TT3525: Mastering Unit Testing (JUnit, Mockito, and PowerMock) – 3 days

Mastering Unit Testing is a three-day, comprehensive hands-on unit testing training course geared for developers who need to get up and running with essential unit testing skills using JUnit, EasyMock, and other tools. Throughout the course students learn the best practices for writing great programs in Java, using unit testing techniques. This comprehensive course also covers essential TDD topics and skills.

Students who attend **Mastering Unit Testing** will leave the course armed with the skills they require to develop solid Java programs, using sound coding testing techniques and best coding practices. This course quickly introduces developers to the features of JUnit and educates them regarding JUnit's strengths and weaknesses.

JUnit and EasyMock make it possible to write higher-quality Java code. These powerful tools are designed to support robust, predictable and automated testing development in the Java enterprise application arena.

This course includes coverage of many of the essential JUnit capabilities, and can be tailored to focus exactly on the areas that you are interested in.

Working within in a dynamic, learning environment, guided by our expert TDD team, attendees will::

- **Understand what unit testing is and what it is not intended to cover**
- **Understand JUnit.**
- **Understand and use the JUnit Test Runner interface.**
- **Use JUnit to drive the implementation of Java code.**
- **Test applications using native IDE support.**
- **Best practices and patterns for unit testing.**
- **Understand JUnit's strengths and weaknesses**
- **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.**
- **Understand how JUnit testing can be used for either state-based or interaction-based testing.**
- **How to extend testing with mock objects using EasyMock.**
- **Look at refactoring techniques available to make code as reusable/robust as possible.**
- **Discuss various testing techniques.**

The following JUnit-based testing frameworks are examined:

- **JUnit**
- **Mockito**
- **PowerMock**

Workshop Topics Covered / Course Syllabus

Session: JUnit

Lesson: JUnit Overview

- **What is Unit Testing?**
- **Purpose of Unit Testing**
- **Successful Unit Testing**
- **Good Unit Tests**
- **Test Stages**
- **Unit Test Stage**
- **Integration Test Stage**
- **Unit Testing Vs Integration Testing**
- **Functional Testing**
- **Non-Functional Testing**

Lesson: Jumpstart: JUnit 4.x

- **Understanding Unit Testing Frameworks**
- **JUnit Overview**
- **JUnit Design Goals**
- **JUnit Features**
- **Reasons to Use JUnit**
- **How JUnit Works**
- **Class to be Tested**
- **Test Case using JUnit**
- **Exploring JUnit**
- **Writing the TestCase**
- **Test Result Verification (Assertions)**
- **Assert**
- **Launching Tests**
- **Failures vs. Errors**
- **Introducing Class Message**
- **Creating Class MessageTest**
- **The First Test Implementation Steps**
- **The Initial Test Code**
- **Testing the Constructor**
- **Running a Test in an IDE**
- **Running a Test From the Command Line**
- **Seeing Results of a Test: JUnit View**
- **Using the Results of a Test**
- **Seeing Results of a Successful Test**
- **Test Suites**
- **Creating a Test Suite**
- **Composing Tests Using Suite**
- **JUnit Test Fixture**

- Managing Resources with Fixtures
- Share Similar Objects
- Share Expensive Setups
- JUnit Method Lifecycle
- JUnit: Resources

Lesson: @Test Annotation
- Test Execution Cycle
- Checking for Exceptions
- Limitation of Expected Parameter
- Testing for an Expected Exception
- Testing Using a Timeout
- Using Timeouts
- Test Annotation: Resources

Lesson: Hamcrest
- About Hamcrest
- junit.assert.Assert.assertThat(...)
- The Hamcrest Matcher Framework
- Using assertThat
- Hamcrest Matchers - Logical
- Hamcrest Matchers - Object
- Hamcrest Matchers - Number
- Hamcrest Matchers - Collections
- Additional Hamcrest Matchers
- Hamcrest: Resources

Lesson: Parameterized Tests
- Parameterize a Test Case
- Injecting the Parameters
- Setting the Parameters
- Write the Test Method
- Writing a Parameterized Test
- Test Execution Cycle
- Observations
- Parameterized Tests: Resources

Lesson: Theories
- Writing Theory Enabled Tests
- Defining DataPoints
- Defining Theories
- Test Execution Cycle
- Observations
- Theories: Resources

Lesson: JUnit Best Practices
- So What is a "Good" Test?
- Good: Readable Equates to Maintainable
- Good: Proper Organization and Structure
- Good: Test the Right Thing
- Good: Run in Solitude
- Practices to Reduce and Manage Dependences
- Good: Reliability
- Importance of Quality of Assertions
- Bad Smell: Primitive Assertions
- Bad Smell: Broad Assertion
- Bad Smell: Hidden Beef
- Bad Smell: Split Personality
- Bad Smell: Split Logic
- Managing Data
- Bad Smell: Parameterized Mess
- Coding Practices
- Legacy Code
- Preparing Unit Test Environment
- Stubs -> Mocks
- White-Box Unit Testing
- Black-Box Unit Testing
- Keys to Success
- Boundary Between Unit and Integration Testing
- Integration Testing
- Purpose of Integration Testing
- Who Does Integration Testing
- The Lowest Bar for Unit Testing
- Automated Testing
- Automation and Coverage
- Working With Coverage Analysis

Session: Testing Tools and Techniques

Lesson: Improving Code Quality Through Refactoring
- Refactoring Overview
- Sample of Refactorings
- Refactoring and Testing
- Suggested Refactoring
- The Impact of Refactoring
- Refactoring to Design Patterns
- Sample Refactorings
- Best Practices
- Refactoring
- Naming conventions
- Organizing test suites

Lesson: Mocking of Components
- Why We use Test Dummies
- Isolation
- Improving Speed and Reliability
- Handling Special Conditions and Hidden Data
- Types of Test Dummies
- Stubs
- Mock Objects
- Working with Mock Objects
- Challenges of Testing User Interfaces
- Using Mocks with the User Interface
- Mock Object Strategies

Lesson: Mock Objects and Mockito
- Mockito Description and Features
- Decoupling System Under Test From Mocks
- Stubbing and Argument Matchers
- Verifying Innvocations
  - Number
  - Order
  - Redundant
  - Timeouts
- Advanced Stubbing Techniques
- do* Methods
- Spying
- Working with Partial Mocks
- Mockito Annotations
- Mockito Limitations
- Testing with Mocks
- Mockito HOWTO
- Mockito Best Practices

Lesson: PowerMock
- PowerMock Description and Features
- Using PowerMock
- PowerMock Object Lifecycle
- Example of Mocking a Static Method
- PowerMock Notes

Appendix: Adding Testing to the Build Process
- JUnit and Ant
- The Ant JUnit Tag
- Example Ant Build File
- Running JUnit Tests From Ant
- Generating a JUnitReport

TT3525/j13