Spring Core Interview Questions

- **What is Spring Framework?**

  Answer: Spring is a lightweight open-source Java framework primarily used for building enterprise applications. It provides comprehensive infrastructure support for developing Java applications and promotes good practices like loose coupling and testability.

- **What is Inversion of Control (IoC)?**

  Answer: IoC is a design principle in which the control of object creation and dependency management is transferred from the program to the Spring container.

- **What is Dependency Injection (DI) in Spring?**

  Answer: DI is a pattern through which Spring injects dependencies (objects) into other objects either via constructor, setter, or field injection, enabling loose coupling.

- **What are the types of Dependency Injection in Spring?**

  Answer:

  - Constructor Injection
  - Setter Injection
  - Field Injection (with annotations like `@Autowired`)

- **What is a Spring Bean?**

  Answer: A Spring Bean is an object managed by the Spring IoC container. Beans are instantiated, assembled, and managed by Spring.

- **What is the difference between BeanFactory and ApplicationContext?**

  Answer:

  - `BeanFactory` is the basic container with lazy initialization.
  - `ApplicationContext` is a more advanced container that supports internationalization, event propagation, and eager initialization.

- **What are the different bean scopes in Spring?**

Answer:

  - Singleton (default)
  - Prototype
  - Request (Web context)
  - Session (Web context)
  - Application (Web context)
  - WebSocket

- **How to define a bean in Spring XML configuration?**

  Answer:

  ```
  <bean id="myBean" class="com.example.MyClass"/>
  ```

- **How to configure a bean using annotations?**

  Answer: Use `@Component`, `@Service`, `@Repository`, or `@Controller` on class and enable component scanning using `@ComponentScan`.

- **What is the use of `@Autowired` annotation?**

  Answer: It allows Spring to automatically resolve and inject collaborating beans.

What is the Spring Bean lifecycle?

Answer: Bean lifecycle stages include:

- Instantiation
- Populate properties
- `BeanNameAware`, `BeanFactoryAware`
- `InitializingBean` / `@PostConstruct`
- Custom init-method
- Use of the bean
- `DisposableBean` / `@PreDestroy`
- Custom destroy-method

How can you define custom initialization and destroy methods for a bean?

Answer:

```
<bean id="myBean" class="com.example.MyClass" init-method="init" destroy-method="cleanup"/>
```

What is the difference between Singleton and Prototype scope?

Answer:

- Singleton: Only one instance per Spring container
- Prototype: A new instance is created every time it's requested

What is the role of `@Qualifier` in Spring?

Answer: It helps disambiguate between multiple beans of the same type to inject the correct one.

What is Java-based configuration in Spring?

Answer: Instead of XML, you can use `@Configuration` and `@Bean` annotations to define beans:

```
@Configuration
public class AppConfig {
    @Bean
    public MyService myService() {
        return new MyServiceImpl();
    }
}
```

Explain the use of `@ComponentScan`.

Answer: It tells Spring where to look for annotated components:

```
@ComponentScan(basePackages = "com.example")
```

What is the difference between `@Component`, `@Service`, and `@Repository`?

Answer: All are specialization of `@Component`, but semantically:

- `@Service`: Business logic
- `@Repository`: Persistence layer
- `@Component`: Generic stereotype

Can you inject a prototype bean into a singleton bean?

Answer: Direct injection results in singleton behavior. To truly get a new prototype instance each time, use `ObjectFactory` or `Provider`.

What is the difference between `@PostConstruct` and `InitializingBean`?

Answer:

- `@PostConstruct` is a JSR-250 annotation, considered more modern and flexible.
- `InitializingBean` is Spring-specific and requires implementing an interface.

How do you handle circular dependencies in Spring?

Answer: Spring handles setter-based circular dependencies. Constructor-based circular dependencies will cause an exception unless resolved using `@Lazy` or `ObjectFactory`.

What is Spring's `BeanPostProcessor` and when is it used?

Answer: It's used to modify bean instances before and after initialization. Useful for custom logic like proxying or modifying configuration.

How does Spring resolve dependencies internally?

Answer: Spring uses `DependencyDescriptor` to resolve dependencies from the ApplicationContext or BeanFactory by matching type, qualifiers, and scope.

What is the role of `ApplicationContextAware` interface?

Answer: It allows a bean to access the `ApplicationContext` that created it.

Explain how annotation-based and XML-based configurations can be combined.

Answer: You can use `@ImportResource` in a Java config class to include XML beans, or include annotated classes in XML using `<context:component-scan>`.

What is the purpose of `@Scope` annotation?

Answer: It defines the scope of the Spring bean:

```
@Scope("prototype")
```

What is Lazy Initialization in Spring?

Answer: With `@Lazy`, Spring defers bean creation until it's actually needed:

```
@Lazy
@Component
```

What is the difference between `@Bean` and `@Component`?

Answer:

- `@Component`: Automatically detected by component scanning
- `@Bean`: Used within `@Configuration` to declare beans explicitly

How does Spring support internationalization (i18n)?

Answer: Via `MessageSource` and `.properties` files. Injected using `@Autowired` or accessed via `ApplicationContext.getMessage()`.

What are Profiles in Spring and how are they used?

Answer: Profiles allow conditional bean registration. Activate with `@Profile("dev")`, and use `spring.profiles.active=dev` in properties.

How to create a custom scope in Spring?

Answer: Implement `Scope` interface and register it using `ConfigurableBeanFactory.registerScope()`.

🔶 Practical Scenario-Based Spring Core Interview Questions

1. **You have a service class with multiple dependencies. Which injection method would you prefer and why?**

   Answer: Constructor injection is preferred because it ensures all required dependencies are provided at object creation, promoting immutability and making the class easier to test.

2. **You need a different instance of a bean for every user session. How would you configure it?**

   Answer: Use `@Scope("session")` on the bean or in XML. Spring will then create one instance per HTTP session.

3. **You want to initialize a database connection pool when the application starts. How would you do it?**

   Answer: Define a bean for the connection pool and use `@PostConstruct` or implement `InitializingBean` to perform initialization logic.

4. **You're using constructor injection and have circular dependencies. How can you resolve this?**

   Answer: Refactor one dependency to use setter injection, or use `@Lazy` to delay bean instantiation.

5. **You need a utility bean that can be accessed from anywhere but only initialized once. What scope should you use?**

   Answer: Use `@Scope("singleton")`, which is the default. Spring creates a single shared instance per application context.

6. **A configuration class needs to return multiple beans of the same type with different values. How do you handle this?**

   Answer: Define multiple `@Bean` methods with unique names and use `@Qualifier` to inject the correct one where needed.

7. **You want to conditionally load a bean only if another class is available on the classpath. How would you configure that?**

   Answer: Use `@ConditionalOnClass(SomeClass.class)` or Spring Boot's conditional annotations (if using Spring Boot).

8. **How would you create a bean that should not be eagerly initialized on application startup?**

   Answer: Use `@Lazy` on the bean definition. This delays the initialization until the bean is first requested.

9. **You're working on a legacy app using XML config. You need to migrate it to annotations. What's your approach?**

   Answer: Gradually replace XML-defined beans with `@Component`, `@Service`, etc., and enable component scanning with `@ComponentScan`.

10. **You want to execute some code during application shutdown. How would you achieve this in Spring?**

    Answer: Implement `DisposableBean` or use the `@PreDestroy` annotation in a bean to perform cleanup during shutdown.