Introduction to Spring and Spring Boot

1 Understanding the Spring Framework
2 Dependency Injection and Inversion of Control
3 Introduction to Spring Boot and Its Advantages
4 Building a Simple Application with Spring Boot

Understanding the Spring Framework

Learning Objectives:

- Understand what the Spring Framework is
- Identify the problems Spring Framework solves
- Learn the key features of Spring Framework

The Spring Framework: An Overview

The Spring Framework is a powerful, lightweight application development framework used for Enterprise Java (JEE). The core features of the Spring Framework can be used to develop any Java application, but there are extensions for building web applications on top of the Java EE platform.

What problems does Spring solve?

Prior to Spring, Java EE (Enterprise Edition) was the go-to platform for enterprise-grade application development. However, JEE was not without its challenges:

- Heavyweight components: JEE applications tended to have high memory consumption and slower startup times due to their "heavyweight" nature.
- Complexity: Development could be slow and complicated, requiring deep understanding of multiple JEE-specific APIs.
- Poor modularity: JEE applications were typically tightly coupled and hard to modularize, making them difficult to test and maintain.

- The Spring Framework emerged as a response to these challenges. It emphasized simplicity, testability, and loose coupling, enabling developers to create enterprise-grade applications that were easier to manage.

Key Features of the Spring Framework

- Dependency Injection (DI) and Inversion of Control (IoC): Spring uses DI and IoC to promote loose coupling and easy unit testing. These techniques make your components modular and easier to integrate.
- Aspect-Oriented Programming (AOP): Spring supports AOP to separate business logic from system services, reducing the coupling between business objects and making them easier to manage.

- Data Access / Integration: Spring provides a JDBC abstraction layer to remove boilerplate code. It also provides support for ORM frameworks like Hibernate.
- Transaction Management: Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (like JTA).
- Model-View-Controller (MVC) framework: Spring's web MVC framework is a well-designed web MVC framework, which provides a great alternative to web frameworks like Struts or other over-engineered or less popular web frameworks.
- Spring Boot: Spring Boot makes it easy to create stand-alone, production-grade Spring-based applications with minimal effort. It's a way to setup and launch Spring applications with ease.

Dependency Injection and Inversion of Control

Dependency Injection and Inversion of Control (IoC) are both design patterns that aim to achieve loose coupling between classes and their dependencies. Here's a simple explanation for each:

1. **Dependency Injection (DI)**: This is a design pattern in which an object (client) is provided with its dependencies (services) by an external entity (the injector). Instead of the client creating or locating the dependent objects, they are injected into it. DI makes the code more modular, testable and easier to manage.

Analogy: Imagine you're at a restaurant. Instead of cooking the meal yourself (which would be like your code creating its own dependencies), you are served the meal (dependencies are given to your code).

Example: Let's say you have a **Car** class and it needs an instance of **Engine** to function. Instead of creating an instance of `**Engine**` inside `**Car**` using `**new Engine()**`, you would pass an `**Engine**` object to the `**Car**`'s constructor or setter method.

2. **Inversion of Control (IoC):** This is a principle where the control flow of a program is inverted: instead of the client controlling the flow of control (what to do and when to do), this responsibility is given to an external entity. IoC is a broad principle and can be implemented in several ways, such as event-driven programming, dependency injection etc.

Analogy*:* Consider you're on a tour guided by a travel agency. Instead of deciding yourself where to go and what to do, you let the travel agency dictate the plan (control is inverted).

Example: In a typical desktop application, the user clicks on buttons, and the program reacts to these events. Here, the user controls the flow of the application. With IoC, the program could lay out a sequence of steps, and it's up to the user to provide input or response at each step.

Remember, Dependency Injection is a form of Inversion of Control - it's a method of implementing IoC. The key difference is while DI is about how one object acquires its dependencies, IoC is about who initiates action.

Introduction to Spring Boot and Its Advantages

Let's imagine for a moment that you're going on a trip. When you travel, you can choose to buy each component of your trip separately — your flight, hotel, rental car, travel insurance, meals, sightseeing tours, etc. This would give you maximum control over each aspect of your trip, but it would also require a lot of time, energy, and knowledge to coordinate and manage. Alternatively, you could buy a travel package that includes everything you need for your trip. This option is much simpler and more convenient, even though it gives you less control over the individual components.

Spring Boot is like the travel package of the Java world. In the "travel-your-own-way" model, you need to handle a lot of setup and configuration on your own. That's akin to the traditional Spring Framework. You have all the parts you need — dependency injection, MVC framework, data integration tools — but you have to assemble and configure them manually.

Enter Spring Boot, the travel package. Spring Boot takes an "opinionated" approach to setting up a Spring application, pre-configuring a lot of things for you based on what it thinks you'll need, much like a travel package that includes flights, hotel, and meals. And just as you can add on extras to a travel package (like sightseeing tours or a rental car), you can add on extras to a Spring Boot application as needed.

Advantages of Spring Boot

1. **Rapid prototyping**: With Spring Boot's automatic configuration, you can focus more on writing your business logic and less on configuration, making it faster to get a working prototype up and running.
2. **Stand-alone applications**: Spring Boot applications are stand-alone, meaning they can be started and run on their own without needing to be deployed on an external server, simplifying the deployment process.
3. **Embedded server**: Spring Boot comes with embedded servers (like Tomcat or Jetty), so you don't have to worry about setting up a server yourself.
4. **Simplified dependency management**: Spring Boot manages dependencies for you, reducing the chances of version conflicts.
5. **Actuator module:** This provides production-ready features like health checks and metrics gathering right out of the box.

In a nutshell, Spring Boot is like a well-planned, pre-packaged vacation. It might not give you the granular control over every little aspect that you'd have if you planned everything yourself, but it certainly makes the process a whole lot easier and quicker. For many developers, that's a trade-off they're more than willing to make!

Spring Boot Annotations

Annotations in Spring Boot are a lot like sticky notes on a project board in an office. They're like concise, targeted instructions that help you manage and organize your tasks (or in this case, your code).

Let's look at some key Spring Boot annotations and explain them with a bit of a twist:

1. **@SpringBootApplication**: This is the cornerstone of any Spring Boot application. It's a composite annotation, which means it actually combines three other annotations: `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`. It's like putting a "Head Office" sign on a building – it marks the main hub of your application where all the primary settings are established.

2. **@Configuration**: This annotation flags the class as a source of bean definitions. Consider it like the blueprint for a model home. It shows the layout and where all the rooms (beans) are supposed to go.

3. **@EnableAutoConfiguration**: Just like a smart home automatically adjusting the lighting or temperature, `@EnableAutoConfiguration` lets Spring Boot make educated guesses about what beans you'll need based on your classpath settings, other beans, and various property settings.

4. **@ComponentScan**: This annotation tells Spring where to look for other components, configurations, and services, allowing it to auto-detect these classes and wire up the beans automatically. Think of it like a scavenger hunt master, telling the participants (Spring) where to look for the items (beans).

5. **@RestController**: This is like the loudspeaker in a theme park, announcing information (data) to all who can hear (client applications). Classes annotated with `@RestController` can serve data directly to clients.

6. **@RequestMapping**:This annotation is like the map you find at the entrance of an amusement park, directing you to different attractions. It maps web requests to specific handler methods and classes.

7. **@Autowired**: It's like the manager at a construction site, making sure all the workers (beans) are in the right place at the right time. It allows Spring to resolve and inject collaborating beans into your beans.

8. **@Entity**: This annotation is like the ID card for an employee, marking a class as an entity so that Spring knows it's something it should manage and map to a database table.

9. **@Service**, **@Repository**, and **@Component**:** These are like nametags on employees, helping Spring know what roles different classes play in your application. These annotations

mark classes as service classes, data access objects, and general components respectively.

10. **@Value**: This annotation is like a personal assistant whispering important, often-changing information into a CEO's ear. It injects values from properties files or defaults into fields or method parameters.

11. **@Qualifier**: Imagine you have several delivery drivers and you need to specify which one to use for a particular task. The `@Qualifier` annotation does just that, helping to select the right bean when there are multiple candidates.

Annotations may seem like simple sticky notes, but they're powerful tools that help organize and manage your code in a Spring Boot application. Each annotation has its own specific role, making the whole development process much smoother and more streamlined.

Quiz:

- What is the Spring Framework?
- What are some of the problems the Spring Framework solves?
- List and describe the key features of the Spring Framework.

Further Reading:

- Spring in Action, Fifth Edition by Craig Walls
- Pro Spring 5 by Iuliana Cosmina, Rob Harrop, Chris Schaefer, and Clarence Ho

Learning Activities:

- Read the official Spring documentation
- Watch the video: Introduction to Spring Framework

Assignment: "Bootstrap Your Spring!" - Students create their own Spring Boot application.