1. What is Spring?

Spring is a powerful, lightweight Java application framework that provides comprehensive infrastructure support for developing Java applications. It is known for promoting good programming practices, such as loose coupling and separation of concerns.

- **Core features include dependency injection, aspect-oriented programming (AOP), transaction management, and support for building web applications, REST APIs, and more.**

- **It helps developers build enterprise-grade applications easily and efficiently.**

2. What is Spring Boot?

Spring Boot is a tool built on top of the Spring framework that makes it easy to create stand-alone, production-ready Spring-based applications with minimal configuration.

- **It comes with embedded servers (like Tomcat or Jetty), auto-configuration, and production-ready actuator endpoints.**

- **Its goal is to simplify setup and boost developer productivity.**

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

- **Spring Platform refers to the entire Spring ecosystem, including Spring Framework, Spring Boot, Spring Data, Spring Security, Spring Cloud, etc.**

- **Spring Boot is a part of the Spring Platform.**

- **Spring Boot uses the Spring Framework under the hood and simplifies the process of working with it.**

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

- **The Spring Framework is the core module of the Spring Platform.**

- **The Spring Platform includes the framework itself plus additional projects (Spring Boot, Spring Data, Spring Security, etc.) that work together to create a full ecosystem for application development.**

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

Dependency Injection (DI) is a design pattern where objects are provided with their dependencies instead of creating them internally.

- **Spring supports DI through:**
    - **XML configuration (old way)**
    - **Java-based configuration (`@Configuration`, `@Bean`)**
    - **Annotation-based injection (`@Autowired`, `@Inject`)**

Example:

```
@Component
public class ServiceA {
   private final RepositoryA repo;

   @Autowired
   public ServiceA(RepositoryA repo) {
      this.repo = repo;
   }
}
```

Spring automatically wires `RepositoryA` into `ServiceA`.

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

Inversion of Control (IoC) is a principle where the control of object creation and lifecycle is shifted from the application to a container or framework.

- **In Spring, the IoC Container (like `ApplicationContext`) is responsible for:**
    - **Creating objects (beans)**
    - **Managing their lifecycle**
    - **Injecting dependencies**

Relation to Spring:

- **The IoC container is at the core of the Spring Framework.**
- **Spring's DI is one way to implement IoC.**