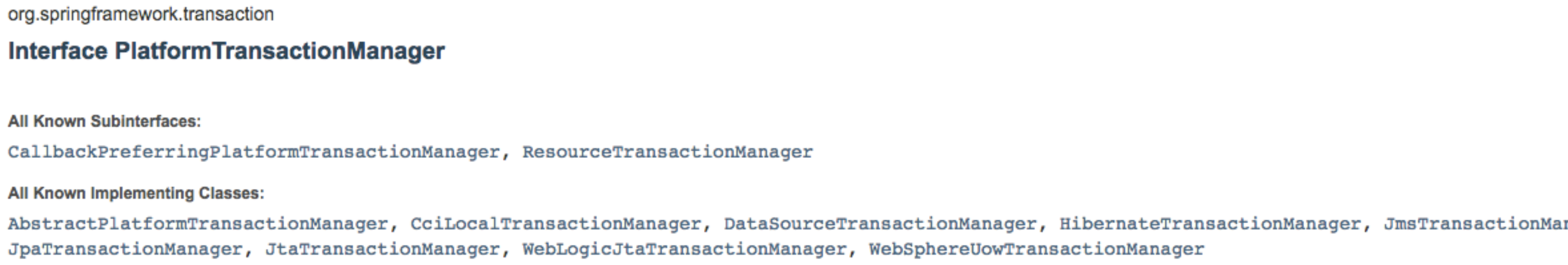
Transaction

Andy

What Spring Transaction?

- Global Transaction & Local Transaction
- How does Spring resolve the disadvantages of them?
    - PlatformTransactionManager



org.springframework.transaction

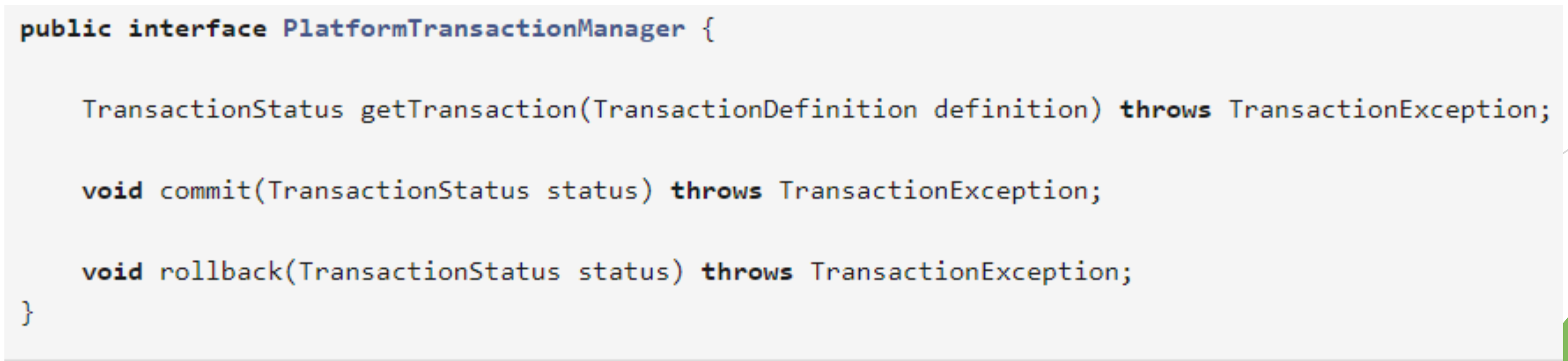
Interface PlatformTransactionManager

All Known Subinterfaces:

CallbackPreferringPlatformTransactionManager, ResourceTransactionManager

All Known Implementing Classes:

AbstractPlatformTransactionManager, CciLocalTransactionManager, DataSourceTransactionManager, HibernateTransactionManager, JmsTransactionMa
JpaTransactionManager, JtaTransactionManager, WebLogicJtaTransactionManager, WebSphereUowTransactionManager



```java
public interface PlatformTransactionManager {

    TransactionStatus getTransaction(TransactionDefinition definition) throws TransactionException;

    void commit(TransactionStatus status) throws TransactionException;

    void rollback(TransactionStatus status) throws TransactionException;
}
```

What does Spring tx do?

- Determine commit and rollback
- Handle exceptions – using AOP

How does Spring tx do it?

- Declarative Transaction Management
  - XML
  - Annotation

- Programmatic Transaction Management

Question: Which one to use?

Declarative TX management --- XML

```xml
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@rj-t42:1521:elvis"/>
    <property name="username" value="scott"/>
    <property name="password" value="tiger"/>
</bean>
```

```xml
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

```xml
    <aop:config>
        <aop:pointcut id="fooServiceOperation" expression="execution(* x.y.service.FooService.*
(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="fooServiceOperation"/>
    </aop:config>
```

```xml
<tx:advice id="txAdvice" transaction-manager="txManager">
    <!-- the transactional semantics... -->
    <tx:attributes>
        <!-- all methods starting with 'get' are read-only -->
        <tx:method name="get*" read-only="true"/>
        <!-- other methods use the default transaction settings (see below) -->
        <tx:method name="*"/>
    </tx:attributes>
</tx:advice>
```

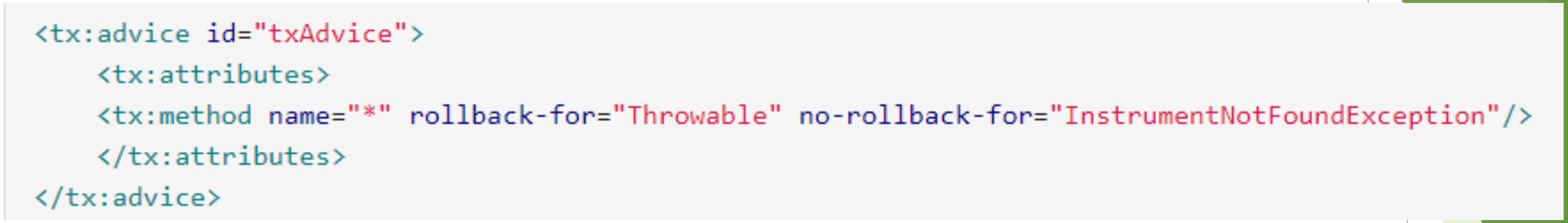
1. Define Data Source

2. Platform-TransactionManager uses datasource

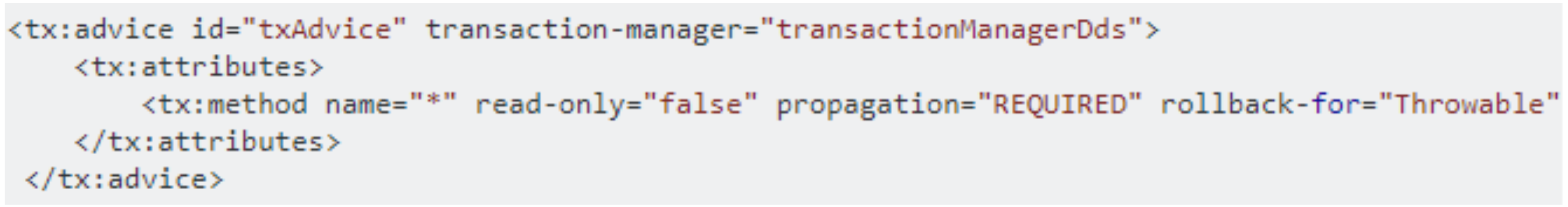
3.Use AOP to define pointcut

4. Define how Spring tx Management be applied To pointcut

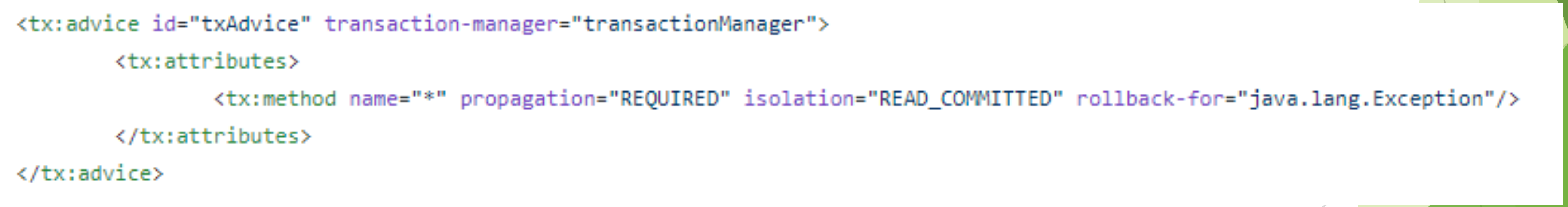
<tx:advice>



```
<tx:advice id="txAdvice">
    <tx:attributes>
    <tx:method name="*" rollback-for="Throwable" no-rollback-for="InstrumentNotFoundException"/>
    </tx:attributes>
</tx:advice>
```

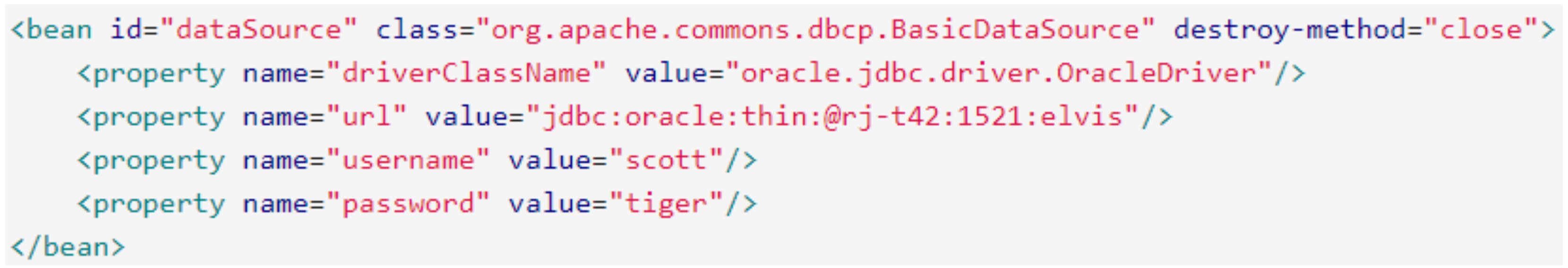


```
<tx:advice id="txAdvice" transaction-manager="transactionManagerDds">
    <tx:attributes>
        <tx:method name="*" read-only="false" propagation="REQUIRED" rollback-for="Throwable"
    </tx:attributes>
 </tx:advice>
```



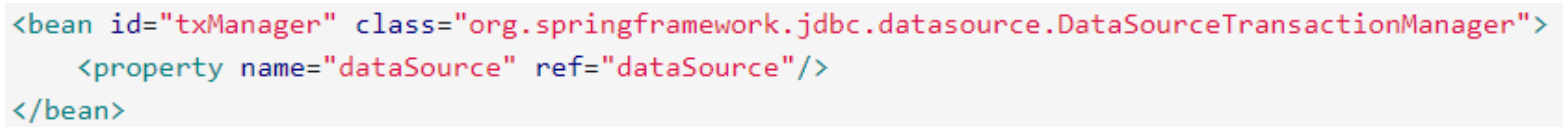
```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
                <tx:method name="*" propagation="REQUIRED" isolation="READ_COMMITTED" rollback-for="java.lang.Exception"/>
        </tx:attributes>
</tx:advice>
```

Declarative TX management --- Annotation



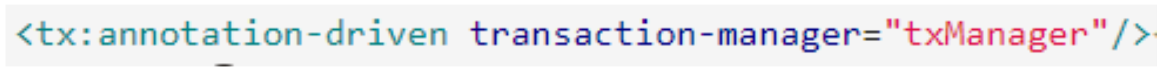
```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@rj-t42:1521:elvis"/>
    <property name="username" value="scott"/>
    <property name="password" value="tiger"/>
</bean>
```

1. Define Data Source



```
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

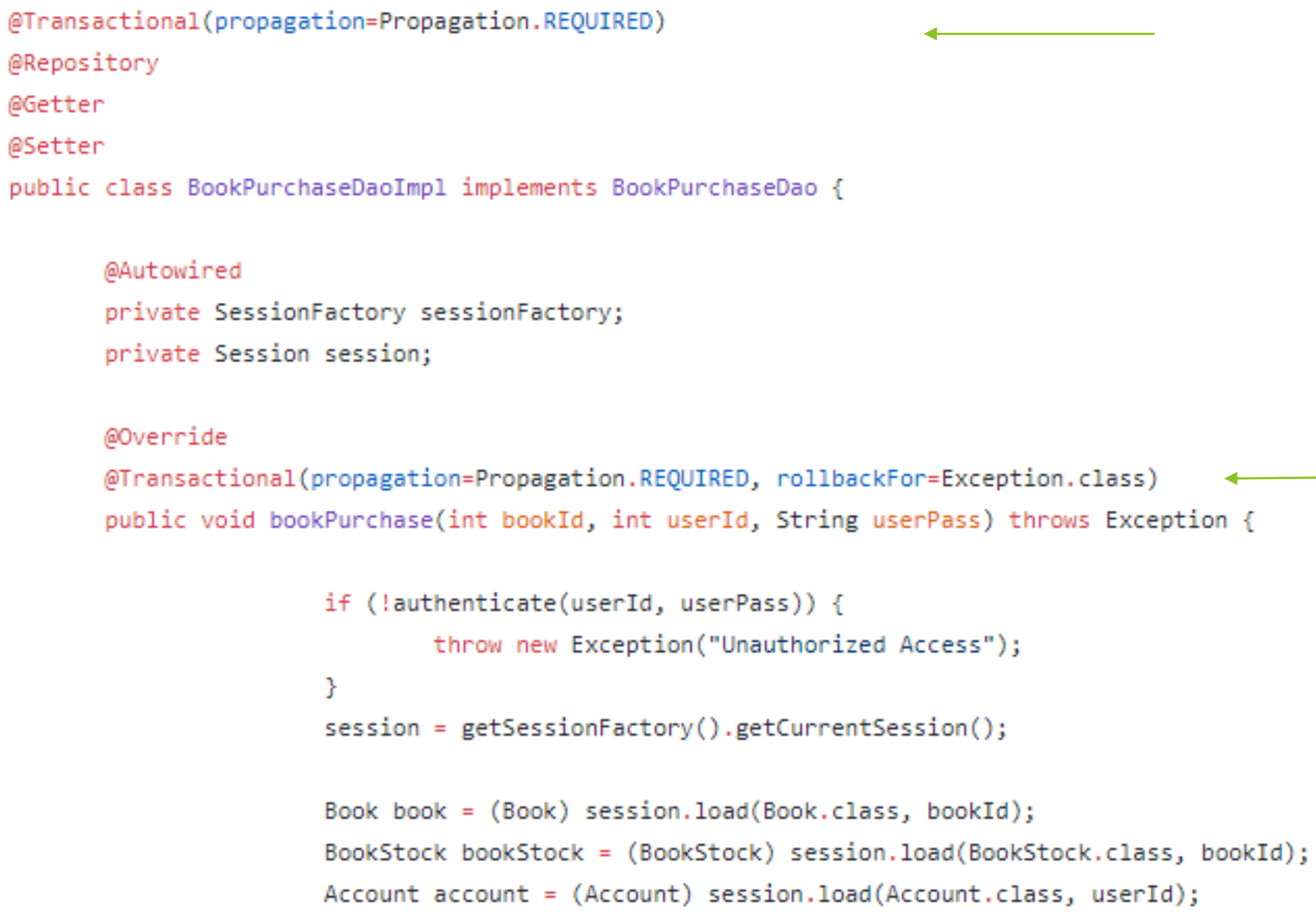
2. Platform-TransactionManager uses datasource



```
<tx:annotation-driven transaction-manager="txManager"/>
```

3.Mark “annotation-Driven”

Declarative TX management --- Annotation



```
@Transactional(propagation=Propagation.REQUIRED)
@Repository
@Getter
@Setter
public class BookPurchaseDaoImpl implements BookPurchaseDao {

       @Autowired
       private SessionFactory sessionFactory;
       private Session session;

       @Override
       @Transactional(propagation=Propagation.REQUIRED, rollbackFor=Exception.class)
       public void bookPurchase(int bookId, int userId, String userPass) throws Exception {

                       if (!authenticate(userId, userPass)) {
                               throw new Exception("Unauthorized Access");
                       }
                       session = getSessionFactory().getCurrentSession();

                       Book book = (Book) session.load(Book.class, bookId);
                       BookStock bookStock = (BookStock) session.load(BookStock.class, bookId);
                       Account account = (Account) session.load(Account.class, userId);
```

Propagation

REQUIRED: Must run in a transaction, create new if no transaction exist

REQUIRED_NEW: Always create a new transaction

SUPPORTS: Run in current transaction or no transaction is needed

NOT_SUPPORTED: do not run in a transaction

MANDATORY: Must run in a transaction or an exception will be thrown

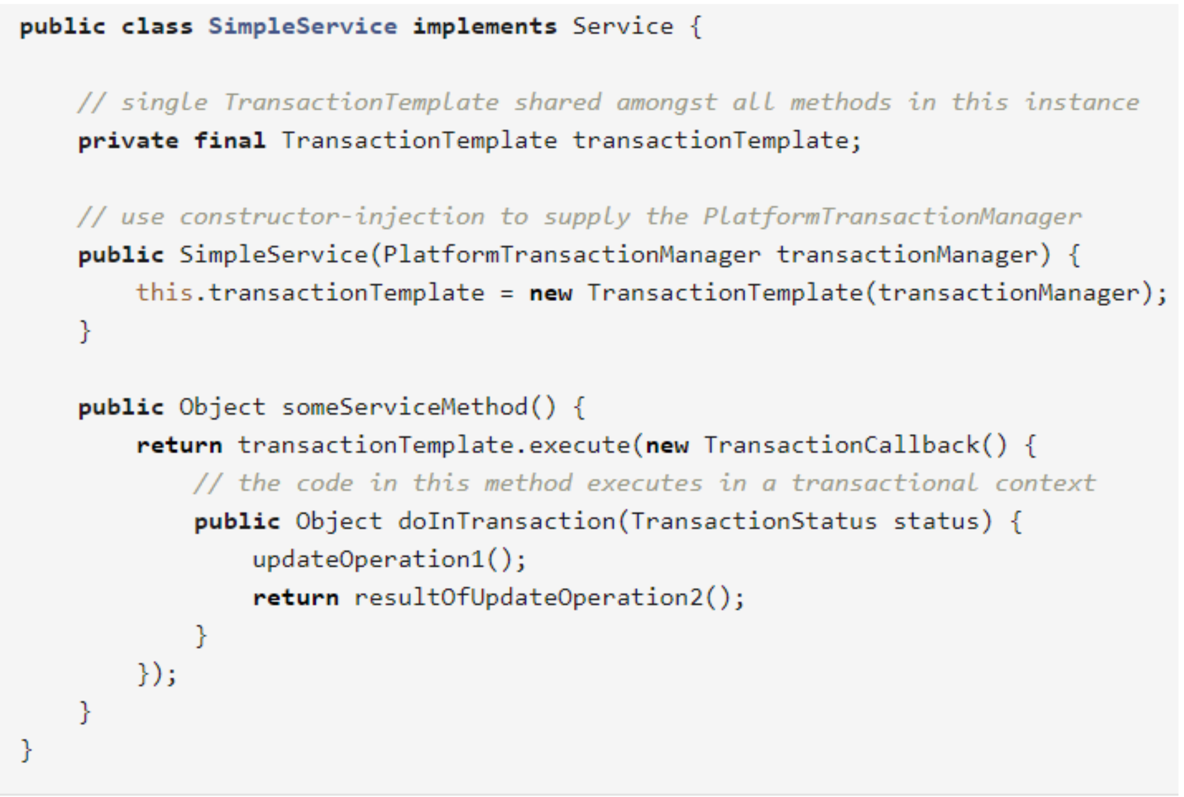
Isolation

Default: Follow underlying database

READ_UNCOMMITTED: Can read uncommitted data

READ_COMMITTED: Only read committed data

* Programmatic tx management

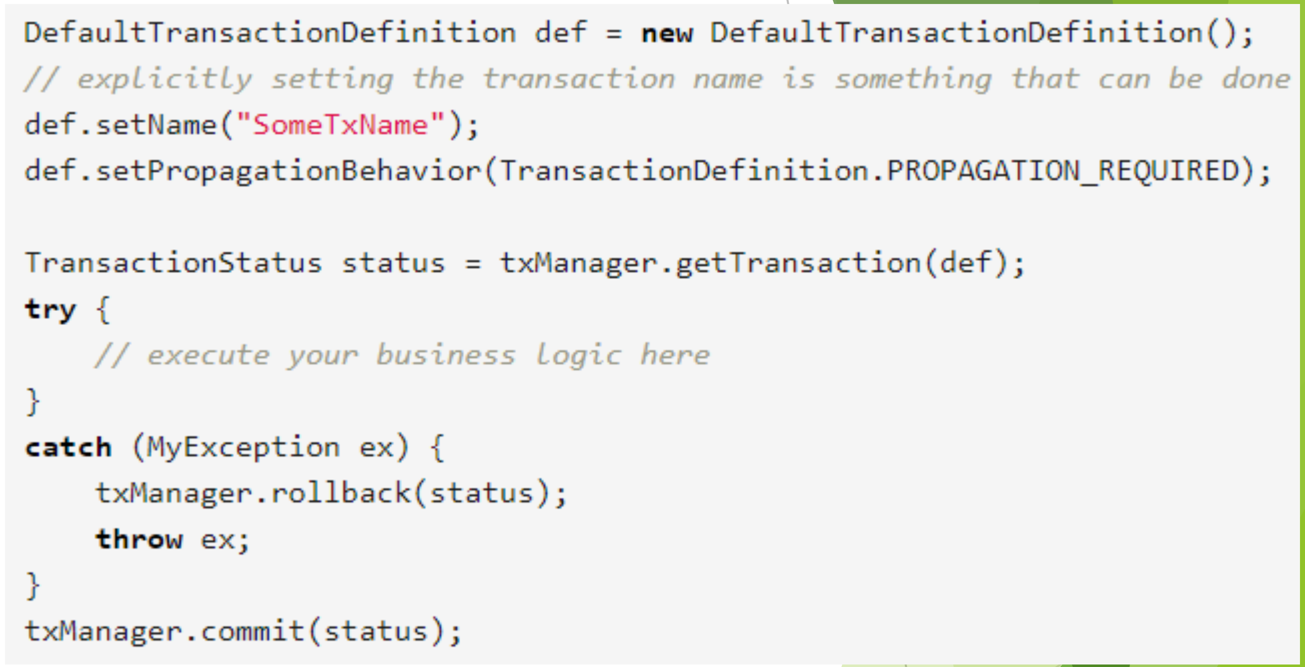


```
public class SimpleService implements Service {

    // single TransactionTemplate shared amongst all methods in this instance
    private final TransactionTemplate transactionTemplate;

    // use constructor-injection to supply the PlatformTransactionManager
    public SimpleService(PlatformTransactionManager transactionManager) {
        this.transactionTemplate = new TransactionTemplate(transactionManager);
    }

    public Object someServiceMethod() {
        return transactionTemplate.execute(new TransactionCallback() {
            // the code in this method executes in a transactional context
            public Object doInTransaction(TransactionStatus status) {
                updateOperation1();
                return resultOfUpdateOperation2();
            }
        });
    }
}
```



```
DefaultTransactionDefinition def = new DefaultTransactionDefinition();
// explicitly setting the transaction name is something that can be done
def.setName("SomeTxName");
def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);

TransactionStatus status = txManager.getTransaction(def);
try {
    // execute your business logic here
}
catch (MyException ex) {
    txManager.rollback(status);
    throw ex;
}
txManager.commit(status);
```

Use “TransactionTemplate”

use default “PlatformTransactionManager”