TestNG Tutorial: A Complete Guide for Beginners to Advanced

Table of Contents

1. Introduction to TestNG

What is TestNG?

TestNG (Test Next Generation) is a testing framework inspired by JUnit and NUnit, introducing innovative functionalities for making testing more powerful and easier.

Key Features

- Annotations
- Test dependencies
- Parallel execution
- Data-driven testing
- Flexible test configuration

2. Core Concepts

Basic Test Structure

import org.testng.annotations.Test;
import org.testng.Assert;

```
public class BasicTest {
    @Test
    public void myFirstTest() {
        String message = "Hello TestNG";
        Assert.assertEquals(message, "Hello TestNG");
    }
}
```

Assertions

TestNG provides various assertion methods:

```
public class AssertionExample {
    @Test
    public void demonstrateAssertions() {
        // Basic assertions
        Assert.assertEquals(5 + 5, 10);
        Assert.assertTrue(10 > 5);
        Assert.assertFalse(5 > 10);

        // Object assertions
        String[] expectedArray = {"one", "two"};
        String[] actualArray = {"one", "two"};
        Assert.assertEquals(actualArray, expectedArray);

        // Null checks
        Object obj = null;
        Assert.assertNull(obj);
    }
}
```

3. Annotations

Common Annotations

```
public class AnnotationsDemo {
    @BeforeSuite
    public void beforeSuite() {
        // Executes before test suite
    }

    @BeforeTest
    public void beforeTest() {
        // Executes before test
    }

    @BeforeClass
    public void beforeClass() {
        // Executes before class
    }

    @BeforeMethod
    public void beforeMethod() {
        // Executes before each test method
    }

    @Test
    public void testMethod() {
        // Test case
```

```
    }

    @AfterMethod
    public void afterMethod() {
        // Executes after each test method
    }

    @AfterClass
    public void afterClass() {
        // Executes after class
    }

    @AfterTest
    public void afterTest() {
        // Executes after test
    }

    @AfterSuite
    public void afterSuite() {
        // Executes after test suite
    }
}
```

4. Test Configuration

testng.xml

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite name="TestSuite">
    <test name="FirstTest">
        <classes>
            <class name="com.example.TestClass1"/>
            <class name="com.example.TestClass2"/>
        </classes>
    </test>
</suite>
```

5. Test Parameters

Data Provider Example

```
public class ParameterizedTests {
    @DataProvider(name = "loginData")
    public Object[][] createData() {
        return new Object[][] {
            {"user1", "pass1"},
            {"user2", "pass2"},
            {"user3", "pass3"}
        };
    }
```

```
    @Test(dataProvider = "loginData")
    public void testLogin(String username, String password) {
        System.out.println("Testing login with: " + username + ", " +
password);
        // Test login logic here
    }
}
```

XML Parameters

```
<suite name="Suite">
    <parameter name="browser" value="chrome"/>
    <test name="ParameterTest">
        <classes>
            <class name="ParameterizedTests"/>
        </classes>
    </test>
</suite>
```

```
public class XMLParameterTest {
    @Parameters({"browser"})
    @Test
    public void testBrowser(String browser) {
        System.out.println("Testing with browser: " + browser);
    }
}
```

6. Groups and Dependencies

Test Groups

```
public class GroupsExample {
    @Test(groups = "smoke")
    public void smokeTest1() {
        // Smoke test
    }

    @Test(groups = {"smoke", "regression"})
    public void smokeAndRegressionTest() {
        // Both smoke and regression
    }

    @Test(groups = "regression")
    public void regressionTest() {
        // Regression test
    }
```

```
}
Dependencies

public class DependencyExample {
    @Test
    public void createUser() {
        // Create user logic
    }

    @Test(dependsOnMethods = {"createUser"})
    public void updateUser() {
        // Update user logic
    }

    @Test(dependsOnMethods = {"updateUser"})
    public void deleteUser() {
        // Delete user logic
    }
}
```

7. Parallel Testing

Parallel Execution Configuration

```
<suite name="ParallelSuite" parallel="methods" thread-count="3">
    <test name="ParallelTest">
        <classes>
            <class name="ParallelTestClass"/>
        </classes>
    </test>
</suite>

public class ParallelTestClass {
    @Test
    public void test1() {
        System.out.println("Test 1 running in thread: " +
Thread.currentThread().getId());
    }

    @Test
    public void test2() {
        System.out.println("Test 2 running in thread: " +
Thread.currentThread().getId());
    }
}
```

8. Test Reports

Listeners

```
public class CustomListener implements ITestListener {
    @Override
    public void onTestStart(ITestResult result) {
        System.out.println("Test started: " + result.getName());
    }

    @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. Real-world Example: Login Test Suite

```
public class LoginTest {
    private WebDriver driver;

    @BeforeClass
    public void setUp() {
        driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    }

    @DataProvider(name = "loginData")
    public Object[][] loginData() {
        return new Object[][] {
            {"validUser", "validPass", true},
            {"invalidUser", "invalidPass", false}
        };
    }
```

```
@Test(dataProvider = "loginData")
    public void testLogin(String username, String password, boolean
expectedResult) {
        driver.get("http://example.com/login");
        driver.findElement(By.id("username")).sendKeys(username);
        driver.findElement(By.id("password")).sendKeys(password);
        driver.findElement(By.id("loginButton")).click();

        boolean actualResult =
```

```
driver.findElement(By.id("welcomeMessage")).isDisplayed();
        Assert.assertEquals(actualResult, expectedResult);
    }

    @AfterClass
    public void tearDown() {
        driver.quit();
    }
}
```

Practice Exercises

1. Basic Test:

```
// Create a simple calculator test
public class CalculatorTest {
    private Calculator calculator;

    @BeforeMethod
    public void setUp() {
        calculator = new Calculator();
    }

    @Test
    public void testAddition() {
        Assert.assertEquals(calculator.add(2, 2), 4);
    }
}
```

Parameterized Test:

```
// Create a string manipulation test
public class StringTest {
    @DataProvider(name = "stringData")
    public Object[][] stringData() {
        return new Object[][] {
            {"hello", "HELLO"},
            {"world", "WORLD"}
        };
    }

    @Test(dataProvider = "stringData")
    public void testUpperCase(String input, String expected) {
        Assert.assertEquals(input.toUpperCase(), expected);
    }
}
```

These examples provide a foundation for understanding TestNG. Practice by implementing these examples and modifying them to suit your needs.