Listing 16-22. *Configuration for Annotation-Based Transaction Management*

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/tx
            http://www.springframework.org/schema/tx/spring-tx.xsd
            http://www.springframework.org/schema/aop
            http://www.springframework.org/schema/aop/spring-aop.xsd">

    <bean id="bankService"
          class="com.apress.prospring2.ch16.services.DefaultBankService">
        <property name="accountDao" ref="accountDao"/>
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager"/>
    <aop:aspectj-autoproxy />

</beans>
```

This XML configuration file shows the standard `bankService` bean declaration, followed by the `<tx:annotation-driven />` and `<aop:aspectj-autoproxy />` tags. The `<tx:annotation-driven />` tag creates the appropriate transaction management aspects using the `@Transactional` annotation. The `<aop:aspectj-autoproxy />` tag then advises the matching beans.

Exploring the tx:annotation-driven Tag

The `<tx:annotation-driven />` tag is at the core of the annotation-driven transaction management support. Table 16-3 lists all attributes of the `<tx:annotation-driven />` tag.

Table 16-3. *Attributes of the <tx:annotation-driven /> Tag*

Attribute	Description
`transactionManager`	Specify a reference to an existing `PlatformTransactionManager` bean that the advices will use.
`mode`	Specify how the Spring transaction management framework creates the advised beans. The allowed values are `proxy` and `aspectj`. The `proxy` value is the default; it specifies that the advised object will be a JDK proxy. The `aspectj` parameter instructs Spring AOP to use AspectJ to create the proxy.
`order`	Specify the order in which the created aspect will be applied. This is applicable if you have more than one advice for the target object.
`proxy-target-class`	Set to `true` to specify that you wish to proxy the target class rather than all interfaces the bean implements.

Exploring the @Transactional Annotation

The `@Transactional` annotation allows you to control all aspects of the transaction definition the advice is going to create. Just as with the `transactionAttributes` property expression, you can specify the propagation, isolation level, timeout, and allowed and disallowed exceptions. Table 16-4 lists all attributes of the `@Transactional` annotation.

Table 16-4. *Attributes of the @Transactional Annotation*

Attribute	Type	Description
`propagation`	`org.springframework.annotaion.transaction.Propagation`	Specifies the propagation to be used in the transaction definition
`isolation`	`org.springframework.annotation.transaction.Isolation`	Sets the isolation level the transaction should have
`timeout`	`int`	Specifies the transaction timeout in seconds
`readOnly`	`boolean`	If `true`, the transaction will be marked as read-only
`noRollbackFor`	`Class<? extends Throwable>[]`	Array of exceptions that the target method can throw but the advice will still commit the transaction
`rollbackFor`	`Class<? extends Throwable>[]`	Array of exceptions that will make the advice roll back the transaction if the target method throws them

Annotation-Based Transaction Management Summary

Using the `@Transactional` annotation is an easy way to declare a method transactional. The advantage is that you can immediately see that the method is transactional, because it has the annotation. The disadvantage is that you have to repeat the `@Transactional` annotation for every transactional method. This is not a problem if you are happy with the default transaction attributes but quickly becomes a clumsy copy-and-paste affair when you are setting additional transaction attributes. An alternative is to annotate the class with the `@Transactional` annotation. This would make all methods in the class transactional. The problem with this approach is that all methods, even simple getters and setters, would run in a transaction, even though there is absolutely no need for that. The XML AOP transaction management handles such situations much better.

■ **Note** In saying that using `@Transactional` will make all methods execute in a transaction, we are being a bit sloppy: more accurately, we should say that all methods of a Spring bean instantiated from a class with the `@Transactional` annotation will be transactional.

Using XML AOP Transaction Management

XML AOP declarative transaction management is the preferred approach in Spring 2.5. Spring comes with the `<tx:advice />` tag, which creates a transaction-handling advice. All we need do to get us started is to create a pointcut that matches all methods we wish to make transactional and reference the transactional advice. Listing 16-23 shows an XML configuration that uses XML AOP transaction management.