Core Java 8 and Development Tools

Lesson 18 : Advanced Testing Concepts

Capgemini

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

18.1: Advanced Testing 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.

18.2: Test Suites
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
}
```

18.2: Test Suites
Demo

Demo on:

- Composing tests into Test Suites
  - TestPersonSuite.java

18.3: Parameterized Tests
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.

18.3: Parameterized Tests
Reusing Tests

Example:

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

18.4: Mocking Concepts
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.

18.4: Mocking Concepts
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.

18.4: Mocking Concepts
EasyMock 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.
- EasyMock instantiates an object based on an interface or class.

18.4: Mocking Concepts
EasyMock Objects in JUnit

The createNiceMock() method creates a mock which returns default values for methods which are not overridden.

```
import org.easymock.EasyMock; // package
.......

public class LoginTest {
private LoginServiceImpl service;
private UserDAO mockDao;

@Before
public void setup() {
service = new LoginServiceImpl();
mockDao = EasyMock.createNiceMock(UserDAO.class);
service.setUserDao(mockDao);
}
```

18.4: Mocking Concepts
Mock Objects in JUnit

Easy mock has different methods which are used to configure the Mock object.

- expect()
- addReturn()
- times()
- replay()
- verify()

```
EasyMock.expect(mockDao.loadByUserNameAndPassword(
      username, password)).andReturn(user);
EasyMock.replay(mockDao);
 .......
EasyMock.verify(mockDao);
```

18.4: Mocking Concepts Demo

Demo on:

- Using Mock Object in JUnit
  - demo.mock.LoginTest.java

Summary

In this lesson, you have learnt:

- Advanced Testing Concepts
- Implementation of Mock object in jUnit

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

Review Question

Question 3: Which method creates a mock which returns default values for methods which are not overridden ?

- Option 1: createNiceMock()
- Option 2: replay()
- Option 3: verify()

Question 4: verify method validates that all expected method calls were executed and in the correct order .

- True / False