ProTech protechtraining.com *"Charting the Course ... ... to Your Success!"*

Java Test Driven Development with JUnit 4 (3 Day)

Course Summary

Description

Test Driven Development (TDD) has become a standard best practice for developers, especially those working in an Agile development environment. TDD is a team and individual code development discipline which, when followed correctly, increases the productivity of both individual developers and entire teams. From the programmer's perspective, TDD has another benefit – it allows programmers to eliminate the tedious tasks of debugging and rewriting code so that they can focus on the creative work of designing and writing code. It makes programming fun again.

The course focuses on two main topic streams. The first is how to start using the TDD process and integrate it into code development or maintenance activities. The best practices for TDD are covered in depth including explanations on why these are best practices. This stream concludes with an introduction into the practice of how TDD is used to refactor and improve existing code as well as code under development.

The second stream is tool oriented. Students learn the features of JUnit through a hands-on exploration of the framework, as well as exploring some of the common assertion and mocking libraries.

The class is designed to be about 50% hands on labs and exercises, about 25% theory and 25% instructor led hands on learning where students code along with the instructor. The course closes with students developing an action plan for implementing what they have learned in class into their own development environment.

Topics

- The TDD process - "red, green, refactor"
- Eliminating technical debt with TDD
- Why TDD works
- Integrating the TDD discipline into a development activites
- TDD and programming and design best practices
- Developing a TDD project using JUnit
- JUnit concepts, architecture and features
- Assertion libraries (hamcrest and AssertJ.)
- Using mocks effectively
- Mocking libraries review (Mockito, JMockit, EasyMock, etc.)
- Best practices for developing good test cases and test suites
- Best practices when using TDD and JUnit to improve development
- Code smells and refactoring
- Using TDD to refactor code
- Migrating to TDD as a programming discipline

Audience

This course is designed for intermediate level Java programmers.

Prerequisites

Knowledge and experience in Java is essential for understanding the course material. No experience or background in software testing is required.

Duration

Three days

ProTech "Charting the Course ...

protechtraining.com

... to Your Success!"

Java Test Driven Development with JUnit 4 (3 Day)

Course Outline

I. Java Test Driven Development Introduction

A. The TDD process as a discipline
B. How TDD improves efficiency and effectiveness of programming
C. Eliminating technical debt, debugging and rework
D. Integrating TDD with best practices in program design and coding
E. TDD as a core Agile practice
F. The Agile testing quadrants
G. The importance of test automation
H. Refactoring: what it is and why we do it

II. An Introduction to JUnit

A. JUnit architecture and functionality
B. Implementing TDD with the JUnit test framework
C. Test runners, fixtures, test classes and test methods
D. JUnit annotations
E. JUnit assertions – how tests pass and fail
F. Writing and validating a JUnit test
G. Introduction to mock objects and unit testing

III. TDD Best Practices I

A. Testing through interfaces and good Java design
B. Command / Query segregation
C. Functional testing concepts for TDD
D. Relationship between good class design and ease of testing
E. Understanding Unit testing – component isolation
F. Planning TDD work flow minimize development effort
G. How to decide what test to add next
H. Errors, faults, failures and exceptions
I. TDD best practices
J. Common errors when implementing TDD

IV. More JUnit

A. Design by contract
B. Writing tests for preconditions, post-conditions and invariants
C. Testing exceptions
D. Using test fixtures effectively
E. Selective execution of tests
F. Ordering the addition of tests into JUnit
G. JUnit reporting
H. Differences between test errors and test failures

V. Testing Concepts

A. Criteria for good testing: validity, accuracy, correctness and reliability
B. Why our tests have to be good
C. Common sources of test case errors
D. Systematic and algorithmic test case development
E. Deriving test cases from acceptance tests
F. Functional coverage measures
G. Determining optimal numbers of tests
H. Using test cases to identify requirements and specification problems
I. Dealing with valid, invalid and outlier test cases
J. Combinatorial versus stateful testing

VI. Assertions and Predicates

A. Four generations of assertions: Java, JUnit, Hamcrest and AssertJ
B. Using hamcrest to develop complex assertion predicates
C. Using hamcrest to examine structures, lists and other complex objects
D. Writing complex predicates in AssertJ
E. Overview of hamcrest and AssertJ features

Java Test Driven Development with JUnit 4 (3 Day)

Course Outline

VII. Mocking and Mock Libraries

A. Mocks, stubs, drivers and incremental unit testing
B. Implementing interfaces with a mock object
C. Overview of mocking libraries: EasyMock, Mockito, JMockit etc.
D. How mocking libraries work – proxy versus instrumented
E. Using the standard mocking libraries in a TDD project
F. Designing and implementing a mocking strategy

VIII. TDD Best Practices II

A. Organizing and maintaining the test environment
B. Testing the test code
C. Co-developing tests and code design
D. Deciding on how many tests are needed
E. TDD implementation patterns
F. Metrics for TDD

IX. Refactoring

A. Refactoring as controlled code changes
B. Using TDD to implement a refactoring
C. Code smells – driver for refactoring
D. Code refactoring versus design refactoring
E. Refactoring to a design pattern
F. Using continuous refactoring to reduce technical debt
G. Refactoring best practices

X. Implementing TDD

A. Review of TDD best practices
B. Review of JUnit best practices
C. Planning a TDD project
D. Integrating TDD into a development process
E. Integrating TDD with coding excellence and software craftsmanship
F. The importance of metrics
G. Developing an implementation plan for migrating to TDD
H. Pitfalls, snares and traps to avoid