**1. Revisiting TestNG Annotations**

Before diving deeper, let’s briefly review key TestNG annotations from the previous session:

* @Test: Defines a test case.
* @BeforeMethod / @AfterMethod: Runs before/after each test method.
* @BeforeClass / @AfterClass: Runs once before/after all tests in the current class.

**2. TestNG Dependency Management**

In TestNG, you can manage the dependency between test methods using the dependsOnMethods or dependsOnGroups attribute in the @Test annotation. This is useful when you want certain tests to run only if some other test has passed.

**Example:**

java

import org.testng.annotations.Test;

public class DependencyTest {

@Test

public void loginTest() {

System.out.println("Login successful");

}

@Test(dependsOnMethods = "loginTest")

public void accountTest() {

System.out.println("Account access after login");

}

@Test(dependsOnMethods = "accountTest")

public void logoutTest() {

System.out.println("Logout successful");

}

}

**Output:**

* If loginTest fails, accountTest and logoutTest won’t execute.

**3. Parameterization in TestNG**

**Using @Parameters Annotation**

@Parameters allows you to pass data into your test methods via the TestNG XML file. This is particularly helpful for running the same test with different data.

**Example:**

java

import org.testng.annotations.Parameters;

import org.testng.annotations.Test;

public class ParameterTest {

@Test

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

public void loginTest(String username, String password) {

System.out.println("Login with Username: " + username + " and Password: " + password);

}

}

**XML Configuration:**

xml

<suite name="Suite">

<test name="Parameter Test">

<parameter name="username" value="user123"/>

<parameter name="password" value="pass123"/>

<classes>

<class name="ParameterTest"/>

</classes>

</test>

</suite>

This allows you to pass the parameters from the XML file during test execution.

**4. Data-Driven Testing with @DataProvider**

@DataProvider is a powerful feature that enables you to run the same test case multiple times with different sets of data.

**Example:**

java

import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;

public class DataProviderTest {

@DataProvider(name = "loginData")

public Object[][] dataProviderMethod() {

return new Object[][] { { "user1", "pass1" }, { "user2", "pass2" } };

}

@Test(dataProvider = "loginData")

public void loginTest(String username, String password) {

System.out.println("Testing login with Username: " + username + " and Password: " + password);

}

}

**Explanation:**

* The @DataProvider annotation provides multiple sets of data to the loginTest method, which is run for each set.

**5. Soft and Hard Assertions in TestNG**

Assertions are used to validate test results. TestNG provides both hard and soft assertions.

**Hard Assertions:**

* If a hard assertion fails, the test method is marked as failed, and the execution of the method is stopped.

**Example:**

java

import org.testng.Assert;

import org.testng.annotations.Test;

public class HardAssertionTest {

@Test

public void testHardAssertion() {

Assert.assertEquals("Actual", "Actual", "Assertion Failed: Values do not match");

System.out.println("This will be executed because assertion passed.");

}

}

**Soft Assertions:**

* Even if a soft assertion fails, the test continues to execute. After execution, you need to call assertAll() to collect the results of all soft assertions.

**Example:**

java

import org.testng.asserts.SoftAssert;

import org.testng.annotations.Test;

public class SoftAssertionTest {

@Test

public void testSoftAssertion() {

SoftAssert softAssert = new SoftAssert();

softAssert.assertEquals("Actual1", "Expected1", "Assertion 1 failed");

softAssert.assertEquals("Actual2", "Expected2", "Assertion 2 failed");

System.out.println("Soft assertions are complete.");

softAssert.assertAll(); // This will throw all assertion failures at once.

}

}

**6. Groups in TestNG**

TestNG allows you to group your tests so you can selectively execute them based on the group name.

**Example:**

java

import org.testng.annotations.Test;

public class GroupingTest {

@Test(groups = "login")

public void testLogin() {

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

}

@Test(groups = "account")

public void testAccount() {

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

}

@Test(groups = { "login", "account" })

public void testAccountAccess() {

System.out.println("Account Access Test");

}

}

**XML Configuration:**

xml

<suite name="Suite">

<test name="Test Groups">

<groups>

<run>

<include name="login"/>

</run>

</groups>

<classes>

<class name="GroupingTest"/>

</classes>

</test>

</suite>

This XML configuration will only run the test methods marked with the login group.

**7. Retry Failed Tests in TestNG**

TestNG has the ability to automatically retry failed tests using IAnnotationTransformer and IRetryAnalyzer.

**Example:**

1. Create a retry class that implements IRetryAnalyzer.

java

import org.testng.IRetryAnalyzer;

import org.testng.ITestResult;

public class RetryAnalyzer implements IRetryAnalyzer {

private int retryCount = 0;

private static final int maxRetryCount = 2;

@Override

public boolean retry(ITestResult result) {

if (retryCount < maxRetryCount) {

retryCount++;

return true;

}

return false;

}

}

1. Attach this retry logic to your test.

java

import org.testng.annotations.Test;

public class RetryTest {

@Test(retryAnalyzer = RetryAnalyzer.class)

public void testRetry() {

System.out.println("Executing retry logic");

assert false; // Fail this test intentionally

}

}

**8. TestNG Listeners**

TestNG listeners allow you to listen to various events in a test execution lifecycle, like test start, test success, test failure, etc. You can use listeners to customize behavior, log test status, or perform specific actions during the test run.

**Example:**

java

import org.testng.ITestListener;

import org.testng.ITestResult;

public class TestListener implements ITestListener {

@Override

public void onTestSuccess(ITestResult result) {

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

}

@Override

public void onTestFailure(ITestResult result) {

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

}

}

**9. TestNG Best Practices:**

* Use @DataProvider for large-scale data-driven testing.
* Utilize groups for better test organization.
* Always add assertion checks to validate expected outcomes.
* Implement retry logic for flaky tests.

**10. Questions and Discussion:**

* When would you prefer soft assertions over hard assertions?
* How can @DataProvider help in testing a login page with multiple credentials?

**11. Conclusion:**

Advanced TestNG features like @Parameters, @DataProvider, and assertions greatly improve test flexibility and scalability. By leveraging these, you can design more comprehensive and maintainable test suites.

**Homework:**

1. Implement a login test using both @Parameters and @DataProvider.
2. Explore TestNG listeners by writing custom listener logic for passed and failed tests.