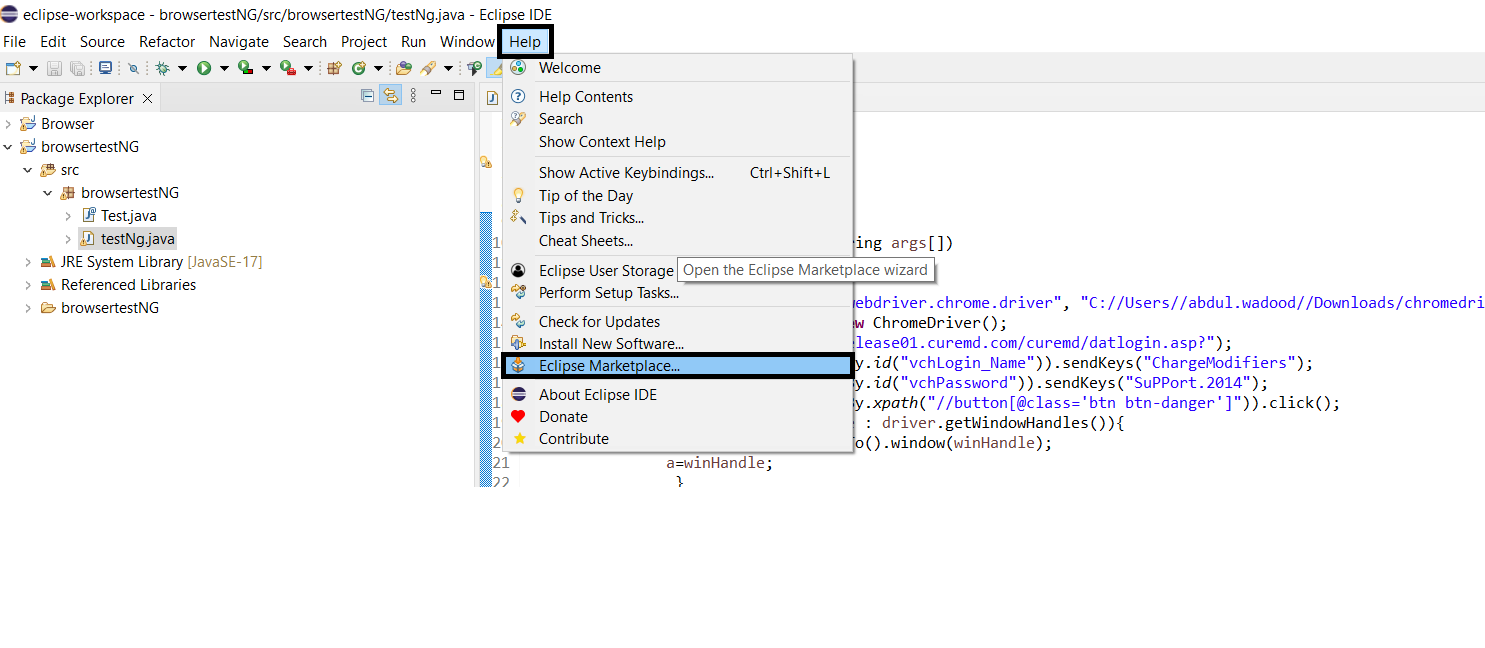
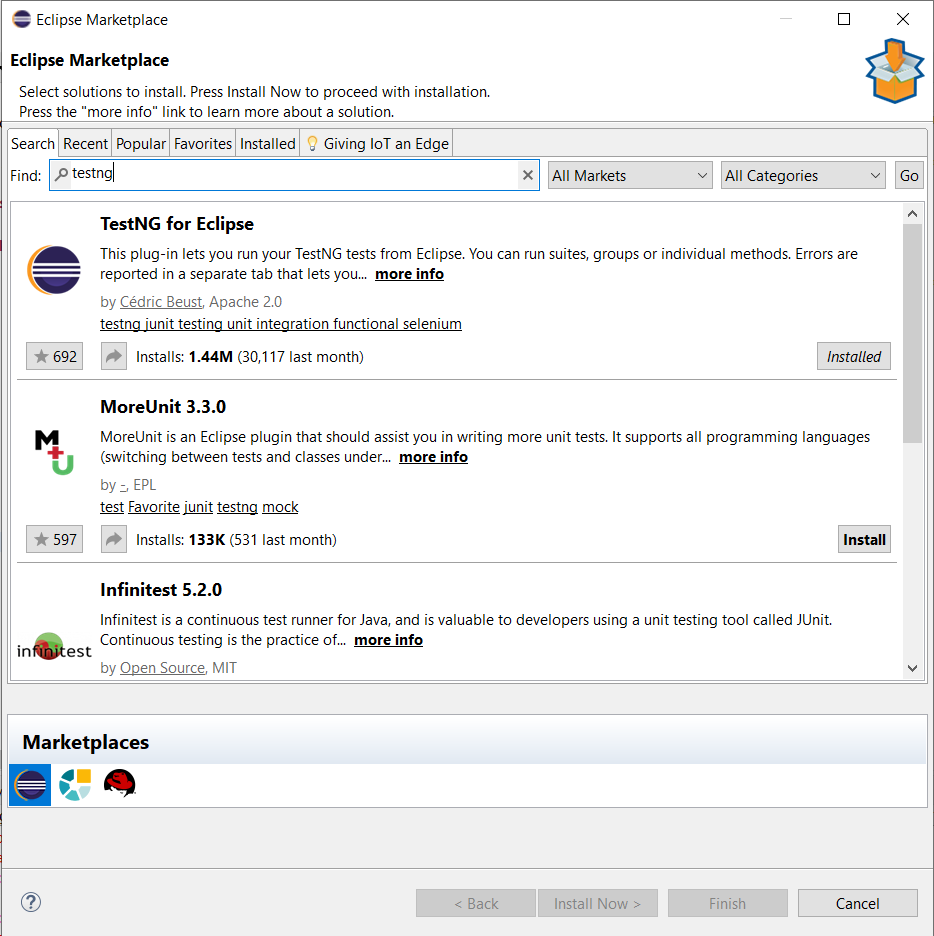
Download & Install Selenium WebDriver

## **TestNG installation**

* Launch the eclipse IDE and go to Help and then to Eclipse Marketplace.



* Write TestNG in the search bar and install it.



1. **Launching Web Browser in Selenium**

* Launch Eclipse IDE.
* Create New Java Project.
* Create POM file.
* Add required dependencies i.e, selenium and web driver manager.

<dependency>

<groupId>io.github.bonigarcia</groupId>

<artifactId>webdrivermanager</artifactId>

<version>5.4.1</version>

</dependency>

<dependency>

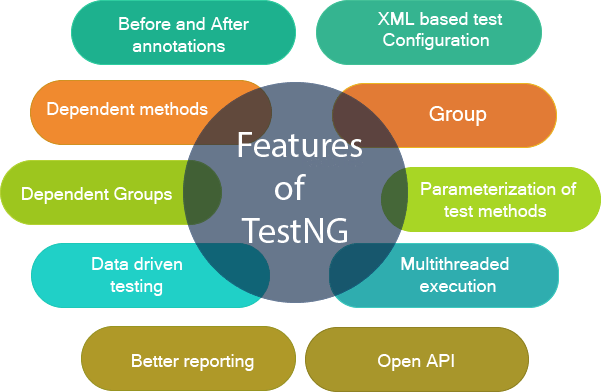
<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-devtools</artifactId>

<version>4.0.0-rc-1</version>

</dependency>

Features of TestNG



### **Multiple Before and After annotation options**

Before and after annotations are used to execute a certain set of code before and after executing the test methods. These annotations are used to set the variables or configuration before the start of the execution of test methods and clean up all the variables after the execution ends. Some of the Before and After annotations are @BeforeSuite, @BeforeTest, @BeforeGroups, @BeforeClass, etc.

### **XML-based test configuration**

Test suites in a Testng are mainly configured by using the XML-based file. Testng.xml file is used to organize and run the test suites. The testng.xml file is used to create the test suites by using classes, test methods, packages as well as by using the test groups. It is also used to pass the parameters to test classes or methods.

### **Dependent methods**

Dependency is a feature of Testng that allows a test method to depend on the single or group of test methods. Dependency works on the principle "depend-on-method" which must be either in the same class or in the inherited base class. This is the most important feature in TestNG that tells the TestNG to run the dependent test method after the execution of a given test method. You can also configure whether you want dependent test method should be executed or not even after the execution of the given test method fails.

### **Groups/group of groups**

TestNG groups allow you to group the test methods. By using TestNG groups, you can declare the methods in a group as well as you can declare the groups within a group. The Testng group can be used to include a certain set of groups and can exclude another set of groups.

### **Dependent groups**

Similar to the Dependent methods, test methods in a group can depend on the test methods of another group.

### **Parameterization of test methods**

One of the most important feature of TestNG is Parameterization. This feature allows you to pass the arguments as parameters and this achieved by using testng@Parameters annotation. We can pass the parameters to test methods in two ways, i.e., testng.xml file and DataProviders.

### **Data-driven testing**

TestNG allows users to perform data-driven testing. This testing allows users to execute the same test multiple times with multiple sets of data. To achieve the data-driven testing, DataProvider feature is used. DataProvider is a data feeder method that executes the test method with multiple sets of data.

### **Multithreaded execution**

Multithreaded execution is the parallel execution of tests. Multithreading means the execution of multiple parts of software at the same time. Based on the configuration in the XML file, multiple threads are started, and test methods are executed in them. Multithreaded execution saves a lot of execution time.

### **Better reporting**

Testng provides XML and HTML reports by default for test execution. You can even add your own custom reports when required.

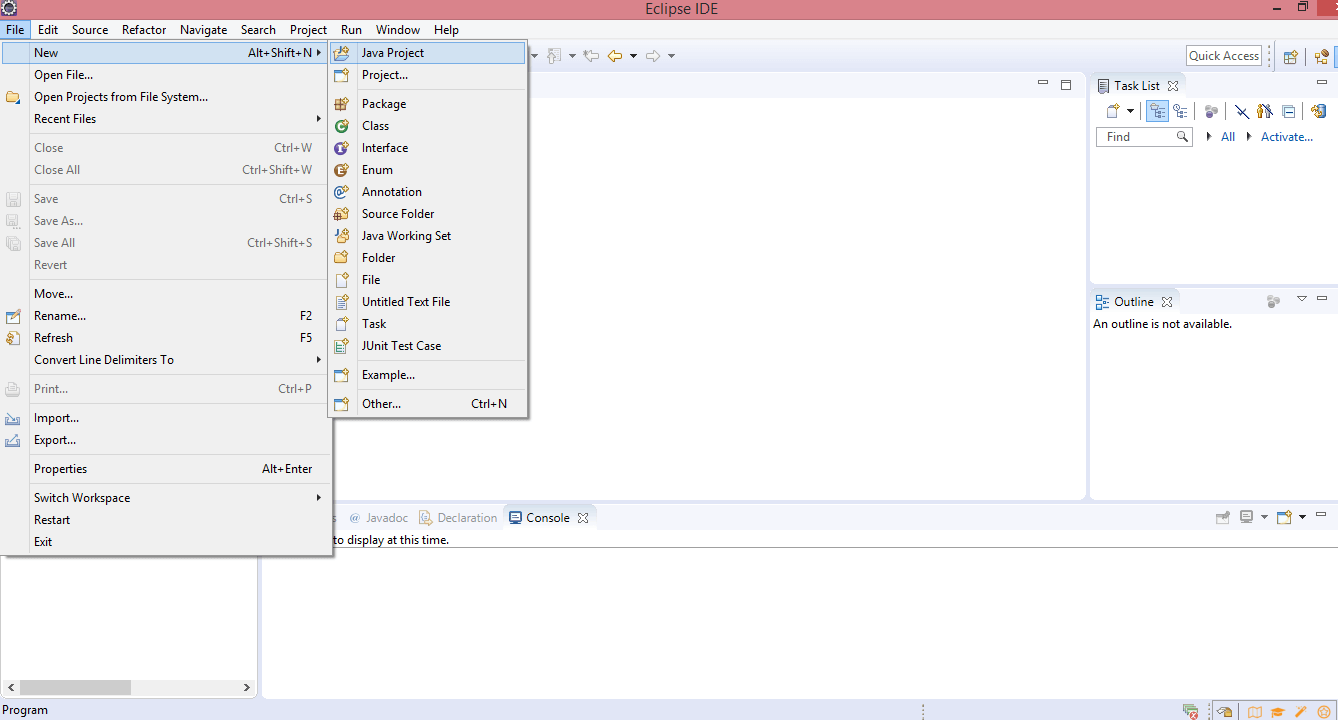
### **Open API**

TestNG contains the open API means API is publicly available to the developers. This feature allows you to create your custom extensions in your framework when required.

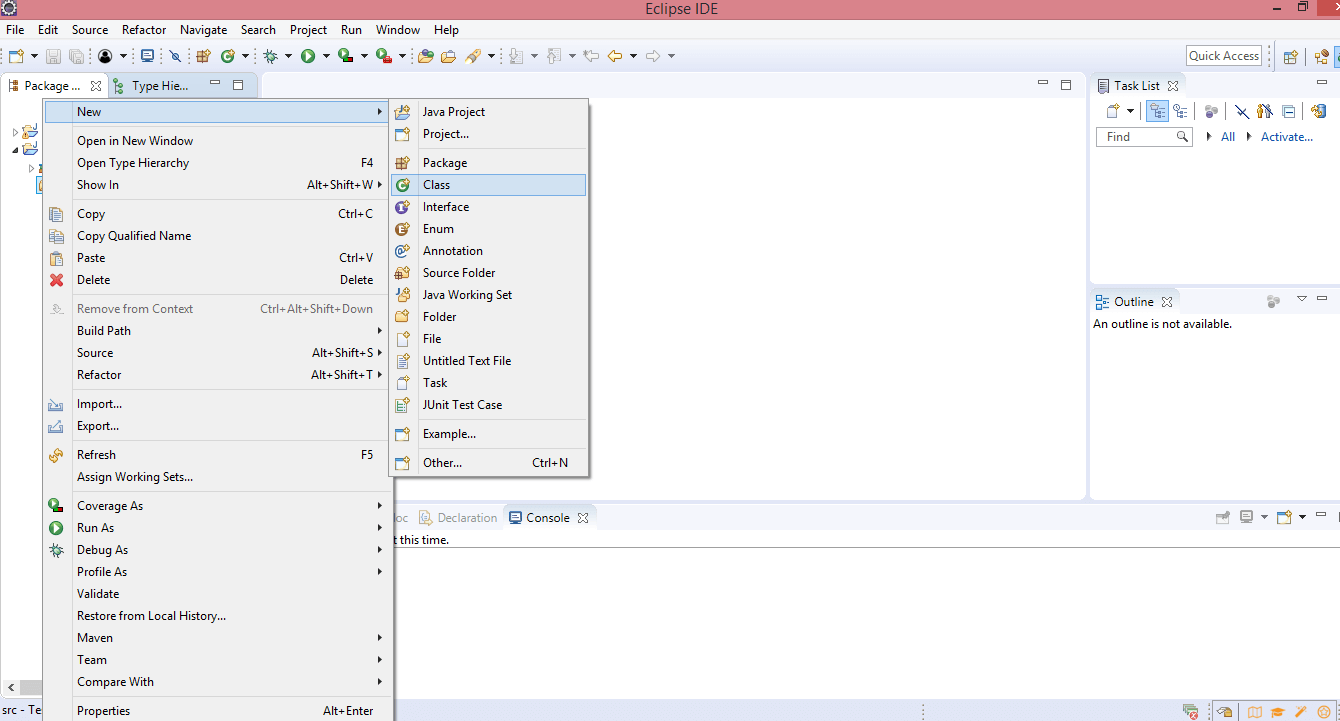
# Running test cases in TestNG without java compiler

**Let's create a new project in TestNG.**

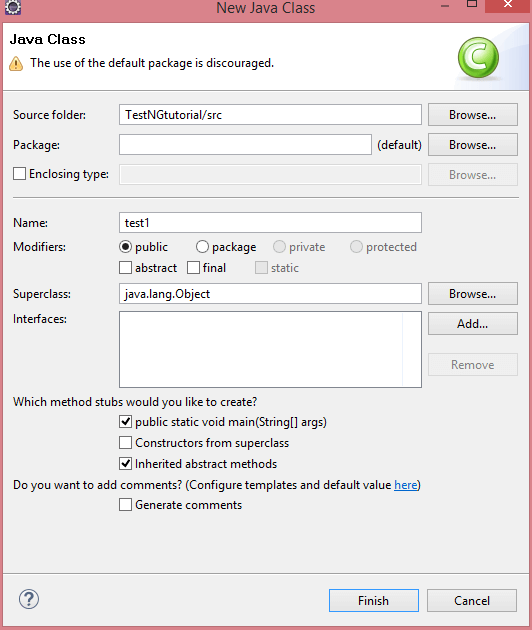
**Step 1:** Create a TestNG project. It is created in the same way as we create the java project. Click on the **File > New > Java project**.



**Step 2:** Now we will create the Java class file in a src folder. Click on the File > New > Java Project.



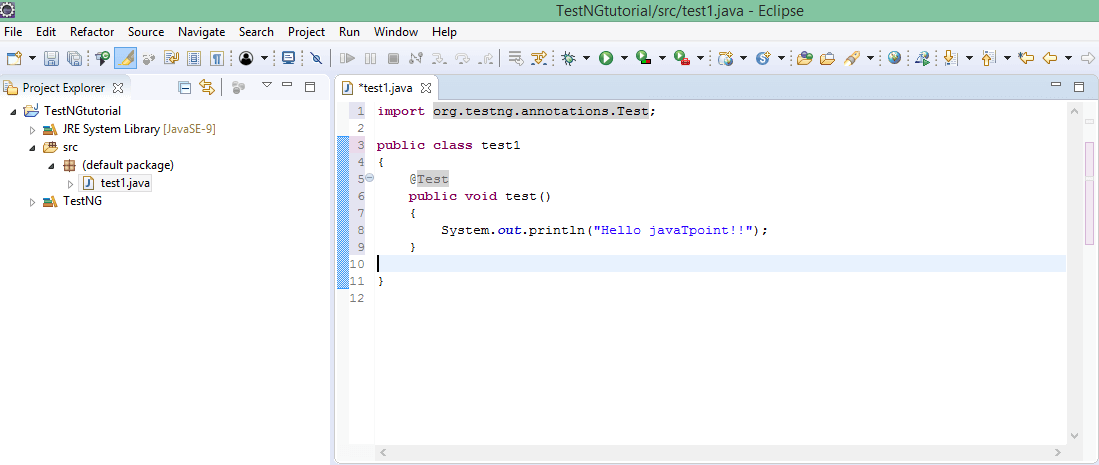
**Step 3:** Enter the class name. Suppose I gave the class name as test1. As we know that execution of java file starts with the main method, check the box of the main method.



In the above screen, we observe that **public static void main()** method has been included in the **test1** class file, so it means that the program will run on Java compiler.

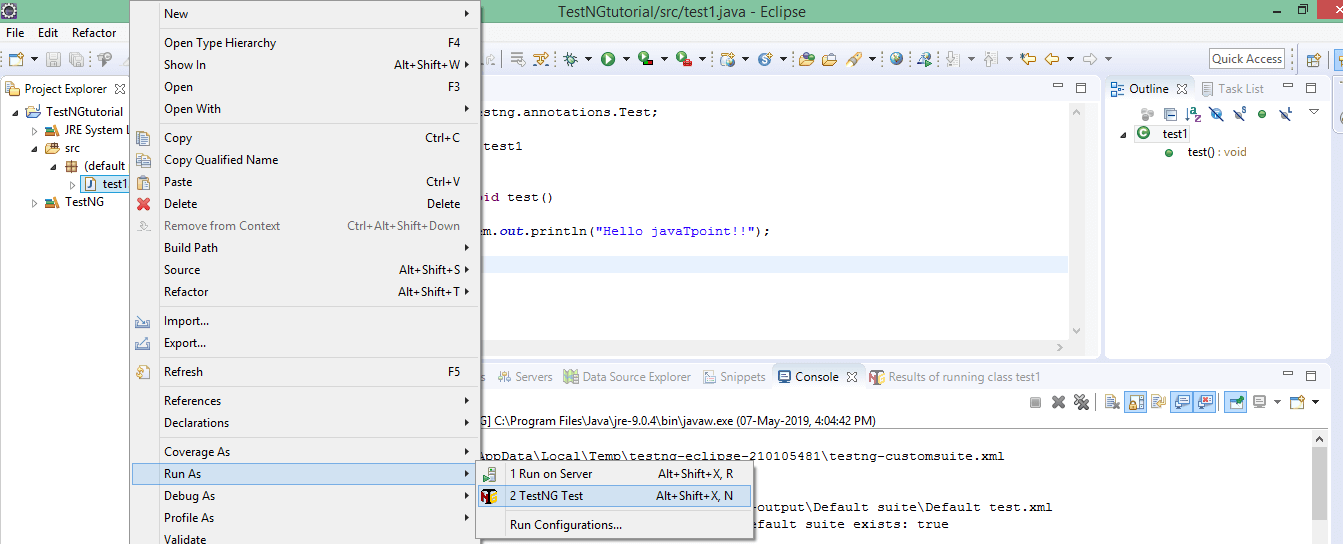
**Step 4:** If you install TestNG framework or TestNG library, then you do not need to compile on Java compiler as TestNG itself is a java compiler that compiles the test cases. In order to achieve this, we need to install the TestNG plug-in and then add the TestNG plug-in in a project.

**Step 5:** Now we will create a simple program.

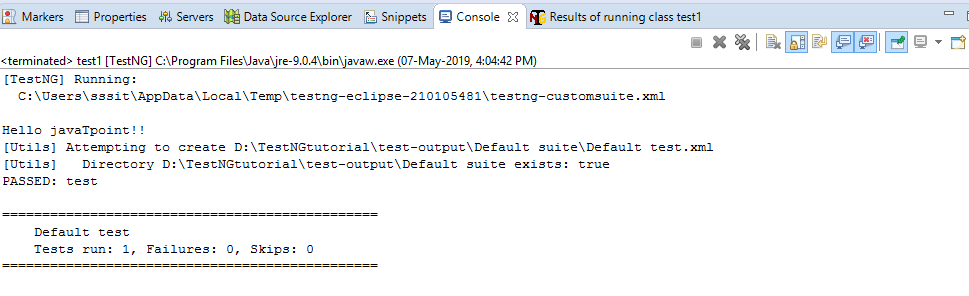


In the above case, we have created a test() method and the class test1 does not contain the main() method. In this case, we use @Test notation which will run the program without any main() method.

**Step 5:** Run the program by using TestNG.



**Output**



# Importance of XML file in TestNG Configuration

In TestNG, you can define multiple test cases in a single class whereas, in Java, you can define only one test in a single class in the main() method. In Java, if you want to create one more test, then you need to create another java file and define the test in the main() method.

Polymorphism in Java | Dynamic Method Dispatch

Instead of creating test cases in different classes, we recommend you to use TestNG framework that allows you to create multiple test cases in a single class.

You can create multiple test cases with the help of **@Test** annotation.

**Let's understand through an example.**

**public** **class** test

{

   @Test

**public** **void** test1()                      // First test case.

  {

    System.out.println("test1");

  }

  @Test

**public**  **void** test2()                     // Second test case.

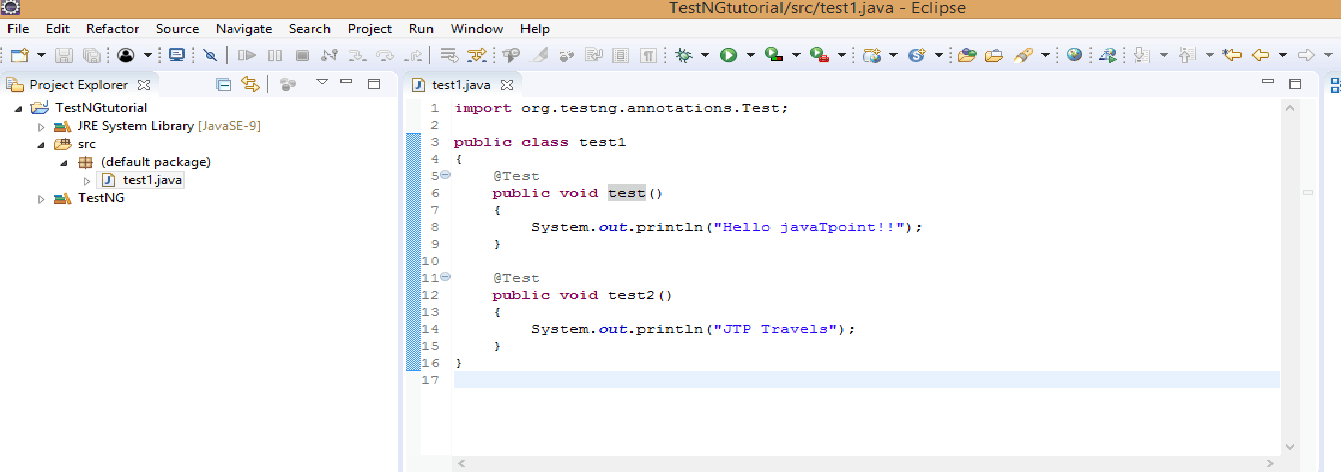
{

 System.out.println("test2");

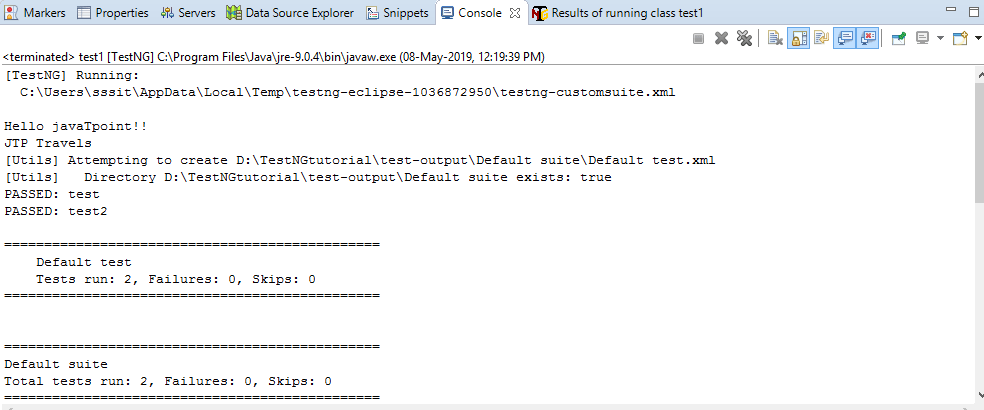
 }}

The above code consists of a class test. The class test consists of two test cases, i.e., test1() and test2(). You can differentiate the test cases by considering the sequence of test cases. In the above code, the test case **test2()** is written in the second @Test annotation, so this test case will be considered as the second case.

**Source code**



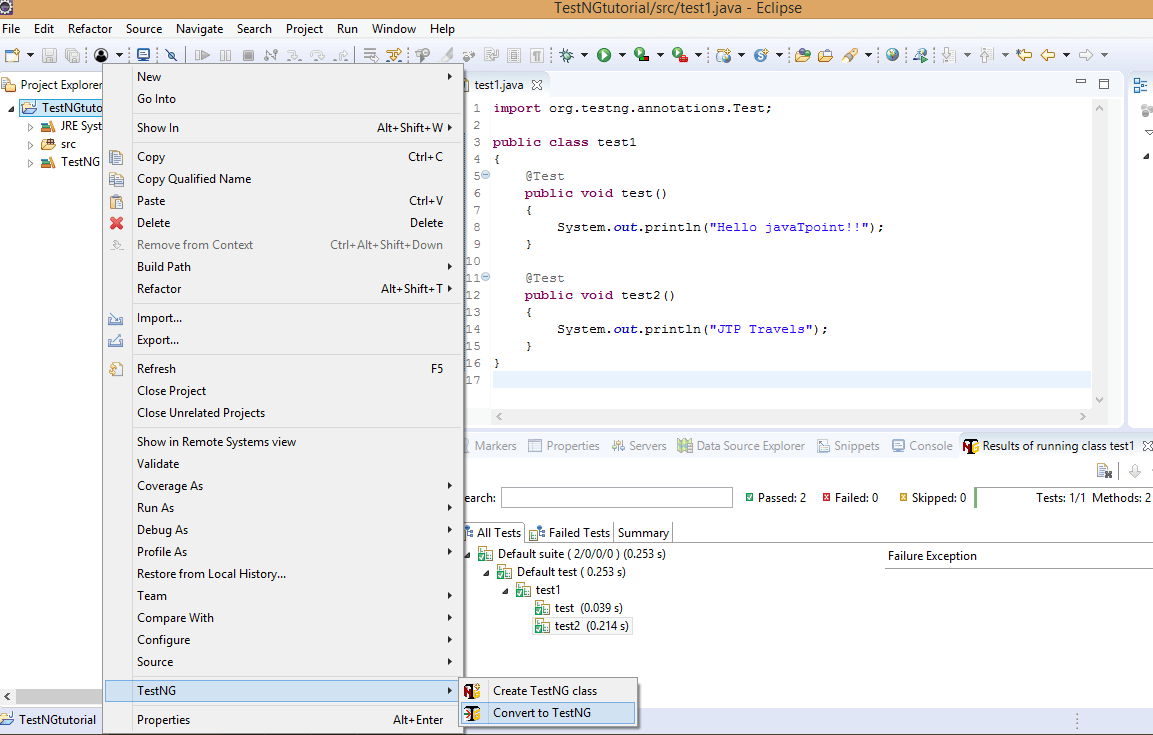
**Output**



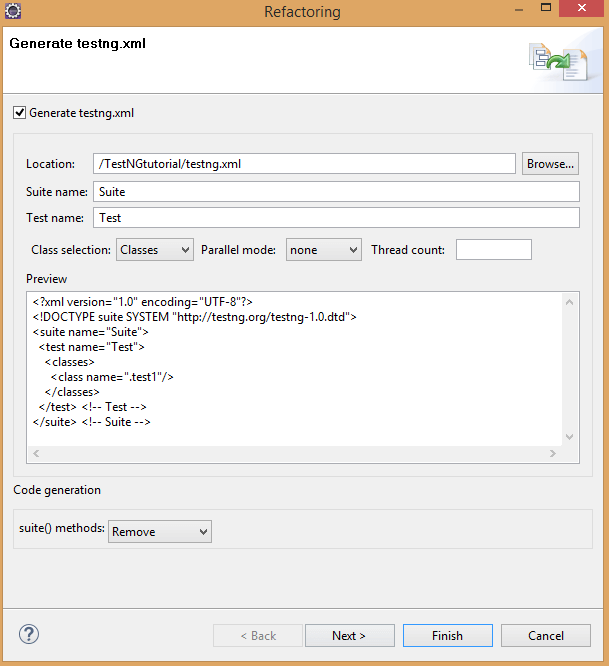
#### **Note: You can trigger all the test cases from a single file known as xml file. Xml file is the heart of TestNG framework.**

## **How to create a xml file**

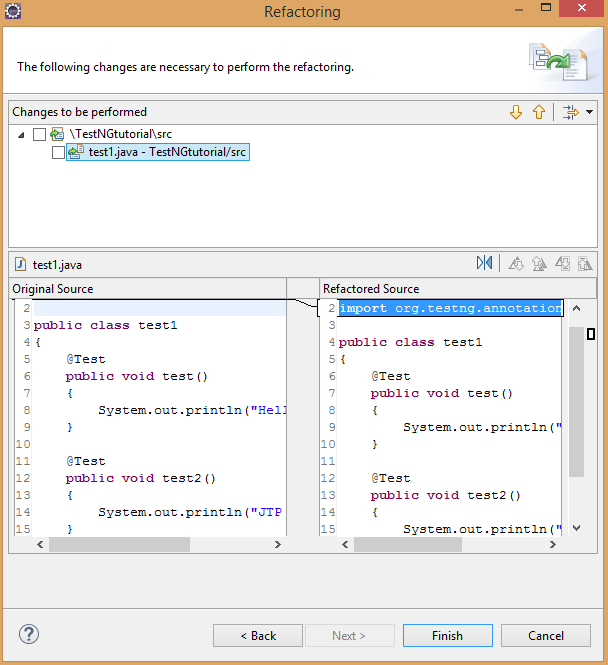
* Right click on the project. Move your cursor down, and you will see TestNG and then click on the **Convert to TestNG**.



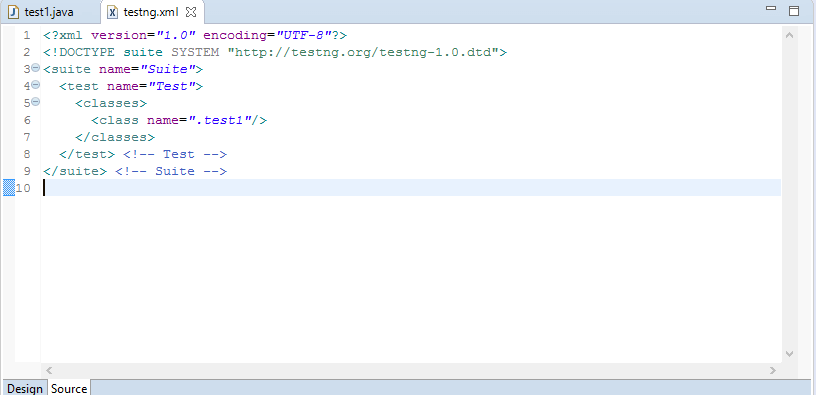
* The below screen shows the preview of the xml file. Click on the **Next** button.



* Click on the **Finish** button.



* The testing.xml file is shown below:



In the above source code of xml file, suite is at the higher hierarchy in TestNG. Inside the , you have to define the test name folder. This test name folder is the name of the folder. For example, In a loan company, there are three different types of modules such as personal loan, home loan and car loan, and each module contain its own test cases. All these test cases are defined in the test name folder.

### **Now we will create the module of personal loan.**

**Step 1:** We first create two java files and both the files contain test cases.

**tes1.java**

**package** day1;

**import** org.testng.annotations.Test;

**public** **class** module1

{

    @Test

**public** **void** test1()

    {

        System.out.println("Hello javaTpoint!!");

    }

    @Test

**public** **void** test2()

    {

        System.out.println("JTP Travels");

    }}

**test2.java**

**package** day1;

**import** org.testng.annotations.Test;

**public** **class** module2

{

    @Test

**public** **void** test3()

  {

      System.out.println("hindi100.com");

  }

}

**Step 2:** Now we will create the xml file.

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

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

<suite name="loan\_department">

  <test name="Personal\_loan">

    <classes>

      <**class** name="day1.module1"/>

      <**class** name="day1.module2"/>

    </classes>

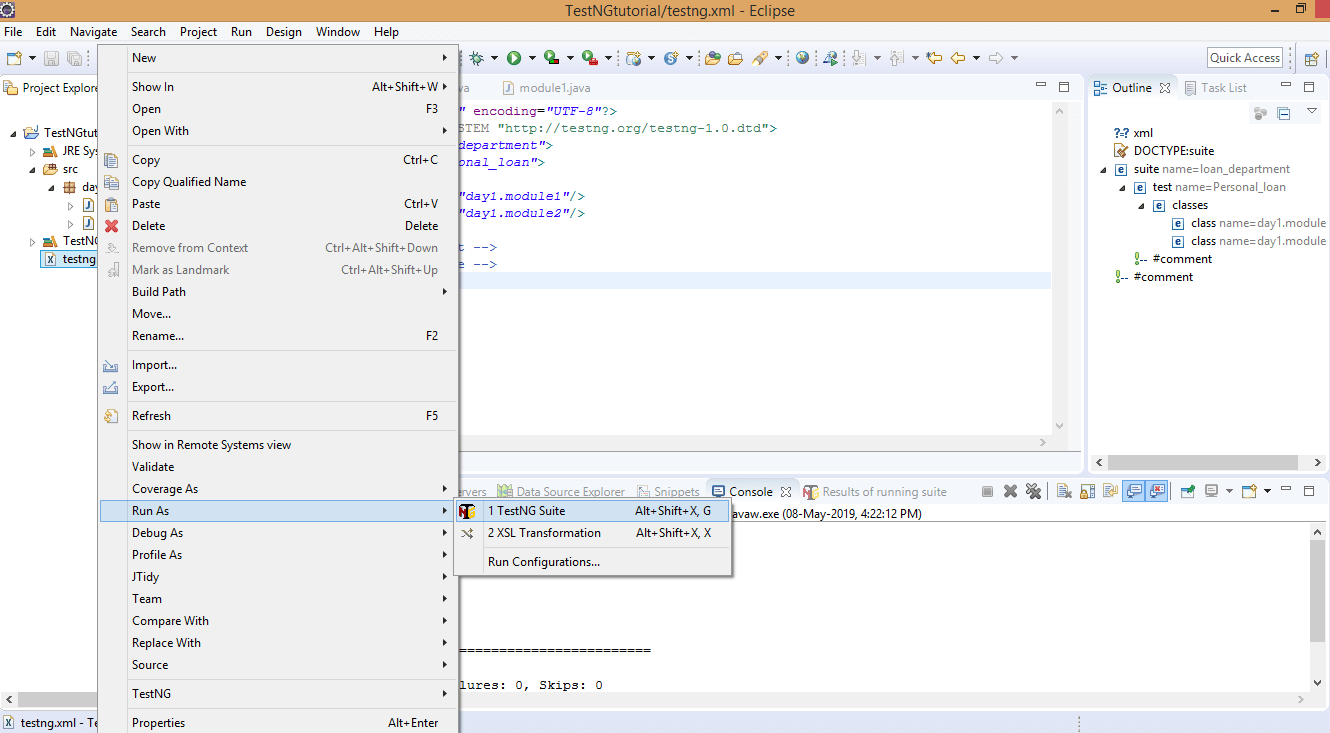
  </test> <!-- Test -->

</suite> <!-- Suite -->

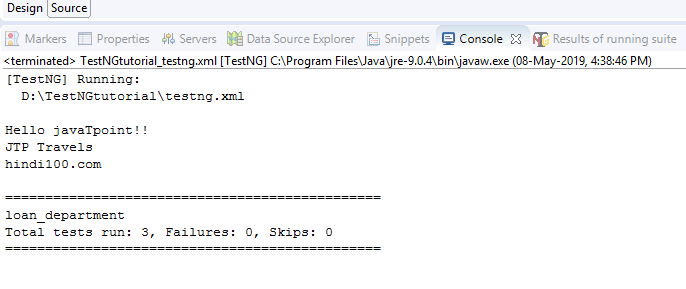
In the above XML file, we have created the suite "**loan\_department**". We have created the module "Personal loan" inside the suite and within this module, we have created the test cases defined in the classes day1.module1 and day1.module2, where day1 is the package name and module1 and module2, are the classes.

**Step 3:** In this step, we will run the test cases. Now we do not need to run the java files individually. We have to run the XML file which will automatically execute all the test cases as we have configured all the class files inside the XML file that are containing test cases.

Right click on the **testng.xml** file and then move down to the **Run As** and then click on **the1 TestNG Suite**.



**Output**



In the above output, we observe that all the test cases run successfully without any failure.

# What is TestNG Annotation?

TestNG Annotation is a piece of code which is inserted inside a program or business logic used to control the flow of execution of test methods.

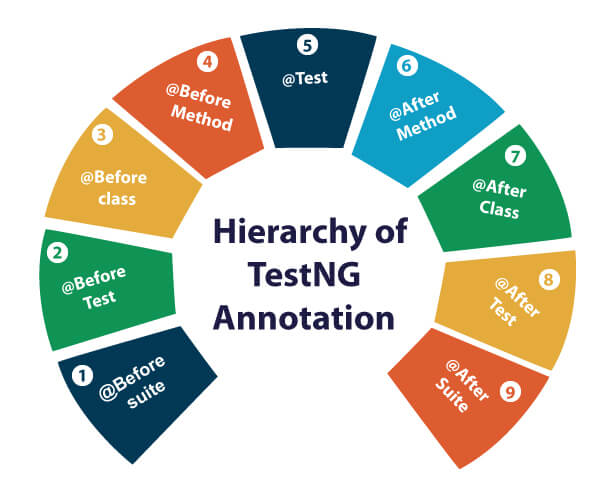
Polymorphism in Java | Dynamic Method Dispatch

## **List of TestNG Annotations**



|  |  |
| --- | --- |
| **TestNG Annotation** | **Description** |
| [@BeforeSuite](https://www.javatpoint.com/testng-beforesuite-annotation) | The @BeforeSuite annotated method will run before the execution of all the test methods in the suite. |
| [@AfterSuite](https://www.javatpoint.com/testng-aftersuite-annotation) | The @AfterSuite annotated method will run after the execution of all the test methods in the suite. |
| [@BeforeTest](https://www.javatpoint.com/testng-beforetest-annotation) | The @BeforeTest annotated method will be executed before the execution of all the test methods of available classes belonging to that folder. |
| [@AfterTest](https://www.javatpoint.com/testng-aftertest-annotation) | The @AfterTest annotated method will be executed after the execution of all the test methods of available classes belonging to that folder. |
| [@BeforeClass](https://www.javatpoint.com/testng-beforeclass-annotation) | The @BeforeClass annotated method will be executed before the first method of the current class is invoked. |
| [@AfterClass](https://www.javatpoint.com/testng-afterclass-annotation) | The @AfterClass annotated method will be invoked after the execution of all the test methods of the current class. |
| [@BeforeMethod](https://www.javatpoint.com/testng-beforemethod-annotation) | The @BeforeMethod annotated method will be executed before each test method will run. |
| [@AfterMethod](https://www.javatpoint.com/testng-aftermethod-annotation) | The @AfterMethod annotated method will run after the execution of each test method. |

## **Hierarchy of the TestNG Annotations:**



* @BeforeSuite
* @BeforeTest
* @BeforeClass
* @BeforeMethod
* @Test
* @AfterMethod
* @AfterClass
* @AfterTest
* @AfterSuite

## **Benefits of using TestNG Annotations:**

* TestNG Annotations made the life of testers very easy. Based on your requirements, you can access the test methods, i.e., it has no predefined pattern or format.
* You can pass the additional parameters to TestNG annotations.
* In the case of TestNG annotations, you do not need to extend any test classes.
* TestNG Annotations are strongly typed, i.e., errors are detected at the compile time.

# **TestNG Annotation Attributes**

While writing the test cases in the TestNG, you need to mention the @Test annotation before the test method.

@Test

**public** **void** testcase1()

{

   System.out.println("This is testcase1");

}

In the above code, we have specified the @Test annotation before the test method, i.e., testcase1().

We can also explicitly specify the attributes in a **@Test** annotation. Test attributes are the test specific, and they are specified at the right next to the **@Test** annotation.

@Test(attribute="value")

**public** **void** testcase2()

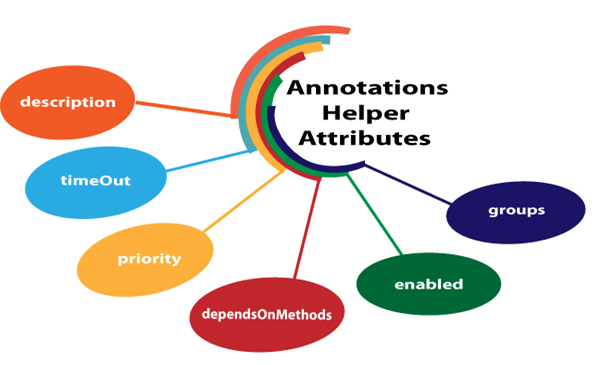
{

 System.out.println("This is testcase2");

}

# TestNG Annotation Attributes

**Some of the common attributes are described below:**



* **description**
* **timeOut**
* **priority**
* **dependsOnMethods**
* **enabled**
* **groups**

## **description**

It is a string which is attached to the @Test annotation that describes the information about the test.

**Let's understand through an example.**

**package** com.javatpoint;

**import** org.testng.annotations.Test;

**public** **class** Class1

{

@Test(description="This is testcase1")

**public** **void** testcase1()

{

System.out.println("HR");

}

@Test(description="This is testcase2")

**public** **void** testcase2()

{

System.out.println("Software Developer");

}

@Test(description="This is testcase3")

**public** **void** testcase3()

{

System.out.println("QA Analyst");

}

}

In the above code, we have added the description attribute in every test. The "**description**" attribute provides information about the test.

## **dependsOnMethods**

When the second test method wants to be dependent on the first test method, then this could be possible by the use of "**dependOnMethods**" attribute. If the first test method fails, then the dependent method on the first test method, i.e., the second test method will not run.

**Let's understand through an example.**

**First case:** When a single value is passed in a parameter.

**package** com.javatpoint;

**import** org.testng.annotations.Test;

**public** **class** Class1

{

@Test

**public** **void** WebStudentLogin()

{

System.out.println("Student login through web");

}

@Test

**public** **void** MobileStudentLogin()

{

System.out.println("Student login through mobile");

}

@Test(dependsOnMethods= {"WebStudentLogin"})

**public** **void** APIStudentLogin()

{

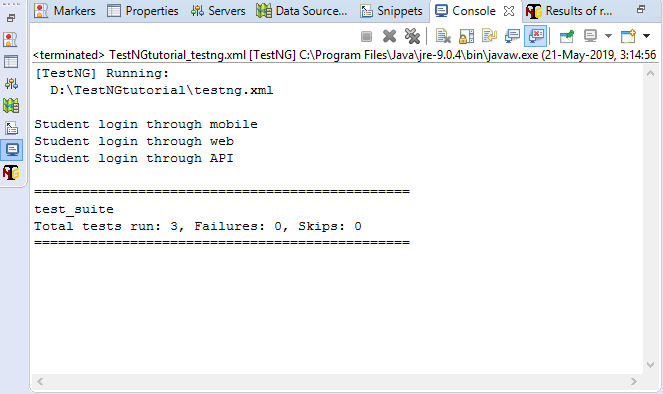
System.out.println("Student login through API");

}

}

We know that the TestNG executes the test methods in alphabetical order so, in the above program, APIStudentLogin() will execute first. However, we want WebStudentLogin() method to be executed before the execution of the APIStudentLogin() method, so this would only be possible through the "dependsOnMethods" attribute. In the above program, we have specified "dependsOnMethods" attribute in an APIStudentLogin() test method and its value is "WebStudentLogin" which means that WebStudentLogin() method will be executed before the APIStudentLogin() method.

**Output**



In the above output, MobileStudentLogin() runs before the WebStudentLogin() method as TestNG runs the test methods in an alphabetical order.

**Second case:** When multiple values are passed in a parameter.

**package** com.javatpoint;

**import** org.testng.annotations.Test;

**public** **class** Depends\_On\_Groups

{

 @Test(dependsOnMethods= {"testcase3","testcase2"})

**public** **void** testcase1()

 {

     System.out.println("This is test case1");

 }

 @Test

**public** **void** testcase2()

 {

     System.out.println("This is test case2");

 }

 @Test

**public** **void** testcase3()

 {

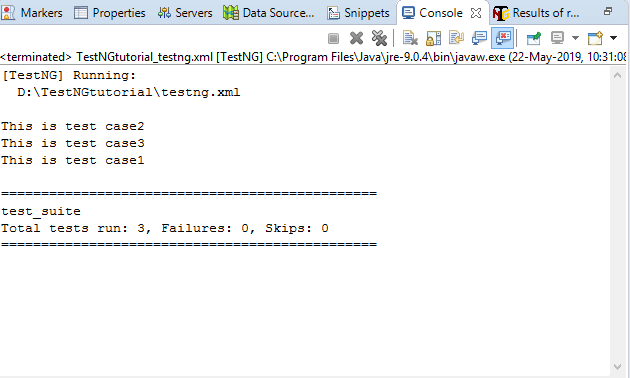
     System.out.println("This is test case3");

 }

}

In the above code, testcase1() is dependent on two methods, i.e., testcase2() and testcase3(), which means that these two methods will be executed before the testcase1().

**Output**



## **priority**

When no 'priority' attribute is specified then the TestNG will run the test cases in alphabetical order. Priority determines the sequence of the execution of the test cases. The priority can hold the integer values between -5000 and 5000. When the priority is set, the lowest priority test case will run first and the highest priority test case will be executed last. Suppose we have three test cases and their priority values are -5000, 0, 15, then the order of the execution will be 0,15,5000. If priority is not specified, then the default priority will be 0.

**Let's understand through an example.**

**package** com.javatpoint;

**import** org.testng.annotations.Test;

**public** **class** Fruits

{

@Test

**public** **void** mango()

{

System.out.println("I am Mango");

}

@Test(priority=2)

**public** **void** apple()

{

System.out.println("I am Apple");

}

@Test(priority=1)

**public** **void** watermelon()

{

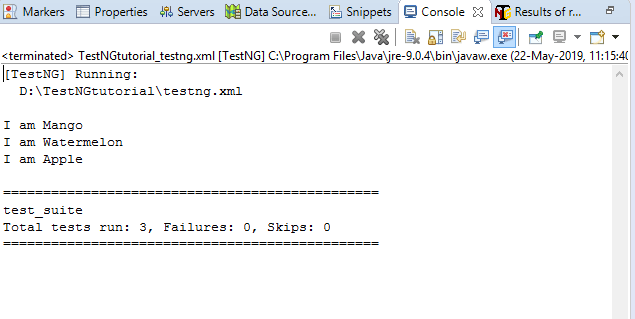
System.out.println("I am Watermelon");

}

}

In the above code, the default priority of mango() test method is 0, so it will be executed first. The watermelon() test method will run after mango() method as the priority of watermelon() test method is 2. The apple() test method has the highest priority, so it will be executed last.

**Output**



## **enabled**

The 'enabled' attribute contains the boolean value. By default, its value is true. If you want to skip some test method, then you need to explicitly specify '**false**' value.

**Let's understand through an example.**

**package** com.javatpoint;

**import** org.testng.annotations.Test;

**public** **class** Programming\_languages

{

@Test

**public** **void** c\_language()

{

System.out.println("C language");

}

@Test(enabled=**false**)

**public** **void** jira()

{

System.out.println("JIRA is a testing tool");

}

@Test

**public** **void** java()

{

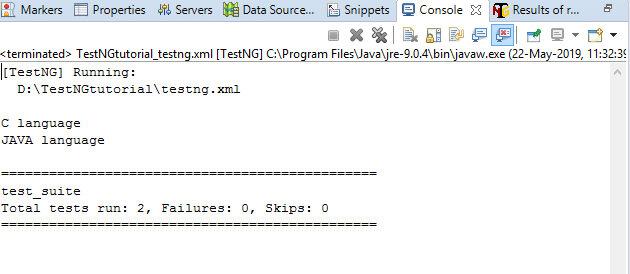
System.out.println("JAVA language");

}

}

In the above code, the value of the enabled attribute in jira() test method is false, so this method will not be invoked.

**Output**



## **groups**

The 'groups' attribute is used to group the different test cases that belong to the same functionality.

**Let's understand through an example.**

**package** com.javatpoint;

**import** org.testng.annotations.Test;

**public** **class** Software\_Company

{

@Test(groups= {"software company"})

**public** **void** infosys()

{

System.out.println("Infosys");

}

@Test

**public** **void** technip()

{

System.out.println("Technip India Ltd");

}

@Test(groups= {"software company"})

**public** **void** wipro()

{

System.out.println("Wipro");

}

}

**testng.xml**

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

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

<suite name="test\_suite">

<test name="Software Company">

<groups>

<run>

<include name="software company"/>

</run>

</groups>

<classes>

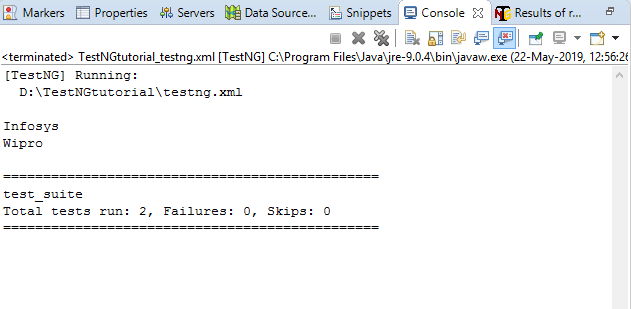
<**class** name="com.javatpoint.Software\_Company"/>

</classes>

</test> <!-- Test -->

</suite> <!-- Suite -->

**Output**



## **timeOut**

If one of the test cases is taking a long time due to which other test cases are failing. To overcome such situation, you need to mark the test case as fail to avoid the failure of other test cases. The timeOut is a time period provided to the test case to completely execute its test case.

**Let's understand through an example.**

**package** com.javatpoint;

**import** org.testng.annotations.Test;

**public** **class** Timeout\_program

{

@Test(timeOut=200)

**public** **void** testcase1() **throws** InterruptedException

{

Thread.sleep(500);

System.out.println("This is testcase1");

}

@Test

**public** **void** testcaes2()

{

System.out.println("This is testcase2");

    }

@Test

**public** **void** testcase3()

{

System.out.println("This is testcase3");

}

}

In the above code, inside the testcase1() method, we have Thread.sleep(500) which means that the testcase1() method will be executed after 500 milliseconds, but we have provided timeOUT attribute with the value 200 means that the testcase1() will be failed after 200 milliseconds.

**testng.xml**

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

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

**<suite name="test\_suite">**

**<test name="TimeOut Program">**

**<classes>**

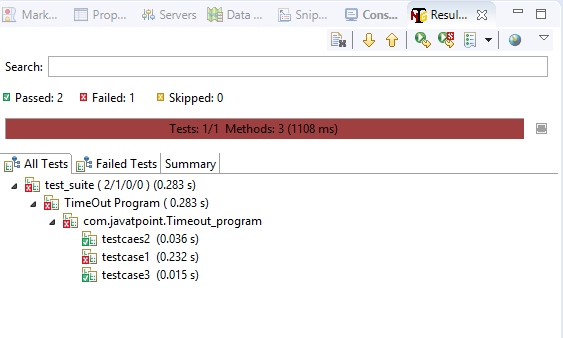
**<class name="com.javatpoint.Timeout\_program"/>**

**</classes>**

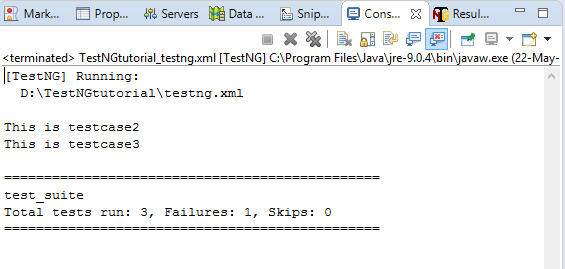
**</test> <!-- Test -->**

**</suite> <!-- Suite -->**

**Output**

****

**The above screen shows that one test case is failed and other test cases are passed.**

****