2. Write short answers to the following questions.

Note: For each question, show it or give an example to support your answer.

### 1. What is Spring?

**Spring** is a **comprehensive open - source Java framework** primarily used to build **enterprise - level applications**. It provides **infrastructure support** for developing Java applications by managing application components and their life cycles.

One of its core features is **Dependency Injection (DI)** which allows loose coupling between classes. Spring also supports features like **transaction management**, **AOP (Aspect - Oriented Programming)**, and **data access abstraction** (like JDBC and JPA integration).

**Example:** Instead of manually managing service classes and their dependencies, Spring can manage them:

@Service

public class UserService {

private final UserRepository userRepository;

// Dependency is injected via constructor

@Autowired

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

}

**2. What is Spring Boot?**

**Spring Boot** is a **rapid development tool** built on top of the Spring Framework. It simplifies the process of creating Spring applications by eliminating boilerplate configuration code. It includes features like:

* **Auto - configuration** (automatically configures beans based on classpath)
* **Embedded servers** (like Tomcat, Jetty - no need to deploy WAR files manually)
* **Opinionated defaults** (sensible default configurations for most apps)

**Example:** You can spin up a REST API in minutes:

@RestController

public class HelloController {

@GetMapping("/hello")

public String sayHello() {

return "Hello, Spring Boot!";

}

}

With just a main class:

@SpringBootApplication

public class App {

public static void main(String[] args) {

SpringApplication.run(App.class, args);

}

}

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

The **Spring platform** is the **entire ecosystem** of Spring projects, and **Spring Boot is one of those projects**. While Spring Framework is the core of the platform, Spring Boot enhances it by providing:

* Pre - configured templates
* Built - in server setup
* Starter dependencies to reduce setup time

In short, **Spring Boot makes it easier to use the Spring platform** without requiring deep knowledge of configuration and setup.

**Analogy:** If Spring Framework is a toolbox, Spring Boot is a fully assembled workstation using that toolbox.

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

The **Spring platform** is a collection of several integrated projects:

* **Spring Framework** - Core dependency injection and IoC container
* Spring Boot - Rapid development
* Spring Data - Simplified data access
* Spring Security - Authentication and authorization
* Spring Cloud - Distributed systems and microservices

The **Spring Framework** is the **foundation**. Every other project in the Spring platform builds on it. Without Spring Framework, the rest of the platform wouldn't work.

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

**Dependency Injection (DI)** is a design pattern where the creation and binding of an object's dependencies are **externalized**, typically handled by a container.

In **Spring**, DI is done through:

* **Constructor Injection** (preferred)
* **Setter Injection**
* **Field Injection** (not recommended for testing)

Spring automatically resolves and injects dependencies based on bean definitions, either declared via annotations or XML.

**Example using Constructor Injection:**

@Component

public class OrderService {

private final PaymentService paymentService;

@Autowired

public OrderService(PaymentService paymentService) {

this.paymentService = paymentService;

}

public void processOrder() {

paymentService.chargeCustomer();

}

}

Here, Spring creates an instance of PaymentService and injects it into OrderService.

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

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

In **Spring**, the **IoC container** is responsible for:

* Creating and configuring objects (beans)
* Managing their life cycles
* Injecting them where needed

This reduces tight coupling and improves modularity and testability.

**Example:** You define a bean:

@Component

public class EmailService {

public void sendEmail(String to, String message) { ... }

}

And Spring will inject it automatically into any class that depends on it:

@Service

public class NotificationService {

private final EmailService emailService;

@Autowired

public NotificationService(EmailService emailService) {

this.emailService = emailService;

}

}

No need for new EmailService() — Spring handles it.