**1. What is the Spring Framework? What are its advantages?**

**Answer:**  
The Spring Framework is an open-source application framework for Java, widely used for enterprise-level applications. It provides comprehensive infrastructure support for developing Java applications.

**Advantages:**

* Lightweight and modular.
* Dependency Injection (DI) and Aspect-Oriented Programming (AOP).
* Supports transaction management.
* Reduces boilerplate code with abstractions for JDBC and ORM.
* Provides integration with other frameworks and technologies like Hibernate, JPA, etc.

**2. What is the Core Container module in Spring?**

**Answer:**  
The **Core Container** module is the foundation of the Spring Framework and consists of:

* **BeanFactory:** The basic container for managing beans.
* **ApplicationContext:** Extends BeanFactory and adds more enterprise-specific functionalities.
* **Core:** Provides the essential functionality of the framework, like DI.
* **SpEL (Spring Expression Language):** Used for querying and manipulating objects.

**3. What is Dependency Injection (DI) in Spring? Explain its types.**

**Answer:**  
**Dependency Injection** is a design pattern where the Spring container injects dependencies into an object at runtime.

**Types of DI:**

1. **Constructor Injection:** Dependencies are provided via the class constructor.
2. **Setter Injection:** Dependencies are injected via setter methods.
3. **Field Injection:** Dependencies are directly injected into fields (requires @Autowired).

**4. What are Spring beans? How are they managed?**

**Answer:**  
A **Spring bean** is an object that is instantiated, configured, and managed by the Spring IoC container.

Beans are defined in:

* XML configuration.
* Java-based configuration using @Configuration and @Bean.
* Component scanning with @Component, @Service, @Repository, and @Controller.

**5. What is the difference between ApplicationContext and BeanFactory?**

**Answer:**

| **Feature** | **BeanFactory** | **ApplicationContext** |
| --- | --- | --- |
| **Lazy Loading** | Yes, beans are created lazily. | No, beans are created eagerly by default. |
| **Enterprise Features** | Limited | Supports internationalization, event propagation, etc. |
| **Bean Post-Processors** | Must be registered manually. | Automatically registered. |
| **Usage** | Lightweight applications. | Enterprise applications. |

**6. What are the different bean scopes in Spring?**

**Answer:**

1. **Singleton (Default):** One instance per Spring container.
2. **Prototype:** A new instance for each request.
3. **Request (Web only):** One instance per HTTP request.
4. **Session (Web only):** One instance per HTTP session.
5. **GlobalSession (Web only):** Scoped for a global session (e.g., Portlet applications).

**7. What are Spring annotations, and why are they important?**

**Answer:**  
Spring annotations provide a more declarative way of configuring beans and wiring dependencies compared to XML.  
Examples:

* @Component, @Service, @Repository, @Controller: Component scanning.
* @Autowired: Dependency injection.
* @Configuration, @Bean: Java-based configuration.
* @Scope: Define bean scope.

**8. What is Spring’s @Autowired annotation?**

**Answer:**  
@Autowired is used for dependency injection. The Spring container automatically resolves the dependency by type.

**Key Features:**

* Can be applied to constructors, fields, and setter methods.
* Requires the bean to be present in the Spring container.

**9. What is the role of the @Qualifier annotation in Spring?**

**Answer:**  
@Qualifier is used to resolve conflicts when multiple beans of the same type exist. It specifies which bean to inject by name.

Example:

java

Copy code

@Autowired

@Qualifier("beanName")

private MyService myService;

**10. What is the Spring Bean Lifecycle?**

**Answer:**

1. Instantiate bean.
2. Populate properties (dependency injection).
3. Call setBeanName() from BeanNameAware.
4. Call setApplicationContext() from ApplicationContextAware.
5. Pre-initialization (BeanPostProcessor).
6. Initialization (@PostConstruct or afterPropertiesSet() from InitializingBean).
7. Post-initialization (BeanPostProcessor).
8. Ready for use.
9. Destroy (@PreDestroy or destroy() from DisposableBean).

**11. What is AOP in Spring? Explain its use.**

**Answer:**  
**Aspect-Oriented Programming (AOP)** separates cross-cutting concerns (e.g., logging, security) from business logic. Spring’s AOP module allows developers to implement these concerns using aspects.

**Core Concepts:**

* **Aspect:** Modularizes a concern.
* **Advice:** Action taken at a join point (e.g., @Before, @After, @Around).
* **Pointcut:** A predicate defining where advice applies.
* **JoinPoint:** A point during execution (e.g., method call).

**12. What is the Spring Expression Language (SpEL)?**

**Answer:**  
SpEL is a powerful expression language for querying and manipulating objects.

**Examples:**

java

Copy code

@Value("#{2 + 4}") // Literal expressions

private int sum;

@Value("#{systemProperties['user.name']}") // Property access

private String username;

**13. What is the @Configuration annotation?**

**Answer:**  
@Configuration indicates a class that contains @Bean-defined methods. It is used for defining Spring beans in a Java-based configuration.

Example:

java

Copy code

@Configuration

public class AppConfig {

@Bean

public MyService myService() {

return new MyServiceImpl();

}

}

**14. How does Spring achieve loose coupling?**

**Answer:**  
Spring achieves loose coupling using Dependency Injection and Interface-Oriented programming. The IoC container handles the creation and wiring of objects, allowing components to interact through interfaces rather than concrete implementations.

**15. What is the purpose of the @ComponentScan annotation?**

**Answer:**  
@ComponentScan configures the package(s) to scan for annotated Spring components (@Component, @Service, etc.).

Example:

java

Copy code

@Configuration

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

public class AppConfig {}