1 . What is Spring?

Spring is an open-source application framework and inversion of control container for the Java platform. It provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so developers can focus on application-level programming.

Example: Spring provides features like Dependency Injection, Aspect-Oriented Programming (AOP), Data Access, Messaging, and more, making it easier to develop enterprise-level Java applications.

2. What is Spring Boot?

Spring Boot is a project that is built on top of the Spring framework. It is designed to simplify the initial setup and development of new Spring applications by providing opinionated defaults and auto-configuration of the Spring platform.

Example: With Spring Boot, you can create a standalone Java application with minimal configuration, just by adding the required dependencies to your project.

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

The Spring platform is a comprehensive programming and configuration model for Java-based enterprise applications, providing a broad range of features and capabilities. Spring Boot is a project built on top of the Spring platform, designed to simplify the development and deployment of Spring-based applications.

Spring Boot leverages the core features of the Spring platform but adds an opinionated approach to configuration and setup, allowing developers to get started quickly with minimal boilerplate code and configuration.

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

The Spring framework is the foundational part of the Spring platform. It provides a comprehensive programming and configuration model for Java applications, including features like Dependency Injection, Aspect-Oriented Programming (AOP), Data Access, Messaging, and more.

The Spring platform encompasses the Spring framework and several other projects, such as Spring Boot, Spring Data, Spring Security, Spring Cloud, and more. These projects build upon the core Spring framework and provide additional capabilities for specific domains or use cases.

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

Dependency Injection (DI) is a design pattern where objects receive other objects that they depend on, rather than creating or finding those dependencies themselves. This promotes loose coupling, testability, and reusability of code.

In the Spring framework, Dependency Injection is achieved through the use of the Inversion of Control (IoC) container. Here's an example:

```
// Service class
public class EmailService {
   private SmtpServer smtpServer;

   // Constructor-based DI
   public EmailService(SmtpServer smtpServer) {
      this.smtpServer = smtpServer;
   }

   // ...
}

// Configuration class
@Configuration
public class AppConfig {
   @Bean
   public SmtpServer smtpServer() {
      return new SmtpServerImpl("smtp.example.com");
   }

   @Bean
   public EmailService emailService(SmtpServer smtpServer) {
      return new EmailService(smtpServer);
   }
}
```

6 .Inversion of Control

Inversion of Control (IoC) is a principle in software engineering that inverts the control flow of a program. Instead of an object creating and managing its dependencies, the control is inverted, and the dependencies are provided to the object from an external source, typically a container or framework.

In the context of Spring, the IoC container (also known as the Spring container) is responsible for creating and managing the lifecycle of objects (beans) and their dependencies. The Spring container reads the configuration metadata (XML, annotations, or Java code) and creates instances of the required beans, wires them together by injecting the dependencies, and manages their lifecycle.

The IoC principle is closely related to and enabled by Dependency Injection in Spring. The Spring container acts as the IoC container, providing a centralized mechanism for creating, configuring, and injecting dependencies into objects. This approach promotes loose coupling, testability, and modularity in applications, as objects no longer need to be responsible for creating or managing their dependencies.