Specialized - Unit Testing Essentials using JUnit and Mockito

Skills Gained
- 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, and PowerMock

Who Can Benefit
This is an intermediate-to-advanced level Java course, designed for developers who wish to get up and running on test-driven development immediately. Attendees should be familiar with Java and object-oriented technologies. Real world programming experience is a must.

Course Details
Experiential Learning – Course Structure
Throughout the course students will be led through a series of progressively advanced topics, where each topic consists of lecture, group discussion, comprehensive hands-on lab exercises, and lab review. This workshop is about 50% hands-on lab and
50% lecture. Multiple complete “mini-projects” are laced throughout the course, designed to reinforce fundamental skills and concepts learned in the lessons, all working in the JUnit environment. Because these lessons, labs and projects are presented in a building block fashion, students will gain a solid understanding of not only the core concepts, but also how all the pieces fit together in a complete application.

**Session: JUnit**

JUnit Overview

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

Jumpstart: JUnit 4.x

- JUnit Overview
- How JUnit Works
- Launching Tests
- Test Suites
- JUnit Test Fixture

@Test Annotation

- Test Execution Cycle
- Checking for Exceptions
- Using Timeouts

Hamcrest

- About Hamcrest
- The Hamcrest Matcher Framework
- Hamcrest Matchers

Parameterized Tests

- Injecting the Parameters
- Setting the Parameters
- Test Execution Cycle
- Observations

Theories

- Writing Theory Enabled Tests
- Defining DataPoints
- Defining Theories
- Observations

JUnit Best Practices
Session: Testing Tools and Techniques
Improving Code Quality Through Refactoring

- Refactoring Overview
- Refactoring and Testing
- Refactoring to Design Patterns
- Naming conventions

Mocking of Components

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

Mock Objects and Mockito

- Mockito Description and Features
- Stubbing and Argument Matchers
- Verifying Invocations
- Working with Partial Mocks
- Mockito HOWTO

PowerMock

- PowerMock Description and Features
- PowerMock Object Lifecycle
- Mocking a Static Method

Appendix: Adding Testing to the Build Process

- JUnit and Ant
- The Ant JUnit Tag
- Running JUnit Tests From Ant
- Generating a JUnitReport

Schedule (as of 3)