

Best Practices in Java Programming: Hands-On - 4 Days

Course 516 Overview

- You Will Learn How To**
- Apply Java best practices to increase productivity and build fast, secure and reliable applications
 - Automate deploying, testing and detecting bugs in software applications
 - Solve architectural problems with proven design patterns and advanced language features
 - Maximize software performance
 - Improve the reliability of threaded applications
 - Code securely in Java and authenticate with industry-standard security frameworks
- Course Benefits** Java provides features to build robust, secure and responsive applications. Knowledge of the language and APIs alone is not enough to exploit the full power of Java. Developers must take advantage of proven best practices and industry-standard software development techniques. This course provides the skills needed to solve real-world software development problems and deliver fast, reliable applications.
- Who Should Attend** Developers, architects and anyone involved in Java projects who wants to expand their Java programming skills. Real-world knowledge of Java at the level of Course 471, "Java Programming Comprehensive Introduction," is assumed.
- Hands-On Training** You apply industry-standard best practices and gain experience using advanced APIs and language features. Exercises include:
- Improving testability by creating a class in tandem with its unit test
 - Implementing key object-oriented design patterns for extensibility and maintainability
 - Optimizing software performance by reordering loops and reducing database calls
 - Invoking dynamic business rules with scripting
 - Enforcing security constraints

Best Practices in Java Programming: 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

Optimizing Software Development with Proven Techniques

Simplifying project build and deployment

- Automating the build process using Ant
- Controlling and configuring logging

Applying test-driven development

- Unit-testing complex components
- Composing and maintaining JUnit tests
- Automating project-wide testing
- Validating application results with functionality tests
- Testing container-managed components such as servlets

Improving Code Quality through Better Design

Expert recommendations

- Balancing extensibility and maintainability
- Avoiding problems with clone()
- Exception best practices

Attaining type safety

- Eliminating run-time errors with generics
- Writing generic classes and methods
- Limiting parameter values with canonicalization

Enforcing encapsulation

- Providing coarse-grained methods with Memento
- Simplifying adaptation to interfaces

Refactoring and design patterns

- Streamlining source code by refactoring
- Designing to interfaces for improved software flexibility
- Key object-oriented design patterns
- Template Method
- Strategy
- Singleton
- Composite
- Factory
- Inversion of control

Tuning for Maximum Performance

Measuring performance

- Applying performance profiling tools

- Assessing response time
- Conducting load and stress tests
- Identifying performance bottlenecks

Strategies for improving performance

- Techniques for dealing with common Java performance issues
- Exploiting generational garbage collectors
- Choosing appropriate JVM and container settings
- Assessing the need for NIO and JNI
- Reordering loops to improve response time
- Processing streaming data to reduce memory overhead

Effective use of the Collections API

- Preventing memory leaks with weak references
- Selecting the best collection classes

Taking Full Advantage of Threads

Improving response time by parallelization

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

Bulletproofing a threaded application

- Synchronizing threads
- Techniques for sharing data between threads
- Managing the performance implications of synchronization

Enforcing Security Constraints

Bulletproofing applications

- Secure coding in Java
- Restricting access to protected resources
- Establishing security policies

Authentication and authorization

- Applying role-based security
- Authenticating users in Web applications

Managing Change with Design Patterns

Limiting the impact of changes

- Centralizing properties using Singleton
- Inserting transparent behavior with Proxy
- Wrapping external libraries using Adapter

Modern design patterns

- Inverting control (IoC) through bean factories
- Injecting behavior with aspects

- Adding scripting abilities to an application
- Scripting end-user behavior