1. What is Spring?

Spring is an open-source application framework and inversion of control container for the Java platform. The core features of the Spring Framework can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework makes the development of Java Enterprise Edition applications easier by providing comprehensive infrastructure support for developing robust Java applications very easily and very rapidly.

Some key features and components of the Spring Framework:

1. Dependency Injection (DI): Spring manages the creation and injection of dependencies into classes, which helps decouple the classes' instantiation and lifecycle management from the application code.

2. Aspect-Oriented Programming (AOP): Spring supports AOP for separating cross-cutting concerns (like logging, security, etc.) from the business logic.

3. Spring MVC: A model-view-controller framework that uses the Spring Framework's features to develop web applications.

Spring ability to facilitate and speed up the development of complex, high-performance applications. It's a central feature of the Spring ecosystem, which includes a variety of other projects that can be used in conjunction with the core framework.

1. What is Spring Boot?

Spring Boot is a spring project that simplifies the process of setting up and developing new Spring applications. It favors convention over configuration by providing a set of default configurations for quick and easy setup, while also offering the flexibility to change these defaults as needed.

Its standout features include the ability to create stand-alone applications with embedded servers, minimal to zero XML configuration, and tools for developing microservices, making it a popular choice for developers looking to quickly create high-quality, production-grade applications.

1. What is the relation between Spring platform and Spring Boot?

The Spring platform, is a comprehensive framework for developing Java applications with enterprise-level features, such as dependency injection and transaction management. Spring Boot, on the other hand, is built on top of the Spring framework; it leverages Spring's infrastructure to simplify the bootstrapping and development of new Spring applications. Essentially, Spring Boot automates much of the configuration and setup process required for a Spring application, adheres to an "opinionated defaults" approach to reduce developer decision-making for boilerplate setup, and facilitates easy packaging and deployment of applications with embedded web servers.

1. What is the relation between Spring platform and Spring framework?

The Spring platform and the Spring framework are essentially the same thing, referred to by different names. The Spring framework, commonly just called "Spring," is the core foundation for creating Java applications, particularly on the server side.

It provides a comprehensive programming and configuration model with a vast array of features such as dependency injection, transaction management, web application development, data access, messaging, and more.

When people refer to the Spring platform, they're usually including the broader set of projects and technologies that are built on top of the Spring framework, extending its capabilities into various domains like cloud, security, and data management. However, the term "Spring platform" is less commonly used than "Spring framework," which is the more recognized and established term in the developer community.

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

Dependency Injection (DI) is a design pattern used to implement IoC (Inversion of Control), where the creation and binding of dependent objects are not done by the objects themselves but by some external entity. In simple terms, DI allows for objects to be given their dependencies rather than creating them internally. This promotes loose coupling, easier testing, and better separation of concerns.

In the Spring framework, DI is accomplished through the Spring container, which manages the instantiation and wiring of beans (objects). Here's how it works:

* Define Beans: You define beans and their dependencies in a Spring configuration file using XML or annotations. With annotations, you might use `@Component` for class-level declaration and `@Autowired` for constructor, field, or method parameter level autowiring.
* Configure Dependencies: Dependencies can be injected via constructor injection, setter injection, or field injection. Constructor and setter injection are the most explicit and are preferred for mandatory dependencies, while field injection is used for optional dependencies and is less invasive.
* Bean Lifecycle Management: The Spring container manages the lifecycle of beans, including creating bean instances, wiring them together, configuring them, and managing their complete lifecycle.

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

Inversion of Control (IoC) is a principle in software engineering where the control of objects or portions of a program is transferred to a container or framework. It's about inverting the flow of control so that the framework calls into the user's code instead of the user's code calling into the framework.

In the context of the Spring framework, IoC refers to the Spring IoC container, which provides a consistent mechanism to configure applications and manage Java objects (known as beans). The container is responsible for assembling these beans (factory pattern), managing their lifecycle, and injecting dependencies where required.

**How IoC relates to Spring**:

* Decoupling: IoC helps in decoupling the execution of a task from the implementation, making code more modular and easier to maintain.
* Bean Management: In Spring, objects are created and managed by the Spring IoC container, which injects these objects (beans) where needed, thereby managing the instantiation and configuration of objects.
* Dependency Injection: IoC is implemented through Dependency Injection in Spring. Beans declare their dependencies without creating or looking for dependent objects. The container injects these dependencies when it creates the beans.

This means that in a Spring application, you typically do not create objects directly using the `new` keyword. Instead, you describe how objects should be created and how they relate to one another in a configuration file or annotations, and the Spring container takes care of wiring them together.

A diagram of a software application

Description automatically generated