1. **What is Spring?**

Spring is a comprehensive and modular framework for building enterprise applications in Java. It provides infrastructure support for developing Java applications, making it easier to manage various aspects of application development such as dependency management, data access, transaction management, and more. Spring is often used to develop Java-based enterprise applications, including web applications, RESTful services, and batch processing.

Example: In a Spring-based web application, you can use Spring's MVC (Model-View-Controller) framework to create controllers, services, and views, making it easier to handle HTTP requests, process business logic, and render responses.

2. **What is Spring Boot?**

Spring Boot is a subproject of the Spring Framework that simplifies the process of building production-ready, stand-alone, and highly customizable Spring-based applications. It provides a set of conventions, templates, and pre-configured settings that reduce the need for boilerplate code and manual configuration. Spring Boot is designed to accelerate application development by offering a "zero-configuration" approach and an embedded web server.

Example: With Spring Boot, you can quickly create a RESTful web service by defining a few annotated classes and dependencies, allowing you to focus on writing business logic rather than configuring the application.

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

Spring Boot is built on top of the Spring Framework. The Spring platform includes the entire ecosystem of Spring projects and modules, including the core Spring Framework, Spring Data, Spring Security, Spring Cloud, and more. Spring Boot, on the other hand, is a specific project within the Spring ecosystem that simplifies the setup and configuration of Spring applications.

Spring Boot can be seen as an extension of the Spring platform that focuses on simplifying the development of stand-alone Spring applications, making it easier to create microservices and web applications.

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

The Spring platform encompasses the entire ecosystem of Spring projects and modules, including the Spring Framework. The Spring Framework is the core of the Spring platform and provides the foundational features for building enterprise applications, such as dependency injection, AOP (Aspect-Oriented Programming), and transaction management.

In summary, the Spring Framework is a critical part of the Spring platform, but the platform includes additional projects and tools that extend its capabilities for various purposes.

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

Dependency Injection (DI) is a design pattern in which the dependencies of a class are injected from the outside rather than being created within the class itself. In Spring, DI is used to manage and resolve object dependencies. Spring provides various ways to achieve DI, with the most common approach being through constructor injection or setter injection.

Example (Constructor Injection in Spring):

```java
public class OrderService {
    private final OrderRepository orderRepository;

    public OrderService(OrderRepository orderRepository) {
        this.orderRepository = orderRepository;
    }

    // ...
}
```

In this example, the `OrderService` class depends on the `OrderRepository`, and the dependency is injected via the constructor.

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

Inversion of Control (IoC) is a design principle in which control over the flow of a program is shifted from the program itself to a container or framework. In the context of Spring, IoC means that the framework manages the creation and lifecycle of application objects (beans) and controls their interactions.

IoC is a fundamental concept in the Spring framework, and it is achieved through Dependency Injection (DI). In Spring, the container is responsible for creating and managing the objects (beans), and it injects their dependencies. This allows for decoupling of components, making applications more modular, testable, and maintainable.

Example (Spring IoC):

In a Spring application, you define beans and their dependencies in a configuration file or using annotations. The Spring container (IoC container) then manages the instantiation and wiring of these beans, as shown in the previous example of Dependency Injection.