Core Spring 3.0 Certification Mock Exam

Question

Container

Question 1

Given the following Spring configuration file, what is the correct answer:

```
<bean class="com.spring.service.MyServiceImpl">
	<property name="repository" ref="jpaDao"/>
</bean>

<bean class="com.spring.repository.JpaDao"/>
```

1. The first declared bean MyServiceImpl is missing an id must be named myService
2. The second declared bean JpaDao is missing an id must be named jpaDao
3. Answers 1 and 2 are both rights
4. Answers 1 and 2 are both wrong

Question 2

Given the Spring configuration file, which are the correct statements?

```
<bean class="com.spring.service.BankServiceImpl"
                    p:bankName="NationalBank">
</bean>
```

1. The p namespace has to be declared
2. Bean id is bankServiceImpl
3. The BankServiceImpl references a NationalBank bean
4. NationalBank is a scalar value

Question 3

How is named the bean that is defined in the following configuration class. Select a single answer.

```
@Configuration
public class ApplicationConfig {

	@Autowired
	private DataSource dataSource;

	@Bean
	ClientRepository clientRepository() {
		ClientRepository accountRepository = new JpaClientRepository();
		accountRepository.setDataSource(dataSource);
		return accountRepository;
```

```
        }
}
```

1. JpaClientRepository
2. jpaClientRepository
3. clientRepository
4. `Two beans are defined : a data souce and a repository`

Question 4

How could you externalize constants from a Spring configuration file or a Spring annotation into a .properties file? Select one or more answers

1. By using the <util:constant /> tag
2. By declaring the ConstantPlaceholderConfigurer bean post processor
3. By using the <context:property-placeholder /> tag
4. By using the c: namespace

Question 5

What statement is not correct in live environment? Select a unique answer.

1. Constuctor and properties autowiring in the same bean are not compatible
2. A bean should have a default or a no-args constructor
3. The <constructor-arg> tag could take type, name and index to reduce ambiguity
4. None of the above
5. All of the above

Question 6

What are the right affirmations about the @PostConstruct, @Resource and the @PreDestroy annotations?

1. Those annotations are specified in the JSR-250
2. The Spring Framework embedded those annotation
3. The <context:component-scan> tag enable them
4. The <context:annotation-config > tag enable them
5. Declaring the CommonAnnotationBeanPostProcessor enable them

Question 7

What is/are typically case(s) where you usually need to manually instanciated an ApplicationContext?

1. In a web application
2. In an integration test running with the SpringJUnit4ClassRunner
3. In a standalone application started with a main method
4. None of the above

Question 8

Select the right statement about referring a Spring configuration file inside the package com.example.myapp in the below example?

```
ApplicationContext context = new
ClassPathXmlApplicationContext("classpath:/com.example.myapp.config.xml");
```

1. The classpath: prefix could be omit
2. Package name with dot is not well formatted using the dot character
3. The slash character preceding com.example could be omit
4. All of the above
5. None of the above

Question 9

How to auto-inject into a field a bean by its name? Select one or more response.

1. With the name attribute of the @Autowired annotation
2. By using the single @Qualifier annotation
3. By using both the @Autowired and the @Qualifier spring annotations
4. By using the @Autowired annotation and naming the field with the bean name

Question 10

What are the main advantages of using interfaces when designing business services? Select one or more answers.

1. Mocking or stubbing the service
2. Be able to use the Spring auto-injection
3. Can do dependency checking
4. Loosely coupled code

Question 11

Select one or many correct answers about spring bean life cycle.

1. The method annoted with @PostConstruct is called after bean instantiation and before properties setting of the bean
2. The method @PreDestroy of a prototype bean is called when the bean is garbage collected
3. The init() method declared in the init-method attribute of a bean is called before the afterPropertiesSet callback method of the InitializingBean interface
4. The method annotated with @PostConstruct is called before before the afterPropertiesSet callback method of the InitializingBean interface

Question 12

Given the following configuration class, what are correct affirmations? Select one or more answers.

```java
public class ApplicationConfig {

	private DataSource dataSource;

	@Autowired
	public ApplicationConfig(DataSource dataSource) {
		this.dataSource = dataSource;
	}

	@Bean(name="clientRepository")
	ClientRepository jpaClientRepository() {
		return new JpaClientRepository();
	}
}
```

1. @Configuration annotation is missing
2. Default or no-arg constructor is missing
3. @Bean name is ambiguous
4. @Bean scope is prototype

Question 13

What are the features of the XML <context: namespace? Select one or many answers.

1. @Transactional annotation scanning
2. @Aspect annotation detection enabling
3. @Autowired annotation enabling
4. @Component annotation scanning

Test

Question 14

Select one or more correct statements about developing integration test with Spring support.

1. A new Spring context is created for each test class
2. To get a reference on the bean you want to test, you have to call the getBean() method of the Spring context
3. Spring context configuration could be inherits from the super class
4. The Spring context configuration file has to be provided to the @ContextConfiguration annotation

Question 15

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

1. Reuse Spring configuration files of the application
2. Create mock or stub
3. Be able to use the rollback after the test pattern
4. Use dependency injection

Question 16

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

1. Reuse Spring configuration files of the application
2. Use dependency injection
3. Provide some mocks for servlet classes
4. All of the above
5. None of the above

Question 17

What is right about the spring test module?

1. It provides an abstraction layer for the main open source mock frameworks
2. Provides the @Mock annotation
3. It dynamically generates mock objects
4. All of the above
5. None of the above

Question 18

Select correct statement(s) about transactional support of the spring test module.

1. Transaction manager could be set within the @TransactionConfiguration annotation
2. Method annotated with @Before is executed outside of the test's transaction
3. Spring test may rollback the transaction of a service configured with the REQUIRES_NEW propagation
4. The transaction of a method annotated with the @Rollback annotation with its default values is rolled back after the method has completed

AOP

Question 19

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(..))
```

1. Matches pubic update methods of the 2 classes, whatever the arguments
2. Matches any update methods of the 2 classes , whatever the arguments and method visibility
3. Matches any update methods of the 2 classes , with one more arguments and whatever method visibility
4. No joint point is defined

Question 20

Using the Spring AOP framework, what is the visibility of the method matches by the following join point?

```
@Pointcut("execution(* *(..))")
private void anyOperation() {};
```

1. All methods, whereas there visibility
2. All methods, except private method
3. Protected and public methods
4. Public methods

Question 21

What are the 2 correct statements about AOP proxy?

1. AOP proxies are created by Spring in order to implement the aspect contracts
2. AOP proxies are always created with a JDK dynamic proxy
3. Only classes that implements a least one interface could be proxied
4. All methods could be proxied
5. Proxies are created by a BeanPostProcessor

Question 22

What is an after throwing advice? Select a unique answer.

1. Advice that could throw an exception
2. Advice to be executed if a method exits by throwing an exception
3. Advice that executes before a join point
4. Spring does not provide this type of advice

Question 23

What is an after returning advice? Select a unique answer.

1. Advice to be executed regardless of the means by which a join point exits
2. Advice that surrounds a method invocation and can perform custom behavior before and after the method invocation
3. Advice to be executed before method invocation
4. Advice to be executed after a join point completes without throwing an exception

Question 24

What is an advice? Select a unique answer.

1. An action taken by an aspect at a particular join point
2. A point during the execution of a program
3. An aspect and a pointcut
4. A predicate that matches join points

Question 25

What is a pointcut? Select a unique answer.

1. Code to execute at a join point
2. An expression to identify joinpoints
3. An advice and a jointpoint
4. None of the above

Question 26

Select method's signatures that match with the following pointcut:

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

1. void com.test.service.MyServiceImpl#transfert(Money amount)
2. void com.test.service.MyServiceImpl#transfert(Account account, Money amount)
3. void com.test.service.account.MyServiceImpl#transfert(Money amount)
4. void com.test.service.account.MyServiceImpl#transfert(Account account, Money amount)
5. None of the above

Question 27

What are the unique correct answers about Spring AOP support?

1. An advice could proxied a constructor's class
2. A point cut could select methods that have a custom annotation
3. Static initialization code could be targeted by a point cut
4. Combination of pointcuts by &&, || and the ! operators is not supported

Question 28

Using the Spring AOP framework, what are the joinpoint methods of the following pointcut expressions?

execution(public * *(..))

1. The execution of all public method
2. The execution of all public method returning a value
3. The execution of all public method having at least one parameter

4. The execution of all public method in class belonging to the default java package

Data Access

Question 29

Why is it a best practice to mark transaction as read-only when code does not write anything to the database? Select one or more answers.

1. It is mandatory for using Spring exception translation mechanism
2. May be improve performance when using Hibernate
3. Spring optimizes its transaction interceptor
4. Provides safeguards with Oracle and some other databases

Question 30

What data access technology is supported by the Spring framework? Select one or more answers.

1. JDBC
2. NoSQL
3. Hibernate
4. JPA

Question 31

What is not provided by the JdbcTemplate? Select a unique answer.

1. Data source access
2. Open/close data source connection
3. JDBC exception wrapping into DataAccess Exception
4. JDBC statement execution

Question 32

Using JdbcTemplate, what is the Spring provided class you will use for result set parsing and merging rows into a single object? Select a unique answer.

1. RowMapper
2. RowCallbackHandler
3. ResultSetExtractor
4. ResultSetMapper

Question 33

What configuration is supported by the LocalSessionFactoryBean? Select a unique answer.

1. Listing entity classes annoted with @Entity
2. Scanning a package to detect annoted entity classes (with @Entity)
3. Listing hibernate XML mapping configuration file (.hbm.xml)

4. All above

Transaction

Question 34

What is/are incorrect statements about XML declaration of the transaction manager bean? Select one or more answers.

1. The tx namespace provides JTA transaction manager declaration shortcut syntax
2. Id of the bean has to be *transactionManager*
3. Depending the application persistence technology, the HibernateTransactionManager or the DataSourceTransactionManager could be used as bean class
4. Default transaction timeout could be given

Question 35

Assuming @Transactional annotation support is enabled and the transferMoney method is called through a Spring AOP proxy, what is the behavior of the following code sample?

```
@Transactional(propagation=Propagation.REQUIRED)
public void transferMoney(Account src, Account target, double amount) {
	add(src, -amount);
	add(src, amount);
}

@Transactional(propagation=Propagation.REQUIRES_NEW)
public void add(Account account, Double amount) {
	// IMPLEMENTATION
}
```

1. The add() method executes code in a new transaction
2. The add() method uses the transaction of the transferMoney() method
3. When calling the add() method, an exception is thrown
4. Other behavior

Question 36

Does Spring provides programmatic transaction management? Select a unique answer.

1. Yes with the TransactionTemplate class
2. Yes with the TransactionService class
3. Yes using the @Transactional bean post processor
4. No

Question 37

What is the transaction behavior of the PROPAGATION_REQUIRES_NEW mode? Select a unique answer.

1. If a transaction exists, the current method should run within this transaction. Otherwise, it should start a new transaction and run within its own transaction.
2. If a transaction is in progress, the current method should run within the nested transaction of the existing transaction. Otherwise, a new transaction has to be started and run within its own transaction.
3. The current method must start a new transaction and run within its own transaction. If there is an existing transaction in progress, it is suspended.
4. None of the above

Question 38

What is the default rollback policy in transaction management?

1. Rollback for any Exception
2. Rollback for RuntimeException
3. Rollback for checked exceptions
4. Always commit

Sping @MVC

Question 39

What could not return a Spring MVC controller? Select a single answer.

1. An absolute path to the view
2. A logical view name
3. A new JstlView
4. void
5. null value

Question 40

Where do you cannot declare Spring MVC controller? Select one or more answers.

1. In a Spring application context XML configuration file
2. Into the web.xml file of the web application
3. Into the java code by using annotations
4. Into the JSP pages

Question 41

What is the easiest method to write a unit test?

1.
```
void displayAccount(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException
```

2.
```
void displayAccount(HttpServletRequest req, HttpSession Session)
              throws ServletException, IOException
```

3.
```
@RequestMapping("/displayAccount")
String displayAccount(@RequestParam("accountId") int id, Model model)
```

4.
```
@RequestMapping("/displayAccount")
String displayAccount(@PathVariable("accountId") int id, Model model)
```

Spring Security

Question 42

How could you secure MVC controller with Spring Security? Select a unique answer.

1. With the @Secured annotation
2. With the @RolesAllowed annotation
3. In a XML security configuration file
4. All of the above
5. None of the above

Question 43

What are the possible mechanisms provided by Spring Security to store user details? Select one or more correct answers.

1. Database
2. JAAS
3. LDAP
4. Properties file

Question 44

What is true about Spring security configuration and the security namespace? Select one or more correct answers.

1. The access attribute of the intercept-url tag support both EL and constants together.

2. The patterns declared into the intercept-url tag are analyzed from up to bottom. Winning is the first that matches.
3. The patterns declared into the intercept-url tag use by default the java regex syntax.
4. Security rules may applied depending request parameter

Remoting

Question 45

What do you have to do even if you are using the RMI Spring Remoting support? Select one or more correct answers.

1. Implements the Remote interface
2. Extends the RemoteObject class
3. Catching the RemoteException exception
4. Implements the Serializable interface

Question 46

What is exact about the HttpInvokerServiceExporter? Select one or more correct answers.

1. Has to run into a HTPP server as Jetty
2. Could process both POST and GET requests
3. Could be used with any http client as Jakarta Commons HttpClient
4. Could consume SOAP http request

JMS

Question 47

What is the method that is not provided by the JmsTemplate Spring class?

1. convertAndSend
2. onMessage
3. receiveAndConvert
4. setDefaultDestination

Question 48

How could you implement a JMS Listener using the Spring JMS support? Select one or more correct answers.

1. By implementing the javax.jms.MessageListener interface
2. By implementing the SessionAwareMessageListener interface provided by Spring
3. Without any code, only using the jms namespace provided by Spring
4. By writing a single POJO without parent class or interface

JMX

Question 49

What is easier to do by using Spring JMS support? Select one or more correct answers.

1. Register any Spring bean as JMX MBean
2. Register an existing MBean with a MBeanServer
3. Accessing to remote MBean
4. Control the attributes and the operations of a Spring bean exposes as a MBean

Question 50

What is the purpose of the @ManageResource annotation? Select a single answer.

1. Expose a bean's property (getter/setter) to JMX
2. Expose a bean's method to JMX
3. Identify a Spring bean as a JMX MBean
4. None of the above

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