**Advanced Java For Developers**

### Training Process:

**70% hands on and 30% theory**

### Evaluation Process

**Pre test and Post test MCQ**

### Business Case Evaluation Duration: 5 Days

Pre Requisites: Knowledge and work experience on Java 6 or 7 is must

Day 1

# Brief overview on Java 8 Lambdas and Streams

## Introduction to Java 8

* + - 1.1.1 Evolution of Java versions
    - 1.1.2 Key features of Java 8

## Introduction to Lambdas

* + - 1.2.1 Understanding functional interfaces
    - 1.2.2 Syntax and structure of lambda expressions
    - 1.2.3 Use cases and benefits of lambdas

## Java Streams

* + - 1.3.1 Stream API basics
    - 1.3.2 Working with intermediate and terminal operations
    - 1.3.3 Parallel streams for concurrent processing

## Functional Interfaces

* + - 1.4.1 Overview of built-in functional interfaces
    - 1.4.2 Creating custom functional interfaces

## Method References

* + - 1.5.1 Types of method references
    - 1.5.2 Simplifying lambda expressions with method references

## Stream Collectors

* + - 1.6.1 Collecting data with predefined collectors
    - 1.6.2 Creating custom collectors

## Exception Handling in Lambdas and Streams

* + - 1.7.1 Handling checked and unchecked exceptions
    - 1.7.2 Best practices for exception handling in streams

# Java 11 and 17 Features

## Overview of Java 11 and 17 features

* + - 2.1.1 Release highlights and significance
    - 2.1.2 Long-term support (LTS) versions

## Local-variable type inference (Java 10 and above)

* + - 2.2.1 Introduction to var keyword
    - 2.2.2 Use cases and benefits
    - 2.2.3 Best practices and considerations

## New APIs and enhancements

* + 1. HTTP Client API (Java 11)
* 2.3.1.1 Basics of the HTTP Client API
* 2.3.1.2 Making asynchronous requests
* 2.3.1.3 Handling responses and errors
  + 1. Pattern Matching (Java 16 and above)
* 2.3.2.1 Introduction to pattern matching
* 2.3.2.2 Use cases in switch expressions and instance of checks
  + 1. Records (Java 16 and above)
* 2.3.3.1 Creating and using record classes
* 2.3.3.2 Immutable data with records
* 2.3.3.3 Record components and automatic methods
  + 1. Sealed Classes (Java 17)
* 2.3.4.1 Overview of sealed classes
* 2.3.4.2 Declaring sealed and non-sealed subclasses
* 2.3.4.3 Pattern matching with sealed classes
  + 1. Deprecation and Removals
* 2.3.5.1 Managing deprecated features
* 2.3.5.2 Understanding removals and migration strategies
  + 1. Performance Improvements
* 2.3.6.1 JIT compiler enhancements
* 2.3.6.2 Garbage collection improvements
  + 1. Other Noteworthy Features
* 2.3.7.1 New tools and utilities
* 2.3.7.2 Security enhancements and updates

Day 2

# Spring Boot 3

## Introduction to Spring Boot

* + - 3.1.1 Spring Boot's role in modern Java development
    - 3.1.2 Key principles and philosophy of Spring Boot
    - 3.1.3 Advantages of using Spring Boot for microservices

## Setting up a Spring Boot project

* + - 3.2.1 Choosing a build tool (Maven or Gradle)
    - 3.2.2 Project structure and conventions
    - 3.2.3 Configuration options for Spring Boot projects

## Creating a simple Spring Boot application

* + - 3.3.1 Bootstrap a Spring Boot application
    - 3.3.2 Defining application properties and profiles
    - 3.3.3 Building and running the application

## Building REST APIs using Spring Boot

* + - 3.4.1 REST architecture principles
      * 3.4.1.1 Understanding RESTful design
      * 3.4.1.2 REST constraints and best practices
    - 3.4.2 Creating RESTful endpoints with Spring Boot
      * 3.4.2.1 Mapping HTTP methods to controller methods
      * 3.4.2.2 Path variables and request parameters
    - 3.4.3 Request and response handling
      * 3.4.3.1 Request and response bodies
      * 3.4.3.2 Content negotiation and media types
    - 3.4.4 Handling different HTTP methods
      * 3.4.4.1 CRUD operations with HTTP methods
      * 3.4.4.2 Idempotence and safety considerations

## Spring Data JPA and Spring Data MongoDB

* + - 3.5.1 Introduction to Spring Data
      * 3.5.1.1 Repository pattern and data access
      * 3.5.1.2 Common features of Spring Data modules
    - 3.5.2 Working with JPA for relational databases
      * 3.5.2.1 Entity modeling and relationships
      * 3.5.2.2 Query methods and custom queries
    - 3.5.3 Using Spring Data MongoDB for NoSQL databases
      * 3.5.3.1 Document modeling and indexing
      * 3.5.3.2 Querying with MongoDB queries and criteria

Day 3

## Testing Spring Boot Application

* + - * 3.6.1 Testing Service Layer
      * 3.6.1.1 Unit testing services and business logic
      * 3.6.1.2 Mocking dependencies and using test doubles
    - 3.6.2 Testing Controllers
      * 3.6.2.1 MockMvc and testing REST controllers
      * 3.6.2.2 Handling request and response in controller tests
    - 3.6.3 Integration Testing
      * 3.6.3.1 Testing database interactions in integration tests
      * 3.6.3.2 Spring Boot testing annotations and configurations

# Introduction To Reactive Programming

## Reactive Architecture - Pros and Cons

* + - 4.1.1 Understanding reactive programming paradigm
    - 4.1.2 Benefits of reactive architecture
      * 4.1.2.1 Scalability and responsiveness
      * 4.1.2.2 Efficient resource utilization
    - 4.1.3 Challenges and considerations
      * 4.1.3.1 Learning curve and mindset shift
      * 4.1.3.2 Debugging and tracing in reactive systems

## Reactive Streams

* + - 4.2.1 Introduction to Reactive Streams API
    - 4.2.2 Publisher, Subscriber, and Subscription
    - 4.2.3 Backpressure and handling data flow
      * 4.2.3.1 Understanding backpressure
      * 4.2.3.2 Strategies for handling backpressure

## Mono, Flux - Introduction

* + - 4.3.1 Mono and Flux as reactive types
      * 4.3.1.1 Understanding Mono for handling a single value
      * 4.3.1.2 Exploring Flux for handling multiple values
    - 4.3.2 Operations on Mono and Flux
      * 4.3.2.1 Transformations with map, flatMap, and filter
      * 4.3.2.2 Aggregating and combining data with reduce and merge

## Reactive Operators (map, filter, flat map, transform)

* + - 4.4.1 Overview of common reactive operators
    - 4.4.2 Using map for data transformation
    - 4.4.3 Filtering data with filter and distinct
    - 4.4.4 Flattening data structures with flatMap
    - 4.4.5 Transformations and conversions with transform

## Get, Put, Post, Delete Operations in Reactive World

* + - 4.5.1 Mapping HTTP methods to reactive operations

Day 4

Introduction to Microservices

## Microservices architecture and benefits

* + - 5.1.1 Principles of microservices
    - 5.1.2 Benefits of microservices over monolithic architecture
    - 5.1.3 Key characteristics and design considerations

## Decomposing monolithic applications

* + - 5.2.1 Identifying and extracting microservices
    - 5.2.2 Challenges and strategies for decomposition
    - 5.2.3 Refactoring techniques for migrating to microservices

## Microservices communication and challenges

* + - 5.3.1 Inter-service communication patterns
    - 5.3.2 Synchronous and asynchronous communication
    - 5.3.3 Challenges in distributed systems and their solutions

# Various Microservices Design Patterns

## Service discovery and registration

* + - 6.1.1 Overview of service discovery
    - 6.1.2 Implementing service registration and discovery
    - 6.1.3 Tools like Eureka for service discovery

## API gateway pattern

* + - 6.2.1 Introduction to API gateways
    - 6.2.2 Benefits of using an API gateway
    - 6.2.3 Implementing API gateway with Spring Cloud Gateway

## Circuit breaker pattern

* + - 6.3.1 Understanding the circuit breaker pattern
    - 6.3.2 Implementing fault tolerance with Hystrix
    - 6.3.3 Monitoring and managing circuit breakers

# Microservices Communication – Synchronous and Asynchronous

## Synchronous communication using REST

* + - 7.1.1 RESTful principles in microservices
    - 7.1.2 Best practices for designing RESTful APIs
    - 7.1.3 Versioning and documentation strategies

## Asynchronous communication using messaging systems

* + - 7.2.1 Messaging patterns in microservices
    - 7.2.2 Introduction to message brokers (e.g., RabbitMQ, Kafka)
    - 7.2.3 Implementing asynchronous communication with messaging

# Service Discovery and Resilience

## Implementing service discovery using Eureka

* + - 8.1.1 Setting up Eureka server and clients
    - 8.1.2 Dynamic registration and discovery of services
    - 8.1.3 Integrating Eureka with Spring Boot applications

## Load balancing and fault tolerance

* + - 8.2.1 Strategies for load balancing in microservices
    - 8.2.2 Circuit breakers and fallback mechanisms
    - 8.2.3 Configuring load balancing with Ribbon

## Resilience patterns for microservices

* + - 8.3.1 Overview of resilience patterns
    - 8.3.2 Retry mechanisms
    - 8.3.3 Bulkheads and isolation strategies

# Configuration Management

* + - 9.1 Externalizing configuration in microservices
    - 9.2 Centralized configuration management tools
    - 9.3 Dynamic configuration updates and reloading

# Spring Boot Actuator

* + - 10.1 Monitoring and management of Spring Boot applications
    - 10.2 Exposing and customizing actuator endpoints

Day 5

# Docker and Containerization Intro

## Introduction to containerization

* + - 11.1.1 Container fundamentals and benefits
    - 11.1.2 Comparison with virtual machines
    - 11.1.3 Container orchestration (e.g., Kubernetes)

## Benefits of Docker

* + - 11.2.1 Lightweight and portable containers
    - 11.2.2 Rapid application deployment
    - 11.2.3 Consistent development and production environments

## Docker architecture and components

* + - 11.3.1 Docker Engine and its components
    - 11.3.2 Docker images, containers, and registries
    - 11.3.3 Networking and storage in Docker

# Running Spring Boot Applications as Docker Containers

## Dockerizing a Spring Boot application

* + - 12.1.1 Creating Dockerfile for Spring Boot
    - 12.1.2 Packaging and optimizing the Docker image
    - 12.1.3 Docker image best practices

## Creating Docker images

* + - 12.2.1 Building Docker images locally
    - 12.2.2 Pushing and pulling images from Docker Hub
    - 12.2.3 Versioning and tagging Docker images

## Running containers and managing images

* + - 12.3.1 Running Spring Boot applications as Docker containers
    - 12.3.2 Managing container lifecycle
    - 12.3.3 Monitoring and troubleshooting Docker containers

# Docker with Microservices

## Containerizing microservices

* + - 13.1.1 Strategies for containerizing microservices
    - 13.1.2 Multi-container applications and microservices architecture
    - 13.1.3 Tools for managing containerized microservices (e.g., Docker Compose)

## Docker Compose for managing multi-container applications

* + - 13.2.1 Defining multi-container applications with Docker Compose
    - 13.2.2 Orchestration and coordination of microservices
    - 13.2.3 Environment variables and service dependencies

## Networking and communication between containers

* + - 13.3.1 Container networking options
    - 13.3.2 Communication patterns between microservices
    - 13.3.3 Challenges and solutions in container communication

# Cloud Native Application

* + - 14.1 Principles of cloud-native application development
    - 14.2 Microservices as a foundation for cloud-native architecture
    - 14.3 Container orchestration and scaling in the cloud

# Using “Twelve-Factor App”

* + - 15.1 Twelve-Factor App methodology
    - 15.2 Adhering to twelve-factor principles in microservices
    - 15.3 Benefits of using the twelve-factor approach

# Design Patterns in Microservices

## Common design patterns in microservices

* + - 16.1.1 Event sourcing and CQRS
    - 16.1.2 Saga pattern for distributed transactions
    - 16.1.3 API Composition and Aggregator pattern