1) Explain the process and significance of the Spring Bean lifecycle. How might understanding this be crucial in a large-scale application?

The Spring Bean lifecycle involves the creation, use, and destruction of beans managed by the Spring container. Understanding this lifecycle is crucial in large-scale applications because it helps in optimizing resource management, ensuring beans are created, used, and disposed of efficiently. This knowledge also aids in troubleshooting issues related to bean dependencies and execution flow within the application.

2) What are the differences between ApplicationContext and BeanFactory?

ApplicationContext and BeanFactory are both used for managing beans in Spring, but ApplicationContext offers more advanced features like event propagation, declarative mechanisms to create a bean, and easier integration with Spring's AOP features. BeanFactory is simpler and lighter, suitable for low-memory scenarios and provides basic container functionality. Generally, ApplicationContext is preferred for most modern Spring applications due to its comprehensive support and ease of use.

3) Mention scenarios where BeanFactory can be used and scenarios where ApplicationContext can be used.

BeanFactory is best used in scenarios where minimal resources are available or when you require only basic bean management functionalities, like in small applications or embedded systems. On the other hand, ApplicationContext is ideal for enterprise-level applications that need advanced features such as event propagation, AOP integration, and declarative services to handle complex business scenarios. It also provides built-in support for internationalization, web contexts, and various other enterprise-level services.

4) What is a circular dependency issue?

A circular dependency issue occurs when two or more beans in a Spring application depend on each other to be created. For example, Bean A requires Bean B to be created, and Bean B simultaneously requires Bean A. This situation leads to a deadlock, as neither bean can be instantiated until the other is, which prevents the application from starting up properly.

5) Explain different ways provided by Spring Boot to resolve circular dependencies.

In Spring Boot, circular dependencies can be resolved by using setter injection instead of constructor injection, allowing beans to be instantiated before their dependencies are

set. Another method is using the @Lazy annotation, which defers the initialization of a bean until it is actually needed, thus breaking the dependency cycle. Additionally, re-designing the application architecture to better separate concerns and reduce coupling between beans can also effectively address circular dependencies.

6) Difference between @Component and @Service. Are these interchangeable?

@Component is a generic stereotype for any Spring-managed component, while @Service is a specialization of @Component that indicates a bean is performing a service task or business logic. Technically, they are interchangeable because they both create Spring beans, but using @Service provides better clarity about the bean's role within the application. It's best practice to use @Service for service-layer beans and @Component for beans that don't fit into more specific categories like @Controller or @Repository.

7) Difference between JpaRepository and CrudRepository, and mention the scenario where CrudRepository is used.

CrudRepository provides basic CRUD (Create, Read, Update, Delete) functionality for handling entities in a database. In contrast, JpaRepository extends CrudRepository and adds additional JPA-specific methods like flushing the persistence context and batch operations. CrudRepository is suitable for applications that require basic database interactions without the need for the advanced capabilities provided by JpaRepository, making it ideal for simpler or less demanding data access scenarios.

8) What is the difference between @Qualifier and @Primary, and where is this annotation used? Difference between @Component and @Service. Are these interchangeable?

@Qualifier is used to specify which bean to inject by name, offering precise control when multiple beans of the same type exist. @Primary marks a bean as the default choice for autowiring when several options are available, streamlining dependency management.

@Component and @Service both create Spring beans, but @Service specifically denotes a bean that handles service tasks, suggesting its role in the service layer. Using @Service over @Component helps clarify the bean's purpose in your application, although they are technically interchangeable.

9) Usage of @Transactional annotation.

The @Transactional annotation in Spring is used to define the scope of a single database transaction. When applied to a method or class, it ensures that the enclosed operations are executed within a transactional context, meaning they either all succeed or all fail together. This is particularly useful for maintaining data integrity and handling complex operations that involve multiple steps or queries to the database.

10) What is Spring Profiles? How do you start an application with a certain profile?

Spring Profiles provide a way to segregate parts of our application configuration and make it only available in certain environments. For example, we can define database configurations for development, testing, and production environments without them interfering with each other. To start an application with a specific profile, we can use the -Dspring.profiles.active=profile_name parameter in our command line when launching the application, or set the spring.profiles.active property in our application's configuration files.

11) How can you inject properties using environment variables?

In Spring, we can inject properties from environment variables using the @Value annotation. Simply specify the environment variable inside the annotation like @Value("${MY_ENV_VAR}") where MY_ENV_VAR is the name of our environment variable. This makes the value of the environment variable available to our Spring bean, allowing your application to adapt to different environments seamlessly.

12) Imagine you have a conflict between beans in your application; how would you resolve it using Spring Boot?

To resolve a bean conflict in Spring Boot, we can use the @Qualifier annotation to specify which bean to use when multiple beans of the same type exist. Simply annotate the injection point with @Qualifier("beanName") where "beanName" is the unique name of the bean you want to use. This directs Spring's dependency injection to use the specified bean, thus resolving the conflict.

13) What happens if multiple AutoConfiguration classes define the same bean?

In Spring Boot, if multiple auto-configuration classes define the same bean, the last one read by the Spring container usually takes precedence, potentially overriding the beans defined earlier. This behavior is influenced by the ordering of auto-configuration classes, which can be controlled using the @AutoConfigureOrder or

@AutoConfigureAfter/@AutoConfigureBefore annotations to specify the load order explicitly. This setup helps manage dependencies and configurations more effectively in complex applications.

14) Do you prefer using XML or annotations for configuration in Spring applications, and why?

Annotations are preferred over XML for configuration in Spring applications because they provide a clearer, more concise way to manage dependencies directly within the Java code. This approach reduces the need for separate configuration files, making the code easier to understand and maintain. Annotations also enhance modularity and make it easier to enable or disable features through simple code changes.

15) What is the difference between the @Spy and @Mock annotations in Mockito?

In Mockito, @Mock is used to create a fully mocked instance of a class where all methods are stubbed and do not execute any actual code. This is useful for isolating dependencies in unit tests. On the other hand, @Spy is used to create a partial mock, meaning it wraps an actual instance of the class and all methods still execute real code unless explicitly overridden. This allows for selectively mocking certain behaviors while keeping the rest of the object's real functionalities intact, making it suitable for more integrated scenarios where some real behaviors are needed.

16) What is the difference between Joint Point and Point Cuts in Spring AOP.

In Spring AOP, a **Joint Point** is a specific point during the execution of a program, such as method calls or field access, where an aspect (a modularization of a concern that cuts across multiple classes) can be applied. **Pointcuts**, on the other hand, are expressions that select one or more joint points and can be used to define where advice (code linked to specific program points) should be applied. Essentially, pointcuts help determine *where* the advice should execute in the application, whereas joint points represent the *actual locations* in the application where those actions take place.

17) What is the use of Spring Batch, have you ever implemented the same, if yes kindly tell me the steps?

Spring Batch is a framework for processing large volumes of data automatically and efficiently, ideal for tasks like data migration, processing daily transactions, or generating reports. It simplifies batch operations by providing essential services, configurations, and enhancements that are required in batch applications. Yes, I implemented Spring Batch myself, the typical steps include defining a job configuration that specifies the steps the batch process will take, setting up a reader to pull data, a processor to apply business

logic, and a writer to output the processed data, all managed within Spring's context to ensure transactional integrity and job monitoring.

18) What type of injection use by @Autowired?

The @Autowired annotation in Spring primarily uses **constructor injection** by default, where dependencies are provided through a class constructor at the time of object creation, promoting immutability and mandatory dependency declaration. However, it can also be used for **field injection**, where Spring directly sets the values of fields on your beans, and **setter injection**, where dependencies are injected through setter methods after the bean is constructed. This flexibility allows for various configurations depending on the needs of the application.

19) Why constructor injection is recommended over setter-based injection?

Constructor injection is recommended over setter-based injection because it ensures that all necessary dependencies for a class are provided when the class is created. This makes objects immutable and stable once constructed, as they can't exist without their required dependencies. Additionally, it prevents the class from being in an incomplete state, reducing errors related to uninitialized dependencies.

20) Define AOP, and share its biggest disadvantage.

Aspect-Oriented Programming (AOP) is a programming paradigm that allows developers to modularize cross-cutting concerns, like logging and security, separate from the main business logic. AOP improves code readability and reduces redundancy by separating these aspects into distinct sections. However, its biggest disadvantage is that it can make the flow of execution harder to follow. This complexity arises because the modularized code executes separately from the main application flow, making it challenging for developers to trace and debug.

21) How can you prevent cyclic dependency in spring?

To prevent cyclic dependencies in Spring, you can redesign your classes to remove direct dependencies, use setter or field injection instead of constructor injection, or introduce interfaces to decouple the components. This approach involves rethinking class designs to reduce tight coupling, employing different types of dependency injections that don't force immediate object creation, or using interfaces that abstract the implementation

details. By doing so, you prevent the scenario where two or more classes depend on each other to be instantiated, which can cause the application to fail at runtime.