Java Design Patterns & Effective Programming

Course 101 – 24 Hours

Overview

Design patterns are a must for every developer. Design patterns help the developers write extensible and maintainable code. Design patterns also provide the developers with common vocabulary for design and allow easy classification of conceptual problems .
Another important aspect for Java programmers is to develop effectively. This means that best practices are taken, object and stacks are not abused, pitfalls are avoided and there is a correct usage of the APIs.
The course will focus on the well-known GoF patterns and their appliance in the Java language. In addition, the course details effective related issues, some taken from "Effective Java" which was written by Joshua Bloch.

Who Should Attend

- Java developers that want to code effective Java
- Java developers that want to implement common DPs in their applications
- Team leaders and software architects in Java environments

Prerequisites

- Experience in JavaSE programming
- Familiarity with Java web development & web-services - optional

Course Contents

Java Design Patterns

- UML Recap
  - Class Diagrams
  - Sequence Diagrams
  - Interaction Diagrams
- Design Principles
  - Open/Closed Principle
  - Design by Contract
  - Inversion of Control & Dependency Injection
  - Composition over Inheritance
- Creational Patterns
  - Factory
  - Abstract Factory
  - Builder
  - Prototype
  - Singleton

- Structural Patterns
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Proxy
- Behavioral Patterns
  - Command
  - Mediator
  - Chain of Responsibility
  - Iterator
  - Observer
  - State
  - Strategy
  - Interpreter
  - Visitor

Effective Java

- All about Objects
  - Object creation with Factories
  - Object creation with Builder
  - Singletons in Java
  - Comparing Objects: equals, Comparable, Comparator, hashCode
  - De-referencing and finalization
  - Cloning objects
  - Object serialization
- Primitives and Strings
  - Boxing and un-boxing
  - Using float and double
  - Numeric overflows
  - String concatenation
- Classes and methods
  - Minimize accessibility, mutability, scope
  - Accessor methods
  - Composition over inheritance
  - Design for inheritance
  - Using interfaces
  - Nested classes
  - Object initialization
  - Overloading
  - Varargs
  - Methods - best practices

- Exceptions
  - Exception performance and use-cases
  - Checked vs. runtime exceptions
  - Standard exceptions
  - Leaky abstractions with exceptions
  - Documenting exceptions
  - Ensuring failure atomicity
  - Catching exceptions
- Generics
  - Generics vs. raw types
  - Unchecked warnings
  - Lists vs. arrays
  - Generic types
  - Generic methods
  - Wildcards
- Enums and Annotations
  - Using enums
  - Enum ordinals
  - EnumSet and EnumMap
  - Enum extensibility with interfaces
  - Annotations use-cases
  - @Override annotation