**TestNG: TestNG**is an automation testing framework in which NG stands for "Next Generation". TestNG is inspired from **JUnit and NUnit** which uses the annotations (@). TestNG overcomes the disadvantages of JUnit and is designed to make end-to-end testing easy.  
TestNG was created by **Cédric Beust**. Current version: 7.x.x

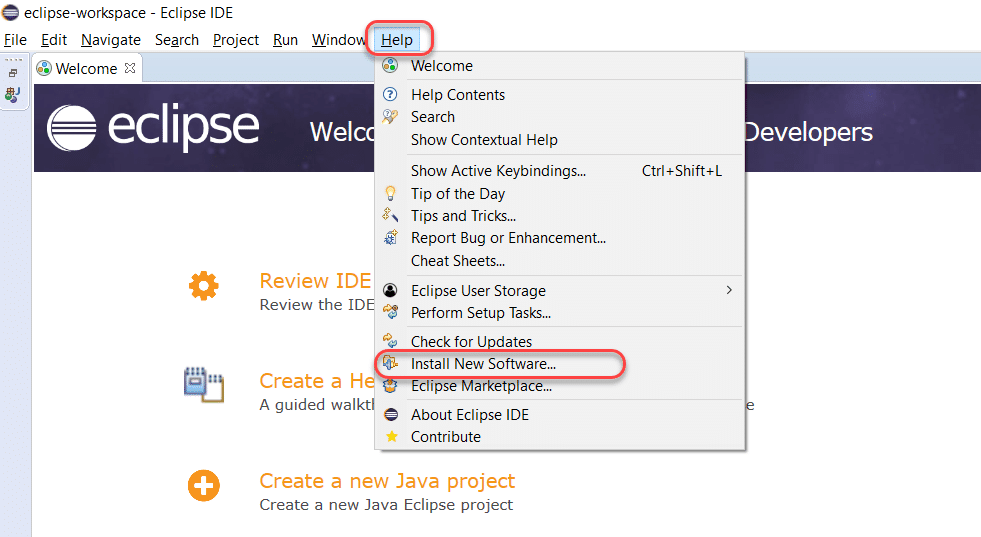
Using TestNG, you can generate a proper report, and you can easily come to know how many test cases are **passed, failed, and skipped**. You can execute the failed test cases separately.

Graphical user interface, diagram, application

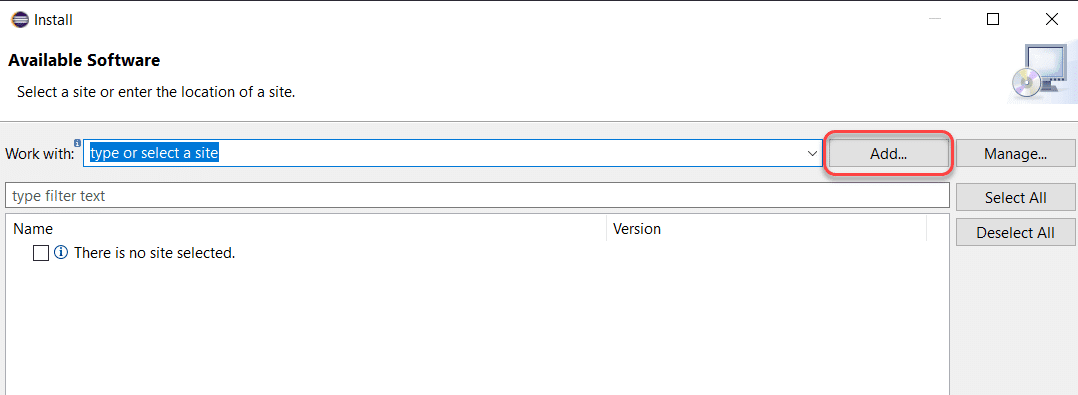
Description automatically generated

**DOWNLOAD AND INSTALL TESTNG**

1) Launch the Eclipse IDE and click “***Install New Software***” in the Help menu.



2) You will see a new installation dialog window, click the “***Add***” button.

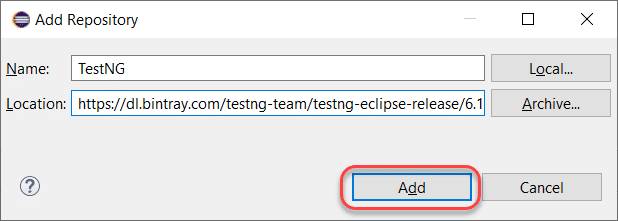


3) Fill out the information as follows:

***Name*:** *TestNG (depends on the user)*

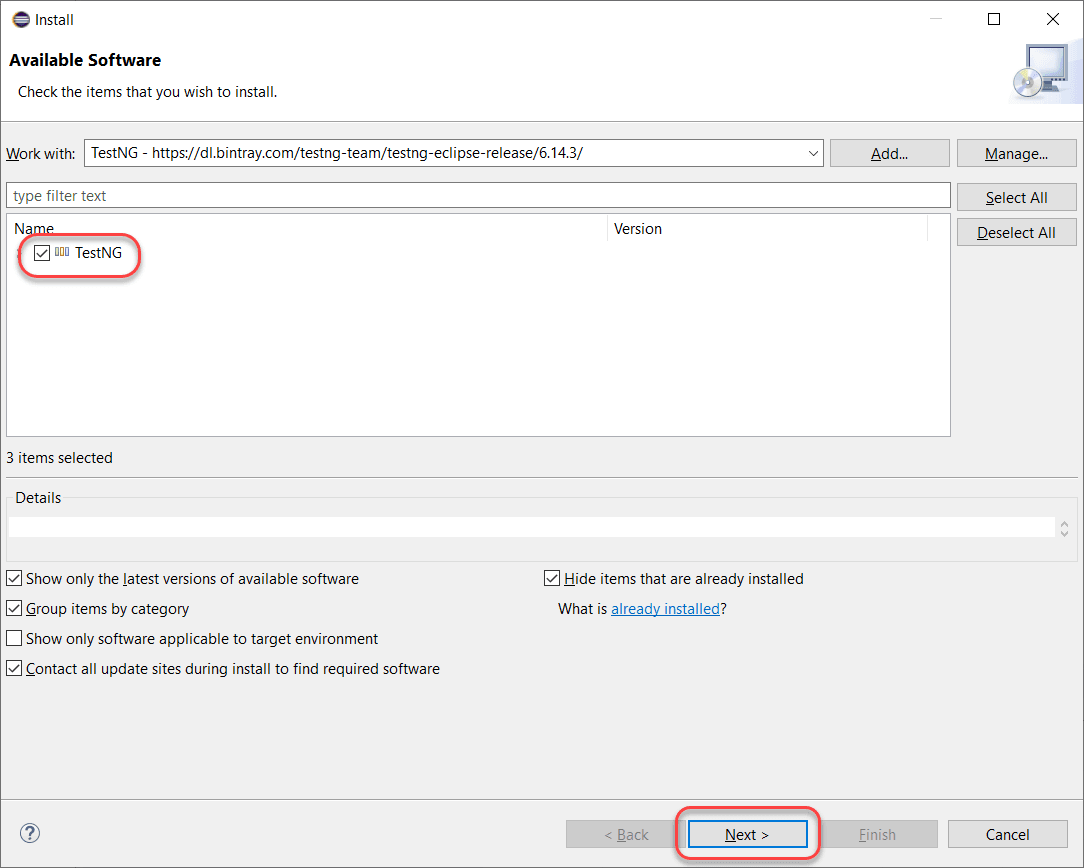
***Location*: *https://dl.bintray.com/testng-team/testng-eclipse-release/6.14.3/***

Click ***Add***.

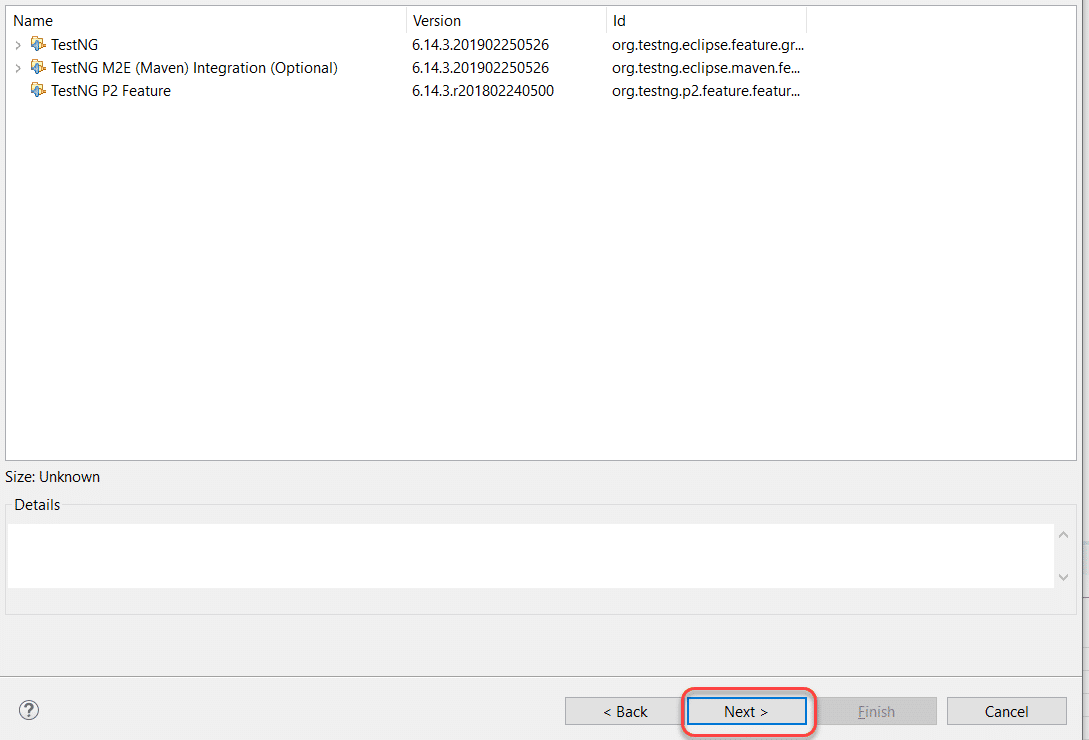


**Note:** *There is a deprecation of****http://beust.com/eclipse/****. For Eclipse06-18, you can install TestNG from the*[***Eclipse Marketplace***](https://marketplace.eclipse.org/content/testng-eclipse)*, and for 09-19, you need to follow the new link as given in this tutorial.*

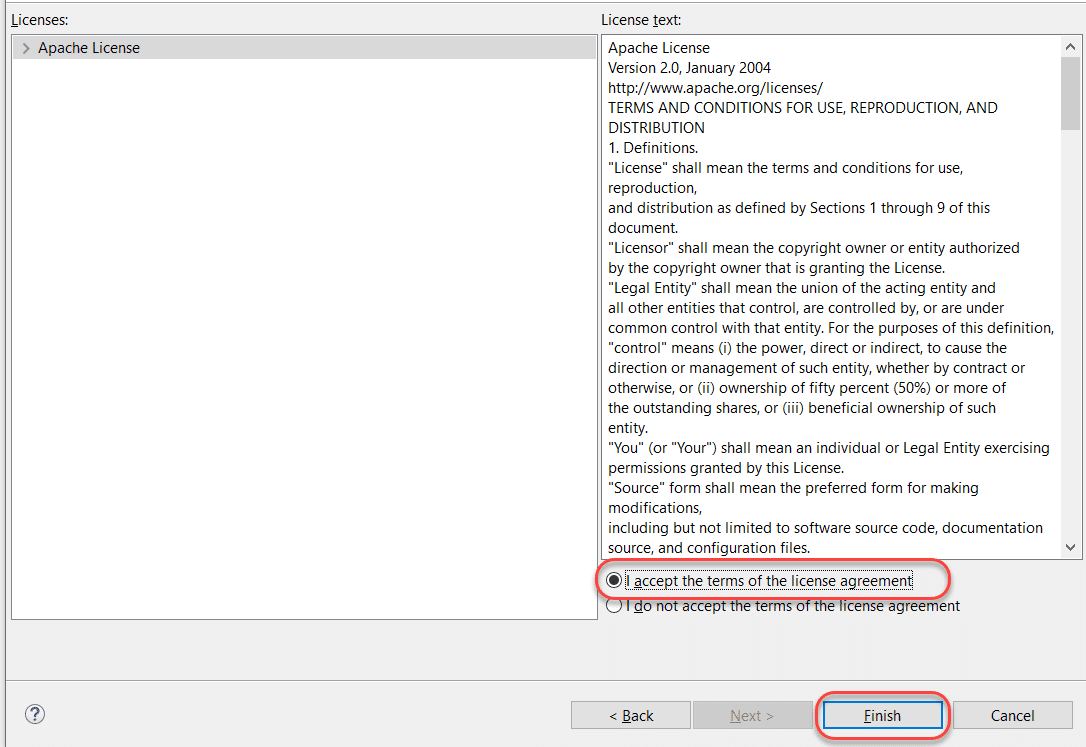
4) Clicking on add redirects us back to the previous window. However, this time you must see the ***TestNG***option in the available software list. After that, check ***“TestNG”*** and click ***Next***.



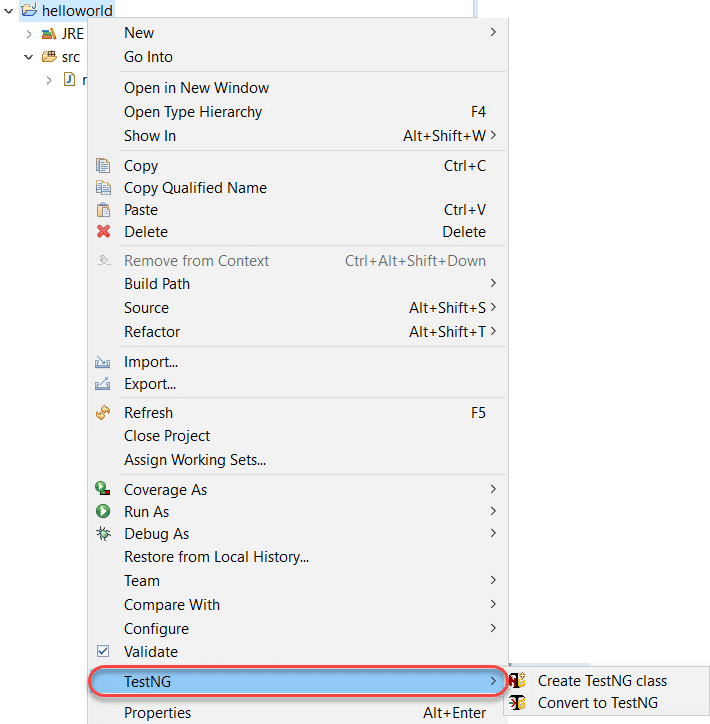
5) Click ***Next*** to install the TestNG dependencies that eclipse calculates by itself.



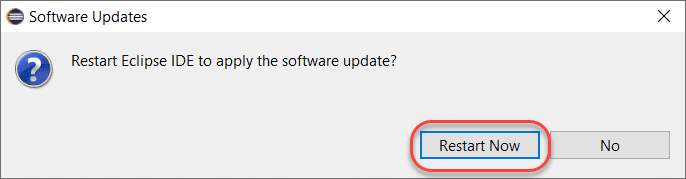
6) After that, ***accept the terms of the license agreement*** then click ***Finish***.



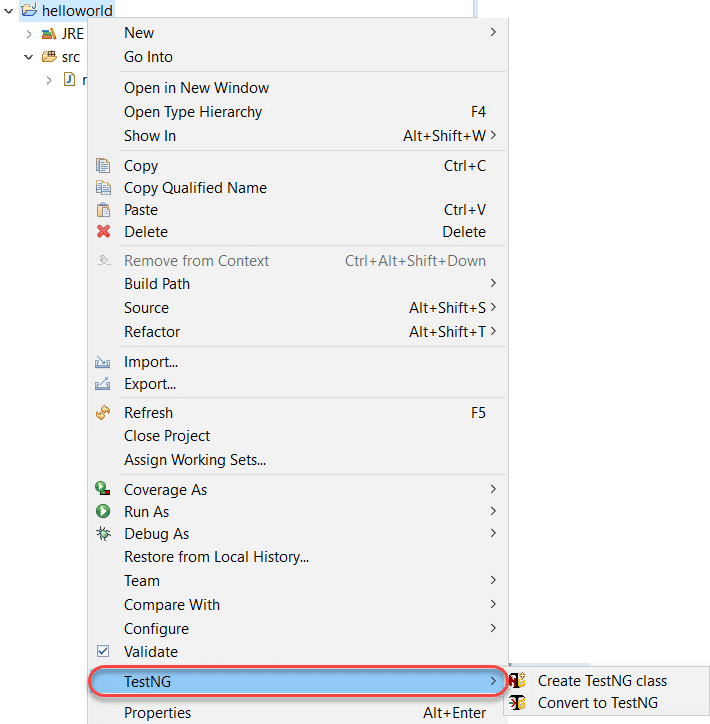
7) You may or may not encounter a Security warning. Click ***Install Anyway***if you do.



8) After that, click “***Restart Now***” to restart the eclipse and finish the installation setup.



9) Finally, after the restart, verify if TestNG installed successfully. Right-click on your project and see if ***TestNG***displays in the opened menu.



If everything has happened as stated in this post, congratulations! You have TestNG now installed on your system. Subsequently, in the next section, we will look at how to install TestNG on another popular IDE for Java called IntelliJ

**What is TestNG.xml?**

TestNG.xml file is a configuration file that helps in organizing our tests. It allows testers to create and handle multiple test classes, define test suites and tests.

It makes a tester’s job easier by controlling the execution of tests by putting all the test cases together and run it under one XML file. This is a beautiful concept, without which, it is difficult to work in TestNG.

Some of the Benefits and Features of TestNG/TestNG.xml is below mentioned:

1. TestNG provides parallel execution of test methods
2. It allows to define dependency of one test method over other method
3. It allows to assign priority to test methods
4. It allows grouping of test methods into test groups
5. It has support for parameterizing test cases using @Parameters annotation
6. It allows data driven testing using @DataProvider annotation
7. It has different assertions that helps in checking the expected and actual results
8. Detailed (HTML) reports

ANNOTATIONS

Text, letter

Description automatically generated with medium confidence

An annotation is a tag that provides additional information about the class or method. It is represented by**‘@’**prefix. TestNG use these annotations to help in making a robust framework. Let us have a look at these annotations of TestNG for [automation testing with Selenium](https://www.lambdatest.com/selenium-automation).

@Test

**The most important annotation in TestNG framework where the main business logic resides.** All functionalities to be automated are kept inside the @Test annotation method. It has various attributes based on which the method can be reformed and executed.

Example of a code snippet below validating the url :

@Test

public void testCurrentUrl() throws InterruptedException

{

driver.findElement(By.xpath(path).click();

String currentUrl= driver.getCurrentUrl();

assertEquals(current\_url, Expected URL);

}

@BeforeTest

**This annotation is run before your first @Test annotation method in your class.** You can use this annotation in TestNG for Selenium to setup your browser profile preferences, for example auto opening your browser in maximize mode, setting up your own customized profile for your browser etc.

Below is the code snippet for BeforeTest method ensuring the browser opens in maximize mode:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | @BeforeTest      public void profileSetup()      {          driver.manage().window().maximize();          } |

@AfterTest

**This annotation in TestNG runs after all your test methods belonging to your class have run.** This is a useful annotation which comes handy in terms of reporting your automation results to your stakeholders. You can use this annotation to generate report of your tests and share it to your stakeholders via email.

Example of the code snippet below:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @AfterTest      public void reportReady()      {          System.out.println("Report is ready to be shared, with screenshots of tests");      } |

@BeforeMethod

**This annotation in TestNG runs before every @test annotated method.** You can use it to check out for the database connections before executing your tests or lets say different functionality been tested in your @test annotated method which requires user login in a certain class. In this case also you can put your login code in the @BeforeMethod annotation method.

Below code snippet is an example, displaying login functionality of the LambdaTest platform:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @BeforeMethod      public void checkLogin()      {            driver.get("https://accounts.lambdatest.com/login");            driver.findElement(By.xpath("//input[@name='email']")).sendKeys("sadhvisingh24@gmail.com");            driver.findElement(By.xpath("//input[@name='password']")).sendKeys("XXXXX");            driver.findElement(By.xpath("//\*[@id='app']/section/form/div/div/button")).click();        } |

@AfterMethod

**This annotation runs after every @test annotated method.** This annotation can be used to take screenshots of every test method ran against test runs.

Below code snippet indicating screenshot taken in the @AfterTest annotation in TestNG for Selenium:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | @AfterMethod      public void screenShot() throws IOException      {          TakesScreenshot scr= ((TakesScreenshot)driver);          File file1= scr.getScreenshotAs(OutputType.FILE);           FileUtils.copyFile(file1, new File("C:\\Users\\navyug\\workspace\\QAPractise\\test-output\\test1.PNG"));        } |

@BeforeClass

**This annotation runs before the first test method in the current class.**This annotation can be used to setup your browser properties, initialize your driver, opening your browser with the desired URL etc.

Below is the code snippet for BeforeClass:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @BeforeClass      public void appSetup()      {          driver.get(url);        } |

@AfterClass

**This annotation runs after the last test method in the current class.** This annotation in TestNG can be used to perform clean up activities during your tests like closing your driver etc

Below is the example of code snippet showing closing activities performed:

|  |  |
| --- | --- |
| 1  2  3  4  5 | @AfterClass      public void closeUp()      {          driver.close();      } |

@BeforeSuite

A suite can consist of multiple classes, this annotation runs before all the tests methods of all the classes. **This annotation marks the entry point of execution.** @BeforeSuite annotation in TestNG can be used to perform the needed and generic functions like setting up and starting Selenium drivers or remote web drivers etc.

Example of @BeforeSuite annotation in TestNG, code snippet showcasing setting up of driver:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @BeforeSuite      public void setUp()      {      System.setProperty("webdriver.chrome.driver", "path to chrome driver");      driver=new ChromeDriver();      } |

@AfterSuite

**This annotation in TestNG runs post all the test methods of all the classes have run.** This annotation can be used to clean up the processes before completing off your tests when you have multiple classes in functioning, for example closing the drivers etc.

Below is the code snippet for @AfterSuite annotation in TestNG for Selenium:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @AfterSuite      public void cleanUp()      {            System.out.println("All close up activities completed");      } |

.

**GROUPS**Groups in TestNG denotes the process of grouping different tests together into a straightforward group and running these tests together by just running the group in a single command. It does not even matter if they belong to different classes.

We have used the group name called “***demo***” but only on two methods,. Once we are done with the code, we need to tell our XML file about the groups (if we want to run according to the groups).

**Example**

<suite name="Test-Suite" >

   <test name="TestName" >

   <groups>

   <run>

   <include name = "demo"></include>

   </run>

   </groups>

       <classes>

          <class name="ClassName" />

       </classes>

   </test>

</suite>

We have included the name of the group “***demo***” inside our XML file within the ***“include”*** tag. By this, we expect to run only two tests because only two tests are included in the group “***demo.”***

***Similar to include, We can also use <exclude> tag which simply means to exclude the mentioned tests.***

<suite name=*"Suite"*>

<test thread-count=*"5"* name=*"Test"*>

<groups>

<run>

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

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

</run>

</groups>

<classes>

<class name=*"Demo.Nitish.launchbrowser"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

Now we have a pattern in the groups i.e. the word “***demo***” is common in both the groups. Include the **regex** in your XML file and check whether TestNG accepts the regular expression or not.

<suite name=*"Suite"*>

<test thread-count=*"5"* name=*"Test"*>

<groups>

<run>

<include name = *"demo.\*"*></include>

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

</run>

</groups>

<classes>

<class name=*"Demo.Nitish.launchbrowser"*/>

<class name=*"Demo.Nitish.secondbrowser"*/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

**PRIORITY AND TEST SEQUENCE**

Prioritization in TestNG is a way to provide a sequence to the methods so that they do not run out of order. Since alphabetically running test cases in TestNG have no logical sequence (***If no priority is provided -Test cases are run Alphabetically***), providing priority to these test cases helps us managing our tests’ execution.

*Priority in TestNG test cases is a parameter with attribute value as “priority.”*

The following is the syntax for allocating a priority to a test case method.

***@Test (priority = 1)***

***public void func(){***

***//test code***

***}***

* *Definition of Priority in TestNG test methods can only be the @Test methods.*
* *Lower the priority number; higher is the priority of the test case method.*
* *Priority in TestNG contains only integer value. The value can be negative, zero, or positive.*
* *One method is allowed to have only one priority in TestNG.*
* *Priority cannot pass through the XML files.*
* *If no priority is provided -The default priority is assigned as 0*
* *If both methods have same priority then it again refers to the basic rule of running the test cases* ***alphabetically***

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

*You can skip the test case by using enabled=false in the argument of @Test*

***DEPENDENT TESTS***

In TestNG, we often require to run the test in a specific order. Along with that, we may require that a test must run only when another test has run. For example, I want testB to run if testA has run. By this, I denote that testB is dependent on testA, and these are called ***Dependent Tests in TestNG***

public class Dependent {

  @Test (dependsOnMethods = { "OpenBrowser" })

  public void SignIn() {

  System.out.println("This will execute second (SignIn)");

  }

  @Test

  public void OpenBrowser() {

  System.out.println("This will execute first (Open Browser)");

  }

}

***Using attributes dependsOnGroups in @Test annotations.***

The dependsOnGroups attribute lets us make a test depend on a whole group rather than a single test. For example, see the code below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public class GroupDependency  {      @Test(dependsOnGroups = { "SignIn" })      public void ViewAcc() {          System.out.println("SignIn Successful");      }        @Test(groups = { "SignIn" })      public void LogIn() {          System.out.println("Logging In Success");      }  } |

**Multiple Dependents**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | import org.testng.annotations.Test;  public class DependsOnTest  {      @Test      public void OpenBrowser() {          System.out.println("Opening The Browser");      }        @Test(dependsOnMethods = { "SignIn", "OpenBrowser" })      public void LogOut() {          System.out.println("Logging Out");      }        @Test      public void SignIn() {          System.out.println("Signing In");      }  } |

**DEPENDENCY IN XML FILE**

You can create dependency in groups in your xml file as well

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >

<suite name="TestNG XML Dependency Suite" >

   <test name="Test" >

   <groups>

   <dependencies>

   <group depends-on= "openbrowser" name= "login"></group>

   <group depends-on= "login" name= "viewaccount"></group>

   <group depends-on= "viewaccount" name= "logout"></group>

   </dependencies>

   </groups>

       <classes>

          <class name="GroupDependency" />

       </classes>

   </test>

</suite>

***TESTNG REPORTERS***

Reporter class in [TestNG](https://testng.org/doc/documentation-main.html) helps the testers log the messages on to the test execution reports. These messages commonly denote the status, information, or any other detail about the test execution. But, with no hard and fast rules, these messages can be anything.

### *****Syntax For Using TestNG Reporter Log:*****

Reporter.log(“message”);

**Import Required:***import org.testng.Reporter;*

* *Refer to index.html in ‘test-output’ folder : Copy the path of html file, Hit the browser and navigate to Reporter Output where you can see your output has been logged successfully*
* *Same logging can also be seen in ‘emailable-report.html’ under same path.*

***TESTNG ASSERTIONS***

***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.

General Syntax:

Assert.Method(actual,expected)  
Assert.Method(actual,expected,message)

**TYPES OF ASSERTIONS**

**HARD ASSERT  
SOFT ASSERT**

**HARD ASSERT  
*Hard Asserts*** are those asserts that stop the test execution when an assert statement fails, and the subsequent assert statements are therefore not validated.

***Hard asserts are the default type of asserts in TestNG***

**SOFT ASSERT**

Soft asserts are just the opposite of hard asserts. In soft asserts, the subsequent assertions keep on running even though one assert validation fails, i.e., the test execution does not stop. ***Soft assert does not include by default in TestNG***. For this, you need to include the package **org.testng.asserts.Softassert.**

***Soft asserts are also known as***“Verify”

Need to create object for Soft Asset:  
SoftAssert softassert = new SoftAssert();  
softassert.assertEquals(originalTitle, expectedTitle);

**DIFFERENT TYPES OF ASSERTIONS**

* ***Assert.assertEqual****(String actual, String expected): Pass the actual string value and the expected string value as parameters. Validates if the actual and expected values are the same or not.*
* ***Assert.assertEqual****(String actual, String expected, String message): Similar to the previous method just that when the assertion fails, the message displays along with the exception thrown.*
* ***Assert.assertEquals****(boolean actual, boolean expected): Takes two boolean values as input and validates if they are equal or not.*
* ***Assert.assertTrue****(condition)****:****This method asserts if the condition is true or not. If not, then the exception error is thrown.*
* ***Assert.assertTrue****(condition, message):  Similar to the previous method with an addition of message, which is shown on the console when the assertion fails along with the exception.*
* ***Assert.assertFalse****(condition): This method asserts if the condition is false or not. If not, then it throws an exception error.*
* ***Assert.assertFalse****(condition, message): Similar to the previous method but with an addition of a message string which is shown on the console when the assertion fails, i.e., the condition is true.*

There are various others & are used as per requirement- Study about them more.

## **TESTNG PARAMETERS What are the TestNG parameters?**

**Parameters** in TestNG is similar to **annotations in TestNG**in their declaration. Similar to parameters in any other ***programming language***, they are declared to pass some values onto the function. A simple reason to use parameters is that they let us run a function many times with different values or to run different functions with the same values. ***Parameters pass the values in the runtime.***

***Syntax:***

***@Parameters ({“a”, “b”})***

### *****How To Run TestNG Parameters?*****

Let’s say we want to add two numbers using the TestNG parameters. Observe the below-given code for the same.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | import org.testng.annotations.Parameters;  import org.testng.annotations.Test;    public class Params  {      @Test      @Parameters ({"val1", "val2"})      public void Sum(int v1, int v2) {       int finalsum = v1 + v2;          System.out.println("The final sum of the given values is " + finalsum);      }  } |

***We cannot run the test as Run As>TestNG Test. You need to specific parameters at the XML level.***

**Definition at XML file level**

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">

<suite name="TestNG Parameters Suite">

   <test name="Params">

      <parameter name="val1" value="2" />

      <parameter name="val2" value="3" />

      <classes>

         <class name="Params" />

      </classes>

   </test>

</suite>

*You can use the parameters are Test and Suite level.*

***CROSS BROWSER TESTING USING PARAMETERS***

***When same test needs to be performed on different browsers then we do cross browser testing as we need to ensure that the application is perfectly working fine on all the supported browsers and versions.***

***Let’s see below as how to perform cross browser testing using parameters***

public class MultiBrowser {

public WebDriver driver;

*@Parameters*("browser")

*@BeforeClass*

  // Passing Browser parameter from TestNG xml

  public void beforeTest(String browser) {

  // If the browser is Firefox, then do this

  if(browser.equalsIgnoreCase("firefox")) {

//Initializing the firefox driver (Gecko)

  driver = new FirefoxDriver();

System.setProperty(‘location’);

  }else if (browser.equalsIgnoreCase("chrome")) {

  //Initialize the chrome driver

  driver = new ChromeDriver();

 System.setProperty(‘location’);

  }

  // Enter the website address in the browser

  driver.get("Site\_URL");

  }

  // Once Before method is completed, Test method will start

*@Test* public void login() throws InterruptedException {

driver.findElement(By.xpath”XPath”).click();

}

*@AfterClass* public void afterTest() {

driver.quit();

}

}

***PARALLEL TESTING  
is more recommended using Selenium Grid.***

## **What is Parallel Testing and Why is it important?**

***Parallel testing or parallel execution, as the name suggests, is a process of running the test case parallelly rather than one after the other. It is important to save a lot of time taken to run hundreds of tests sequentially.***

### *****Advantages of Parallel Testing*****

If we look at the bigger picture, parallel testing has the following advantages:

* ***Reduces Time***: Running the tests in parallel reduces the overall execution time.
* ***Allow Multi-Threaded Tests***: Using the parallel execution in TestNG, we can allow multiple threads to run simultaneously on the test case providing independence in the execution of different components of the software.

### *****Running test methods parallelly in TestNG using Selenium*****

In the following section, we will be running the test methods in parallel using TestNG. So, all the methods that come under the annotation @Test will run parallel when we execute the test suite. The following code will initialize the drivers of two different browsers (Chrome and Firefox) in parallel.

* **Write 2 different browser invocation method for Chrome, Google with @Test annotation and configure parallel attribute in your XML file and run to see the result, Both the browsers will be launched simultaneously achieving parallel execution.**
* **Thread attribute is also provided : Thread is an instance say on which these tests run so if you need to specific thread count as per how many parallel execution you want do.**
* **Default value for Thread count is 5 so it can handle 5 executions in parallel even if you do not provide the attribute.**
* **If you want to run the parallel execution on class that can also be done and you just need to modify parallel attribute value to ‘classes’ instead of methods and include your class in classes and you are good to go to execute all your parallel executions.**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >  <suite name = "Parallel Testing Suite">     <test name = "Parallel Tests" **parallel = "methods**" **thread-count = "2"** >        <classes>           <class name = "ParallelTest" />        </classes>     </test>  </suite> |

## **Configuring the Test Methods to run parallelly in TestNG**

In this complete tutorial, we discussed how to run the test methods, classes, and suites parallelly in TestNG using selenium web driver. It included mentioning the thread-count to tell TestNG how many threads we would like to create. But, ***in TestNG, we also get the liberty to run a single test method parallelly by configuring it inside the test code itself***. We do it by making a few changes in @Test annotation. Observe the following code and the parameters used in @Test annotation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | import org.testng.annotations.Test;  public class TestNG  {      @Test(threadPoolSize = 4, invocationCount = 4, timeOut = 1000)      public void testMethod()      {          System.out.println("Thread ID Is : " + Thread.currentThread().getId());      }  } |

In the @Test annotation, we need to mention three parameters:

* ***threadPoolSize***: The number of threads we would like to create and run the test parallelly.
* ***invocationCount***: The number of times we would like to invoke this method.
* ***timeOut***: The maximum time a test execution should take. If exceeded, the test fails automatically.

## **What is a DataProvider in TestNG?**

Similar to TestNG Parameters, DataProviders are a means to pass data to test scripts in TestNG. Using DataProvider in TestNG, we can easily inject multiple values into the same test case. It comes inbuilt in TestNG and is popularly used in data-driven frameworks.

The syntax for a DataProvider in TestNG is as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @DataProvider(name = ”name of the data provider”)  public Object[][] dataProviderfunc(){  Return new Object[][]{  values  }  } |

Now, let’s try to understand the different components of this syntax.

* The DataProvider annotation has a single attribute called name, which you can select as per your convenience.
* DataProviders are separate methods used in test functions, which means that this annotation is not used on test functions like the testNG parameters.
* The DataProvider method returns a 2D list of objects.
* In case you do not define a name for the DataProvider, the DataProvider method name is considered its default name. So, the name of the DataProvider calls the DataProvider method.

## **Using DataProvider in TestNG**

Let’s see how to implement it:

|  |
| --- |
| public class SimpleTest { |
|  |  |
|  | WebDriver driver; |
|  |  |
|  | @DataProvider(name = "test-data") |
|  | public Object[][] dataProvFunc(){ |
|  | return new Object[][]{ |
|  | {"Lambda Test"},{"Automation"}, {“”} |
|  | }; |
|  | } |
|  |  |
|  | @BeforeMethod |
|  | public void setUp() { |
|  |  |
|  |  |
|  |
|  |
|  |
|  |
|  |
|  | **YOUR CODE** |
|  | } |
|  | //Passing the dataProvider to the test method through @Test annotation |
|  | @Test(dataProvider ="test-data") |
|  | public void search(String keyword, String second, String name){ |
|  | WebElement txtBox = driver.findElement(By.xpath("//input[@class='gLFyf gsfi']")); |
|  | txtBox.sendKeys(keyWord); |
|  | Reporter.log("Keyword entered is : " +keyWord); |
|  | txtBox.sendKeys(Keys.ENTER); |
|  | Reporter.log("Search results are displayed."); |
|  | } |
|  |  |
|  | @AfterMethod |
|  | public void burnDown(){ |
|  | driver.quit(); |
|  | } |
|  |  |
|  | } |

**Inherting Data Provider in TESTNG**

Create separate class and call the method from the class wherever you would want to use it and as the name suggest it will inherit the method.

Data Provider Class:

|  |  |
| --- | --- |
| 3  4  5  6  7  8  9  10 | package dataProviders;  import org.testng.annotations.DataProvider;  public class DPClass {     @DataProvider(name = "test-data")     public static Object[][] dataProvFunc(){           return new Object[][]{                 {"Lambda Test"},{"Automation"}           };     }  } |

Now this will be inherited in any of the @Test you like as mentioned below:

**@Test(dataProvider=”name”, dataProviderClass=DPClass.class)**