springpeople

Advanced Java for Developers

Duration: 5 Days

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

Lab set up:

Java 21
Intellij Ultimate
Mysql Community Server 8.0
Mysql Workbench
Docker Desktop with Kubernetes
RAM 32 GB

Day 1

1. Brief overview on Java 8 Lambdas and Streams

1.1 Introduction to Java 8

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

1.2 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

1.3 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

1.4 Functional Interfaces

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

1.5 Method References

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

1.6 Stream Collectors

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

1.7 Exception Handling in Lambdas and Streams

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

2. Java 11 and 17 Features

2.1 Overview of Java 11 and 17 features

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

2.2 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

2.3 New APIs and enhancements

2.3.1 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

2.3.2 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

2.3.3 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

Day 2

2.3.4 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

2.3.5 Deprecation and Removals

- 2.3.5.1 Managing deprecated features
- 2.3.5.2 Understanding removals and migration strategies

2.3.6 Performance Improvements

- 2.3.6.1 JIT compiler enhancements
- 2.3.6.2 Garbage collection improvements

2.3.7 Other Noteworthy Features

- 2.3.7.1 New tools and utilities
- 2.3.7.2 Security enhancements and updates

3. Spring Boot 3

3.1 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

3.2 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

3.3 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

3.4 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

3.5 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

Introduction to Microservices

3.6 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

3.7 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

3.8 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

4. Various Microservices Design Patterns

4.1 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

4.2 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

4.3 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

Day 4

5. Microservices Communication – Synchronous and Asynchronous

5.1 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

5.2 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

6. Service Discovery and Resilience

6.1 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

6.2 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

6.3 Resilience patterns for microservices

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

7. Configuration Management

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

8. Spring Boot Actuator

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

Day 5

9. Docker and Containerization Intro

9.1 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)

9.2 Benefits of Docker

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

9.3 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

10. Running Spring Boot Applications as Docker Containers

10.1 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

10.2 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

10.3 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

11. Docker with Microservices

11.1 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)

11.2 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

11.3 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

12. 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

13. 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

14. Design Patterns in Microservices

14.1 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