* **What is Spring Framework?**
  + **Answer**: The Spring Framework is an open-source application framework for Java. It provides comprehensive infrastructure support for developing Java applications, including data access, transaction management, and web applications. It is designed to simplify the development process by offering various components such as dependency injection, aspect-oriented programming, and more.
* **What is the Spring Core Container?**
  + **Answer**: The Spring Core Container is the core component of the Spring Framework that manages the lifecycle and configuration of application objects. It includes the BeanFactory and ApplicationContext.
* **What is a Bean in Spring?**
  + **Answer**: A bean is an object that is instantiated, assembled, and managed by the Spring IoC container. Beans are created based on the configuration metadata provided to the container (via XML or annotations).
* **What is Dependency Injection (DI)?**
  + **Answer**: Dependency Injection is a design pattern used to implement IoC (Inversion of Control). It allows an object to have its dependencies supplied by an external source rather than creating them itself. Spring supports various DI methods, such as constructor injection, setter injection, and field injection.
* **What is Inversion of Control (IoC)?**
  + **Answer**: IoC is a principle in which the control flow of a program is inverted. Instead of the application calling the framework, the framework calls the components or services of the application. In Spring, IoC is achieved through Dependency Injection.
* **What are the different types of IoC containers in Spring?**
  + **Answer**: The main types of IoC containers in Spring are:
    1. **BeanFactory**: The simplest container that provides basic DI support.
    2. **ApplicationContext**: A more advanced container that builds on BeanFactory and adds more enterprise-specific functionality like event propagation, declarative mechanisms to create a bean, and various means to look up beans.
* **What is the difference between BeanFactory and ApplicationContext?**
  + **Answer**: BeanFactory is a basic IoC container that provides basic dependency injection. ApplicationContext extends BeanFactory with additional features such as easier integration with Spring’s AOP features, message resource handling (for i18n), event propagation, and declarative mechanisms to create a bean.
* **How does Spring’s Bean Lifecycle work?**
  + **Answer**: The lifecycle of a Spring bean includes several stages:
    1. **Instantiation**: The container instantiates the bean.
    2. **Populating Properties**: Dependencies are injected into the bean.
    3. **BeanNameAware**: If the bean implements BeanNameAware, the factory calls setBeanName().
    4. **BeanFactoryAware**: If the bean implements BeanFactoryAware, the factory calls setBeanFactory().
    5. **ApplicationContextAware**: If the bean implements ApplicationContextAware, the setApplicationContext() method is called.
    6. **Pre-initialization**: If there are any BeanPostProcessors, the pre-initialization method postProcessBeforeInitialization() is called.
    7. **InitializingBean**: If the bean implements InitializingBean, the afterPropertiesSet() method is called. If the bean has a custom init method, it is called.
    8. **Post-initialization**: BeanPostProcessor postProcessAfterInitialization() method is called.
    9. **Bean is ready for use**.
    10. **Destruction**: If the bean implements DisposableBean, the destroy() method is called. If the bean has a custom destroy method, it is called.
* **What are Spring Bean scopes?**
  + **Answer**: Spring provides several bean scopes:
    1. **Singleton**: A single bean instance per Spring IoC container.
    2. **Prototype**: A new bean instance each time it is requested.
    3. **Request**: A single bean instance per HTTP request (web applications only).
    4. **Session**: A single bean instance per HTTP session (web applications only).
    5. **Global Session**: A single bean instance per global HTTP session (web applications only, typically for Portlet applications).
* **What is the difference between @Component, @Controller, @Service, and @Repository annotations in Spring?**
  + **Answer**:
    1. **@Component**: A generic stereotype for any Spring-managed component.
    2. **@Controller**: A specialized @Component used to define a Spring MVC controller.
    3. **@Service**: A specialized @Component used to define a service layer component.
    4. **@Repository**: A specialized @Component used to define a data access object (DAO) component.
* **What is a Proxy in Spring AOP?**
  + **Answer**: A proxy is an object that wraps another object to intercept method calls and introduce additional functionality such as logging, transaction management, or security checks. Spring AOP uses proxies to implement aspect-oriented programming.
* **Explain the different types of autowiring in Spring.**
  + **Answer**: Spring supports several types of autowiring:
    1. **No**: Default setting. No autowiring.
    2. **ByName**: Autowires by property name.
    3. **ByType**: Autowires by property type.
    4. **Constructor**: Autowires by constructor.
    5. **Autodetect**: Spring checks the constructors first, then by type.
* **What is Spring Boot and how does it relate to the Spring Framework?**
  + **Answer**: Spring Boot is an extension of the Spring Framework aimed at simplifying the development of new Spring applications. It provides pre-configured templates and out-of-the-box functionality to speed up the setup and development process. Spring Boot makes it easier to configure and run applications with minimal configuration.
* **What is the role of the DispatcherServlet in Spring MVC?**
  + **Answer**: The DispatcherServlet is the central servlet that receives and processes all incoming requests in a Spring MVC application. It coordinates the different components such as controllers, view resolvers, and exception handlers to process requests and generate responses.
* **How does Spring handle transaction management?**
  + **Answer**: Spring supports both programmatic and declarative transaction management. Declarative transaction management is the most commonly used approach and is achieved using annotations like @Transactional or XML configuration. Spring abstracts the transaction management code so that business logic can be transaction-aware without being dependent on transaction management APIs.
* **How do you configure a Spring application without using XML?**
  + **Answer**: You can configure a Spring application using Java-based configuration with @Configuration and @Bean annotations. Alternatively, Spring Boot provides application.properties or application.yml files for configuration.
* **What are the different ways to inject dependencies in Spring?**
  + **Answer**: Dependencies can be injected in the following ways:
    1. **Constructor Injection**: Using constructor arguments.
    2. **Setter Injection**: Using setter methods.
    3. **Field Injection**: Using @Autowired directly on fields (not recommended due to lack of immutability and difficulty in testing).
* **What is the @Primary annotation in Spring?**
  + **Answer**: The @Primary annotation is used to give a higher preference to a bean when multiple beans of the same type exist. It marks a bean as the primary bean to be injected in such scenarios.
* **What is the @Qualifier annotation in Spring?**
  + **Answer**: The @Qualifier annotation is used along with @Autowired to specify which bean should be injected when multiple beans of the same type are present in the Spring context.
* **Can you create a Singleton bean in a Prototype-scoped bean in Spring?**
  + **Answer**: Yes, you can. A singleton-scoped bean can inject a prototype-scoped bean. This can be achieved using @Scope("prototype") annotation on the prototype bean and ensuring the singleton bean retrieves the prototype bean from the ApplicationContext or using a scoped proxy.
* **What is a scoped proxy and when would you use it?**
  + **Answer**: A scoped proxy is a way to inject beans with a specific scope (e.g., request, session) into beans with a different scope (e.g., singleton). It is often used in web applications to manage session or request scoped beans. Spring creates a proxy that can hold the target bean, ensuring the correct scope is used.
* **Explain the @Lazy annotation in Spring.**
  + **Answer**: The @Lazy annotation is used to mark a bean for lazy initialization. This means the bean will only be created and initialized when it is first requested, rather than at application startup.
* **What is Aspect-Oriented Programming (AOP) in Spring?**
  + **Answer**: AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns (e.g., logging, transaction management, security). In Spring, AOP is implemented using proxies that can intercept method calls and apply additional behavior before or after method execution.
* **What are the different types of advice in Spring AOP?**
  + **Answer**: The different types of advice in Spring AOP are:
    1. **Before advice**: Executed before a join point (method execution).
    2. **After returning advice**: Executed after a join point completes normally.
    3. **After throwing advice**: Executed if a method exits by throwing an exception.
    4. **After (finally) advice**: Executed after a join point completes, whether it completes normally or by throwing an exception.
    5. **Around advice**: Executed before and after the method execution.
* **How do you enable annotation-driven transaction management in Spring?**
  + **Answer**: Annotation-driven transaction management can be enabled by adding @EnableTransactionManagement in a configuration class. Methods that require transaction management can be annotated with @Transactional.
* **What is Spring Boot Actuator?**
  + **Answer**: Spring Boot Actuator provides production-ready features to help you monitor and manage your application. It includes a set of endpoints that can be used to gather various metrics, monitor your application health, and trace requests.
* **How do you create a custom Spring Boot starter?**
  + **Answer**: A custom Spring Boot starter can be created by:
    1. Creating a new project and defining dependencies.
    2. Writing auto-configuration classes using @Configuration.
    3. Creating a spring.factories file in META-INF directory to register the auto-configuration classes.
    4. Publishing the starter as a library.
* **How can you test Spring applications?**
  + **Answer**: Spring applications can be tested using:
    1. **Unit Testing**: Using frameworks like JUnit or TestNG, along with Mockito for mocking.
    2. **Integration Testing**: Using @SpringBootTest, @ContextConfiguration, and other Spring test annotations to load the application context.
    3. **MockMvc**: For testing Spring MVC controllers.
* **What is Spring’s support for property placeholders?**
  + **Answer**: Spring supports property placeholders using the @Value annotation to inject values from properties files. Additionally, PropertySourcesPlaceholderConfigurer can be used in XML configuration to achieve the same.
* **What are some best practices for designing Spring applications?**
  + **Answer**: Best practices for designing Spring applications include:
    1. Use constructor injection over field injection for better testability and immutability.
    2. Use @Configuration and Java-based configuration over XML configuration.
    3. Keep configuration and business logic separate.
    4. Use Spring profiles to manage environment-specific configurations.
    5. Avoid circular dependencies.
    6. Leverage Spring Boot for rapid development and convention over configuration.
* **How does Spring handle exceptions in an application?**
  + **Answer**: Spring provides various ways to handle exceptions:
    1. **@ExceptionHandler**: Used to handle exceptions at the controller level.
    2. **@ControllerAdvice**: Global exception handling across all controllers.
    3. **HandlerExceptionResolver**: Used to resolve exceptions throughout the application.
* **How can you inject a collection (List, Set, Map) in Spring?**
  + **Answer**: Collections can be injected using @Autowired or @Resource annotations along with @Qualifier if necessary. For example, a list of beans can be injected as follows:

java

@Autowired

private List<MyBean> myBeanList;

* **What is the @Required annotation in Spring?**
  + **Answer**: The @Required annotation is used on setter methods to indicate that the property must be injected. If the property is not set, the container will throw a BeanInitializationException. Note that this annotation is deprecated as of Spring 5, and it is recommended to use constructor injection or explicit checks in setters instead.
* **Explain the use of @PostConstruct and @PreDestroy annotations.**
  + **Answer**: @PostConstruct is used on a method to be called after the bean has been initialized, whereas @PreDestroy is used on a method to be called before the bean is destroyed. These annotations are part of the JSR-250 standard.
* **How do you configure a bean in Spring using annotations?**
  + **Answer**: Beans can be configured using annotations like @Component, @Service, @Repository, and @Controller for different types of beans. These beans are then discovered through classpath scanning using @ComponentScan. Dependencies are injected using @Autowired or @Inject.
* **How can you define custom scopes in Spring?**
  + **Answer**: Custom scopes can be defined by implementing the Scope interface and registering the custom scope with the ConfigurableBeanFactory using the registerScope method. For example:

java

@Component

public class CustomScopeConfigurer implements BeanFactoryPostProcessor {

@Override

public void postProcessBeanFactory(ConfigurableListableBeanFactory beanFactory) throws BeansException {

beanFactory.registerScope("customScope", new CustomScope());

}

}

* **What is the @Scope annotation and how is it used?**
  + **Answer**: The @Scope annotation is used to define the scope of a Spring bean. It can be applied at the class level or on factory methods in @Configuration classes. For example:

java

@Component

@Scope("prototype")

public class MyPrototypeBean { }

* **How does Spring manage transactions in a multi-threaded environment?**
  + **Answer**: Spring manages transactions using thread-local storage, ensuring that each thread has its own transaction context. This is achieved using transaction propagation settings and proper configuration of the transaction manager. Additionally, the @Transactional annotation can be used to specify the transactional behavior.
* **What are transaction propagation behaviors in Spring?**
  + **Answer**: Spring supports various transaction propagation behaviors:
    1. **REQUIRED**: Supports a current transaction; creates a new one if none exists.
    2. **REQUIRES\_NEW**: Suspends the current transaction and creates a new one.
    3. **MANDATORY**: Supports a current transaction; throws an exception if none exists.
    4. **NESTED**: Executes within a nested transaction if a current transaction exists.
    5. **SUPPORTS**: Supports a current transaction but doesn't require one.
    6. **NOT\_SUPPORTED**: Does not support a current transaction; suspends the current transaction if it exists.
    7. **NEVER**: Does not support a current transaction; throws an exception if one exists.
* **How does Spring Boot simplify the configuration of Spring applications?**
  + **Answer**: Spring Boot simplifies configuration by providing:
    1. **Auto-configuration**: Automatically configures Spring and third-party libraries based on the project's dependencies.
    2. **Starter POMs**: A set of dependency descriptors that you can include in your application to get a working project setup quickly.
    3. **Embedded servers**: Allows running web applications with embedded servers like Tomcat, Jetty, or Undertow.
    4. **Convention over configuration**: Provides sensible defaults to reduce the need for extensive configuration.
* **How do you create a RESTful web service using Spring Boot?**
  + **Answer**: A RESTful web service can be created using @RestController and @RequestMapping annotations in Spring Boot. Here’s a simple example:

java

@RestController

@RequestMapping("/api")

public class MyRestController {

@GetMapping("/hello")

public String sayHello() {

return "Hello, World!";

}

}

* **What is Spring Cloud and how does it relate to microservices?**
  + **Answer**: Spring Cloud is a set of tools and frameworks designed to support the development and deployment of microservices. It builds on top of Spring Boot and provides features like configuration management, service discovery, circuit breakers, routing, and load balancing, essential for building resilient microservices.
* **What is a Spring BeanFactoryPostProcessor and how is it used?**
  + **Answer**: A BeanFactoryPostProcessor is a bean that modifies the bean definitions before the container instantiates any beans. It is used for tasks like changing bean properties, adding new bean definitions, or modifying existing ones.
* **What is the Environment abstraction in Spring?**
  + **Answer**: The Environment abstraction in Spring provides a way to access properties and profiles in a type-safe manner. It allows you to read property values, check active profiles, and manage environment-specific configurations.
* **What is the Profile annotation in Spring?**
  + **Answer**: The @Profile annotation is used to specify that a bean or configuration should only be loaded in certain environments. It can be applied to @Component classes, @Configuration classes, or individual @Bean methods. For example:

java

@Profile("dev")

@Configuration

public class DevConfig {

// Development-specific beans

}

* **Explain the @Conditional annotation in Spring.**
  + **Answer**: The @Conditional annotation allows conditional bean registration based on specified criteria. It can be used to enable or disable beans under certain conditions. For example:

java

@Configuration

@ConditionalOnProperty(name = "feature.enabled", havingValue = "true")

public class FeatureConfig {

// Beans that are only created when the feature is enabled

}

* **What is the difference between @Autowired and @Inject in Spring?**
  + **Answer**: Both @Autowired (from Spring) and @Inject (from JSR-330) are used for dependency injection. @Autowired has more features, such as required and optional dependencies using the required attribute. @Inject is more standard and doesn't have these additional attributes.
* **How do you handle circular dependencies in Spring?**
  + **Answer**: Circular dependencies can be handled by:
    1. **Setter Injection**: Instead of constructor injection, use setter injection.
    2. **Lazy Initialization**: Use @Lazy on one of the beans.
    3. **ObjectFactory**: Use ObjectFactory or Provider to lazily resolve dependencies.
* **What is @Value used for in Spring?**
  + **Answer**: @Value is used to inject values from properties files into Spring beans. For example:

java

Copy code

@Value("${property.name}")

private String propertyName;

* **How do you create and register a custom BeanPostProcessor?**
  + **Answer**: A custom BeanPostProcessor can be created by implementing the BeanPostProcessor interface and overriding postProcessBeforeInitialization and postProcessAfterInitialization methods. It can be registered as follows:

java

Copy code

@Component

public class CustomBeanPostProcessor implements BeanPostProcessor {

@Override

public Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {

// Custom logic before initialization

return bean;

}

@Override

public Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {

// Custom logic after initialization

return bean;

}

}

* **What is the purpose of @Order in Spring?**
  + **Answer**: The @Order annotation is used to define the order of execution for components, such as BeanPostProcessor or HandlerInterceptor. Lower values have higher priority. For example:

java

Copy code

@Component

@Order(1)

public class FirstBeanPostProcessor implements BeanPostProcessor {

// Implementation

}

@Component

@Order(2)

public class SecondBeanPostProcessor implements BeanPostProcessor {

// Implementation

}

* **How do you programmatically manage transactions in Spring?**
  + **Answer**: Programmatic transaction management is done using the PlatformTransactionManager and TransactionTemplate or TransactionStatus. For example:

java

Copy code

@Autowired

private PlatformTransactionManager transactionManager;

public void executeInTransaction() {

TransactionTemplate transactionTemplate = new TransactionTemplate(transactionManager);

transactionTemplate.execute(new TransactionCallbackWithoutResult() {

@Override

protected void doInTransactionWithoutResult(TransactionStatus status) {

// Transactional code

}

});

}

* **Explain the @Transactional propagation attribute and its importance.**
  + **Answer**: The propagation attribute defines the transactional behavior of a method when an existing transaction is already in progress. Its importance lies in determining how transaction boundaries are handled. For instance:
    1. **REQUIRED**: Joins the existing transaction or creates a new one if none exists.
    2. **REQUIRES\_NEW**: Suspends the existing transaction and creates a new one.
    3. **NESTED**: Executes within a nested transaction.
* **How do you configure different profiles in Spring Boot?**
  + **Answer**: Different profiles can be configured using the application-{profile}.properties or application-{profile}.yml files. Profiles can be activated using the spring.profiles.active property. For example:

properties

# application-dev.properties

server.port=8081

java

Copy code

@SpringBootApplication

public class Application {

public static void main(String[] args) {

SpringApplication.run(Application.class, "--spring.profiles.active=dev");

}

}

* **What is Spring Boot DevTools and how does it help in development?**
  + **Answer**: Spring Boot DevTools is a set of tools to enhance developer productivity. It includes features like automatic restarts, live reload, and configurations to disable caching, which helps in rapid development by providing immediate feedback on code changes.
* **What is the @ConditionalOnProperty annotation and how is it used?**
  + **Answer**: The @ConditionalOnProperty annotation is used to conditionally enable or disable beans based on the presence of a specific property. For example:

java

@Configuration

@ConditionalOnProperty(name = "feature.enabled", havingValue = "true")

public class FeatureConfig {

// Beans that are only created when the feature is enabled

}

* **How do you create and register a custom property editor in Spring?**
  + **Answer**: A custom property editor can be created by extending PropertyEditorSupport and registered in the CustomEditorConfigurer:

java

public class CustomDateEditor extends PropertyEditorSupport {

private final SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");

@Override

public void setAsText(String text) throws IllegalArgumentException {

try {

setValue(dateFormat.parse(text));

} catch (ParseException e) {

throw new IllegalArgumentException("Invalid date format");

}

}

}

java

@Configuration

public class CustomEditorConfig {

@Bean

public CustomEditorConfigurer customEditorConfigurer() {

CustomEditorConfigurer configurer = new CustomEditorConfigurer();

Map<Class<?>, Class<? extends PropertyEditor>> editors = new HashMap<>();

editors.put(Date.class, CustomDateEditor.class);

configurer.setCustomEditors(editors);

return configurer;

}

}

* **What is the role of HandlerInterceptor in Spring MVC?**
  + **Answer**: HandlerInterceptor is used to intercept requests in a Spring MVC application. It provides three main methods:
    1. preHandle: Called before the actual handler is executed.
    2. postHandle: Called after the handler is executed but before the view is rendered.
    3. afterCompletion: Called after the complete request is finished, for cleanup purposes.
* **What are Spring Profiles and how do they work?**
  + **Answer**: Spring Profiles provide a way to segregate parts of your application configuration and make it only available in certain environments. Beans can be defined to be active only when a specific profile is active using the @Profile annotation. For example:

java

@Profile("dev")

@Bean

public DataSource devDataSource() {

// Configuration for development data source

}

* **Explain the difference between @Component and @Bean.**
  + **Answer**: @Component is a class-level annotation used to mark a class as a Spring-managed component, and it is detected through classpath scanning. @Bean is a method-level annotation in a @Configuration class that defines a bean which is managed by the Spring container. @Component is used for automatic detection, while @Bean is used for manual creation and configuration.
* **What is the role of @RestControllerAdvice in Spring?**
  + **Answer**: @RestControllerAdvice is used to handle exceptions across the whole application in RESTful web services. It combines @ControllerAdvice and @ResponseBody to provide global exception handling for REST controllers. For example:

java

@RestControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(ResourceNotFoundException.class)

@ResponseStatus(HttpStatus.NOT\_FOUND)

public ErrorResponse handleResourceNotFound(ResourceNotFoundException ex) {

return new ErrorResponse("Resource not found", ex.getMessage());

}

}