Download VMware 2V0-72.22 Exam Dumps Free

Shared by Cameron on 17-06-2026

For More Free Questions and Preparation Resources

Check the Links on Last Page

Question 1

Question Type: MultipleChoice

Which statement is true? (Select the best answer.)

Options:

A- @ActiveProfiles is a class-level annotation that is used to instruct the Spring TestContext Framework to record all application events that are published in the ApplicationContext during the execution of a single test.
B- @ActiveProfiles is a class-level annotation that you can use to configure how the Spring TestContext Framework is bootstrapped.
C- @ActiveProfiles is a class-level annotation that you can use to configure the locations of properties files and inlined properties to be added to the set of PropertySources in the Environment for an ApplicationContext loaded for an integration test.
D- @ActiveProfiles is a class-level annotation that is used to declare which bean definition profiles should be active when loaded an ApplicationContext for an integration test.

Answer:

D

Explanation:

A bean definition profile is a named logical grouping of bean definitions that can be activated or deactivated in different environments. For example, we can define different profiles for development, testing, and production environments. Spring provides the @Profile annotation to mark a bean definition or a configuration class as belonging to a specific profile. To activate a profile, we can use the spring.profiles.active property or the ConfigurableEnvironment.setActiveProfiles() method.

The @ActiveProfiles annotation is a class-level annotation that is used in conjunction with the Spring TestContext Framework to declare which profiles should be active when loading an ApplicationContext for an integration test. This annotation can be applied to any class annotated with @ContextConfiguration or a meta-annotation that is composed of @ContextConfiguration.

Question 2

Question Type: MultipleChoice

Which two statements are true regarding a Spring Boot "fat" JAR? (Choose two.)

Options:

A- The 'fat' JAR contains both the class files and the source files for your project.
B- The 'fat' JAR requires an external Servlet container.
C- The 'fat' JAR contains compiled classes and dependencies that your code needs to run.
D- The 'fat' JAR can contain multiple embedded application servers.
E- The 'fat' JAR is created by the Spring Boot Maven plugin or Gradle plugin.

Answer:

C, E

Question 3

Question Type: MultipleChoice

Which two options are valid optional attributes for Spring's @Transactional annotation? (Select two.)

Options:

A- isolation
B- writeOnly
C- nestedTransaction
D- readWrite
E- propagation

Answer:

A, E

Explanation:

https://www.baeldung.com/transaction-configuration-with-jpa-and-spring

@Transactional(propagation = Propagation.SUPPORTS, readOnly = true)

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/annotation/Transactional.html

Question 4

Question Type: MultipleChoice

Which strategy is correct for configuring Spring Security to intercept particular URLs? (Select the best answer.)

Options:

A- The URLs can be specified via configuration (using authorizeRequests () and request matchers), with the most specific rule first and the least specific last.
B- Spring Security can obtain URLs from Spring MVC controllers, the Spring Security configuration just needs a reference to the controller to be protected.
C- The URLs are specified in a special properties file, used by Spring Security.
D- The URLs can be specified via configuration (using authorizeRequests () and request matchers), with the least specific rule first and the most specific last.

Answer:

A

Explanation:

Spring Security provides a fluent API for configuring web security based on URL patterns and request matchers. The authorizeRequests() method returns an expression that allows specifying access rules for different URLs using methods such as antMatchers(), regexMatchers(), mvcMatchers(), etc. The order of these rules matters, as they are evaluated from top to bottom. Therefore, it is recommended to put the most specific rules first and the least specific ones last.

Question 5

Question Type: MultipleChoice

Which two statements are correct regarding the Actuator info endpoint? (Choose two.)

Options:

A- It provides configuration options through which only an authenticated user can display

application information.
B- It is not enabled by default.
C- It can be used to display arbitrary application information.
D- It can be used to change a property value on a running application.
E- Typically it is used to display build or source control information.

Answer:

B, C

Question 6

Question Type: MultipleChoice

Refer to the exhibit.

```
@Configuration
public class AppConfig {
     @Bean
     public ClientService clientService() {
          return new ClientServiceImpl();
     }
}
```

What is the id/name of the declared bean in this Java configuration class? (Choose the best answer.)

Options:

A- clientServiceImpl (starting with lowercase ''c'')
B- clientServiceImpl (starting with uppercase ''C'')
C- clientService (starting with lowercase ''c'')
D- ClientService (starting with uppercase ''C'')

Answer:

C

Explanation:

This is true because the id/name of a bean declared by the @Bean annotation is derived from the name of the method that returns the bean. In this case, the method name is clientService, so the bean name will be clientService as well. By default, Spring uses a lower-case first letter for bean

names, unless explicitly specified otherwise by using the name attribute of the @Bean annotation. For example, we can use @Bean(name = ''ClientService'') to change the bean name to ClientService (starting with uppercase ''C'').

Question 7

Question Type: MultipleChoice

Which two annotations indicate that the transaction for a transactional test method should be committed after the test method has completed? (Choose two.)

Options:

A- @SqlMergeMode(false)
B- @Rollback(false)
C- @Commit
D- @Sql(alwaysCommit=true)
E- @Transactional(commit=true)

Answer:

B, C

Question 8

Question Type: MultipleChoice

Given an ApplicationContext containing three bean definitions of type Foo with bean ids foo1, foo2, and foo3, which three @Autowired scenarios are valid and will allow the ApplicationContext to initialize successfully? (Select three.)

Options:

A- @Autowired public void setFoo (Foo foo) {...}
B- @Autowired @Qualifier (''foo3'') Foo foo;
C- @Autowired public void setFoo (@Qualifier (''foo1'') Foo foo) {...}
D- @Autowired private Foo foo;
E- @Autowired private Foo foo2;
F- @Autowired public void setFoo(Foo foo2) {...}

Answer:

B, C, F

Explanation:

The @Autowired annotation can be used to inject a dependency into a field, a constructor, or a setter method. However, if there are multiple beans of the same type in the application context, Spring will not be able to determine which one to inject by default. To resolve this ambiguity, we can use the @Qualifier annotation to specify the bean id of the desired dependency. Alternatively, we can use the bean id as the name of the field or the parameter of the setter method, and Spring will match it with the corresponding bean.

Question 9

Question Type: MultipleChoice

Refer to the exhibit.

```
public class ClientServiceImpl implements ClientService{
      @Transactional(propagation=Propagation.REQUIRED)
      public void update() {
            update2();
      }
      @Transactional(propagation=Propagation.REQUIRES_NEW)
      public void update2() {
      }
}
```

Assume that the application is using Spring transaction management which uses Spring AOP internally.

Choose the statement that describes What is happening when the update1 method is called? (Choose the best answer.)

Options:

A- There are 2 transactions because REQUIRES_NEW always runs in a new transaction.
B- An exception is thrown as another transaction cannot be started within an existing transaction.
C- There is only one transaction because REQUIRES_NEW will use an active transaction if one already exists.
D- There is only one transaction initiated by update1() because the call to update2() does not go through the proxy.

Answer:

D

Explanation:

When using Spring transaction management with annotation-driven mode, the @Transactional annotation will be processed by a TransactionInterceptor that implements the AOP advice interface. This interceptor will be applied to the target bean through a proxy that implements the same interface as the target bean. Therefore, when a method of the target bean is called from outside, it will actually invoke the proxy method, which will delegate to the interceptor and then to the actual target method.

However, when a method of the target bean is called from within the same bean, it will not go through the proxy and thus bypass the interceptor logic. In this case, when update1() calls update2(), it will not start a new transaction as specified by REQUIRES_NEW propagation level, but rather join the existing transaction initiated by update1() itself.

Question 10

Question Type: MultipleChoice

Which two statements are correct regarding Spring Boot auto-configuration? (Choose two.)

Options:

A- Auto-configuration uses @Conditional annotations to constrain when it should apply.
B- Auto-configuration could apply when a bean is missing but not when a bean is present.
C- Auto-configuration is applied by processing candidates listed in META-INF/spring.factories.
D- Auto-configuration could apply when a bean is present but not when a bean is missing.
E- Auto-configuration is applied before user-defined beans have been registered.

Answer:

A, C

Question 11

Question Type: MultipleChoice

What two options are auto-configured Spring Boot Actuator HealthIndicators? (Choose two.)

Options:

A- DataSourceHealthIndicator
B- GoogleCloudDataStoreHealthIndicator
C- DynamoDBHealthIndicator
D- RabbitHealthIndicator
E- OktaHealthIndicator

Answer:

A, D

Question 12

Question Type: MultipleChoice

Which two annotations are meta-annotations on the @SpringBootApplication composed annotation? (Choose two.)

Options:

A- @Configuration
B- @ComponentScan
C- @SpringBootConfiguration
D- @SpringApplication
E- @AutoConfiguration

Answer:

A, B

Explanation:

A . @Configuration This annotation indicates that the class has @Bean definition methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtime. B. @ComponentScan This annotation configures component scanning directives for use with @Configuration classes. Provides support parallel with Spring XML's context:component-scan element.