Unit Tests Best Practices

Source Code

It's a good idea to keep the test classes separate from the main source code.

Package Naming Convention

We should create a similar package structure in the *src/main/test* directory for test classes, this way improving the readability and maintainability of the test code.

Test Case Naming Convention

The test names should be insightful, and users should understand the behavior and expectation of the test by just glancing at the name itself.

It's often helpful to name the test cases in *given_when_then* to elaborate on the purpose of a unit test

Expected vs Actual

A test case should have an assertion between expected and actual values.

Prefer Simple Test Case

We should create a simple test case that asserts hard-coded expected value against the actual one.

Appropriate Assertions

Always use proper assertions to verify the expected vs. actual results. We should use various methods available in the *Assert* class of JUnit or similar frameworks such as AssertJ.

Specific Unit Tests

Instead of adding multiple assertions to the same unit test, we should create separate test cases.

Test Production Scenarios

Unit testing is more rewarding when we write tests considering real scenarios in mind.

Mock External Services

Although unit tests concentrate on specific and smaller pieces of code, there is a chance that the code is dependent on external services for some logic.

We can use various frameworks such as Mockito, EasyMock and JMockit for mocking external services.

Avoid Code Redundancy

Create more and more helper functions to generate the commonly used objects and mock the data or external services for similar unit tests.

Annotations

Often, testing frameworks provide annotations for various purposes, for example, performing setup, executing code before and tearing down after running a test.

Various annotations such as JUnit's *@Before, @BeforeClass and @After* and from other test frameworks such as TestNG are at our disposal.

80% Test Coverage

As a rule of thumb, we should try to cover 80% of the code by unit tests.

TDD Approach

Test-Driven Development (TDD) is the methodology where we create test cases before and in ongoing implementation. The approach couples with the process of designing and implementing the source code.

Automation

We can improve the reliability of the code by automating the execution of the entire test suite while creating new builds.