TestNG

Contents

[**1. Introduction to TestNG** 1](#_Toc185423425)

[**2. TestNG Setup and Installation** 1](#_Toc185423426)

[**3. TestNG Annotations** 1](#_Toc185423427)

[**4. TestNG Suite and XML Configuration** 2](#_Toc185423428)

[**5. TestNG Grouping** 3](#_Toc185423429)

[**6. TestNG Priorities** 3](#_Toc185423430)

[**7. Data-Driven Testing (Parameters & DataProvider)** 3](#_Toc185423431)

[**8. TestNG Assertions** 4](#_Toc185423432)

[**9. TestNG Listeners** 4](#_Toc185423433)

[**10. Parallel Test Execution** 5](#_Toc185423434)

[**11. TestNG Report Generation** 5](#_Toc185423435)

[**12. Advanced Features** 5](#_Toc185423436)

[**13. TestNG Configuration and Suite Management** 6](#_Toc185423437)

[**14. TestNG Assertions and Soft Assertions** 7](#_Toc185423438)

[**15. Dependency and Conditional Execution** 7](#_Toc185423439)

[**16. Custom Annotations** 8](#_Toc185423440)

[**17. TestNG Parallel Execution Strategies** 8](#_Toc185423441)

[**18. Listeners and Reporters** 9](#_Toc185423442)

[**19. TestNG in Continuous Integration** 10](#_Toc185423443)

[**20. TestNG in Real-Time Projects** 10](#_Toc185423444)

**1. Introduction to TestNG**

TestNG is a testing framework inspired by JUnit and NUnit but introduces some new functionality that makes it more powerful and flexible, particularly for integration testing. It is widely used for unit, functional, end-to-end, and integration testing in Java projects.

**2. TestNG Setup and Installation**

**Maven Dependency**: To get started with TestNG, add the dependency to your pom.xml (for Maven projects):

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.4.0</version> <!-- Check for the latest version -->

<scope>test</scope>

</dependency>

**Gradle Dependency**: Alternatively, if you are using Gradle, you can add:

testImplementation 'org.testng:testng:7.4.0'

**3. TestNG Annotations**

TestNG provides several annotations to define tests and configure the execution flow:

**@Test**: Marks a method as a test method.

@Test

public void testMethod() {

// Test logic

}

**@BeforeSuite**: Executed before any tests in the suite run.

@BeforeSuite

public void setupSuite() {

System.out.println("Setting up suite...");

}

**@AfterSuite**: Executed after all tests in the suite finish.

@AfterSuite

public void teardownSuite() {

System.out.println("Teardown suite...");

}

**@BeforeTest**: Executed before any tests in the <test> tag in the XML file run.

@BeforeTest

public void setupTest() {

System.out.println("Setting up test...");

}

**@AfterTest**: Executed after all tests in the <test> tag in the XML file complete.

@AfterTest

public void teardownTest() {

System.out.println("Teardown test...");

}

**@BeforeClass**: Executed before the first method in the class is run.

@BeforeClass

public void setupClass() {

System.out.println("Setting up class...");

}

**@AfterClass**: Executed after the last method in the class has run.

@AfterClass

public void teardownClass() {

System.out.println("Teardown class...");

}

**@BeforeMethod**: Executed before each test method.

@BeforeMethod

public void setupMethod() {

System.out.println("Setting up method...");

}

**@AfterMethod**: Executed after each test method.

@AfterMethod

public void teardownMethod() {

System.out.println("Teardown method...");

}

**4. TestNG Suite and XML Configuration**

TestNG allows you to configure tests, their execution order, and their groupings using an XML file (testng.xml).

Example:

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

<suite name="Test Suite">

<test name="Test 1">

<classes>

<class name="com.example.TestClass"/>

</classes>

</test>

</suite>

* **<suite>**: Defines a suite of tests.
* **<test>**: Defines a set of test cases.
* **<classes>**: Contains all test classes to be run.
* **<class>**: Specifies the test class to execute.

**5. TestNG Grouping**

You can group tests and run them selectively, making TestNG more flexible than many other testing frameworks.

Example:

@Test(groups = {"smoke"})

public void smokeTest() {

System.out.println("Running Smoke Test...");

}

@Test(groups = {"regression"})

public void regressionTest() {

System.out.println("Running Regression Test...");

}

In your XML file, you can specify groups to run:

<groups>

<run>

<include name="smoke"/>

</run>

</groups>

**6. TestNG Priorities**

TestNG allows you to define the priority of tests. The lower the priority value, the earlier the test will run.

@Test(priority = 1)

public void firstTest() {

System.out.println("First Test");

}

@Test(priority = 2)

public void secondTest() {

System.out.println("Second Test");

}

**7. Data-Driven Testing (Parameters & DataProvider)**

**Using Parameters in XML**: You can pass parameters to your tests via the testng.xml file.

<parameter name="username" value="admin"/>

<parameter name="password" value="password123"/>

Then, access them in the test:

@Test

@Parameters({"username", "password"})

public void loginTest(String username, String password) {

System.out.println("Logging in with " + username + " and " + password);

}

**Using DataProvider**: TestNG's @DataProvider allows for parameterization of tests in a more flexible way.

@DataProvider(name = "testData")

public Object[][] getData() {

return new Object[][] {

{"admin", "password123"},

{"user", "pass456"}

};

}

@Test(dataProvider = "testData")

public void loginTest(String username, String password) {

System.out.println("Logging in with " + username + " and " + password);

}

**8. TestNG Assertions**

TestNG supports a wide range of assertions:

* **assertEquals()**: Checks that two values are equal.
* **assertNotEquals()**: Checks that two values are not equal.
* **assertTrue()**: Verifies that the condition is true.
* **assertFalse()**: Verifies that the condition is false.
* **assertNull()**: Verifies that the object is null.
* **assertNotNull()**: Verifies that the object is not null.

Example:

@Test

public void testAssertions() {

Assert.assertEquals(1, 1);

Assert.assertTrue("Test".equals("Test"));

}

**9. TestNG Listeners**

TestNG allows you to create custom listeners to monitor the execution flow, such as before/after methods, test failures, and other events.

Example:

public class TestListener implements ITestListener {

public void onTestStart(ITestResult result) {

System.out.println("Test started: " + result.getName());

}

public void onTestSuccess(ITestResult result) {

System.out.println("Test passed: " + result.getName());

}

public void onTestFailure(ITestResult result) {

System.out.println("Test failed: " + result.getName());

}

}

To register listeners in the XML:

<listeners>

<listener class-name="com.example.TestListener"/>

</listeners>

**10. Parallel Test Execution**

TestNG supports parallel test execution, which allows you to run tests concurrently to reduce execution time.

In testng.xml, you can set the parallel execution mode:

<suite name="Parallel Test Suite" parallel="tests" thread-count="3">

<test name="Test 1">

<classes>

<class name="com.example.TestClass"/>

</classes>

</test>

<test name="Test 2">

<classes>

<class name="com.example.TestClass"/>

</classes>

</test>

</suite>

**11. TestNG Report Generation**

TestNG provides built-in reports (HTML and XML) after test execution. You can configure the report output path using the testng.xml.

Example configuration in testng.xml:

<suite name="Suite" verbose="1">

<listeners>

<listener class-name="org.testng.reporters.XMLReporter"/>

</listeners>

</suite>

**12. Advanced Features**

**Dependency Testing**: Tests can depend on the results of other tests using the dependsOnMethods attribute.

@Test(dependsOnMethods = {"testMethod2"})

public void testMethod1() {

// Test logic

}

**Timeouts**: You can specify the maximum time a test should take using the timeOut attribute.

@Test(timeOut = 1000)

public void testMethod() {

// Test logic

}

**13. TestNG Configuration and Suite Management**

**Suite and Test Configuration**

TestNG allows you to organize tests in a suite with multiple tests, groups, and methods. A suite can include tests grouped by feature or functionality.

**Example of Suite Configuration**:

<suite name="Regression Suite" parallel="tests" thread-count="2">

<test name="Test Set 1">

<classes>

<class name="com.example.ClassA"/>

<class name="com.example.ClassB"/>

</classes>

</test>

<test name="Test Set 2">

<classes>

<class name="com.example.ClassC"/>

</classes>

</test>

</suite>

* **parallel="tests"**: Executes tests in parallel.
* **thread-count="2"**: Specifies the number of threads.

**Data Providers and Parallel Execution**

You can use **Data Providers** with parallel execution to run data-driven tests in parallel.

@DataProvider(name = "userData", parallel = true)

public Object[][] getUserData() {

return new Object[][] {

{"admin", "password123"},

{"user", "pass456"}

};

}

@Test(dataProvider = "userData")

public void loginTest(String username, String password) {

System.out.println("Logging in with: " + username + " and " + password);

}

This will run both sets of data in parallel.

**14. TestNG Assertions and Soft Assertions**

While **hard assertions** (like Assert.assertTrue()) stop execution when they fail, **soft assertions** allow tests to continue even if an assertion fails, which is useful for reporting multiple failures.

**SoftAssertions:**

import org.testng.asserts.SoftAssert;

SoftAssert softAssert = new SoftAssert();

softAssert.assertTrue(condition, "Condition failed");

softAssert.assertEquals(actual, expected, "Values do not match");

softAssert.assertAll(); // Reports all soft assertion failures at the end of the test

Use softAssert.assertAll() to aggregate all assertion failures and report them after the test method has completed.

**15. Dependency and Conditional Execution**

**Test Dependencies**

You can control the execution order of tests using the dependsOnMethods or dependsOnGroups attributes.

* **Test Method Dependencies**:

@Test(dependsOnMethods = {"testMethod1"})

public void testMethod2() {

// Will execute only if testMethod1 passes

}

* **Group Dependencies**:

@Test(groups = {"smoke"})

public void testMethod1() { /\* Some test \*/ }

@Test(groups = {"smoke"}, dependsOnGroups = "smoke")

public void testMethod2() { /\* This will run after all "smoke" tests \*/ }

**Enabling/Disabling Tests Conditionally**

You can use the enabled attribute to conditionally disable tests.

@Test(enabled = false)

public void testThatIsDisabled() {

// This test will not run

}

**Running Tests Based on Conditions**

Using @Test(enabled = true) or checking the System.getProperty() within tests can control when tests should be executed. For example:

@Test

public void conditionalTest() {

if (System.getProperty("runTests").equals("yes")) {

// Test logic here

}

}

In the XML:

<parameter name="runTests" value="yes"/>

**16. Custom Annotations**

You can create custom annotations by combining **TestNG annotations** with **Reflection**. For example, you can create an annotation like @MyTestAnnotation that can be executed based on certain conditions or before/after every test method.

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.METHOD)

public @interface MyTestAnnotation {

String value() default "testValue";

}

public class CustomAnnotationTest {

@MyTestAnnotation(value = "Custom Test")

public void myTestMethod() {

System.out.println("Running Custom Annotated Test");

}

}

This allows you to create more complex behaviors or configurations that go beyond TestNG’s built-in annotations.

**17. TestNG Parallel Execution Strategies**

TestNG offers several ways to parallelize test execution:

**Parallel Methods Execution:**

You can specify to run individual test methods in parallel using the parallel attribute.

<suite name="Parallel Test Suite" parallel="methods" thread-count="5">

<test name="Test 1">

<classes>

<class name="com.example.TestClass"/>

</classes>

</test>

</suite>

**Parallel Class Execution:**

Similarly, test classes can run in parallel.

<suite name="Parallel Suite" parallel="classes" thread-count="2">

<test name="Test Set 1">

<classes>

<class name="com.example.TestClass1"/>

<class name="com.example.TestClass2"/>

</classes>

</test>

</suite>

**Parallel Test Execution Across Groups:**

This allows running groups in parallel, making it useful for executing tests belonging to different modules.

<suite name="Parallel Groups" parallel="groups" thread-count="2">

<test name="Test Group 1">

<groups>

<run>

<include name="smoke"/>

<include name="regression"/>

</run>

</groups>

</test>

</suite>

**18. Listeners and Reporters**

**Custom Reporters**

You can extend TestNG's reporting features by creating custom reporters. Reporters allow you to modify or create your own test output formats.

public class CustomReporter implements IReporter {

public void generateReport(List<XmlSuite> xmlSuites, List<ISuite> suites, String outputDirectory) {

// Custom reporting logic

}

}

Add the custom reporter in the testng.xml:

<listeners>

<listener class-name="com.example.CustomReporter"/>

</listeners>

**Built-in Listeners**

TestNG provides several built-in listeners to hook into the lifecycle of tests:

* ITestListener: Used to track the status of each test method (e.g., onTestStart(), onTestSuccess()).
* ISuiteListener: Used to track the suite lifecycle (e.g., onStart(), onFinish()).
* IInvokedMethodListener: Used to monitor the invocation of methods.

Example usage:

public class MyTestListener implements ITestListener {

public void onTestStart(ITestResult result) {

System.out.println("Test Started: " + result.getName());

}

public void onTestSuccess(ITestResult result) {

System.out.println("Test Passed: " + result.getName());

}

}

**19. TestNG in Continuous Integration**

TestNG works well with popular CI tools like **Jenkins**, **TeamCity**, and **Bamboo**. You can integrate TestNG with CI tools to run tests automatically after each build and display results in the build logs.

* **Jenkins Integration**:
  + Install the **TestNG Plugin** in Jenkins.
  + Configure the TestNG report path in Jenkins' build configuration (testng-results.xml).
  + Jenkins will show the test results after each build.

**20. TestNG in Real-Time Projects**

TestNG can be integrated into real-world projects in many ways, including:

* **API Testing**: Automating API tests using RestAssured combined with TestNG.
* **Web Automation**: Using **Selenium WebDriver** along with TestNG to automate web applications.
* **Performance Testing**: Combining TestNG with tools like **JMeter** or **Gatling** to test performance and load handling.

For example, with **Selenium**:

public class SeleniumTest {

WebDriver driver = new ChromeDriver();

@Test

public void openWebsite() {

driver.get("http://example.com");

Assert.assertTrue(driver.getTitle().contains("Example"));

}

}

These additional concepts should provide you with a more thorough understanding of TestNG and its power to handle various types of testing scenarios. Let me know if you need a deeper dive into any of these topics!