

Course 478:

Refactoring and Maintaining Legacy Code

(4 days)

Course Description...

This is an intermediate course for programmers who know the basics of Java. The course covers advanced techniques useful in enterprise development including the maintenance and improvement of already developed code. The course uses a test-driven development (TDD) approach for exercises and where practical working with an existing code set. Exercises show techniques and approaches needed in production ready code. The course is approximately 50% lecture and 50% hands-on work.

Learning Objectives...

- Learn strategies to maintain, change and improve legacy code
- Manage changes to legacy code
- Make changes without breaking existing code
- Organize data and methods better
- Simplify the implementation of logic
- Build a test harness around inflection points
- Break dependencies using design patterns

Who should attend...

This is an intermediate level Java programming course, designed for developers who wish to learn what's new in Java. The student should be an experienced Java programmer, with practical development experience in Java.

Prerequisites...

Practical development experience in Java.

See next page for a detailed course outline...



Course Outline...

- **Introduction and Overview**
 - Course Objectives
- **What is legacy code?**
 - Maintaining code without unit tests
 - Test coverings
 - Unit tests vs. test covering
 - Identifying change points
- **Managing changes to legacy code**
 - Legacy code management strategy
 - Finding inflection points
 - Identifying internal and external dependencies
 - Breaking external dependencies
 - Breaking internal dependencies
 - Making changes
 - Refactoring
- **Testing legacy code changes**
 - Unit testing with JUnit
 - Testing exception throwing
 - Functionality testing
 - Testing container-managed objects
 - Mock objects
 - Testing services
 - Testing Data Access Objects
 - DBUnit
- **Refactoring**
 - Identifying code smells
 - Simplifying code structure
 - Organizing data and methods
 - Simplifying and encapsulating logic
 - Extracting subclasses and interfaces
- **Testing in hard-to-test situations**
 - Hidden dependencies
 - Multi-level dependencies
 - Global dependencies
 - Undetectable side effects
 - Hidden methods
 - Library dependencies
 - Large classes
 - Large methods
 - API-heavy applications



- **Breaking dependencies with design patterns**
 - Adapter
 - Façade
 - Command Design Pattern
 - Dependency Inversion
 - Generalize methods
 - Strategy Design Pattern
- **Working effectively with legacy code**
 - Legacy code management strategy
 - Targeted refactoring
 - Improved design
 - Managed change

Please contact your ROI representative to discuss course customization!!!