Response

Container

Question 1

Answer 2 is correct. Those beans are anonymous because no id is supplied explicitly. Thus Spring container generates a unique id for that bean. It uses the fully qualified class name and appends a number to them. However, if you want to refer to that bean by name, through the use of the `ref` element you must provide a name (see Naming Beans section of the Spring reference manual). To be correct, the 2nd bean has to declare a `jpaDao` id attribute in order to be reference by the `repository` property of the first bean.

Question 2

Answers 1 and 4 are correct.

1. To set bean's property with the p:propertyName shortcut, you have to declare the http://www.springframework.org/schema/p in your xml configuration file. No xsd is required.
2. The bean is anonymous. Spring generates a unique id: `com.spring.service.BankServiceImpl#0`
3. To reference another bean with the p namespace, you have to use the p:propertyName-ref syntax
4. Due to the above explanation, NationalBank is not a bean reference, so it is a simple String and thus a scalar value.

Question 3

Correct answer is 3.
The @Bean annotation defines a String bean with the id "clientRepository". JpaClientRepository is the implementation class of the bean. The data source is injected and is not declared in this class.

Question 4

The only possible answer is the number 3.

1. The <util:constant static-field="constant name"/> tag enables to reference a Java constant or enumeration into a spring configuration file
2. ConstantPlaceholderConfigurer does not exist. You may think about the PropertyPlaceholderConfigurer bean post processor.
3. The <context:property-placeholder location="file:/myApp.properties" /> tag activates the replacement of ${...} placeholders, resolved against the specified properties file.
4. The c: namespace is for simplifying constructor syntax (since Spring 3.1) and don't provide such feature.

Question 5

The statements number 5 is right.

1. You may auto-wiring properties by constructor, setter or properties in the same bean
2. The <constructor-arg> tag helps to instanciated a bean without default or no-args constructor
3. The <constructor-arg> tag could take type and index to reduce ambiguity, but not name which requires debug symbols.

Question 6

Answers 1, 3, 4 and 5 are rights.

1. The @PostConstruct, @PreDestroy and @Resource annotations are defined in the JSR-250
2. They belong to the javax.annotation package. You should add an external jar to use them in Java 5. Java 6 and 7 integrates them.
3. The <context:component-scan> automatically detects stereotyped classes and turns on the <context:annotation-config>
4. The <context:annotation-config > activates the Spring infrastructure for various annotations to be detected in bean classes, including the JSR 250 annotations
5. The CommonAnnotationBeanPostProcessor supports common Java annotations out of the box, in particular the JSR-250 annotations.

Question 7

Correct answer in the number 3.

1. In a web application, the ContextLoaderListener is in charge to create an WebApplicationContext.
2. In an integration test based on Spring, the SpringJUnit4ClassRunner creates the application context for you. The @ContextConfiguration annotation allows to specified application context configuration files.
3. In a main method, you have to instanciated a class implementing the ApplicationContext interface (examples: ClassPathXmlApplicationContext or FileSystemXmlApplicationContext)

Question 8

Answer number 4 is right.

1. `When using the ClassPathXmlApplicationContext,` the classpath: prefix is default one so you could omit it
2. In a Spring location resource, package separator is a slash and not a dot. Thus the com/example/myapp/config.xml syntax has to be used.
3. ClassPathXmlApplicationContext starts looking from root of the classpath regardless of whether specify "/"

Question 9

Answers number 3 and 4 are valid.

1. The @Autowired annotation has no name property, just a required one.
2. Autowiring a field, the @Inject or the @Autowired or the @Resource annotations are mandatory.
3. The @Qualifier("name") annotation complements the use of the @Autowired annotation by specifying the name of the bean to inject
4. When 2 beans are eligible to auto-injection, Spring uses the field name to select the appropriate one.

Question 10

Answers number 1 and 4 are valid.

1. With modern mock API like Mockito or EasyMock, interfaces are not mandatory for mocking or stubbing the service. But using interface remains easier when you have to manually mock the service in unit test.
2. Auto-injection is possible with class. Spring uses CGLIB.
3. Dependency checking is an advantage of dependencies injection.
4. The Inversion of Control pattern requires an interface to separate 2 classes. This pattern provides code more flexible, unit testable, loosely coupled and maintainable.

Question 11

Correct answers: 4

1. In the bean lifecycle, method annotated with @PostConstruct is called after the properties set step and the BeanPostProcessors#postProcessBeforeInitialization step
2. Destroy methods of prototype beans are never called
3. In the bean lifecycle, the afterPropertiesSet callback method of the InitializingBean is called after the method annotated with the @PostConstruct annotation and before the init-method declared in the XML configuration file.
4. In the bean lifecycle, the method annotated with the @PreDestroy annotation is called before the destroy callback of the DisposableBean interface and before the destroy-method declared in the XML configuration file.

Question 12

Correct answers are 1 and 2.

1. In order to be taken into account by Spring, the ApplicationConfig class has to be annotated with the @Configuration annotation
2. Default or no-arg constructor is mandatory. Here, the provided constructor with a dataSource parameter is not taken into account
3. The bean name is clientRepository. The name property of the @Bean annotation is specified thus the method name jpaClientRepository is ignored.
4.

Question 13

Correct answers are 3 and 4

1. Use <tx:annotation-driven /> to enable @Transactional annotation scanning
2. Use <aop:aspectj-autoproxy /> to enable detection of @Aspect bean
3. Turns on <context:annotation-config /> or <context:component-scan /> to enable @Autowiring annotation
4. Turns on <context:component-scan /> to enable @Component annotation scanning

Test

Question 14

The only correct answer is number 3.

1. The Spring context is cached across tests unless you use @DirtiesContext annotation
2. With the Spring test module, dependency injection is available in test case. So you may auto-wired the bean to test
3. By default, a @ContextConfiguration annoted class inherits the spring context configuration file locations defined by an annotated superclass. The inheritLocations of this attribute allows to change this default behavior.
4. If no context configuration file is provided to the @ContextConfiguration annotation, Spring use a file convention naming. It try to load a file named with the test class name and suffices by the "-context.xml" suffice (i.e. MyDaoTest-context.xml)

Question 15

Correct answers are 1, 3 and 4.

What are the main advantage(s) for using Spring when writing integration tests?

1. More than testing multiple classes together, integration test may allow to test your spring configuration file and/or to reuse it.
2. Mocking or stubbing is more frequent in unit tests than in integration tests. And Spring does not provide any implementation or abstraction of mock framework.
3. The framework may create and roll back a transaction for each test method. Default rollback policy could be change by using the @TransactionConfiguration annotation. And default mode could be overridden by the @Rollback annotation.
4. DependencyInjectionTestExecutionListener provides support for dependency injection and initialization of test instances.

Question 16

The correct answer is the number 3.

What are the main advantage(s) for using Spring when writing unit tests?

1. You don't need Spring container to writer unit test
2. Refer to the answer number 1.
3. The org.springframework.mock package provides mock classes like MockHttpSession or MockHttpContext. They could be helpful for unit test in the presentation layer and when you don't use any mock framework such as Mockity or EasyMock.

Question 17

Answer 5 is correct.

What is right about the spring test module?

1. The spring test module does not provide an abstraction layer for open source mock frameworks like EasyMock, JMock or Mockito
2. The @Mock annotations comes from the Mockito framework
3. The spring test module does not provide mechanism to generate mock objects at runtime

Question 18

Correct statements are number 1 and 4.

1. The transactionManager property of the @TransactionConfiguration annotation enable to set the bean name of the PlatformTransactionManager that is to be used to drive transactions.
2. Method annotated with @Before is executed inside the test's transaction. You have to use the @BeforeTransaction to execute code outside the test's transaction.
3. The REQUIRES_NEW propagation suspends the current test's transaction then creates a new transaction that will be used to execute the service. A commit at the service level could not be changed by the test.
4. The transaction for the annotated method should be rolled back after the method has completed.

AOP

Question 19

The correct answer is the number 4.

Considering 2 classes AccountServiceImpl and ClientServiceImpl. Any of these 2 classes inherits from each other. What is the result of the pointcut expressions?

```
    execution(* *..AccountServiceImpl.update(..))
&& execution(* *..ClientServiceImpl.update(..))
```

Poincut expression could not satisfied both first and second execution point. Do not confuse the && operator and || operator.

Question 20

Correct answer is the number 4.

Due to the proxy-based nature of Spring's AOP framework, protected methods are by definition not intercepted, neither for JDK proxie nor for CGLIB proxies. As a consequence, any given pointcut will be matched against public methods only!

To intercept private and protected methods, AspecJ weaving should be used instead of the Spring's proxy-bases AOP framework.

Question 21

The 2 correct statements are 1 and 5.

What are the 2 correct statements about AOP proxy.

1. An object created by the AOP framework in order to implement the aspect contracts
2. If the target object does not implement any interfaces then a CGLIB proxy will be created. You could also use CGLIB proxy instead of JDK dynamic proxy
3. If the target object does not implement any interfaces then a CGLIB proxy will be created.
4. When CGLIB proxy is used, final methods cannot be advised, as they cannot be overridden.
5. AOP Proxies are created by the AbstractAutoProxyCreator#postProcessAfterInitialization method.

Question 22

The answer number 2 is correct.

1. A before advice could throw an exception
2. An after throwing advice is executed if a method exits by throwing an exception
3. An advice that executes before a join point is named a before advice
4. Spring supports after throwing advices

Question 23

Correct answer: 4

1. This is an after (finally) advice
2. This is an around advice
3. This is a before advice
4. True

Question 24

Correct answer: 1

1. Definition of an advice
2. Definition of a joint point
3. Represents nothing
4. Definition of a point cut

Question 25

Correct answer: 2

1. Definition of an advice
2. Definition of a pointcut
3. Represents nothing

Question 26

Correct answers: 1, 3

Select methods that match with the following pointcut:

execution(* com.test.service..*.*(*))

1. True
2. The pattern (*) matches a method taking one parameter of any type
3. The com.test.service.account sub-package matches the pointcut
4. False for the same reason as answer number 2.

Question 27

Correct answers: 2

1. Interception of constructors requires the use of Spring-driven native AspectJ weaving instead of Spring's proxy-based AOP framework
2. The @annotation designator enables to select methods that are annotated by a given annotation
3. The staticinitialization AspectJ designator is not supported by Spring AOP
4. Pointcut expressions can be combined using &&, || and !

Question 28

Correct answers: 1

1. The execution of all public method
2. The * return type pattern indicates any return value or void
3. The (..) param pattern indicates 0, 1 or many parameters

4. No package name is specified. So classes of any package could match.

Data Access

Question 29

Correct answers: 2 , 4

1. Spring exception translation mechanism has nothing to do with read-only transaction
2. Read-only transaction prevents Hibernate from flushing its session. Hibernate do not do dirty checking and it increases its performance.
3. No
4. When jdbc transaction is marked as read-only, Oracle only accepts SELECT SQL statements.

Question 30

Correct answers: 1, 3, 4

1. JDBC is supported: JdbcTemplate, JDBCException wrapper …
2. Some NoSQL databases are supports through the Spring Data project
3. Hibernate is supported: HibernateTemplate, AnnotationSessionFactoryBean …
4. JPA is supported: LocalEntityManagerFactoryBean, @PersistenceContext annotation support

Question 31

Correct answer: 1

1. A JdbcTemplate requires a DataSource as input parameters
2. JdbcTemplate uses the probided datasource to open then close a JDBC connection
3. Callback methods of JdbcTemplate throw SQL Exception and Spring converts into DataAccessException
4. For example, the queryForInt method executes an SQL statement

Question 32

Correct answer: 3

1. RowMapper : result set parsing when need to map each row into a custom object
2. RowCallbackHandler : result set parsing without returning a result to the JdbcTemplate caller
3. ResultSetExtractor : for result set parsing and merging rows into a single object
4. ResultSetMapper : this class does not exist

Question 33

Correct answer: 3

1. False. This is supported by the AnnotationSessionFactoryBean using annotedClasses
2. False. This is supported by the AnnotationSessionFactoryBean using packagesToScan
3. True using mappingLocations
4. False

Transaction

Question 34

Correct answer: 2

1. <tx:jta-transaction-manager />
2. Id of the transaction manager bean could be customized (ie. txManager)
3. DataSourceTransactionManager is a transaction manager for a JDBC data source. HibernateTransactionManager may be used to manage transaction with Hibernate.
4. The AbstractPlatformTransactionManager has a defaultTimeout property that could be customized

Question 35

Correct answer: 2

In proxy mode, only external method calls coming in through the proxy are intercepted. In the code snippet, the add() method is self-invocated. This means that, the @Transactional annotation of the add() method is not interpreted. The REQUIRES_NEW propagation level is not taken into account. To summary, when the transferMoney() methods calls add() method directly, the transaction attributes of add() method are not used

Question 36

Correct answer: 1

1. The TransactionTemplate class provides an execute(TransactionCallback) method
2. The TransactionService class does not exists
3. The @Transactional annotation is for declarative transaction management

Question 37

Correct answer: 3

1. PROPAGATION_REQUIRED
2. PROPAGATION_NESTED
3. PROPAGATION_REQUIRES_NEW

Question 38

Correct answer: 2

1. False.
2. True
3. False
4. False

Sping @MVC

Question 39

Correct answer: 1

1. Spring does not allow to return an absolute path to the view
2. Controller could return a String that matches with a logical view name
3. A JstlView with the .jsp path (i.e. /WEB-INF/accountList.jsp)
4. void forward to the default view
5. null forward to the default view

Question 40

Correct answer: 2, 4

1. Spring MVC controllers are beans. So you can declare them into a Spring application context XML configuration file that could be loaded by the DispatcherServlet.
2. In the web.xml, you may declarer and a ContextLoaderListener and a DispatcherServlet that are in charge to load XML Spring configuration files. But you cannot declare controllers directly in these file.
3. The @Controller annotation may be used to annoted Spring MVC Controller beans that handle HTTP requests.
4. JSP is the View of the MVC Pattern. Thus this is not the right place to declare controllers.

Question 41

Correct answer: 3

1. HttpServletRequest and HttpServletResponse have to be mocked. Id of the account to display could be set into the http request parameters.

2. HttpServletRequest and HttpSession have to be mocked. Id of the account to display could be set into the http request parameters.

3. This method is not dependent of the servlet API. Id of the account to display may be directly passed through the call stack. Thus test methods are simplified.

4. The *@PathVariable* annotation has to be bound to a URI template variable. This is not the case.

Sping Security

Question 42

Correct answer: 4

1. @Secured annotation is a Spring Security annotation
2. @RolesAllowed is a JSR-250 annotation that is supported by Spring Security
3. Spring Security could be configured in a XML way to intercept particular URLs

Question 43

Correct answer: 1, 2, 3 and 4

Question 44

Correct answer: 2

1. You cannot mix EL and constant in the same configuration file
2. If more than one intercept-url matches, the top one is used
3. Ant pattern is used by default. But you can change to use regular expression.
4. Security rules may apply to request URL, request method (GET, POST …) but not to request parameters.

Remoting

Question 45

Correct answer: 4

1. No more interface to implement. RMI Client and Server could be POJO.
2. No more class to extend. RMI Client and Server could be POJO.
3. Spring Remoting wraps the cheched RemoteException into RuntimeExcpetion.
4. Object that are transferred via RMI are serializabled/unserializabled. So they have to implement the Serializable interface.

Question 46

Correct answers: 1, 3

1. HttpInvokerServiceExporter requires a HTTP web server to process incoming http request. Tomcat or Jetty is possible candidates. Spring also supports the Oracle/Sun's JRE 1.6 HTTP server.
2. Only the POST method is supported. Maybe due to the 256 characters limit of the GET method.
3. Spring comes with 2 http client implementations: for Commons HttpClient and classic JavaSE API. You can create a custom one by extending the AbstractHttpInvokerRequestExecutor class.
4. Does not support SOAP web service. Use the Spring web service module or use the JAX-WS or JAX-RPC remoting support.

JMS

Question 47

Correct answer: 2

1. The convertAndSend method sends a given object to a destination, converting the object to a JMS message.
2. The onMessage method does not exist.
3. The receiveAndConvert method receives a message synchronously then convert the message into an object
4. The setDefaultDestination method sets the destination to be used on send/receive operations that do not have a destination parameter.

Question 48

Correct answers: 1, 2 , 4

1. The javax.jms.MessageListener interface could be used with the SimpleMessageListenerContainer
2. The SessionAwareMessageListener interface could be used with DefaultMessageListenerContainer and SimpleMessageListenerContainer
3. Business code is required to handle and process the JMS message.
4. A JMS Listener could be a POJO. The name of the handler method to invoke has to be specified in the <jms:listener /> tag.

JMX

Question 49

Correct answers: 1, 2, 3, 4

1. The MBeanExporter class allow to expose any Spring bean as a JMX MBean
2. Exiting MBean could be declared as Spring bean. Then the <context:mbean-export /> directive enables their registration to the MBeanServer

3. Remote MBean could be access through a proxy
4. Implementations of the MBeanInfoAssembler interface do the job

Question 50

Correct answer: 3

1. @ManageAttribute exposes a bean's property (getter/setter) to JMX
2. @ManageOperation exposes a bean's method to JMX
3. @ManageResources identify a Spring bean as a JMX MBean