## Java 8 Patterns, AntiPatterns and TDD

**Prerequisites:**

1. **Java/JEE Developers with 2-3 years of experience.**
2. **Java 8 working knowledge is a must.**

**Duration: 8 Days**

**Course Outline**

**Day 1**

## Introduction to Java Patterns?

* Explaining design patterns
* Understanding the advantages of design patterns
* Understanding the basic design patterns of the Java world
* Why they are important
* How they help us to keep code clean and robust
* How to decide which one to pick when in confusion
* Why to learn AntiPatterns

## Creational Patterns

* Abstract Factory
* Builder
* Factory Method
* Object Pool
* Prototype
* Singleton

## Structural Patterns

* Adapter
* Bridge
* Composite
* Decorator

**Day 2**

## Structural Patterns (Contd.)

* Facade
* Private Class Data
* Proxy

## Java 8 Specific Patterns

* How Java 8 Lambdas changed the way we think about Java
* How Java now supports functional programming
* How Java 8 changed object oriented programming
* Java 8 Patterns
* Functional Programming Patterns
  + Behavioral Injection Pattern
  + Prefer Named Functions Over Anonymous Lambdas Pattern
  + Stream Wrecks Pattern
  + Overcoming the Greatest Beast of All Null Pointer Pattern
  + The Loan Pattern Passing a block Pattern
  + Ways to Implement Type-Specific Logic Pattern
  + MapReduce Pattern

**Day 3**

## Java 8 Specific Patterns (Contd.)

* Old Patterns Java 8 way
  + Factory
  + Adapter
  + Chain of responsibility
  + Command
  + Decorator
  + Iterator
  + Observer
  + Strategy
  + Template method

## Java Anti Patterns?

* Why to learn AntiPatterns
* How AntiPatterns help us to keep code in good shape and secure
* Java AntiPatterns
* Measure time intervals using System.currentTimeMillis()
* Null returned from a method returning a collection or an array
* Unneccessary thread safety of StringBuffer
* Comparing URLs with ‘URL.equals()’
* ‘equals()’ and ‘hashCode()’ are context-sensitive
* Not taking advantage of ‘toString()’
* Instatiation of immutable objects
* Unbuffered I/O

**Day 4**

## Java Anti Patterns? (Contd.)

* ‘equals()’ does not check for null argument
* ‘compareTo()’ is incompatible with ‘equals()’
* LinkedLists as arrays
* Accessing the Map values using keySet iterator
* Synchronized collections everywhere
* ‘equals()’ is overridden while ‘hashCode()’ is not
* String concatenation
* Lost StringBuffer performance
* Testing for string equality
* Converting numbers to Strings
* Parsing and converting numbers
* Not taking advantage of immutable objects
* Platform dependent filenames
* Unbuffered streams
* Unbuffered operations on InputStreamReader, OutputStreamWriter
* Using PrintWriter for file I/O
* Infinite heap
* Infinite time
* Assuming a cheap timer call
* Catch all: I don't know the right runtime exception
* Exceptions are annoying
* Re-wrapping RuntimeException
* Not properly propagating the exception
* Silly exception messages
* Catching to log
* Incomplete exception handling
* The exception that never happens
* The transient trap
* Overkill initialization
* Chosing the wrong class loader
* Poor use of reflection

**Day 5**

## Java Anti Patterns? (Contd.)

* Synchronization overkill
* Wrong list type
* The HashMap size trap
* Hashtable, HashMap and HashSet are overrated
* Lists are overrated
* Object arrays are soooo flexible
* Premature object decomposition
* Modifying setters
* Unnecessary Calendar
* Relying on the default TimeZone
* Having a global Configuration/Parameters/Constants class
* Using == with float or double
* Storing money in floating point variables
* Abusing finalize()
* Spawning thread from static initializers
* Canceled timer tasks that keep state
* Holding strong references to ClassLoaders and unflushable caches
* Nested synchronized statements
* Doing random file access via RandomAccessFile

**Day 6**

## Microservices Design Pattern

* Explaining microservices patterns
* Explaining how microservices architecture works
* Explaining when to use microservices architecture
* Advantages and drawbacks of a microservices-based application
* Microservices architecture patterns
* Aggregator Microservice pattern
* Chained Microservice Pattern
* Shared Data Microservice Pattern
* Asynchronous Messaging Microservice Pattern

## TDD

* The TDD mindset
* Why is TDD important?
* Red/green/refactor
* Behavior driven development
* How TDD helps you design software
* Why testing first is better than testing after
* Testing frameworks (e.g. JUnit)

**Day 7**

## Writing testable code

* How to write tests that run fast and are easy to maintain
* What makes code difficult or impossible to test
* The dependency inversion principle

## JUnit

* Introduction to Programmer Testing
* Getting started with JUnit
* Running JUnit
* Core JUnit Classes and Interfaces
* Elementary Tests like
  + Test a method
  + Test a constructor
  + Test getters and setters
  + Test an Interface
  + Test a Java Bean
  + Test throwing the exception
* Build Test from the command line
* Organizing and building JUnit Tests into TestSuits
* Managing TestSuits

## Advanced TDD

* Single responsibility principle, and how TDD helps you design small classes
* How to use mocking frameworks (e.g. Mockito)
* How to effectively translate technical specs into tests
* How to test legacy code
* Other TDD ninja tricks that we’ve learned over the years

## Mockito

* Mocking Concepts
* Mockito Overview
* Mockito Demo
* Creating Mock Instances
* Stubbing Method Calls
* Verifications
* Summary

**Day 8**

## Mockito Feature Deep-Dive

* Argument Matching
* Stubbing Consecutive Calls
* Demo - Stubbing Consecutive Calls
* Verification Order
* Capturing Arguments
* Demo - Capturing Arguments
* Partial Mocks
* When Mockito is not Enough - PowerMock
* Mocking Static Method Invocations
* Replacing Object Instantiation
* Stubbing Final & Private Methods
* Whitebox Test Utility Class

## BDD

* Introduction to Behavior Driven Development
* Introduction to Specification by Example and Behavior Driven Development
* User Stories
* Discovering Users
* Discovering User Stories
* Planning an Iteration Signing off
* Write plain-text Features and Scenarios
* Write Step Definitions
* Scenarios and Steps
* Example Workshop
* Collaborative specifications in practice
* Introduction to BDD scenarios with Cucumber
* Converting examples to BDD scenarios
* Best practices for BDD scenarios
* Stakeholder analysis and focusing on software that matters

## BDD Frameworks ( Cucumber, JBehave, JDave, Easyb)

* BDD using Cucumber
* Cucumber automation for developers and testers
* Automating basic scenarios
* Integrating into a development process
* Automating complex scenarios

## Conclusion