1. In Aspect Oriented Programming using Spring, what is a Joinpoint? 1 / 1 point

- Defines when an Advice should be applied, such as a method invocation
- [✓ Defines an expression that represents and entry point into the application]
- Defines additional behavior to apply when a method is executed in the application
- A class that implements enterprise application concerns that cut across multiple classes

Correct
Correct; in the form of an expression, it illustrates the entry point into an application, this will be used by a Pointcut to associate to an Advice to identify when the Advice behavior should be triggered

2. When would you use an @Around advice? Select all that apply. 1 / 1 point

- [✓ To block access to a target bean]

Correct
Correct; without the ProceedingJointpoint proceed method being executed, the request is never passed onto the target, this is the basis for security

- [✓ To get at the target method arguments]

Correct
Correct; using a JoinPoint expression of "args(types)", we can inject the target methods arguments into the advice method as its arguments and change them (if mutable), or even replace them altogether. This is the basis for data sanitization or transformation routines where the back end expects a certain format but the incoming request has if in a different format

- To catch and log Exceptions and force a throw of a more general Exception
- [✓ To change the return object from a target bean]

Correct
Correct; we can not only get at the return type from the target bean, we can replace it. This is the basis for transforming data from back to front end formats

3. Which Advice would you use to always log the return from a target bean method execution. 1 / 1 point

- @AfterThrowing
- @AfterReturning
- @Before
- [✓ @After]

Correct
Correct; regardless if an exception is thrown or not, this annotated advice will always be triggered. It is much like a finally in a try and catch block

4. How would you annotate a method to Guarantee that it will run in its own transaction and rollback on a CheckedException 1 / 1 point

- [✓ @Transactional(noRollbackFor = CheckedException.class, propagation =Propagation.REQUIRED)]
- @Transactional(noRollbackFor = CheckedException.class, propagation =Propagation. REQUIRES_NEW)
- @Tranactional
- @Transactional(propagation= REQUIRED)

Correct
Correct; A Checked exception does not automatically rolled back a Transaction, only RuntimeExceptions do that, If the method sis called in a transaction, that transaction is suspended and the method will run in its own transaction. Upon commit or rollback of this second transaction the first transaction will be resumed

5. What are cross-cutting concerns in Spring AOP

1 / 1 point

- ● Behavior which is applicable throughout the application
- ○ Behavior in one class only
- ○ Behavior which we want to have in a core business modules of an application
- ○ Behavior that is External to the application

✓ **Correct**
True Applicable to the application and non evasive to core logic modules

6. What is the definition of a proxy

1 / 1 point

- ○ Create object without exposing the creation logic to the client and refer to newly created object using a common interface
- ○ Create an object from a “copy” of an existing object
- ● An object that looks like another object, but adds special functionality behind the scene.
- ○ Select an algorithm at runtime

✓ **Correct**
True, a proxy looks like the real target object that is desired and in fact can wrap the real target object thereby adding functionality without touching the target object code itself

7. Which of the following are true about @AfterThrowing – Pick all that apply

1 / 1 point

- ☐ You can catch the application and log it and stop exception bubbling
- ☑ You can throw another Exception

✓ **Correct**
True you can throw another Exception out of the method to replace the existing one

- ☑ It will continue to throw the original Exception

✓ **Correct**
True you can log it and that’s it the original exception will bubble up through the stack

- ☐ Logs access before executing the target method

8. What does this pointcut expression mean, **execution(* clockIn(*, String))**: 1 / 1 point

- ○ Matches any invocation of clockIn() invoked with a String argument
- ○ Matches any invocation of clockIn() invoked with no arguments
- [✓ ● Matches any invocation of clockIn() invoked with two arguments, with the 2nd being a String]
- ○ Matches any invocation of clockIn() invoked with any number of arguments

✓ **Correct**
True the wildcard is for one argument and the String stipulates the 2nd argument type

9. What class would you inject into a @Before annotated method to identify the Signature of the method that is being invoked 1 / 1 point

- [✓ ● Joinpoint]
- ○ ProceedingJoinpoint
- ○ Aspect
- ○ Proxy

✓ **Correct**
True, this class has the getSignature method

10. Which type of exceptions does Spring Transaction Manager automatically mark for Rollback 1 / 1 point

- ○ Exception
- [✓ ● RuntimeException]
- ○ CheckException
- ○ Throwable

✓ **Correct**
True Unchecked or runtime exceptions such as RuntimeException are marked for rollback