Java Best Practices and Design Patterns: Hands-On - 4 Days

Course 516 Overview

You Will Learn How To

- Employ best practices to rapidly build reliable and scalable Java applications
- Effectively apply Test Driven Development to enhance program maintainability
- Solve architectural problems with proven design patterns
- Write robust threaded applications and avoid concurrency hazards
- Code securely in Java, authenticate users and enforce access control

Who Should Attend

Developers, architects and those involved in Java projects who want to expand their Java programming skills and have familiarity with Java code at the level of Course 471, "Java Programming Comprehensive Introduction".

Hands-On Exercises

- Creating tests in tandem with classes
- Implementing key object-oriented design patterns for extensibility and maintainability
- Profiling a program and improving response time
- Refactoring code to improve maintainability
- Authenticating users and controlling access to programs

Java Best Practices and Design Patterns: Hands-On - 4 Days

Course 516 Outline

Effective Programming in Java

- Clarifying the goals of best practices
- Identifying the key characteristics of high-quality software

Applying Test Driven Development

Exploiting unit testing

- Composing and maintaining JUnit tests
- Taking advantage of advanced JUnit features

Creating matchers and mock objects

- Writing custom Hamcrest matchers
- Testing with fake objects and mocks

Leveraging Design Patterns

Employing common design patterns

- Observer
- Iterator
- Template method
- Strategy
- Factory
- Singleton

Refactoring legacy code

- Identifying reasons to change software
- Clarifying the mechanics of change
- Writing tests for legacy classes and methods

Extending Applications with Java Meta Programming

Improving type safety with generics

- Creating generic classes and methods
- Navigating generic class hierarchies

Adding metadata by writing annotations

- Leveraging the built-in and custom annotations
- Annotating with meta-annotations

Modifying runtime behaviour with reflection

- Retrieving class and method data dynamically
- Assessing disadvantages of reflection

Tuning for Maximum Performance

Measuring and improving performance

- Assessing response time
- Conducting load and stress tests
- Specifying strategies for improving performance

Exploiting garbage collectors

- Choosing appropriate algorithms for garbage collection
- Avoiding the hazards of finalisers
- Preventing memory leaks with reference types

Taking full advantage of threads

- Writing reliable thread-safe code
- Avoiding race hazards and deadlocks
- Employing the Executors framework

Bulletproofing a threaded application

- Synchronising and sharing data between threads
- Managing the performance implications of synchronisation

Architecting for Separation of Concerns

Allocating responsibilities to components

- Translating method calls with the adaptor pattern
- Adding behaviour with a proxy
- Implementing inversion of control

Persisting objects

- Abstracting persistence using data accessor objects
- Integrating the Java Persistence Architecture (JPA)

Employing Java Frameworks

Supporting multiple types of views with a Model-View-Controller (MVC) pattern

- Alleviating developer effort with Java web application frameworks
- Comparing action- and pull-based frameworks

Leveraging the Java EE server

- Developing the client tier and interacting with the web tier
- Encapsulating logic and workflow in the business tier
- Integrating the enterprise information systems tier

Enforcing Security Constraints

- Coding securely in Java
- Restricting access to protected resources
- Applying role-based security
- Avoiding security pitfalls