02 - Spring Boot and Annotations

Table of Contents

Agenda

- Introduction to Spring Boot
- How Spring Boot is better than Spring
- Spring Initializer
- What are Spring Annotations?
- Hands-on - Important Annotations (@Component, @Autowired, @Qualifier, and @Bean)
- Bean Scopes with Spring Boot
- Hands-on configure Bean Scopes in Spring Boot

Introduction to Spring Boot

- What is Spring Boot?
  - An extension of the Spring Framework that simplifies the bootstrapping and development of new Spring applications.
  - Opinionated framework: Makes assumptions about how you want to build things, leading to faster setup and less configuration.
  - Embraces "convention over configuration": Provides sensible defaults that you can override if needed.
  - Goal: Rapid application development, easy deployment.
- Core Features
  - Starter Dependencies: Pre-packaged dependencies to easily add functionalities (e.g., Spring Web, Spring Data JPA).

- **Auto-configuration:** Automatically configures components based on your classpath and beans.
- **Embedded Servers:** Run your application as a standalone JAR file with embedded Tomcat, Jetty, or Undertow.
- **Actuator:** Provides production-ready features for monitoring and managing your application.
- **Command-Line Interface (CLI):** Optionally use the CLI to create, run, and test Spring Boot apps.

How Spring Boot is Better than Spring

Feature	Spring	Spring Boot
Configuration	Extensive XML configuration	Reduced XML, leverages auto-configuration and annotations
Boilerplate Code	More boilerplate for setup and configuration	Minimal boilerplate, convention-based approach
Deployment	Requires external servlet container	Embedded server support for standalone JAR deployments
Monitoring	Requires additional libraries	Actuator provides built-in monitoring and management endpoints
Development Time	Longer due to manual configuration	Significantly faster due to auto-configuration and sensible defaults

Spring Initialiser

- **What is Spring Initializr?**
  - A web-based tool and API for generating a basic Spring Boot project structure.
  - Select dependencies, project metadata, and build tool (Maven or Gradle).
  - Quickly get started without manually configuring everything.
- **Hands-on:**
  - Use the Spring Initializr website: https://start.spring.io/
  - Create a basic project with Spring Web dependency.
  - Explore the generated project structure.

What are Spring Annotations?

- **Annotations in Spring:**
  - Form of metadata that provides information to the Spring Framework.
  - Used to configure beans, dependency injection, web requests, and more.
  - Reduce or eliminate the need for XML configuration.

Hands-on - Important Annotations

- `@Component`:
  - Marks a class as a Spring bean.
  - Spring will automatically create and manage an instance/object of this class.
- `@Autowired`:
  - Used for dependency injection.
  - Spring automatically finds and injects a bean of the required type.
- `@Qualifier`:
  - Used with `@Autowired` to specify which bean to inject when there are multiple candidates of the same type.
- `@Bean`:
  - Used to define and configure a bean within a configuration class.
- Hands-on Activity:
  - Create a few example components using `@Component`.
  - Inject them into each other using `@Autowired`.
  - Explore scenarios where `@Qualifier` is needed.
  - Define custom beans using `@Bean`.

From the last project add below code files and add Annotations to them:

`CricketCoach.java`

```
@Component
public class CricketCoach implements Coach{

    @Autowired
    Greetings greetings;

    public CricketCoach(Greetings greetings) {
        this.greetings = greetings;
    }

    public String getWorkout(){
        return "Practice 50 Coverdrives";
    }
    public String greetings(){
        return greetings.sayHello();
    }
}
```

`TennisCoach.java`

```
@Component
public class TennisCoach implements Coach {
```

```
    Greetings greetings;
    @Autowired
    public void setGreetings(Greetings greetings) {
        this.greetings = greetings;
    }

    public String getWorkout(){
        return "Practice 50 back hands";
    }

    @Override
    public String greetings() {
        return greetings.sayHello();
    }
}
```

`Greetings.java`

```
@Component
public class Greetings {
    public String sayHello(){
        return "Hello! Welcome to Spring Boot Framework Training";
    }
}
```

`Coach.java`

```
public interface Coach {
    String getWorkout();
    String greetings();
}
```

Now to your main application file - `SpringBootDemoApplication.java`

- Add context
- `ApplicationContext context = SpringApplication.run(SpringBootDemo5JulyApplication.class, args);`

```
@SpringBootApplication
public class SpringBootDemoApplication {

    public static void main(String[] args) {
       ApplicationContext context =
SpringApplication.run(SpringDemo1002Application.class, args); // Start
Spring Boot app
```

```
        Coach coach = context.getBean("cricketCoach", Coach.class);
        System.out.println(coach.getWorkout());
        System.out.println(coach.greetings());

        Coach coach1 = context.getBean("tennisCoach", Coach.class);
        System.out.println(coach1.getWorkout());
        System.out.println(coach1.greetings());
    }
}
```

This way we are using annotations using SpringBoot-framework in place of creating an `applicationContext.xml` file separately in Spring-framework.

Bean Scopes with Spring Boot

- **Bean Scopes:** Define the lifecycle and visibility of a bean instance.
- **Common Scopes:**
  - **Singleton (Default):** Only one instance per Spring container.
  - **Prototype:** A new instance created each time it is requested.
  - **Request:** One instance per HTTP request.
  - **Session:** One instance per user session.

Hands-on - Configure Bean Scopes

- **Hands-on Activity:**
  - Create beans with different scopes.
  - Observe the behavior when requesting these beans multiple times.

  Lets add singleton and prototype scopes
- Note:*By default scope of a class in SpringBoot is* `singleton`

Add `@Scope` annotation
`CricketCoach.java`

```
@Component
@Scope("singleton")
public class CricketCoach implements Coach{

    @Autowired
    Greetings greetings;

    public CricketCoach(Greetings greetings) {
        this.greetings = greetings;
    }
```

```
    public String getWorkout(){
        return "Practice 50 Coverdrives";
    }
    public String greetings(){
        return greetings.sayHello();
    }
}
```

`TennisCoach.java`

```
@Component
@Scope("prototype")
public class TennisCoach implements Coach {


    Greetings greetings;
    @Autowired
    public void setGreetings(Greetings greetings) {
        this.greetings = greetings;
    }

    public String getWorkout(){
        return "Practice 50 back hands";
    }

    @Override
    public String greetings() {
        return greetings.sayHello();
    }
}
```

In `Main.java` make changes to observe

```
@SpringBootApplication
public class SpringBootDemoApplication {

    public static void main(String[] args) {

       ApplicationContext context =
SpringApplication.run(SpringBootDemoApplication.class, args);

       Coach coach = context.getBean("cricketCoach", Coach.class);
       System.out.println(coach.getWorkout());
        System.out.println(coach.greetings());

       Coach coach2 = context.getBean("cricketCoach", Coach.class);
```

```
        System.out.println(coach.hashCode());
        System.out.println(coach2.hashCode());

        Coach coach3 = context.getBean("tennisCoach", Coach.class);
         System.out.println(coach3.getWorkout());
         System.out.println(coach3.greetings());

        Coach coach4 = context.getBean("tennisCoach", Coach.class);
        System.out.println(coach3.hashCode());
        System.out.println(coach4.hashCode());
    }

}
```

By the output, you can observe the concept of the `singleton` and `prototype` Scope.
