**Spring**

Spring is a Java-based open-source framework that provides comprehensive infrastructure support for developing Java applications. It focuses on making Java development easier and more modular by supporting:

* Dependency Injection (DI)
* Aspect-Oriented Programming (AOP)
* Transaction management
* Integration with various data access frameworks (JPA, JDBC)
* Building web applications (Spring MVC)

At its core, Spring is a platform for building enterprise-level applications with loose coupling and testability.

**Spring Boot**

Spring Boot is an extension of the Spring framework that simplifies the process of building and running Spring applications.

Spring Boot provides:

* Auto-configuration: Automatically configures your application based on the dependencies you’ve added.
* Embedded servers: Like Tomcat or Jetty, so you don’t need to deploy WAR files.
* Production-ready features: Metrics, health checks, externalized configuration, etc.
* Starter dependencies: Predefined sets of libraries for different functionalities (e.g., spring-boot-starter-web).

Spring Boot makes it faster and easier to build stand-alone, production-ready Spring applications with minimal configuration.

**Relation between Spring platform and Spring Boot**

The Spring platform is a broader ecosystem or umbrella that includes many projects:

* Spring Framework
* Spring Boot
* Spring Security
* Spring Data
* Spring Cloud

Spring Boot is one of these projects. It sits on top of the Spring Framework and provides a simplified, opinionated setup.

**Relation between Spring platform and Spring framework**

The Spring platform is the ecosystem or collection of Spring projects. The Spring framework is the core project in this ecosystem. It provides the foundational features like:

* IoC container
* Dependency injection
* AOP
* Transaction management
* MVC for web apps

So, the Spring framework is part of the Spring platform.

**What is Dependency Injection (DI) and how is it done in Spring**

Dependency Injection is a design pattern where an object receives its dependencies from an external source, rather than creating them itself. In Spring, DI can be done in several ways:

* Constructor Injection (recommended)
* Setter Injection
* Field Injection (less preferred)

Example – Constructor Injection:

@Component

public class UserService {

private final UserRepository userRepository;

@Autowired

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

}

Spring’s IoC container is responsible for injecting these dependencies at runtime.

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

Inversion of Control (IoC) is a design principle where the control of object creation and management is transferred from the application code to the framework or container.

In Spring: The IoC container manages your objects (beans), wiring them together automatically.

You just declare the relationships and dependencies via annotations or XML. This allows for loose coupling and easier testing. Spring’s IoC container enables Dependency Injection. it’s how Spring implements IoC.