Spring : Fundamental Concepts

Abhinav Vinci Follow 5 min read · Jul 23, 2023

Spring is a widely used **open-source framework** for building Java applications.

Key Concepts related to Spring:

Inversion of Control (IoC)

Inversion of Control refers to the **reversal of the flow of control in a program.**

In traditional programming, a class directly controls the creation and management of its dependencies.

- In IoC, the **control is shifted to an external entity (the IoC container, in this case, Spring**) that manages the creation and injection of dependencies.
- The IoC container creates and wires the objects together, allowing objects to focus on their main responsibilities.

What is inversion of control exactly?

- In normal flow, a program's custom code calls reusable libraries to take care of generic tasks, but with inversion of control, it is the framework that calls the custom code. In essence, this means delegating execution of tasks to generic code over specific code allowing for more versatility.
- Inversion of control carries the philosophy that reusable code and problem-specific code be developed independently even though they operate together in an application.

How to achieve inversion of control ?

- Callbacks, schedulers, event loops, Dependency Injection, and template method are examples of design patterns that follow the inversion of control principle

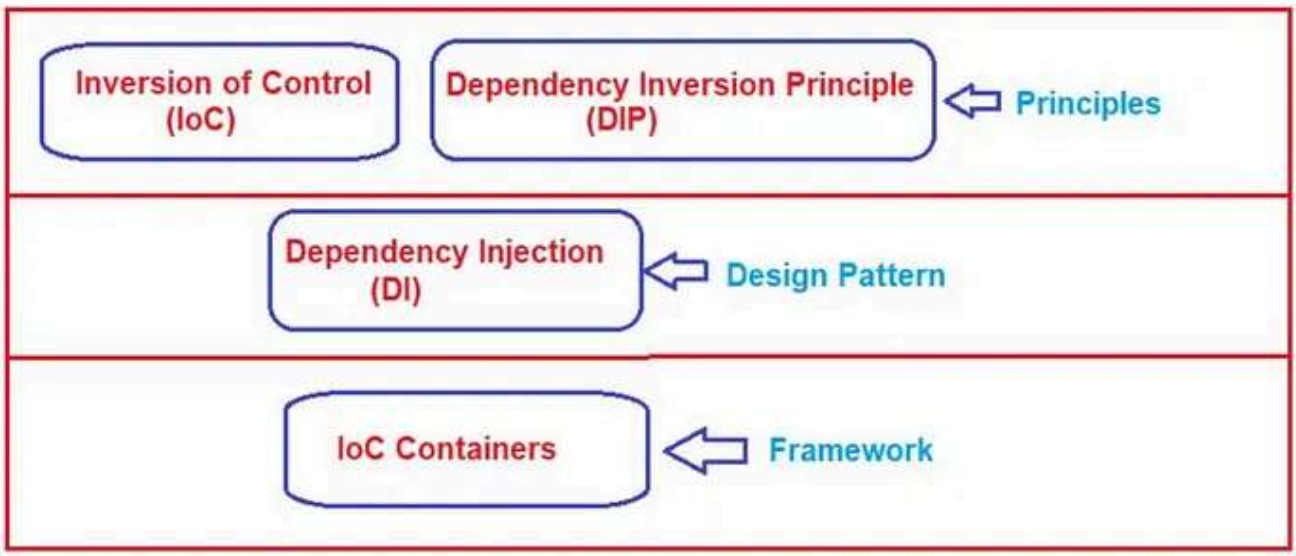
Dependency Injection (DI)

Dependency Injection is a **design pattern.** It main use is to **remove maunal hard-coded dependencies** in classes by externalizing the creation and management of objects (dependencies).

- Instead of an object creating its dependencies itself, it relies on an external entity (container) to provide the required dependencies.

Why Dependency Injection ?

Dependency Injection promotes loose coupling and improves the flexibility and testability of the code. The `Dependency-Injection` (DI) pattern is one of the ways to implement IoC



https://dotnettutorials.net/lesson/introduction-to-inversion-of-control/

How spring works?

Basically Spring is a framework for implementing **dependency-injection** which is a pattern that allows **building very decoupled systems.**

Spring (Dependency Injection) approach : What Spring does is that it handles all the objects instantiation and initialization and *injection* in the

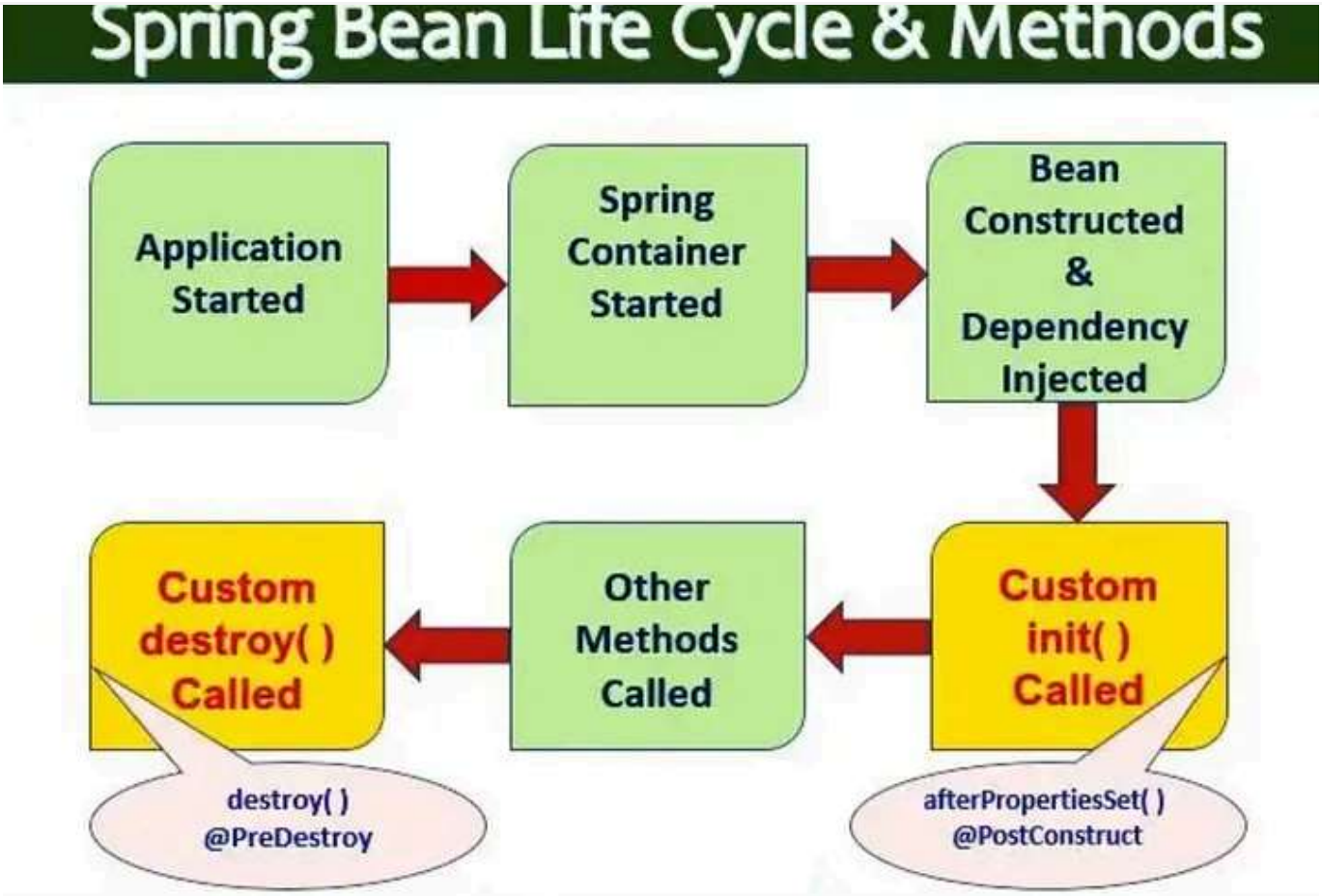
right places

Spring Container : The Spring Container, also known as the IoC container, is responsible for managing the Spring objects.

Spring Bean : In Spring, a **bean is an object** that is created and manged by the Spring IoC container.

Spring IOC container is like a Bag of Beans.

- Bean creation, maintainence and deletion all are responsibilities of Spring Container.
- The container reads the bean definitions (from XML files or annotations), creates bean instances, and manages their lifecycle.



https://medium.com/@sendvjs/spring-bean-life-cycle

We can put the bean into Spring by Wiring and Auto Wiring.

- Wiring mean we manually configure it into the XML file.
- Auto Wiring mean we put the annotations in the Java file then Spring automatically scan the root-context where java configuration file, make it and put into the bag of Spring.

Spring Beans in detail :

- Spring beans are just object instances that are managed by the Spring IOC container.
- Beans are Java objects that are instantiated, assembled, and managed by the Spring framework.

They represent the basic building blocks of a Spring application. Beans can be defined using XML configuration or annotated with Spring annotations.

Spring Configuration

Spring configuration defines how beans are created, wired, and managed.

- Configuration can be done using XML-based configuration or Java-based configuration (using annotations or Java config classes).

Spring annotations are like special keywords that you can use in your Java code to tell the Spring framework how to do certain things automatically.

They make it easier to work with Spring and save you from writing a lot of repetitive code.

Annotations vs XML

Annotations and XML are two different approaches used in the Spring framework for configuring and customizing applications. Both serve the

purpose of defining the behavior and dependencies of Spring components, but they have some key differences :

Annotations:

1. **Readability and Ease of Use:** They are embedded directly in the Java code alongside classes, methods, or fields. Annotations are typically more concise and easier to read since they are integrated directly into the Java code. This makes the code more self-explanatory and reduces the need for separate configuration files.
2. **Limited Flexibility:** While annotations are powerful and flexible in many cases, there are scenarios where the complexity of configurations may require additional XML configuration or are simply not achievable with annotations alone.

XML:

XML (Extensible Markup Language) is a separate markup language that uses tags to define configuration elements. Spring configuration in XML is typically done in separate XML files, often named `applicationContext.xml`.

1. **Explicit Configuration**: XML configuration provides explicit and detailed control over the configuration of Spring components. Developers can specify beans, dependencies, and other settings in a structured and well-defined format.
2. **Flexibility**: XML allows for more complex and dynamic configurations since it is not tied to the Java language. XML configuration can be externalized from the code, which is useful in scenarios where

configurations need to be modified without touching the application's source code.

> *Which One to Use?*

In modern Spring applications, annotations are often favored due to their simplicity, readability, and compile-time checking. They promote a more "code-centric" approach to configuration, making it easier to manage and understand. However, XML configuration can still be useful, especially in cases where complex or dynamic configurations are required, or when externalizing configurations is a priority.

Bean

Bean: It is a Java object that is created based on a configuration and can be accessed and used throughout the application.

- Beans are the building blocks of a Spring application, representing various components such as services, data sources, controllers, and more. Each bean is uniquely identified within the IoC container by its name or ID.
- Beans can be defined using XML-based configuration files or through annotations in Java-based configuration classes. The configuration specifies how to create the bean, its dependencies, and other properties.

Why we use Spring?

- **Simplifies Development:** Spring provides a comprehensive and consistent framework that simplifies the development of Java applications.
- **To implement Inversion of Control (IoC) and Dependency Injection (DI):** Promotes loose coupling and makes applications easier to maintain and test.
- **Provides Comprehensive Ecosystem**: Offers a wide range of modules and extensions (e.g., Spring Boot, Spring Data, Spring Security) for various application needs.
- **Integration Friendly:** Seamlessly integrates with other technologies and frameworks, making it easy to leverage existing tools and libraries.

Spring Java Programming Coding Software Development

Written by Abhinav Vinci

970 followers · 37 following

SWE Indeed | Ex : Amazon, Codenation (Trilogy) | IIIT-Allahabad
https://www.linkedin.com/in/abhinav-vinci-163343ab/

Follow

No responses yet

Write a response