Test Driven Development (TDD) and Unit Testing Essentials Training (3 days)

Objectives

Working in a hands-on learning environment, guided by our expert team, you will:

- Understand JUnit.
- Understand and use the JUnit Test Runner interface.
- Use JUnit to drive the implementation of Java code.
- Best practices and patterns for test development.
- Understand the role of debugging when done in conjunction with tests.
- Understand not only the fundamentals of the TDD using Java, but also its importance, uses, strengths and weaknesses.
- Understand how JUnit affects your perspective on development and increases your focus on a task.
- Learn good JUnit coding style.
- Create well-structured JUnit programs.
- Compile and execute programs using JUnit and DBUnit
- How to extend testing with mock objects using Mockito.
- Look at refactoring techniques available to make code as reusable/ robust as possible.
- Discuss various testing techniques.

Audience

This programming course is for experienced Java developers.

Prerequisites

Students should have development skills at least equivalent to the following course(s) or should have attended as a pre-requisite:

Duration

Three days

Introducing Test-driven Development

- Rationale for TDD
- The process of TDD

- Advantages to TDD
- Side-effects of TDD
- Tools to support TDD
- Tutorial: Setup IntelliJ for Using Maven

Unit Testing using JUnit

- Purpose of Unit Testing
- Good Unit Tests
- Test Stages
- Unit Testing Vs Integration Testing
- Understanding Unit Testing Frameworks

Jumpstart: JUnit 5.x

- Understand and work with the features of JUnit
- Write unit tests using @Test annotation
- Test Result Verification (Assertions)
- Manage fixtures using @BeforeEach, @AfterEach, @BeforeAll and @AfterAll annotations
- Maven setup using Surefire plugin
- Lab: Demo JUnit
- Lab: Build JUnit Case Study
- Lab: Jumpstart JUnit

Annotations

- Use @DisplayName to specify a custom name for the test
- Check for exceptions thrown by test
- Use @Disabled to prevent a test class or method from running
- Use timeouts to fail test that take longer than required
- Test Execution Order
- Lab: Working with @Test Annotation

Hamcrest

- Learn the notation of assertThat
- Know the objective of Hamcrest library
- Use Hamcrest's logical and object matchers
- Use Hamcrest's number and collection matchers
- Lab: Working with Hamcrest

Parameterized Tests

- The @ParameterizedTest annotation
- A parameterized test to test code under several conditions

- Define different sources for test data (@ValueSource, @CsvSource, @CsvFileSource,@EnumSource, @MethodSource, @ArgumentSource)
- Lab: Working with Parameterized Tests

Advanced Features

- JUnit 4 vs JUnit 5
- Nested Unit Tests
- Repeated Tests
- JUnit Extensions
- ExecutionConditions
- Lambda Support
- Grouped Assertions
- Lab: Working with Advanced Features

JUnit Best Practices

- "Good" Tests
- Bad Smell
- White-Box Unit Testing
- Black-Box Unit Testing
- Automation and Coverage

Mocking of Components

- Why We use Test Dummies
- Working with Mock Objects
- Using Mocks with the User Interface
- Mock Object Strategies

Mock Objects and Mockito

- Mockito Description and Features
- Mockito Object Lifecycle
- JUnit 5 and Mockito Dependency Injection
- Stubs Using ArgumentMatchers
- Verifying Behavior in Mockito
- Partial Mock Objects
- The Spy annotation
- Lab: Mock Object and Mockito

PowerMock

- PowerMock Description and Features
- Using PowerMockito
- @PrepareForTest
- Mocking a final class or final method
- Mocking a Static Method

State-based vs. Interaction-based Testing

- State-based Testing
- Interaction-based Testing
- Mock Objects Support Each Approach
- Three Areas to Check in a Test
- Lab: Interaction-based Testing

Improving Code Quality Through Refactoring

- Refactoring Overview
- Refactoring and Testing
- Refactoring to Design Patterns
- Lab: Refactoring
- Lab: Best Practices - Refactoring Tests

Testing Messaging frameworks

- RabbitMQ Overview
- Writing test cases for RabbitMQ topics and Queues
- Test Connections and resources

Database Testing: DbUnit

- Setting up DbUnit
- Defining a Dataset File in XML, CSV or Excel
- Writing a DbUnit Test Class
- Assert the results
- Use the FailureHandler and ValueComparer
- Using Date and Time in test sets
- Export a data set
- Lab: Introduction to DbUnit
- Lab: DbUnit Assertions
- Lab: Selenium and DbUnit