

Building Java Enterprise Applications with Design Patterns: Hands-On - 4 Days

Course 318 Overview

- You Will Learn How To**
- Architect Java EE applications using industry-recognized best practices
 - Select design patterns and identify opportunities for their usage
 - Create flexible and powerful designs for core business logic
 - Design a data layer that manages transactions and optimizes queries
 - Centralize control logic in the Web presentation tier using Java EE patterns
 - Compare the designs of popular Java EE frameworks and choose the right one for your projects

Course Benefits The wide variety of Java enterprise technologies presents many challenges to designing an effective Java system. Java EE design patterns help by providing best practices, design ideas and proven techniques. In this course, you gain experience building scalable and maintainable Java EE applications. You learn to apply Java EE patterns to solve commonly recurring design problems.

Who Should Attend Anyone currently designing or developing Java EE applications. As the emphasis is on software design, familiarity with Java code at the level of Course 471, "Java Programming Comprehensive Introduction," is required. Practical experience with Java is required and knowledge of Java EE is beneficial.

Hands-On Training Throughout this course, you gain experience designing flexible, robust Java EE applications. Exercises include:

- Choosing a topology based on requirements
- Writing a simple distributed chat application
- Designing a flexible domain model
- Utilizing persistence mechanisms on the integration tier
- Designing detailed Web application workflows
- Implementing a complex Web-based Java EE application
- Profiling performance of JEE applications

Building Java Enterprise Applications with Design Patterns: Hands-On - 4 Days

Course 318 Outline

Java EE and Design Patterns

Design principles and OO design patterns

- Leveraging OO design patterns which adhere to best practices
- Determining the appropriate design patterns for requirements
- Singleton
- Strategy
- Template
- Proxy
- Observer

Design patterns and Enterprise Java

- Analyzing goals of Enterprise Java applications
- Planning for distributed applications
- Communicating between JVMs
- Implementing Remote Method Invocation
- Annotations and dependency injection
- Registering and locating remote objects with JNDI
- EJB 3.1 global JNDI names

Building the Business Tier

Modeling entities and use cases

- Realizing an application's domain model
- Business Object
- Application Service

Reducing the impact of known performance bottlenecks

- Eliminating inter-tier dependencies
- Service Facade
- Session Facade
- Business Delegate

Locating objects

- Singleton
- Factory
- Inversion of Control
- Service Locator

Implementing the business logic with Session Beans

- Injecting services to business logic using Session Beans
- Conversing with client using Stateful Session Beans
- Message-driven beans
- Exposing beans as Web services with annotations

Managing Resources in the Integration Tier

Abstracting the data layer

- Implementing effective Data Access Objects (DAO)
- Highlighting difficulties associated with Object/Relational Mapping
- Analyzing persistence technologies: Hibernate, JPA 2.0, EJB 3.1

Handling transactions effectively

- Considering local and global transaction needs
- Selecting optimistic or pessimistic locking

Structuring the Presentation Tier

Separating control and presentation logic

- Realizing the role of JSPs and servlets
- Constructing Model View Control (MVC) architectures

Planning and implementing complex workflows

- Front Controller
- Dispatcher View
- Service to Worker

Localizing disparate logic

- Improving maintainability of algorithms
- Writing modular JSPs
- Intercepting Filter
- View Helper
- Composite View

Leveraging Web frameworks

- Determining evaluation criteria
- Handling duplicate form submission with the Synchronizer Token pattern
- JSF 2.0
- Spring MVC
- Google Web Toolkit (GWT)
- Tapestry
- Wicket

Employing Lightweight Frameworks and Architecture

Overview of Spring Lightweight Framework

- Inversion of Control (IoC) design pattern
- Configuring the Spring IoC container

Promoting code reuse

- Aspect-Oriented Programming
- Sending e-mail using Spring

- Utilizing Spring data access templates

Maintaining Performance and Scalability

Designing for performance

- Distributed components and performance
- Measuring runtime performance
- Optimizing Java EE applications
- Caching
- Connection Pooling

Planning for scalability

- Analyzing design trade-offs in distributed architectures
- Clustering applications across servers
- Managing session state effectively