Spring

**Q: What is Spring IoC Container and Dependency Injection?**

The **Spring IoC Container** is the **core component** of the Spring Framework that is responsible for **managing the life cycle and configuration of application objects**. "IoC" stands for **Inversion of Control**, which means the control of object creation and dependency management is transferred from the application code to the container.

In simpler terms, instead of manually creating objects using new, we allow the container to create and inject the required dependencies into our classes.

The **IoC Container** does this through a process known as **Dependency Injection (DI)**.

**What is Dependency Injection?**

**Dependency Injection** is a design pattern used to **remove the dependency between objects**. Instead of a class instantiating its own dependencies, the dependencies are **injected externally**—by the Spring container—usually through:

1. **Constructor Injection**
2. **Setter Injection**
3. **Field Injection** (less recommended, but possible)

For example, if class A depends on class B, we don’t create B inside A. Instead, we let the container inject an instance of B into A.

**Why is it useful?**

* It promotes **loose coupling**
* It improves **testability** (e.g., with mocks)
* It enhances **code maintainability and readability**
* It aligns with **SOLID principles**, especially the **Dependency Inversion Principle**

**Types of IoC Containers in Spring:**

1. **BeanFactory** – Lightweight and lazy-loading container.
2. **ApplicationContext** – More advanced, feature-rich container (used in most applications).

**Real-world Example:**

Suppose we have a StudentService class that depends on a StudentRepository. Instead of doing:

StudentRepository repo = new StudentRepository();

We declare:

@Autowired

private StudentRepository repo;

Then, Spring injects the appropriate implementation automatically at runtime.

**Summary:**

So, to summarize:

The **Spring IoC container** is responsible for managing beans and their dependencies using **Dependency Injection**, which helps achieve loose coupling and better modular code.