Introduction to Spring Framework

Lahfari Bilal

December 22, 2024

1 Spring framework

1.1 What is Spring framework?

The **Spring Framework** is an open-source framework designed for building enterprise-level applications in Java. It simplifies the development process by providing a comprehensive programming and configuration model, making it easier to create robust and scalable applications.

Figure 1: Spring framework

1.2 Spring core concepts

1.2.1 Inversion of Control (IoC) concept

Inversion of Control is a design principle where the control over object creation and management is transferred from the application code to a container or framework—in this case, **the Spring IoC container**. This means that instead of objects creating their dependencies, the IoC container manages their lifecycle and dependencies. This leads to more modular and maintainable code, as components can be easily replaced or modified without affecting other parts of the application.

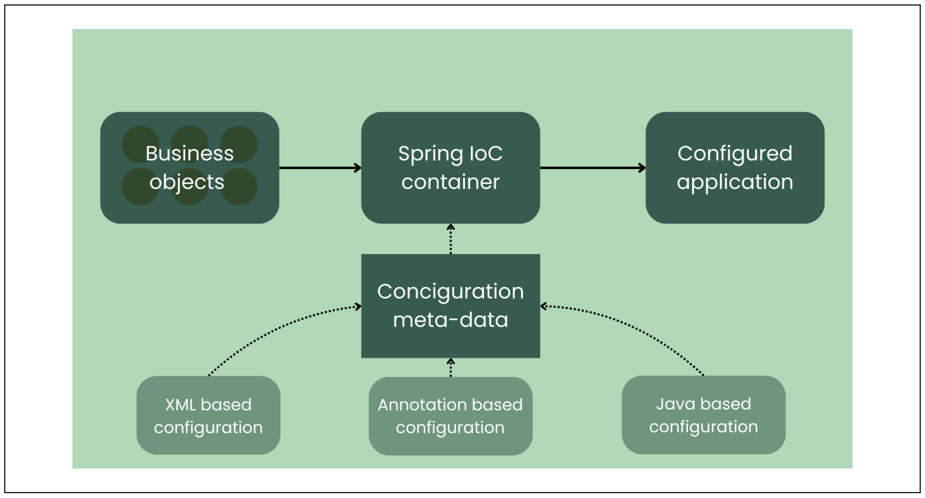


Figure 2: Spring IoC container concept

1.2.2 IoC Example

To demonstrate Inversion of Control, here's a Java code example showing a simple Spring configuration:

- **Business objects**: POJO stands for Plain Old Java Object. These are basic Java classes that represent the business logic or data for your application. They don't have any specific Spring code in them—they're just simple Java objects that define your data and operations.
- **Spring IoC container**: The IoC (Inversion of Control) Container is the central part of Spring. It's responsible for creating, managing, and assembling your objects (the POJOs) and their dependencies.
- **Configuration metadata**: The Configuration Metadata is where you define the settings or instructions for the IoC container. It tells the container what objects (beans) to create and how they should be wired together.
  - **XML-based**: You write XML files to specify which beans (objects) to create and how they're connected.
  - **Annotation based**: You use Java annotations directly in your code to tell Spring which classes to use and how to connect them.
  - **Java-based**: You use Java code to configure the beans and dependencies, typically through `@Configuration` classes.

- **Application**: Once the IoC container reads the configuration metadata and assembles the beans (POJOs), it results in a fully configured application.

1.2.3 Spring beans and dependency injection

- **Spring Beans** A Spring bean is an object that is instantiated, assembled, and managed by the Spring IoC container. These beans represent the backbone of a Spring application, encapsulating the application's logic and functionality. The management of beans includes their lifecycle, configuration, and dependencies, which are defined through metadata provided by the developer.
- **Dependency injection** : is a design pattern used to implement IoC. It allows the Spring container to provide an object's dependencies rather than having the object create them itself. This leads to loose coupling between components and enhances testability.

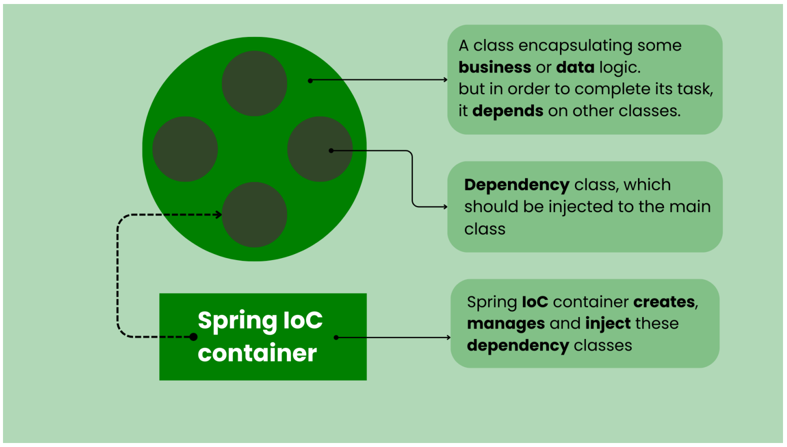

A class encapsulating some **business** or **data** logic.
but in order to complete its task, it **depends** on other classes.
Dependency class, which should be injected to the main class
Spring IoC container
Spring **IoC** container **creates**, **manages** and **inject** these **dependency** classes

Figure 3: Dependency injection of beans

1.2.4 Dependency Injection Examples

1. **XML based configuration**
2. **Annotation based configuration**
3. **Java based configuration**

2 Spring boot

Spring Boot is an extension of the Spring Framework that simplifies the process of developing Spring applications. For beginners, understanding how Spring Boot relates to the Spring Framework can help clarify its purpose and advantages.

Figure 4: Spring boot

2.1 Key features of Spring boot

- **Auto-Configuration** : Spring Boot automatically configures your application based on the libraries you include. For example, if you add a library for web development, it will set up a web server for you without needing extensive configuration.
- **Standalone applications** : With Spring Boot, you can create standalone applications that run independently. It includes an embedded web server (like Tomcat), so you don't need to deploy your application to an external server.
- **Starter dependencies** : Instead of manually specifying each dependency in your project configuration file, Spring Boot provides `starters` that bundle commonly used libraries together. For example, spring-boot-starter-web includes everything needed for a web application.

3 Setting up a Spring boot application

3.1 Prerequisites

3.1.1 Java development kit (JDK)

You need to have Java installed on your system. Spring Boot 3.x requires at least **Java 17 or higher** Ensure that your Java version is compatible with the Spring Boot version you plan to use.

3.1.2 Build tools

You need a build tool to manage your project dependencies and build processes. The two most common build tools used with Spring Boot are :

- **Maven** : Version 3.6.3 or later is required.
- **Gradle** : Versions 7.x (7.5 or later) and 8.x are supported

3.1.3 Integrated development environment (IDE)

- **Eclipse IDE**
- **Intellij IDEA**
- **Visual studio code**

3.2 Creating Our First Project

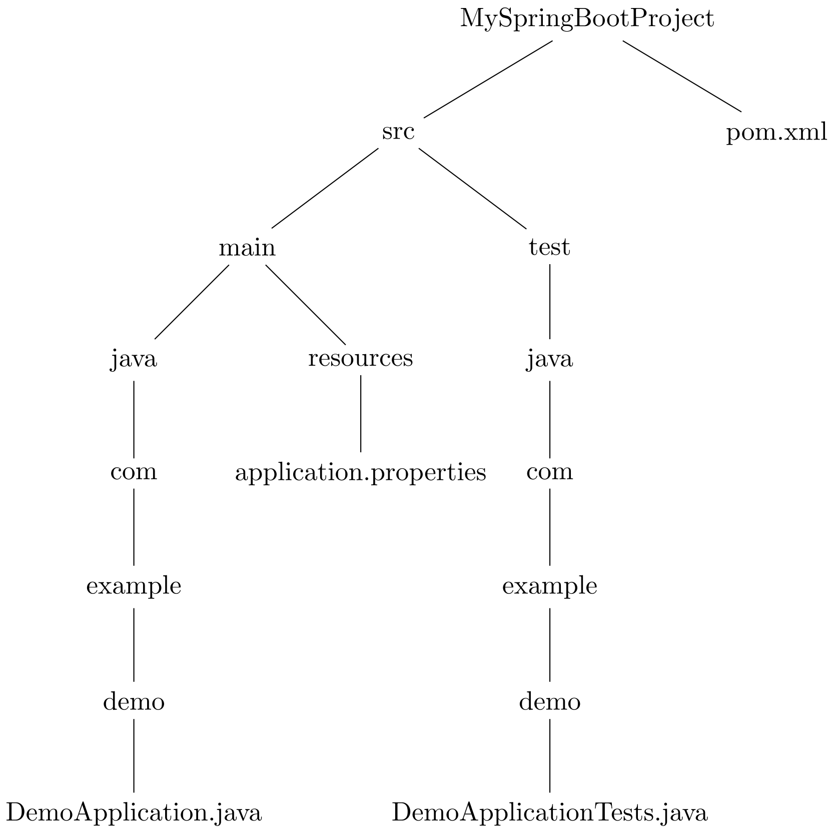
3.2.1 Bootstrapping the project

It is recommended to use Spring Initializr (https://start.spring.io/) to set up a Spring Boot project due to the following reasons:

- Quickly generates project structure and dependencies.
- Allows selection of required modules (e.g., Spring Web, Spring Data JPA).
- Provides pre-configured Maven/Gradle builds.
- Saves time and simplifies the setup process.

Figure 5: Spring initializr

3.2.2 Project strcuture



1. **src/main/java**:

- This directory contains the main source code for your application.
- It is organized by package, typically following the reverse domain name convention (e.g., `com.example.yourapplication`).
- The main class (e.g., `YourApplication.java`) contains the `main` method that starts the Spring Boot application.

2. **src/main/resources**:
    - This folder is for resource files that your application needs, such as configuration files and static assets.
        - `static`: Contains static files like CSS, JavaScript, and images that can be served directly.
        - `templates`: Contains server-side templates (e.g., Thymeleaf or FreeMarker) used for rendering dynamic web pages.
        - `application.properties`: A key configuration file where you define various settings for your application, such as server port, database connection details, and other properties.
3. **src/test/java**:
    - This directory is for test classes that correspond to your main application code.
    - It follows the same package structure as `src/main/java`, allowing you to keep tests organized alongside the code they test (e.g., `YourApplicationTests.java`).
4. **pom.xml**:
    - This is the Maven build file that manages project dependencies, plugins, and other configurations.
    - It specifies the Spring Boot starter dependencies you are using (like `spring-boot-starter-web` for web applications) and other libraries necessary for your project.

4 Conclusion

The Spring Framework and Spring Boot provide powerful tools for building scalable and maintainable enterprise-level applications. By leveraging concepts like Inversion of Control (IoC) and Dependency Injection, developers can create modular, testable, and easily configurable systems. Spring Boot further simplifies the development process by offering auto-configuration, standalone applications, and pre-configured dependencies, making it an ideal choice for modern Java-based applications. Understanding the key features and setup process of both Spring and Spring Boot will help developers streamline their application development and improve productivity.