Java Programming Best Practices

Course Summary

Description

This four-day course provides the intermediate-level Java programmer with techniques and tools, but more importantly with the ability to recognize coding pitfalls and to take the opportunity to apply a variety of best practices when developing enterprise-class software. We review Java exception handling and logging APIs and discuss best practices for error handling, logging, and tracing in Java. We consider modern Java threading and concurrency solutions such as atomic operations, semaphores, and thread pools. And we look at Java Reflection, especially because it is essential to solving certain problems that arise later in the course, such as isolating test environments, certain factory patterns, and dependency injection.

A multi-chapter module in the middle of the course introduces automated testing, JUnit, and test-driven development (TDD) practices, along with mocking tools and practices. The programmer is encouraged not only to test, but to write solid, fine-grained, and well-isolated tests, and to consider the design of classes under test to foster the best testability.

Another module on code factoring, refactoring, and design patterns concludes the course. We start by recognizing some common coding pitfalls, for each of which we identify and explore better code-factoring approaches. This leads naturally to a discussion of design patterns, and we will have plenty of exercise in “gang of four” patterns to round out the class. Note that we intentionally provide more patterns material at the end of the course than we will be able to fit into the four-day timeline. We cover as much or as little of this as time allows -- most often a day to a day-and-a-half.

Topics

- Exception Handling and Logging
- Threads and Concurrency
- Reflection
- Automated Testing with JUnit
- Writing Tests
- Test-Driven Development
- Mocking
- Testing Java EE Components
- Refactoring to Best OO Practices
- Recognizing and Applying Patterns
- Creational Patterns
- Behavioral Patterns
- Structural Patterns

Audience

This course is designed for intermediate-level Java programmers.

Prerequisites

Before taking this course, solid Java programming experience is essential - especially object-oriented use of the language, including interfaces and abstract classes, generics, and the Collections API.

Duration

Four days
Java Programming Best Practices

Course Outline

I. Exception Handling and Logging
   A. Reporting and Trapping Errors
   B. Exception Handling
   C. Throwing Exceptions
   D. Declaring Exceptions per Method
   E. Catching Exceptions
   F. The finally Block
   G. Catch-and-Release
   H. Chaining Exceptions
   I. try-with-resources
   J. Logging
   K. The Java SE Logging API
   L. Loggers
   M. Logging Levels
   N. Handlers
   O. Configuration
   P. Best Practices

II. Threads and Concurrency
   A. Java Thread Model
   B. Creating and Running Threads
   C. Manipulating Thread State
   D. Thread Synchronization
   E. Volatile Fields vs. Synchronized Methods
   F. wait and notify
   G. join and sleep
   H. The Concurrency API
   I. Atomic Operations
   J. Thread-Safe Collections
   K. Concurrency Primitives
   L. Executors and the ExecutorService
   M. Using Thread Pools

III. Reflection
    A. Uses for Meta-Data
    B. The Reflection API
    C. The Class< T > Class
    D. The java.lang.reflect Package
    E. Reading Type Information
    F. Navigating Inheritance Trees
    G. Dynamic Instantiation
    H. Dynamic Invocation
    I. Reflecting on Generics
    J. JavaBeans Properties
    K. Reflection in Dependency Injection
    L. Reflection with SQL Databases
    M. Reflection in XML Binding

IV. Automated Testing with JUnit
    A. Unit Testing
    B. JUnit
    C. The @Test Annotation
    D. Writing Tests
    E. The Asserts Class Utility
    F. The Test Runner
    G. Tool Support

V. Writing Tests
    A. Test Granularity
    B. Test Isolation
    C. Test Coverage
    D. Fake Objects
    E. Recording and Comparing Output
    F. Test Outcomes, Not Implementation
    G. Non-Invasive Testing
    H. Controlling the Test Environment
    I. Designing Testable Classes
    J. Refactoring for Testability
    K. Managing Dependencies
    L. Factories

VI. Test-Driven Development
    A. Writing the Test First
    B. The TDD Cycle
    C. Refactoring
    D. Benefits of TDD
    E. Resistance to TDD

VII. Mocking
    A. Why We Use Mock Objects
    B. Static Mock Classes
    C. Dynamic Mock Classes
    D. Stubbing
    E. Verifying
    F. Matching and Capturing
    G. Mocking Libraries
    H. Mockito
    I. Stubbing with when().thenReturn()
    J. Verifying with verify()
    K. Argument Matchers
    L. Argument Captors
    M. Verifying Multiple Calls
    N. Using a Spy

Due to the nature of this material, this document refers to numerous hardware and software products by their trade names. References to other companies and their products are for informational purposes only, and all trademarks are the properties of their respective companies. It is not the intent of ProTech Professional Technical Services, Inc. to use any of these names generically.
Java Programming Best Practices

Course Outline (cont’d)

VIII. Testing Java EE Components
   A. (This is an overview chapter, with no lab work. To be covered as time allows.)
   B. Testing Persistence Components
   C. Test Databases
   D. Mocking Database Interactions
   E. Auto-Rollback
   F. Mocking the Entity Manager
   G. Verifying Transactionality and Cleanup
   H. The Web Tier
   I. Mocking JNDI
   J. Mocking the Container
   K. Testing JSF Applications
   L. Web Services
   M. Messaging
   N. Mocking Sessions, Queues, and Messages
   O. Verifying Transactionality and Cleanup

IX. Refactoring to Best OO Practices
   A. Recognizing Pitfalls
   B. Refactoring to Better Practices
   C. Reliance on Magic Numbers and Literal Strings
   D. Enumerations
   E. Externalization via Properties
   F. Failure to Encapsulate
   G. Stateful and Behavioral Enumerations
   H. Over-Encapsulation
   I. Separation of Concerns
   J. Multi-Tier Architecture
   K. Factoring Code Effectively
   L. Strategy and Template Method Patterns
   M. Delegation Over Inheritance
   N. Decorator Pattern
   O. Hard-Coding Type Information
   P. Dependency Injection

X. Recognizing and Applying Patterns
   A. Design Patterns
   B. Defining a Pattern
   C. Unified Modeling Language
   D. Seeing Patterns
   E. Warning Signs and Pitfalls

XI. Creational Patterns
   A. Factory Patterns
   B. The Singleton Pattern
   C. APIs and Providers
   D. Cascading Factories

XII. Behavioral Patterns
    A. The Strategy Pattern
    B. The Template Method Pattern
    C. The Observer Pattern
    D. The Model/View/Controller Pattern
    E. The Command Pattern
    F. The Chain of Responsibility Pattern

XIII. Structural Patterns
     A. The Composite Pattern
     B. The Adapter Pattern
     C. The Decorator Pattern
     D. The Façade Pattern
     E. The Flyweight Pattern