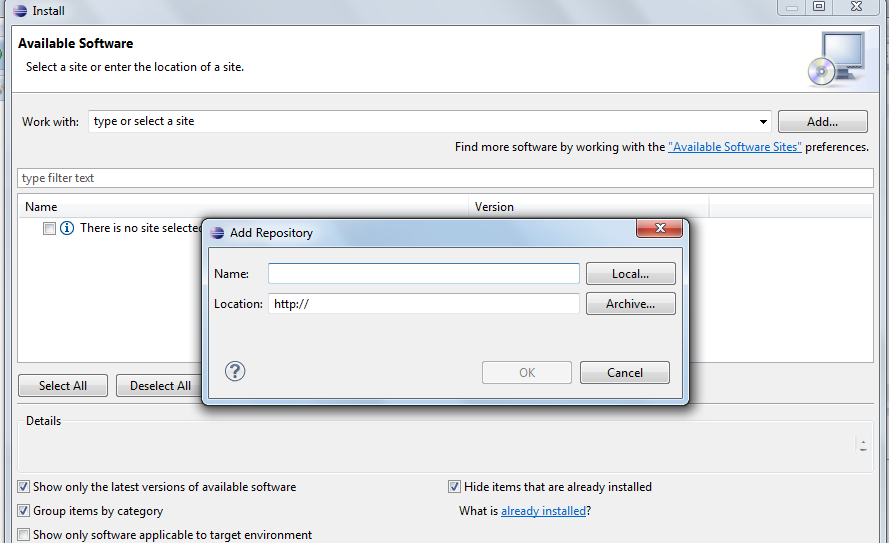
Navigate to the below mentioned section in the webpage. Depending on the eclipse version copy the link.

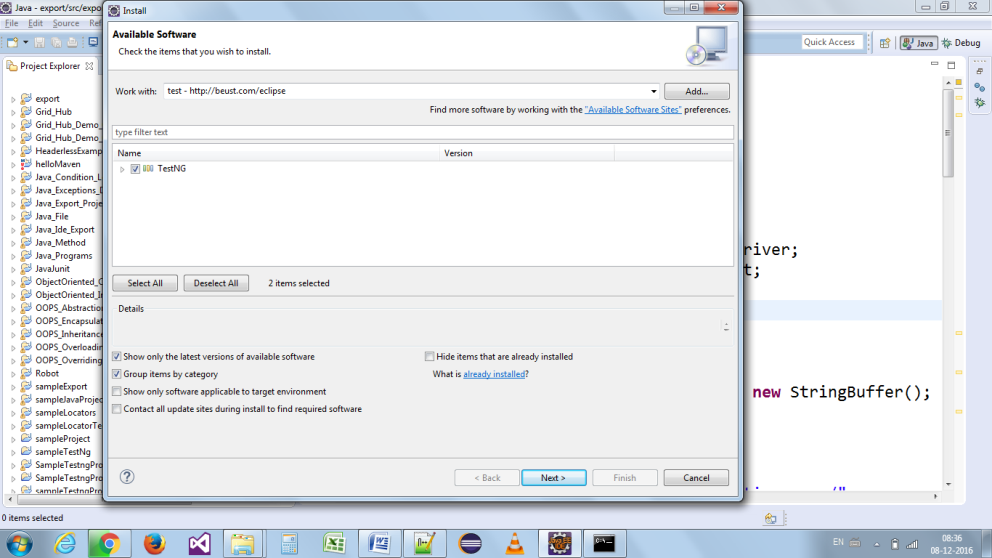


Now open Eclipse navigate to Help -> Install New Software the below screen will be available



Enter the copied site in the Work with text box and click on Add button

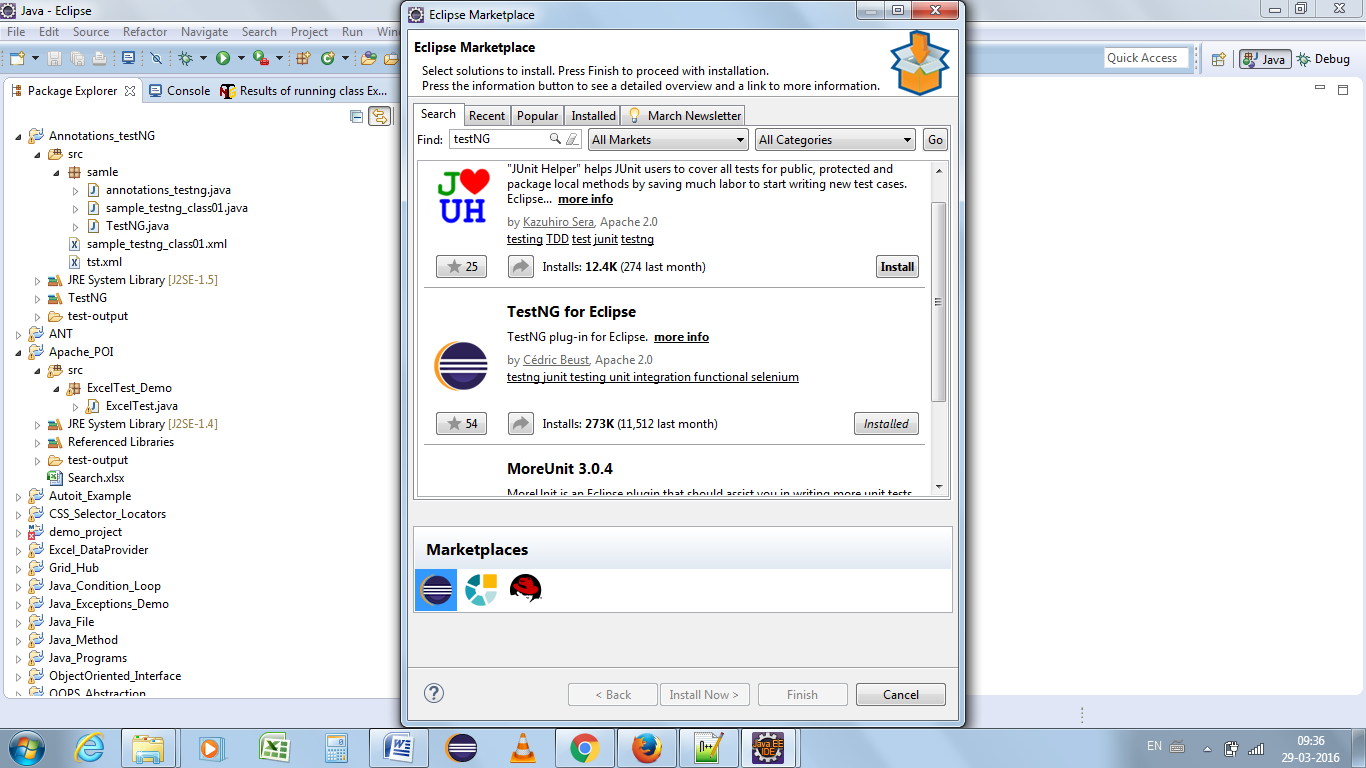




Now enter a name and copied link and click ok and follow the procedure.

(OR)

Navigate to Eclipse Help -> **Eclipse Marketplace** and type testNG in the Find text box



**What are Annotations?**

Annotations are lines of code that can control how the method below them will be executed. They are preceded by @ symbol followed by specific name.

Example: @Test,@BeforeMethod,@AfterMethod…etc

Reference file that we use it,

import org.testng.annotations.\*;

**What is meant by hierarchy of Annotations?**

Hierarchy is the order by which the annotations are executed.

->Suite ->(@Beforesuite & @Aftersuite)

->Test(@BeforeTest & @AfterTest)

->classes (@Beforeclass & @Afterclass)

->Method(@BeforeMethod & @AfterMethod)

->Testcase(@Test)

**Annotation in TestNG**

Annotations are those things in TestNG which guides it for what to do next or which method should be executed next. Let us discuss the annotations used in TestNG

**@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.

**@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.

**Benefits of using annotations**

* TestNG identifies the methods it is interested in by looking up annotations. Hence method names are not restricted to any pattern or format.
* We can pass additional parameters to annotations.
* Annotations are strongly typed, so the compiler will flag any mistakes right away.
* Test classes no longer need to extend anything (such as Test Case, for JUnit 3).

**Hierarchy of Annotations**

<@Beforesuite>

<@Before test>

<@Before classes>

<@Before method>

<@test> (parameters)

</@After method>

<@Before method>

<@test>(parameters)

</@After method>

</@After classes>

</@After test>

</@After suite>

**Grouping of Test case:**

It represents one more annotation in TESTNG which is used in execution of multiple test cases.

Example:

 @Test (groups = {“sample"})

**Dependant Test case:**

In times we need to methods in test case in a specific order or need to share data and state between methods. These dependencies are possible in TESTNG.

They are classifieds as

* *dependsOnMethods – not used*
* *dependsOnGroups – not used*

Example:

@Test (dependsOnMethods = { "OpenBrowser" })

Public void method(){

}

**What is Prioritizing and Sequencing in TestNG?**

We need to use “priority” parameter in order to execute the test cases in order. They usually modify the annotation functions.

Example:

@Test (priority = 0)

@Test (priority = 1)

**How to we skip a specific test case?**

Another concept is skipping of test cases where we can ignore specific test case without being executed.

Example:

@Test (enabled = false)

We can also use two keywords as below,

@Test (priority = 3, enabled = false)

**What is data drive testing?**

There are times when we would like to run our tests against different data sets. Instead of hard coding the data set within tests itself, it will be more appropriate if we could pass our data as parameters to the test method. Thus the test logic remains same but the data on which the test is run changes and this is called data-driven testing.

TestNG supports two different ways of injecting parameter values.

1. Parameterization through testng.xml - **@Parameter(specify name of the parameter)**
2. Data Provider – Array/Apache POI- **@Dataprovider(name= dataprovider name)**

**By Parameter:**

Parameterization through testng.xml is one of the methods of injecting parameter values. This method is useful if the data set is limited to just a few rows and the values are of simple types like String, int etc. The parameters are declared in testng.xml, where the name attribute defines name of the parameter and value attribute defines the parameter value.

**By Data Provider:**

If you want to provide the test data, the Data Provider way, then we need to declare a method that returns the data set in the form of two dimensional object array Object[][]. The first array represents a data set whereas the second array contains the parameter values.

The Data Provider method can be in the same test class or one of its super classes. It is also possible to provide Data Provider in another class but then the method has to be static.

Once we have added the method we need to annotate it using @DataProvider to let TestNG know that it is a Data Provider method. You can also provide a name to it using the name attribute of the Data Provider annotation but this is optional. If one hasn’t provided the name, name of the method will be used to refer to it.

Example:

@dataprovider(name=”test”)

Public void object[][]method(){

}

@Test(dataprovider=” test”)

If a test wants to use the Data Provider, it can do so by specifying the name of the Data Provider in data Provider attribute of @Test annotation.

**Reporters and Asserts:**

**Reporters:** TestNG provides logging facility which is helpful to insert comments or track the events happening in the framework. The information can be of any details which solely rely on the purpose.

In selenium we use two types of logging *High Level* and *Low level*.

* High level – we keep track only of the main events occurring
* Low level – we try to produce logs for each and every steps

**Asserts/Assertion:** The decisions which are made in the middle of a test run are referred as assertion/assert. Used for creating checkpoints in tests. In selenium there will be many situations in the test where you just like to check the presence of an element. All you need to do is to put an assert statement on to it to verify its existence.

Assert ->assertEquals,assertTrue,assertFalse