Practical No. 11 TestNg Annotations

Date: _____

Aim:

To study TestNg Annotations.

Theory:

Annotation is a feature introduced in Java 5 and is used to add metadata (data about data) to Java source code.

This will allow you to add information to an existing data object in your source code.

It can be applied for classes, methods, variables, and parameters.

Annotations may affect the way different programs or tools use your source code.

There are certain predefined set of annotations defined in Java.

For example, @Override, @Deprecated, @SupressWarnings, and so on, but Java allows users to define their own annotations too.

TestNg makes use of the same feature provided by Java to define its own annotations and build an execution framework by using it.

The following is a table containing information about all the annotations provided by TestNG and a brief description of them:

Annotation	**Description**
@BeforeSuite or @AfterSuite	The annotated method will be executed before and after any tests declared inside a TestNG suite.
@BeforeTest or @AfterTest	The annotated methods will be executed before and after each test section declared inside a TestNG suite.
@BeforeGroups or @AfterGroups	These annotations are associated with the groups feature in TestNG. BeforeGroups annotated method will run before any of the test method of the specified group is executed.

	AfterGroups annotated method will run after any of the test method of the specified group gets executed. For this method to be executed, the user has to mention the list of groups this method belongs to using groups attribute with the said annotation. You can specify more than multiple groups if required.
@BeforeClass or @AfterClass	BeforeClass annotated method is executed before any of the test method of a test class. AfterClass annotated method is executed after the execution of every test methods of a test class are executed.
@BeforeMethod or @AfterMethod	These annotated methods are executed before/after the execution of each test method.
@DataProvider	Marks a method as a data providing method for a test method. The said method has to return an Object double array (Object[][ ]) as data.
@Factory	Marks a annotated method as a factory that returns an array of class objects (Object[ ]). These class objects will then be used as test classes by TestNG. This is used to run a set of test cases with different values.
@Listeners	Applied on a test class. Defines an array of test listeners classes extending org.testng.ITestNGListener. Helps in tracking the execution status and logging purpose.
@Parameters	This annotation is used to pass parameters to a test method. These parameter values are provided using the testng.xml configuration file at runtime.
@Test	Marks a class or a method as a test method. If used at class level, all the public methods of a class will be considered as a test method

Test annotation

One of the basic annotations of TestNG is the Test annotation.

This annotation marks a method or a class as part of the TestNG test.

If applied at class level this annotation will mark all the public methods present inside the class as test methods for TestNG test.

It supports lot of attributes which you can use along with the annotation, which will enable you to use the different features provided by TestNG.

The following is a list of attributes supported by the Test annotation:

Supported attributes	**Description**
alwaysRun	Takes a true or false value. If set to true this method will always run even if its depending method fails.
dataProvider	The name of the data provider, which will provide data for data-driven testing to this method.
dataProviderClass	The class where TestNG should look for the dataprovider method mentioned in the dataProvider attribute. By default its the current class or its base classes.
dependsOnGroups	Specifies the list of groups this method depends on.
dependsOnMethods	Specifies the list of methods this method depends on.
description	The description of this method
enabled	Sets whether the said method or the methods inside the said class should be enabled for execution or not. By default, its value is true.
expectedExceptions	This attribute is used for exception testing. This attribute specifies the list of exceptions this method is expected to throw. In case a different exception is thrown.
groups	List of groups the said method or class belongs to.

timeOut	This attribute is used for a time out test and specifies the time (in millisecs) this method should take to execute.

Parameterization of test

One of the important features of TestNG is parameterization.

This feature allows user to pass parameter values to test methods as arguments. This is supported by using the Parameters and DataProvider annotations.

There are mainly two ways through which we can provide parameter values to test-methods:

Through testng XML configuration file

Through DataProviders

Parameterization through testng.xml

If you need to pass some simple values such as String types to the test methods at runtime, you can use this approach of sending parameter values through TestNG XML configuration files.

You have to use the Parameters annotation for passing parameter values to the test method.

DataProvider

One of the important features provided by TestNG is the DataProvider feature.

It helps the user to write data-driven tests, that means same test method can be run multiple times with different datasets.

DataProvider is the second way of passing parameters to test methods. It helps in providing complex parameters to the test methods as it is not possible to do this from XML.

To use the DataProvider feature in your tests you have to declare a method annotated by DataProvider and then use the said method in the test method using the dataProvider attribute in the Test annotation.

Groups

Grouping test methods is one of the most important features of TestNG.

In TestNG users can group multiple test methods into a named group.

You can also execute a particular set of test methods belonging to a group or multiple groups.

This feature allows the test methods to be segregated into different sections or modules.

For example, you can have a set of tests that belong to sanity test where as others may belong to regression tests.

You can also segregate the tests based on the functionalities/features that the test method verifies.

This helps in executing only a particular set of tests as and when required

Implementation

1. Create a test class with @BeforeClass/@AfterClass, @BeforeMethod/@AfterMethod annotations and execute it using a testng.xml

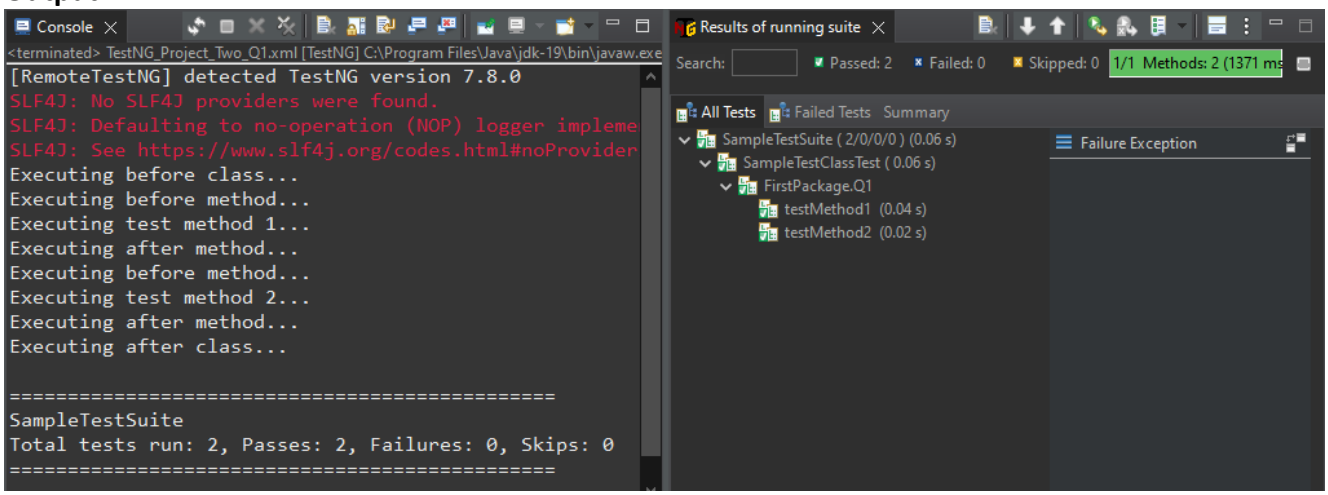
   Program:
   Q1.java

```
package FirstPackage;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
public class Q1 {
	@BeforeClass
	public void beforeClass() {
	System.out.println("Executing before class...");}
	@AfterClass
	public void afterClass() {
	System.out.println("Executing after class...");}
	@BeforeMethod
	public void beforeMethod() {
	System.out.println("Executing before method...");}
	@AfterMethod
	public void afterMethod() {
	System.out.println("Executing after method...");}
	@Test
	public void testMethod1() {
	System.out.println("Executing test method 1...");}
	@Test
	public void testMethod2() {
	System.out.println("Executing test method 2...");}    }
```

   testing.xml

```
<suite name="SampleTestSuite">
	<test name="SampleTestClassTest">
		<classes>
			<class name="FirstPackage.Q1"/>
		</classes>
	</test>
</suite>
```

Output:



2. Create and execute a TestNG class using test annotation on class. The class contains two public methods and one private method.

Program:

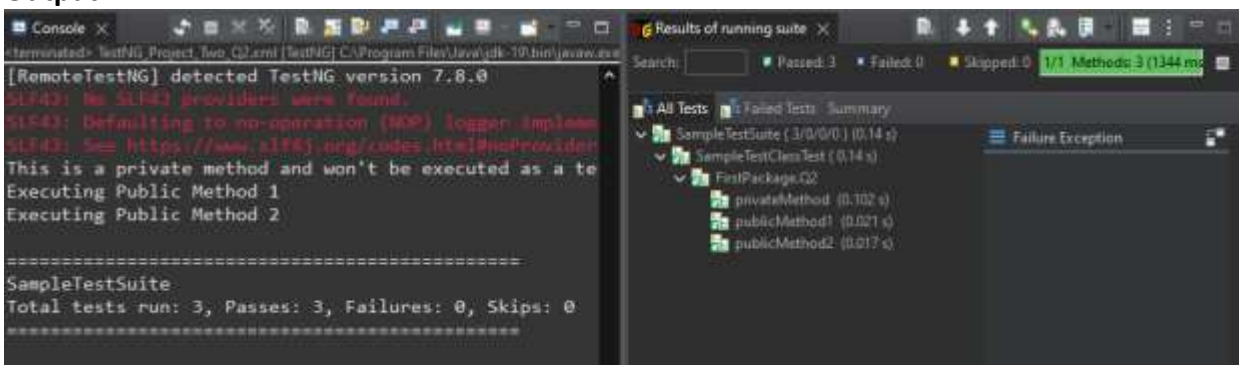
Q2.java

```
package FirstPackage;
import org.testng.annotations.Test;
public class Q2 {
	@Test
	public void publicMethod1() {
	System.out.println("Executing Public Method 1");
}
	@Test
	public void publicMethod2() {
	System.out.println("Executing Public Method 2");
}
	@Test
	private void privateMethod() {
	System.out.println("This is a private method and won't be executed as a test");
	}
}
```

testing.xml

```
<suite name="SampleTestSuite">
	<test name="SampleTestClassTest">
		<classes>
			<class name="FirstPackage.Q2"/>
		</classes>
	</test>
</suite>
```

Output:



3. Create TestNG class containing three test methods using test annotation out of which any two methods are enabled and remaining method is disabled. Use appropriate attributes of test annotation.

Program:

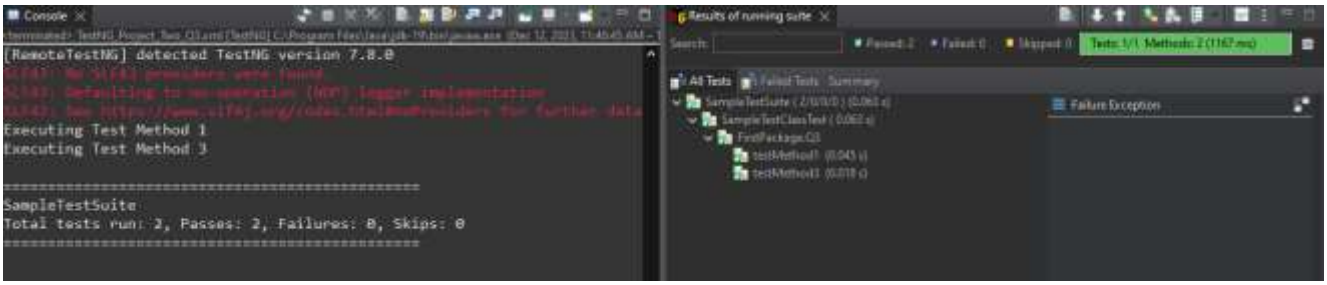
Q3.java

```
package FirstPackage;
import org.testng.annotations.Test;
public class Q3 {
	@Test(enabled = true)
	public void testMethod1() {
		System.out.println("Executing Test Method 1");
	}
	@Test(enabled = false)
	public void testMethod2() {
		System.out.println("Executing Test Method 2");
	}
	@Test(enabled = true)
	public void testMethod3() {
		System.out.println("Executing Test Method 3");
	}
}
```

tesing.xml

```
<suite name="SampleTestSuite">
	<test name="SampleTestClassTest">
		<classes>
			<class name="FirstPackage.Q3"/>
		</classes>
	</test>
</suite>
```

Output:



4. Create a test class with @BeforeSuite/@AfterSuite, @BeforeTest/@AfterTest annotations and execute it using a testng.xml.

Program:

Q4.java

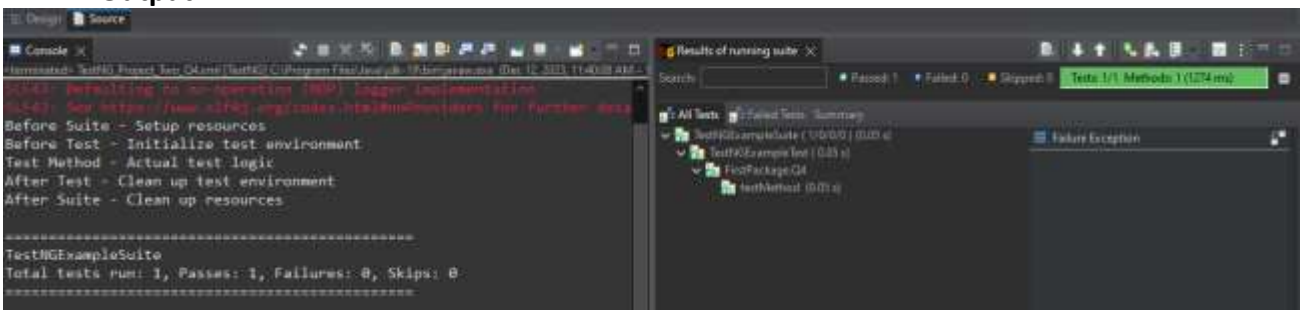
```
package FirstPackage;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
public class Q4 {
	@BeforeSuite
	public void beforeSuite() {
		System.out.println("Before Suite - Setup resources");}
	@AfterSuite
	public void afterSuite() {
		System.out.println("After Suite - Clean up resources");}
	@BeforeTest
	public void beforeTest() {
		System.out.println("Before Test - Initialize test environment");}
	@AfterTest
	public void afterTest() {
		System.out.println("After Test - Clean up test environment");}
	@Test
	public void testMethod() {
		System.out.println("Test Method - Actual test logic");	}	}
```

tesing.xml

```
<suite name="TestNGExampleSuite">
	<before-suite name="BeforeSuite" class-name="FirstPackage.Q4" method-name="beforeSuite" />
	<after-suite name="AfterSuite" class-name="FirstPackage.Q4" method-name="afterSuite" />
	<test name="TestNGExampleTest">
		<before-test name="BeforeTest" class-name="FirstPackage.Q4" method-
```

```
name="beforeTest" />
            <after-test name="AfterTest" class-name="FirstPackage.Q4" method-
name="afterTest" />
            <classes>
                <class name="FirstPackage.Q4"/>
            </classes>
        </test>
</suite>
```

Output:



5. Create a test class that contains four test methods. Two of which should belong to one group and the remaining two to another group. Create testing.xml file to execute tests in a particular group.

Program:

Q5.java

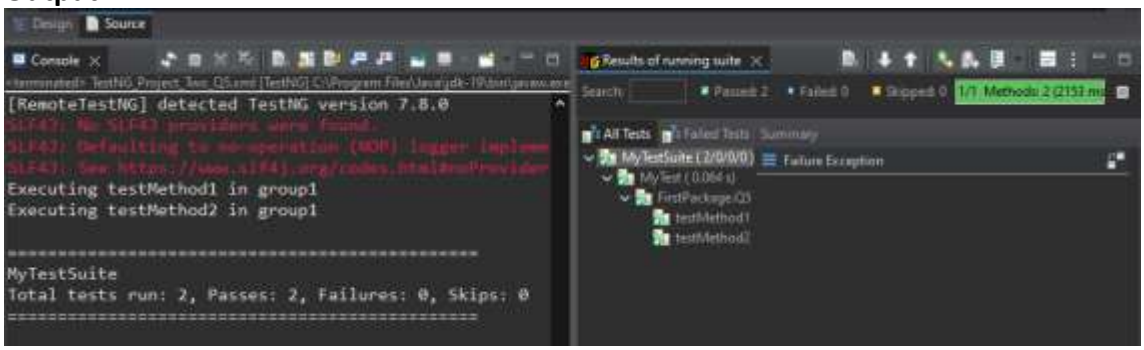
```
package FirstPackage;
import org.testng.annotations.Test;
public class Q5 {
        @Test(groups = "group1")
        public void testMethod1() {
        System.out.println("Executing testMethod1 in group1");}
        @Test(groups = "group1")
        public void testMethod2() {
        System.out.println("Executing testMethod2 in group1");}
        @Test(groups = "group2")
        public void testMethod3() {
        System.out.println("Executing testMethod3 in group2");}
        @Test(groups = "group2")
        public void testMethod4() {
        System.out.println("Executing testMethod4 in group2");}    }
```

testing.xml

```
<suite name="MyTestSuite">
        <test name="MyTest">
            <groups>
                <run>
                    <include name="group1"/>
                </run>
            </groups>
```

```
        <classes>
            <class name="FirstPackage.Q5"/>
        </classes>
    </test>
</suite>
```

Output:



6. Write a test class containing test method that calculates the average marks that awarded by two reviewers prints whether writer is shortlisted if average is >4. The marks are passed as parameters whose values are passed from testing.xml at test level.

Program:

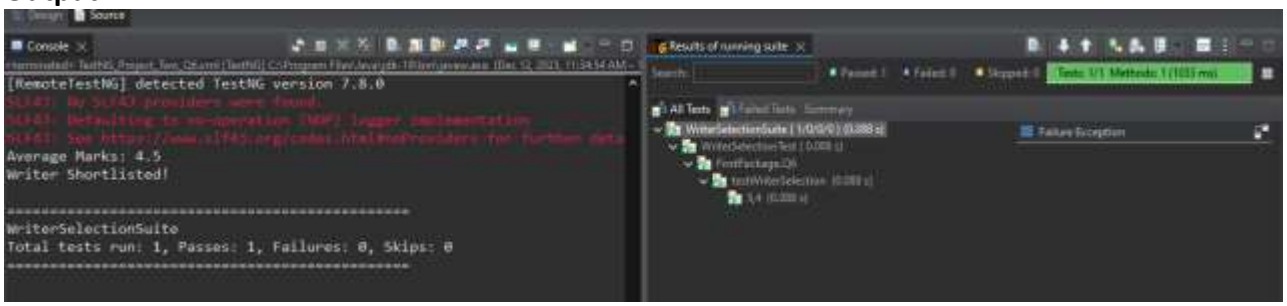
Q6.java

```
package FirstPackage;
import org.testng.annotations.Parameters;
import org.testng.annotations.Test;
public class Q6 {
    @Test
    @Parameters({"reviewer1Marks", "reviewer2Marks"})
    public void testWriterSelection(int reviewer1Marks, int reviewer2Marks)
    { // Calculate average marks
    double averageMarks = (reviewer1Marks + reviewer2Marks) / 2.0;
    // Print the average marks
    System.out.println("Average Marks: " + averageMarks);
    // Check if the writer is shortlisted
    if (averageMarks > 4) {
    System.out.println("Writer Shortlisted!");
    } else {
    System.out.println("Writer Not Shortlisted.");} } }
```

testing.xml

```
<suite name="WriterSelectionSuite">
    <test name="WriterSelectionTest">
        <parameter name="reviewer1Marks" value="5" />
        <parameter name="reviewer2Marks" value="4" />
        <classes>
            <class name="FirstPackage.Q6" />
        </classes>
    </test>
</suite>
```

Output:



7. Write a test class containing test method that prints the value of parameters which are passed from data provider.

Program:

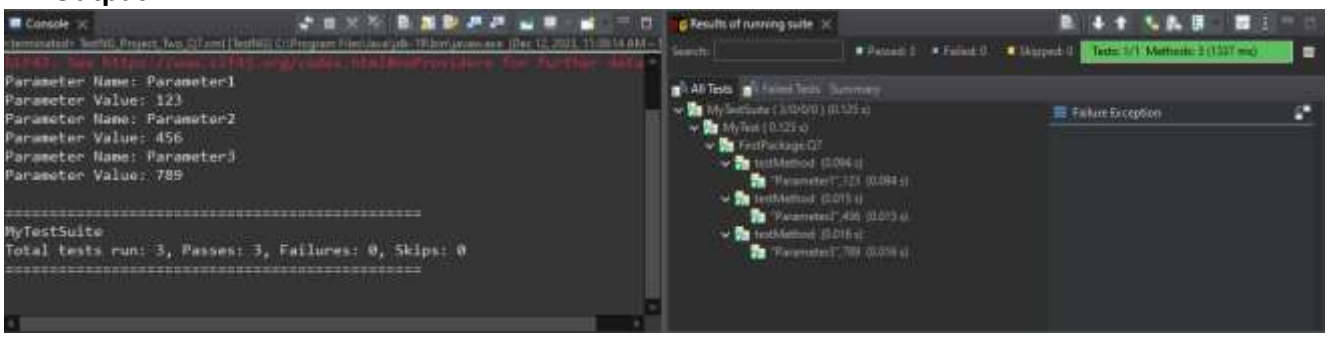
Q7.java

```
package FirstPackage;
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;
public class Q7 {
	@DataProvider(name = "testData")
	public Object[][] testData() {
	return new Object[][]{
	{"Parameter1", 123},
	{"Parameter2", 456},
	{"Parameter3", 789}
	};
}
	@Test(dataProvider = "testData")
	public void testMethod(String parameterName, int parameterValue)
	{ System.out.println("Parameter Name: " + parameterName);
	System.out.println("Parameter Value: " + parameterValue);
	}
}
```

testing.xml

```
<suite name="MyTestSuite">
	<test name="MyTest">
		<classes>
			<class name="FirstPackage.Q7" />
		</classes>
	</test>
</suite>
```

Output:



Conclusion: Understood how to use TestNG Annotations.