SPRING EXPLAINED

ZERO TO HERO

Sadiq Tech Solutions

Spring Framework

Table of Content

Chapter 1: Getting Started with Spring Framework

Section 1.1: Setup (XML Configuration)

Introduction

Spring Framework is one of the most popular frameworks for building Java applications. It provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so developers can focus on application logic. In this chapter, we'll cover the basics of getting started with Spring, including setting up your environment and understanding the core features of the framework.

Setup (XML Configuration)

To start with Spring, we need to set up a basic project structure and configure Spring using XML configuration. Here are the steps:

1. **Create a Maven Project:**
   First, create a Maven project. Maven is a build automation tool used for managing project dependencies and building the project.

```
<project xmlns="<http://maven.apache.org/POM/4.0.0>"
         xmlns:xsi="<http://www.w3.org/2001/XMLSchema-in
stance>"
         xsi:schemaLocation="<http://maven.apache.org/PO
M/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd>">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.9</version>
        </dependency>
```

2. **Create Spring Configuration File:**
   Create an XML configuration file named `applicationContext.xml` in the `src/main/resources` directory.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="<http://www.springframework.org/schema/beans>"
       xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"
       xsi:schemaLocation="<http://www.springframework.org/schema/beans>

    <!-- Bean Definitions -->
    <bean id="myBean" class="com.example.MyBean">
/>
    </bean>

</beans>
```

3. **Create a Simple Bean:**
   Create a Java class named `MyBean` in the `src/main/java/com/example` directory.

```
package com.example;

public class MyBean {
    private String property;

    public void setProperty(String property) {
        this.property = property;


    public void printProperty() {
        System.out.println("Property Value: " + property);
    }
}
```

4. **Initialize the Spring Context:**
   Create a main class to load the Spring context and retrieve the bean.

```
package com.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlA
  ...

public class SpringExample {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApp
licationContext("applicationContext.xml");

        MyBean myBean = (MyBean) context.getBean("myBea
n");

        myBean.printProperty();
```

Running the Application

1. **Compile the Project:**
   Run the following Maven command to compile the project:

```
mvn compile
```

2. **Run the Application:**
   Run the main class:

```
mvn exec:java -Dexec.mainClass="com.example.SpringExampl
e"
```

You should see the following output:

```
Property Value: Hello, Spring!
```

This simple setup demonstrates how to configure Spring using XML. In the next sections, we will explore more about Spring features and how to leverage them

in your applications.

Section 1.2: Showcasing Core Spring Features by Example

Introduction

Spring Framework offers a multitude of features that facilitate the development of robust, scalable, and maintainable applications. In this section, we will showcase some of the core features of Spring by example.

Dependency Injection (DI)

Dependency Injection is a fundamental concept in Spring, which allows the creation of dependent objects outside of a class and provides those objects to a class in different ways.

Example:

```
<!-- applicationContext.xml -->
<bean id="dependencyBean" class="com.example.DependencyBean">
    <property name="message" value="Injected Dependency!" />
`

<bean id="dependentBean" class="com.example.DependentBean">
    <property name="dependency" ref="dependencyBean" />
```

```
package com.example;

public class DependencyBean {
    private String message;

    public void setMessage(String message) {
        this.message = message;
    `

    public String getMessage() {
        return message;
```

```
    }
}
```

```
package com.example;

public class DependentBean {

    private DependencyBean dependency;

    public void setDependency(DependencyBean dependency) {
        this.dependency = dependency;
    }

    public void showDependencyMessage() {
        System.out.println(dependency.getMessage());
    }
}
```

```
package com.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlAppl
ationContext;

public class SpringExample {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplic
ationContext("applicationContext.xml");

        DependentBean dependentBean = (DependentBean) conte
xt.getBean("dependentBean");

        dependentBean.showDependencyMessage();
```

Output:

```
Injected Dependency!
```

Aspect-Oriented Programming (AOP)

Spring AOP allows for the modularization of concerns such as transaction management, logging, or security.

Example:

```
<!-- applicationContext.xml -->
<aop:config>
    <aop:aspect ref="myAspect">
        <aop:pointcut id="myPointcut" expression="execution
(* com.example.MyService.*(..))" />
        <aop:before method="beforeAdvice" pointcut-ref="myP
ointcut" />

<bean id="myAspect" class="com.example.MyAspect" />
<bean id="myService" class="com.example.MyService" />
```

```
package com.example;

public class MyService {
    public void doSomething() {
        System.out.println("Doing something in MyService");
    }
```

```
package com.example;

public class MyAspect {
    public void beforeAdvice() {
        System.out.println("Before Advice: Method is about
to be called.");
    }
```

```
package com.example;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlAppl

public class SpringExample {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplic
ationContext("applicationContext.xml");

        MyService myService = (MyService) context.getBean
("myService");

        myService.doSomething();
```

Output:

```
Before Advice: Method is about to be called.

Doing something in MyService
```

These examples highlight some of the core features of Spring, including Dependency Injection and Aspect-Oriented Programming. As you continue through this book, you'll discover many more capabilities and best practices for using Spring Framework in your projects.

Section 1.3: What is Spring Framework, Why Should We Go for It?

Introduction

Spring Framework is a comprehensive framework for enterprise Java development. It provides a wide range of functionalities that can be used to develop any kind of Java application. Its popularity is due to its simplicity, power, and ease of use.

What is Spring Framework?

Spring Framework is an open-source framework created to address the complexity of enterprise application development. It offers a comprehensive programming and configuration model for modern Java-based enterprise applications.

Key Features:

- **Inversion of Control (IoC):** Manages the dependencies of your application.
- **Aspect-Oriented Programming (AOP):** Modularizes cross-cutting concerns.
- **Data Access Framework:** Simplifies database interaction.
- **Transaction Management:** Manages transactions declaratively.
- **MVC Framework:** Simplifies the development of web applications.
- **Batch Processing:** Supports the creation of robust batch processing applications.
- **Integration:** Integrates with various enterprise services.

Why Should We Go for Spring Framework?

1. **Modularity:**
   Spring promotes modularity by breaking down the application into modules. This separation of concerns makes the code easier to manage and understand.
2. **Dependency Injection:**
   Spring's IoC container manages the components' lifecycle and configuration, promoting loose coupling and better testability.
3. **Aspect-Oriented Programming:**
   AOP helps separate cross-cutting concerns, such as logging and transaction management, from the business logic.
4. **Comprehensive Ecosystem:**
   Spring provides a rich ecosystem with various projects

Spring Boot

1. Spring Boot Introduction

- What is Spring Boot?
- Benefits of Using Spring Boot
- Comparison with Spring Framework
- Key Features and Concepts

2. Spring Boot - Bootstrapping

- Setting Up Your Development Environment
- Creating a Spring Boot Project
- Using Spring Initializr
- Running Your First Spring Boot Application

3. Spring Boot - Tomcat Development

- Embedded vs External Tomcat
- Configuring Embedded Tomcat
- Customizing Tomcat Settings
- Deploying Spring Boot Application to External Tomcat

4. Spring Boot - Build System

- Maven Configuration
- Gradle Configuration
- Dependency Management
- Building and Packaging Applications

5. Spring Boot - Code Structure

- Project Directory Layout

- Package Naming Conventions
- Organizing Your Codebase
- Best Practices

6. Spring Boot - Spring Beans and Dependency Injection

- Overview of Spring Beans
- Defining and Managing Beans
- Constructor and Setter Injection
- @Component, @Service, @Repository Annotations

7. Spring Boot - Runners

- CommandLineRunner Interface
- ApplicationRunner Interface
- Executing Code on Startup
- Use Cases and Examples

8. Spring Boot - Application Properties

- Configuration Files Overview
- application.properties vs application.yml
- Common Configuration Properties
- Externalizing Configuration

9. Spring Boot - Logging

- Logging Overview
- Configuring Logback
- Customizing Log Output
- Logging Best Practices

10. Spring Boot - Building Restful Web Service

- Introduction to RESTful Web Services
- Creating REST Controllers
- Handling HTTP Methods (GET, POST, PUT, DELETE)
- Consuming JSON and XML

11. Spring Boot - Exception Handling

- Global Exception Handling with @ControllerAdvice
- Custom Exception Handling
- Returning Custom Error Responses
- Best Practices for Error Handling

12. Spring Boot - Interceptor

- Introduction to Interceptors
- Creating and Configuring Interceptors
- Pre-processing and Post-processing Requests
- Use Cases and Examples

13. Spring Boot - Servlet Filter

- Overview of Servlet Filters
- Creating Custom Filters
- Filter Registration
- Common Use Cases

14. Spring Boot - Rest Template

- Introduction to RestTemplate
- Making HTTP Requests
- Handling Responses
- Advanced Features and Configurations

15. Spring Boot - File Handling

- File Upload and Download
- Configuring Multipart File Support
- Handling Large Files
- File Storage Best Practices

16. Spring Boot - Service Components

- Creating Service Classes
- Service Layer Best Practices
- Transaction Management
- Integrating with Repositories

17. Spring Boot - Consuming Restful Web Services

- Using RestTemplate
- Error Handling
- Authenticating API Requests
- Consuming External APIs

18. Spring Boot - CORS Support

- Introduction to CORS
- Configuring CORS in Spring Boot
- Global and Local CORS Configurations
- Common CORS Issues and Solutions

19. Spring Boot - Internationalization

- Introduction to Internationalization (i18n)
- Configuring Message Sources
- Localizing Messages

- Switching Locales

20. Spring Boot - Scheduling

- Introduction to Scheduling
- @Scheduled Annotation
- Configuring Scheduled Tasks
- Advanced Scheduling Techniques

21. Spring Boot - Enabling HTTPS

- Configuring SSL/TLS
- Generating and Using Keystores
- Redirecting HTTP to HTTPS
- Best Practices for Secure Communication

22. Spring Boot - Eureka Server

- Introduction to Eureka
- Setting Up Eureka Server
- Configuring Eureka Properties
- Running and Testing Eureka Server

23. Spring Boot - Service Registration With Eureka

- Registering Services with Eureka
- Configuring Service Discovery
- Load Balancing with Eureka
- Monitoring Eureka Services

24. Spring Boot - Cloud Configuration Server

- Introduction to Spring Cloud Config
- Setting Up Configuration Server

- Managing Configuration Properties
- Securing Configuration Server

25. Spring Boot - Cloud Configuration Client

- Configuring Clients to Use Config Server
- Refreshing Configuration Properties
- Handling Configuration Changes
- Best Practices for Configuration Management

26. Spring Boot - Actuator

- Introduction to Spring Boot Actuator
- Monitoring and Management Endpoints
- Customizing Actuator Endpoints
- Securing Actuator Endpoints

27. Spring Boot - Admin Server

- Introduction to Spring Boot Admin
- Setting Up Admin Server
- Monitoring and Managing Applications
- Customizing Admin Server

28. Spring Boot - Admin Client

- Registering Clients with Admin Server
- Monitoring Client Applications
- Configuring Client Properties
- Advanced Client Features

29. Spring Boot - Enable Swagger2

- Introduction to Swagger

- Setting Up Swagger2
- Generating API Documentation
- Customizing Swagger UI

30. Spring Boot - Creating Docker Image

- Introduction to Docker
- Creating Dockerfile for Spring Boot Application
- Building Docker Images
- Running Spring Boot Application in Docker

31. Spring Boot - Tracing Micro Service Logs

- Introduction to Distributed Tracing
- Using Sleuth and Zipkin
- Configuring Trace and Span IDs
- Visualizing Traces

32. Spring Boot - Flying Database

- Introduction to Database Migration
- Using Flyway for Database Migrations
- Configuring Flyway
- Managing Database Versions

33. Spring Boot - Web Socket

- Introduction to WebSockets
- Setting Up WebSocket in Spring Boot
- Creating WebSocket Endpoints
- Handling WebSocket Messages

34. Spring Boot - Batch Service

- Introduction to Spring Batch
- Configuring Batch Jobs
- Creating Batch Processes
- Monitoring and Managing Batch Jobs

35. Spring Boot - Spring for Apache Kafka

- Introduction to Apache Kafka
- Setting Up Kafka with Spring Boot
- Producing and Consuming Messages
- Advanced Kafka Configurations

36. Spring Boot - Database Handling

- Configuring DataSources
- Using Spring Data JPA
- Database Transactions
- Performance Tuning

37. Spring Boot - Securing Web Application

- Introduction to Spring Security
- Configuring Authentication and Authorization
- Securing Web Endpoints
- OAuth2 and JWT Integration

38. Spring Boot - OAuth2 with JWT

- Introduction to OAuth2
- Setting Up OAuth2 Authorization Server
- Implementing OAuth2 Resource Server
- Using JWT for Token Management

Chapter 1: Spring Boot Introduction

1.1 What is Spring Boot?

Spring Boot is an open-source, Java-based framework used to create stand-alone, production-grade Spring-based applications with minimal effort. It is built on top of the Spring Framework and provides a range of features and tools to simplify the process of developing and deploying applications.

1.2 Benefits of Using Spring Boot

- **Convention Over Configuration:** Reduces the need for manual configuration by providing sensible defaults.
- **Embedded Servers:** Comes with embedded servers like Tomcat, Jetty, and Undertow, allowing you to run your application as a standalone Java application.
- **Microservices Ready:** Facilitates the development of microservices with features like Spring Cloud.
- **Production-Ready Features:** Includes features like health checks, metrics, and externalized configuration.
- **Rapid Development:** Provides numerous starter templates to quickly set up projects with necessary dependencies.
- **Spring Ecosystem Integration:** Seamlessly integrates with other Spring projects like Spring Data, Spring Security, and Spring Batch.

1.3 Comparison with Spring Framework

Feature	Spring Framework	Spring Boot
Setup and Configuration	Requires extensive configuration (XML/Java)	Minimal configuration with sensible defaults
Embedded Server	Requires external server setup	Comes with embedded server
Production-Ready	Requires additional setup	Built-in production-ready features

Microservices Support	Requires additional setup	Out-of-the-box support with Spring Cloud
Dependency Management	Manual dependency management	Starter templates for dependency management

14 Key Features and Concepts

- **Auto-Configuration:** Automatically configures Spring application based on the dependencies present in the classpath.
- **Spring Boot Starters:** A set of convenient dependency descriptors you can include in your application. For example, `spring-boot-starter-web` for web applications.
- **Spring Boot CLI:** Command-line interface to run and test Spring Boot applications.
- **Spring Initializr:** Web-based tool to generate Spring Boot project structure with desired dependencies.
- **Actuator:** Provides production-ready features like monitoring and management over HTTP or JMX.
- **Spring Boot DevTools:** Enhances the development experience by enabling features like automatic restarts and live reload.

15 Example: Creating a Simple Spring Boot Application Step 1: Setting Up Your Development Environment

Ensure you have JDK 8 or higher installed, and an IDE like IntelliJ IDEA, Eclipse, or STS (Spring Tool Suite).

Step 2: Creating a New Spring Boot Project

You can create a Spring Boot project using Spring Initializr (https://start.spring.io/).

1. Open Spring Initializr.

2. Select the following:
    - Project: Maven Project
    - Language: Java

- Spring Boot: 2.7.0 or latest stable version
- Project Metadata:
  - Group: com.example
  - Artifact: demo
  - Name: demo
  - Package Name: com.example.demo
  - Packaging: Jar
  - Java Version: 11 (or your preferred version)
- Dependencies: Select "Spring Web"

3. Click "Generate" to download the project.
4. Unzip the project and open it in your IDE.

Step 3: Writing Your First Spring Boot Application

In your `src/main/java/com/example/demo` directory, create a class named `DemoApplication.java`.

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootAppli
cation;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestControlle

@SpringBootApplication

public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);

    `
```

```
    @RestController
    class HelloController {
        @GetMapping("/")
        public String hello() {
            return "Hello, Spring Boot!";
}
```

Step 4: Running the Application

1. Run the application from your IDE by running the `DemoApplication` class.
2. Open your web browser and go to `http://localhost:8080/`. You should see the message "Hello, Spring Boot!".

1.6 Summary

In this chapter, we introduced Spring Boot, its benefits, key features, and how it simplifies the development of Spring-based applications. We also compared it with the traditional Spring Framework and provided a step-by-step guide to creating your first Spring Boot application. In the subsequent chapters, we will delve deeper into various features and functionalities of Spring Boot, providing detailed explanations, examples, and advanced usage scenarios.

Chapter 2: Spring Boot -Bootstrapping

2.1 Introduction

Bootstrapping a Spring Boot application involves setting up the project and configuring it to run with minimal effort. This chapter covers the various ways to bootstrap a Spring Boot application, including using Spring Initializr, Maven, and Gradle. We will also discuss the project structure and configuration files.

2.2 Setting Up Your Development Environment

Before you start bootstrapping your Spring Boot application, ensure you have the following installed:

- JDK 8 or higher
- An IDE like IntelliJ IDEA, Eclipse, or Spring Tool Suite (STS)
- Maven or Gradle (depending on your build system preference)

23 Creating a Spring Boot Project with Spring Initializr

Spring Initializr is a web-based tool to generate Spring Boot project structure with desired dependencies.

Step-by-Step Guide:

1. Open Spring Initializr.
2. Fill in the project metadata:
    - Project: Maven Project or Gradle Project
    - Language: Java
    - Spring Boot: 2.7.0 or the latest stable version
    - Project Metadata:
        - Group: com.example
        - Artifact: demo
        - Name: demo
        - Package Name: com.example.demo
        - Packaging: Jar
        - Java Version: 11 (or your preferred version)
    - Dependencies: Select the necessary dependencies like "Spring Web" for a web application.
3. Click "Generate" to download the project.
4. Unzip the downloaded project and open it in your IDE.

24 Creating a Spring Boot Project with Maven

You can also create a Spring Boot project manually using Maven.

Step-by-Step Guide:

1. Create a new directory for your project.
2. In the project directory, create a `pom.xml` file with the following content:

```
<project xmlns="<http://maven.apache.org/POM/4.0.0>"
    xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"
    xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <
http://maven.apache.org/xsd/maven-4.0.0.xsd>">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>demo</name>
    <description>Demo project for Spring Boot</description>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository --
>

    </parent>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
```

```
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifac
tId>
            </plugin>
```

1. Create a `src/main/java/com/example/demo/DemoApplication.java` file with the following content:

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootAppli


@SpringBootApplication

public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);

    }
```

1. Open a terminal and navigate to your project directory. Run the following command to build and run the application:

```
mvn spring-boot:run
```

25 Creating a Spring Boot Project with Gradle

Similarly, you can create a Spring Boot project using Gradle.

Step-by-Step Guide:

1. Create a new directory for your project.
2. In the project directory, create a `build.gradle` file with the following content:

```
plugins {
    id 'org.springframework.boot' version '2.7.0'
    id 'io.spring.dependency-management' version '1.0.11.RELE
ASE'

group = 'com.example'
version = '0.0.1-SNAPSHOT'

repositories {
    mavenCentral()
`

dependencies {
    implementation 'org.springframework.boot:spring-boot-star
ter-web'
    testImplementation 'org.springframework.boot:spring-boot-
starter-test'

test {
    useJUnitPlatform()
```

1. Create a `src/main/java/com/example/demo/DemoApplication.java` file with the following content:

```
package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootAppli

@SpringBootApplication

public class DemoApplication {

    public static void main(String[] args) {

        SpringApplication.run(DemoApplication.class, args);
```

1. Open a terminal and navigate to your project directory. Run the following command to build and run the application:

```
./gradlew bootRun
```

26 Project Directory Structure

A typical Spring Boot project has the following structure:

```
demo
|
|   ├── main
|   |   ├── java
|   |   |   └── com
|   |   |       └── example
|   |   |           └── demo
|   |   └── resources
|   |       └── application.properties
|   └── test
```

```
|       |   └── com
|       |       └── example
|       |           └── demo
|       |               └── DemoApplicationTests.java
```

27 Application Properties

Spring Boot uses a file named `application.properties` (or `application.yml` for YAML format) to store configuration properties.

Example: `application.properties`

```
server.port=8081
spring.application.name=demo
```

Example: `application.yml`

```
server:
  port: 8081

spring:
  application:

    name: demo
```

28 Running Your Spring Boot Application

You can run your Spring Boot application in several ways:

- **From your IDE:** Run the **Using Maven:** Run **Using Gradle:** Run
- `mvn spring-boot:run`
- `./gradlew bootRun`

```
│       └── resources
├── .gitignore
├── build.gradle (if using Gradle)
├── pom.xml (if using Maven)
```

class directly in your terminal.

Note:

- This is a **preview Spring & Spring Boot e-Book** containing **only few pages**.
- It is provided to help you understand **how the Spring & Spring Boot and is structured**.
- The **complete Spring & Spring Boot e-Book** includes detailed concepts, real-world examples, and career guidance.
- Buy now from our official website: https://topitcourses.com/spring-spring-boot/