# What is Spring?

**Spring** is a popular, open-source **Java platform** that provides comprehensive infrastructure support for developing robust, highly performant, and easily testable Java applications. It is not just a single framework but an umbrella project encompassing multiple modules (like Spring MVC, Spring Data, Spring Security, etc.).

**Example:** The core of Spring is the **Spring Framework**, which provides features like **Dependency Injection** and **Aspect-Oriented Programming (AOP)**. It allows you to build enterprise applications without the "heavy lifting" of traditional Java EE (now Jakarta EE).

# What is Spring Boot?

**Spring Boot** is an opinionated, convention-over-configuration extension of the Spring platform that makes it quick and easy to create standalone, production-grade Spring applications.

**Example:** Instead of manually setting up a web server like Tomcat and writing verbose XML configuration, you can add the spring-boot-starter-web dependency, and Spring Boot will **automatically configure an embedded Tomcat server** and all necessary settings.

# What is the relation between Spring platform and Spring Boot?

**Spring Boot is built on top of the Spring Platform (specifically the Spring Framework) and is part of the larger Spring Ecosystem.**

Spring Boot's main goal is to **simplify and accelerate** the development of new Spring applications. It takes the power of the Spring Framework and abstracts away much of the initial setup, dependency management, and boilerplate configuration.

**Relation:**

* **Spring Platform (The "What"):** Provides the fundamental programming and configuration model (IoC, DI, Transactions, etc.).
* **Spring Boot (The "How"):** Provides the tools to easily and quickly build, configure, and run a standalone Spring application (Auto-configuration, Starter dependencies, Embedded server).

# What is the relation between Spring platform and Spring framework?

The **Spring Framework** is the **core module** (the foundation) of the broader **Spring Platform**.

**Relation:**

* **Spring Platform (Umbrella):** This is the entire ecosystem, including the core framework, plus other major projects like Spring Boot, Spring Data, Spring Security, Spring Cloud, etc.
* **Spring Framework (Core):** This is the foundational module that defines the core concepts: the **Inversion of Control (IoC) container**, Dependency Injection, core utility classes, data access abstraction, and web/MVC capabilities. All other Spring projects rely on this core framework.

# What is Dependency Injection and how is it done in the Spring platform/framework?

**Dependency Injection (DI)** is a design pattern where an object receives its dependent objects (its "dependencies") from an external source, rather than creating them itself. This makes the code modular, flexible, and easy to test.

**How it's done in Spring:** Spring uses its **IoC Container** to manage the lifecycle of objects (called **Beans**). A developer uses annotations to tell Spring where and what to inject.

**Example (Field Injection):**

A black screen with blue text

AI-generated content may be incorrect.

# What is Inversion of Control (IoC) and how is it related to Spring?

**Inversion of Control (IoC)** is a fundamental design principle where the flow of a program is "inverted." Instead of the application manually controlling the creation and lifecycle of its objects, a specialized container (the IoC container) takes over this responsibility.

**How it is related to Spring:** **The Spring Framework is primarily defined by its IoC Container.**

* In traditional programming, you call methods and create objects yourself (e.g., new BookService()).
* In a Spring application, you simply tell the Spring **IoC Container** which objects you need (by using annotations like @Controller, @Service, or @Component), and the Container handles their creation, configuration, and injection (DI) into other objects.

**IoC Example (The "Hollywood Principle"):** IoC is often described using the "Hollywood Principle": **"Don't call us, we'll call you."** You don't call Spring methods to create objects; instead, you provide the components, and Spring calls your methods or instantiates your classes when it needs them.