Core Java 8 and Development Tools

Lesson 14 : Advanced Testing Concepts

Lesson Objectives

- After completing this lesson, participants will be able to
  - Understand advanced testing concepts
  - Work with test suites
  - Implement parameterized tests and mocking concepts

Composing Test into Test Suites

- A testsuite comprises of multiple tests and is a convenient way to group the tests, which are related.
- It also helps in specifying the order for executing the tests.
- JUnit provides the following:
  - org.junit.runners.Suite class : It runs a group of test cases.
  - @RunWith : It specifies runner class to run the annotated class.
  - @Suite.SuiteClasses : It specifies an array of test classes for the Suite.Class to run.
    - The annotated class should be an empty class but may contain initialization and cleanup code.

Composing Test into Test Suites

- Example:

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
@RunWith(Suite.class)
@Suite.SuiteClasses({ TestCalAdd.class, TestCalSubtract.class,
TestCalMultiply.class, TestCalDivide.class })
public class CalSuite {
// the class remains completely empty,
// being used only as a holder for the above annotations
}
```

Demo

- Demo on:
  - Composing tests into Test Suites
    - TestPersonSuite.java

Reusing Tests

- Parameterized tests allow you to run the same test with different data.
- To specify parameterized tests:
  - Annotate class with @RunWith(Parameterized.class).
  - Add a public static method that returns a Collection of data.
    - Each element of the collection must be an Array of the various parameters used for the test.
  - Add a public constructor that uses the parameters.

Reusing Tests

- Example:

```
@RunWith(Parameterized.class)
 public class SomethingTest {
@Parameters
public static Collection<Object[]> data() { …. }
public SomethingTest()
{…..}
@Test
public void testValue()
{…..}
}
```

Testing in Isolation

- Unit Testing of any method should be ideally done in isolation from other methods.
- For testing in isolation, you need to be independent of expensive resources.
- Use mock objects to perform testing in isolation.
- Mock object is created to represent an object that your code will be collaborating with.

Advantages of Using Mock Objects

- There are obvious advantages of using mock objects:
  - You get the ability to test code that is not yet written.
  - They help teams to unit test one part of the code independently.
  - They help to write focused tests that will test only a single method.
  - They are helpful when the application integrates with expensive external resources.

Mock Objects in JUnit

- Mock objects can either program these classes manually or use EasyMock to simulate these classes.
  - EasyMock provides mock objects for interfaces in JUnit tests.
  - EasyMock is an open source software that is available under the terms of the Apache 2 license.

Demo

- **Lab-5**
  - Using Mock Object in JUnit

Summary

- In this lesson, you have learnt:
  - Advanced Testing Concepts

Review Question

- Question 1: While writing unit tests you should test ___.
  - Option 1: Only public methods
  - Option 2: Only constructors
  - Option 3: Should test all methods
- Question 2: Use constant expected values in assertions.
  - True / False