Comprehensive Introduction to Testing with TestNG

Objective: Equip students with a foundational understanding of software testing using the TestNG framework. By the end of this lab, students should be able to set up TestNG, write and group tests, and use advanced features like data providers.

Prerequisites:

- Familiarity with Java programming concepts.
- A working Java Development Kit (JDK) installed on the computer.
- An Integrated Development Environment (IDE) – preferably IntelliJ IDEA or Eclipse.

PART A: Setting Up and Basics of TestNG

A1: Setting Up TestNG

1. Launch the IDE and create a new Java project named TestNGLab.
2. Install the TestNG plugin for your IDE.
3. Add the TestNG library to your project.

A2: Your First TestNG Test

- Construct a simple Calculator class within your project. This class should contain basic arithmetic methods – add(), subtract(), multiply(), and divide().
- Develop a CalculatorTest class. Within this class:
- Instantiate the Calculator class.
- Use the @Test annotation before methods to indicate they are test methods.
- Create methods testAdd(), testSubtract(), testMultiply(), and testDivide(). Use the Assert class from TestNG to verify outcomes.

1. TASK: Execute the CalculatorTest and note down observations.

PART B: Diving Deeper into TestNG Features

B1: Grouping Tests

- Using the groups attribute of the @Test annotation, categorize tests into two groups: BasicOperations (add, subtract) and AdvancedOperations (multiply, divide).

1. TASK: Run only one group at a time. Document any differences observed.

B2: Priority and Dependencies in Tests

- Assign priorities to tests using the priority attribute of the @Test annotation. For instance, make testAdd() run before testSubtract().

- Set up dependencies between tests using the dependsOnMethods attribute.

1. TASK: Intentionally fail a test that another test depends on. Observe and record the outcome.

B3: Using Data Providers

- Introduce the @DataProvider annotation. Explain its purpose: to allow a test method to be executed multiple times with different sets of data.
- Modify the testAdd() method to use a data provider that supplies several different sets of numbers to add.

1. TASK: Extend the use of the @DataProvider annotation for other arithmetic operations. Share results.

PART C: Best Practices and Clean Up

C1: TestNG XML Suite

- A testng.xml file should be available with configurations for running both BasicOperations and AdvancedOperations groups.

1. Answers to TASK: The XML suite should allow execution of both groups of tests in a controlled manner. The output should reflect this controlled execution.

C2: Before and After Annotations

- Logging (simple print statements can suffice for this lab) should appear before and after each test method or class, based on the usage of annotations like @BeforeClass, @AfterClass, @BeforeMethod, and @AfterMethod.

1. Answers to TASK: When running tests, students should observe logs appearing in sequence, showcasing the order of execution and demonstrating the use of the aforementioned annotations.

Answers

PART A: Setting Up and Basics of TestNG

A2: Your First TestNG Test

```
// Calculator.java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero!");
        }
        return a / b;
    }
}

// CalculatorTest.java
import org.testng.Assert;
import org.testng.annotations.Test;

public class CalculatorTest {
    Calculator calculator = new Calculator();

    @Test
    public void testAdd() {
        Assert.assertEquals(calculator.add(3, 2), 5);
    }

    @Test
    public void testSubtract() {
        Assert.assertEquals(calculator.subtract(3, 2), 1);
    }

    @Test
    public void testMultiply() {
        Assert.assertEquals(calculator.multiply(3, 2), 6);
    }
```

```
    @Test
    public void testDivide() {
        Assert.assertEquals(calculator.divide(6, 2), 3);
    }
}
```

PART B: Diving Deeper into TestNG Features

B1: Grouping Tests

```
@Test(groups = "BasicOperations")
public void testAdd() {...}

@Test(groups = "BasicOperations")
public void testSubtract() {...}

@Test(groups = "AdvancedOperations")
public void testMultiply() {...}

@Test(groups = "AdvancedOperations")
public void testDivide() {...}
```

B2: Priority and Dependencies in Tests

```
@Test(priority = 1)
public void testAdd() {...}

@Test(priority = 2, dependsOnMethods = {"testAdd"})
public void testMultiply() {...}
```

B3: Using Data Providers

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

@DataProvider(name = "dataForAddition")
public Object[][] dataForAdd() {
    return new Object[][] {
        { 1, 2, 3 },
        { 2, 3, 5 },
        { 3, 3, 6 }
    };
}
@Test(dataProvider = "dataForAddition")
public void testAdd(int a, int b, int expectedResult) {
    Assert.assertEquals(calculator.add(a, b), expectedResult);
}
```

PART C: Best Practices and Clean Up

C2: Before and After Annotations

```
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.AfterMethod;

@BeforeMethod
public void logBeforeMethod() {
    System.out.println("About to execute a test method...");
}

@AfterMethod
public void logAfterMethod() {
    System.out.println("Finished executing a test method...");
}
```

testng.xml (this is just a simple example; it can be much more complex):

```
<suite name="Calculator Suite">
  <test name="Calculator Test">
    <groups>
      <run>
        <include name="BasicOperations" />
        <include name="AdvancedOperations" />
      </run>
    </groups>
    <classes>
      <class name="CalculatorTest" />
    </classes>
  </test>
</suite>
```