+91 8318815041
chaitanyaa292@gmail.com

TestNG Concepts

Que 1: Introduction to TestNG

Ans 1: TestNG is a testing framework inspired by JUnit and NUnit. It is designed to make testing easier and more powerful by offering advanced features like annotations, grouping, parameterization, and parallel execution.

Key Features:

- Annotation-driven flexible configuration
- Support for data-driven testing
- Ability to run tests in parallel
- Detailed reporting"

Que 2: Assertions in TestNG

Ans 2: Assertions in TestNG are used to validate expected results in your test cases. They help ensure that the actual output matches the expected output.

Common Assertion Methods:

- **assertEquals(actual, expected)**
  - Validates that the actual result is equal to the expected result.
  - `Assert.assertEquals(actualTitle, expectedTitle, "Title does not match");`
- **assertTrue(condition)**
  - Checks if a condition is true.
  - `Assert.assertTrue(isElementVisible, "Element is not visible");`
- **assertNotNull(object)**
  - Ensures that the object is not null.
  - `Assert.assertNotNull(userDetails, "User details should not be null");`

  `, etc.`

+91 8318815041
chaitanyaa292@gmail.com

Que 3: Types of Assertions

Ans 3: Hard Assertions: Stop test execution on failure of Assertion.
Soft Assertions: Continue test execution on failure of Assertion.

- **Hard Assertions**: Report the failure immediately and stop further test execution.
- `Assert.assertEquals()`, `Assert.assertTrue()`, `Assert.assertFalse()`, etc.

- **Soft Assertions**: Collect all failures and report them together at the end of the test.
- `SoftAssert.assertEquals()`, `SoftAssert.assertTrue()`, `SoftAssert.assertFalse()`, etc.

Hard Assertions Example:

```
import org.testng.Assert;
import org.testng.annotations.Test;
public class HardAssertionTest {
    @Test
    public void testHardAssertion() {
        Assert.assertEquals(1, 2, "Values are not equal");
        // This will fail
        System.out.println("This line will not be executed if the
above assertion fails.");
    }
}
```

Soft Assertions Example:

```
import org.testng.asserts.SoftAssert;
import org.testng.annotations.Test;
public class SoftAssertionTest {
    @Test
    public void testSoftAssertion() {
        SoftAssert softAssert = new SoftAssert();
        softAssert.assertEquals(1, 2, "Values are not equal");
        // This will fail but the test will continue
        System.out.println("This line will be executed even if the
above assertion fails.");
```

```
        softAssert.assertAll();
        // This will report all the soft assertion failures
    }
}
```

Que 4: Core TestNG Annotations

Ans 4: Commonly Used Annotations:

Annotation	Description
@Test	Marks a method as a test case.
@BeforeSuite	Executes before all tests in the suite.
@AfterSuite	Executes after all tests in the suite.
@BeforeTest	Executes before each test in the `<test>` tag.
@AfterTest	Executes after each test in the `<test>` tag.
@BeforeClass	Executes before the first method in the class.
@AfterClass	Executes after the last method in the class.
@BeforeMethod	Executes before each `@Test` method.
@AfterMethod	Executes after each `@Test` method.

Example:

```
import org.testng.annotations.*;
public class AnnotationExample {
    @BeforeSuite
    public void beforeSuite() {
        System.out.println("Before Suite");
    }

    @BeforeMethod
    public void beforeMethod() {
```

```
        System.out.println("Before Method");
    }

    @Test
    public void testMethod() {
        System.out.println("Test Method");
    }

    @AfterMethod
    public void afterMethod() {
        System.out.println("After Method");
    }

    @AfterSuite
    public void afterSuite() {
        System.out.println("After Suite");
    }
}
```

Que 5: testng.xml file

Ans 5: `Testng.xml file` defines how TestNG should execute tests in your project.

To create an automated `testng.xml`:

1. After adding a few test methods -> Right click on the project -> TestNG -> Convert to TestNG -> Select the location `'src/test/resources'`.

Use of `testng.xml` file:

- **Organize tests**: Control which tests to run and in what order.
- **Grouping**: Run groups of tests together.
- **Parallel execution**: Execute tests in multiple threads.
- **Parameterization**: Pass parameters to tests.
- **Suites**: Run multiple test classes together.

Sample of `testng.xml` file:

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd">
<suite name="MyTestSuite" parallel="false">
    <!-- Define a Test -->
    <test name="LoginTests">
        <classes>
            <class name="tests.LoginTests">
                <methods>
                    <include name="testLogin" />
                    <exclude name="testForgotPassword" />
                </methods>
            </class>
        </classes>
    </test>
</suite>
```

Que 6: Grouping Tests in TestNG

Ans 6: You can group tests using the groups attribute in the @Test annotation.

Example:

```
public class GroupingExample {

    @Test(groups = {"smoke"})

    public void smokeTest() {

        System.out.println("This is a smoke test.");

    }

    @Test(groups = {"regression"})

    public void regressionTest() {

        System.out.println("This is a regression test.");

    }

    @Test(groups = {"smoke"})

    public void smokeTest2() {
```

```
        System.out.println("This is a smoke test 2.");
    }
}
```

Run specific groups by configuring the `testng.xml` file:

```
<suite name="Suite">
    <test name="Test">
        <groups>
            <run>
                <include name="smoke"/>
            </run>
        </groups>
        <classes>
            <class name="GroupingExample"/>
        </classes>
    </test>
</suite>
```

Output (printed):

```
This is a smoke test
This is a smoke test2
```

+91 8318815041
chaitanyaa292@gmail.com

Que 7: Data-Driven Testing with @DataProvider

Ans 7: The @DataProvider annotation is used to supply test data.

Example:

```
import org.testng.annotations.*;

public class DataProviderExample {

    @DataProvider(name = "testData")

    public Object[][] dataProvider() {

        return new Object[][] {

            {"John", 25},

            {"Jane", 30}

        };

    }

    @Test(dataProvider = "testData")

    public void testWithData(String name, int age) {

        System.out.println("Name: " + name + ", Age: " + age);

    }

}
```

If the number of arguments mismatches, it fails the test as shown in the given code.

Example of failure due to argument mismatch:

```
import org.testng.annotations.DataProvider;

import org.testng.annotations.Test;


public class DataPro3 {
```

```
    @Test(dataProvider="ca")
    public void tst(String fname, String lname) {
        System.out.println(fname + " " + lname);
    }

    @DataProvider(name="ca")
    public Object[][] dp() {
        return new Object[][] {
            {"David", "Warner", "Sports"},  // will fail
            {"Tom", "Cruise"}  // will pass
        };
    }
}
```

Que 8: Parallel Execution

Ans 8: Parallel execution allows multiple test methods or classes to run simultaneously.

- **Parallel execution can be configured in the testng.xml file using the parallel attribute and the thread-count attribute.**

Example:

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="Parallel Execution Suite" parallel="methods" thread-count="2">
    <test name="Test 1">
        <classes>
            <class name="com.example.TestClass1"/>
```

```
            <class name="com.example.TestClass2"/>
        </classes>
    </test>
</suite>
```

- **parallel="methods" specifies that methods within the classes will be executed in parallel.**
- **thread-count="2" sets the number of threads to use for executing tests concurrently.**

Execution Order:

- **Thread 1: A1() (from TestClass1)**
- **Thread 2: A2() (from TestClass2)**
- **Thread 1: After A1() completes, it can execute B1()**
- **Thread 2: After A2() completes, it can execute B2()**
- **Thread 1: After B1() completes, it can execute C1()**
- **Thread 2: After B2() completes, it can execute C2()**

Que 9: Dependency Tests

Ans 9: Use the dependsOnMethods attribute to define dependencies.

Example:

```
import org.testng.annotations.*;

public class DependencyExample {

    @Test

    public void loginTest() {

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

    }
```

```
    @Test(dependsOnMethods = {"loginTest"})
    public void dashboardTest() {
        System.out.println("Dashboard Test");
    }
}
```

Que 10: @BeforeSuite vs @BeforeTest

Ans 10:
@BeforeSuite

- Establish a connection to the database that all tests will use.
- To generate values with specific details using Random Generator that will be used in the framework.
- Start any necessary services (like web services) that will be required for tests.
- Initialize a test environment (like a test server) that all tests will run against.
- Set up environment variables specific to your test environment (like API keys).

@BeforeTest

- Web Driver Initialization:
- Open Browser
- Apply Implicit waits at global level
- Maximize the window
- Configuring API Endpoints

Que 11: @AfterSuite vs @AfterTest

Ans 11:
@AfterSuite

- Close the database connection used during all tests in the suite.
- Create a summary report after all tests have run.
- Send an email notification after all tests are complete.
- Delete any temporary files created during the tests.

@AfterTest

- Ending WebDriver Session
- Log results for tests executed in the specific group.
- Logout the application session (During automated tests for a web application, user sessions are created when users log in to the application. After the tests, these sessions should be cleared to prevent any data leakage or unintended access in subsequent tests.)

Que 12: How can you implement a retry mechanism for failed test cases in TestNG?

Ans 12: TestNG allows you to define a custom class that implements the IRetryAnalyzer interface. In this class, you can specify the logic to determine whether a test should be retried and how many times.

Step 1: Create a class that implements the IRetryAnalyzer interface.

```
import org.testng.IRetryAnalyzer;

import org.testng.ITestResult;

public class RetryAnalyzer implements IRetryAnalyzer {

    private int retryCount = 0; // Counter for retries

    private static final int maxRetryCount = 2;

    @Override

    public boolean retry(ITestResult result) {

        if (retryCount < maxRetryCount) {

            retryCount++;

            return true; // Retry the test

        }

        return false; // Do not retry

    }

}
```

Step 2: Use the Retry Analyzer in your test class.

```
import org.testng.Assert;

import org.testng.annotations.Test;

public class RetryTest {

    @Test(retryAnalyzer = RetryAnalyzer.class)

    public void testMethod() {

        System.out.println("Executing test method");

        Assert.fail("Failing the test intentionally");

        // This will fail and trigger a retry

    }
```

Que 13: How can exceptions be handled in TestNG?

Ans 13: TestNG provides the `expectedExceptions` attribute in the `@Test` annotation, which allows you to specify one or more exceptions that a test method is expected to throw. If the specified exception is thrown during the test execution, the test will be marked as passed; if no exception is thrown, or if a different exception is thrown, the test will fail.

Example:

```
import org.testng.Assert;

import org.testng.annotations.Test;

public class ExceptionHandlingExample {

    // Test method that is expected to throw an ArithmeticException

    @Test(expectedExceptions = ArithmeticException.class)

    public void testDivisionByZero() {

        System.out.println("Testing division by zero");
```

```
        int result = 1 / 0; // This will throw ArithmeticException
    }
    // Test method that is expected to throw a NullPointerException
    @Test(expectedExceptions = NullPointerException.class)
    public void testNullPointer() {
        System.out.println("Testing null pointer exception");
        String str = null;
        str.length(); // This will throw NullPointerException
    }
}
```

Que 14: What is the purpose of the timeout feature in TestNG?

Ans 14: The timeout feature in TestNG allows you to set a maximum execution time for a test method. If the method does not complete within this time frame, TestNG will automatically fail the test, helping to catch performance issues or infinite loops.

Example:

```
@Test(timeOut = 2000)
public void testWithTimeout() throws InterruptedException {
    // Test will fail if sleep > 2000
    Thread.sleep(1000);
    System.out.println("Test completed within the timeout");
}
```

+91 8318815041
chaitanyaa292@gmail.com

Que 15: How can I skip a test in TestNG?

Ans 15: To skip a test in TestNG, you can throw a SkipException within the test method. This will mark the test as skipped when the test suite is executed.

Example:

```
import org.testng.SkipException;

import org.testng.annotations.Test;

public class SkipTestExample {

    @Test

    public void testToBeSkipped() {

        throw new SkipException("Skipping this test intentionally");

    }

    @Test

    public void testToBeExecuted() {

        System.out.println("This test will be executed");

    }

}
```

In the testToBeSkipped() method, we throw a SkipException with a message indicating that the test is being skipped.