BY NAGOOR BABU

SPRING
COURSE MATERIAL
BY
NAGOOR BABU

{SPRING TRANSACTIONS}

CONTACT US:
Mobile: +91- **8885 25 26 27**
+91- **7207 21 24 27/28**
US NUM: **4433326786**
Mail ID: **durgasoftonlinetraining@gmail.com**
WEBSITE: **www.durgasoftonline.com**
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

CONTACT US:

Mobile: +91- **8885 25 26 27**

+91- **7207 21 24 27/28**

US NUM: **4433326786**

Mail ID: **durgasoftonlinetraining@gmail.com**

WEBSITE: **www.durgasoftonline.com**

FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

BY NAGOOR BABU

SPRING TRANSACTIONS MODULE INDEX

CONTACT US:

Mobile: +91- **8885 25 26 27**

+91- **7207 21 24 27/28**

US NUM: **4433326786**

Mail ID: **durgasoftonlinetraining@gmail.com**

WEBSITE: **www.durgasoftonline.com**

FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

BY NAGOOR BABU

Spring Transactions

Intorduction:

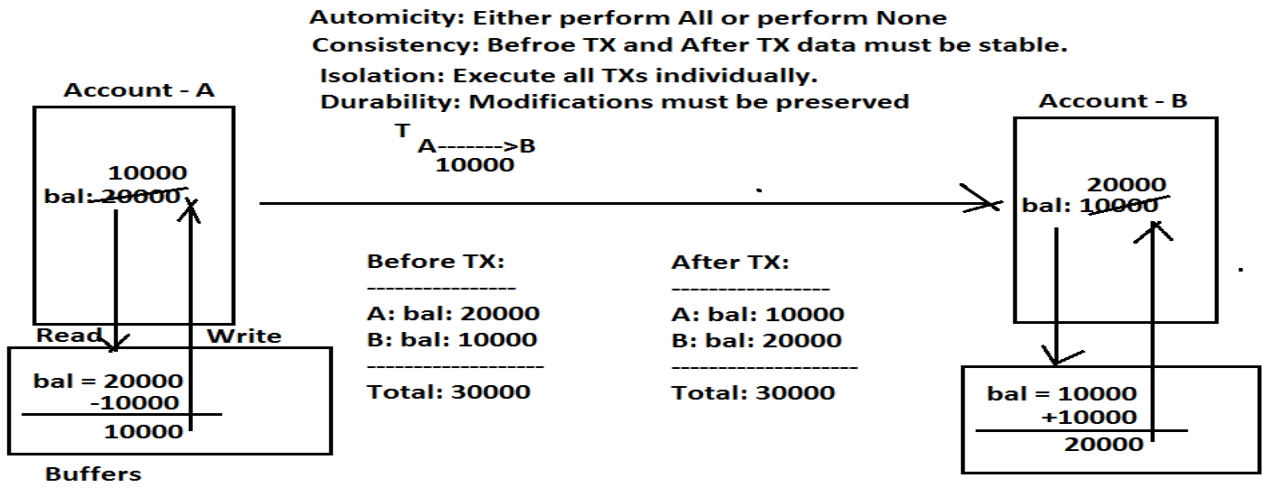
Transaction Management:

Transaction is an unit of work performed by Front End applications on Back End System.

EX: Deposit some amount in an Account.
Withdraw some amount from an Account
Transfer some amount from one account to another account.

IN database applications, Every Transaction must follow ACID properties

1. Atomicity: This property will make the Transaction either in SUCCESS state or in FAILURE state.
In Database related applications, if we perform all operations then the Transaction is available in SUCCESS State, if we perform none of the operations then the Transaction is available in FAILURE state.
2.Consistency: In database applications, Before the Transaction and After the Transaction Database state must be in stable.
3. Isolation: If we run more than one Transaction on a single Data item then that Transactions are called as "Concurret Transactions". In Transactions Concurrency , one transaction execution must not give effect to another Transaction, this rule is called as "Isolation" property.
4. Durability: After committing the Transaction, if any failures are coming like Power failure, OS failure,... after getting the System if we open the transaction then the modifications which we performed during the transaction must be preserved.



CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

In JDBC, to perform Automicity property we have to change Connections auto-commit nature and we have to perform either commit() or rollback() at the end of Transaction.

EX:

```
1. Connection con = DriverManager.getConnection(---);
2. con.setAutoCommit(false);
3. try{
4. ---instructions-----
5. con.commit();
6. }catch(Exception e){
7. e.printStacktrace();
8. con.rollback();
9. }
```

In Hibernate applications, if we want to manage Transactions Automicity property then we have to use the following steps.

1. Declare Transaction Before try.
2.Create Transaction object inside try block.
3.Perform commit() operation at end of Transaction.
4.Perform rollback() operation at catch block.

Transaction tx = null;

```
1. try{
2. -----
3. tx = session.beginTransaction();
4. ----
5. -----
6. tx.commit();
7. }catch(Exception e){
8. tx.rollback();
9. }
```

If we execute more than one transaction on a single data item then that transactions are called as Concurrent Transactions.

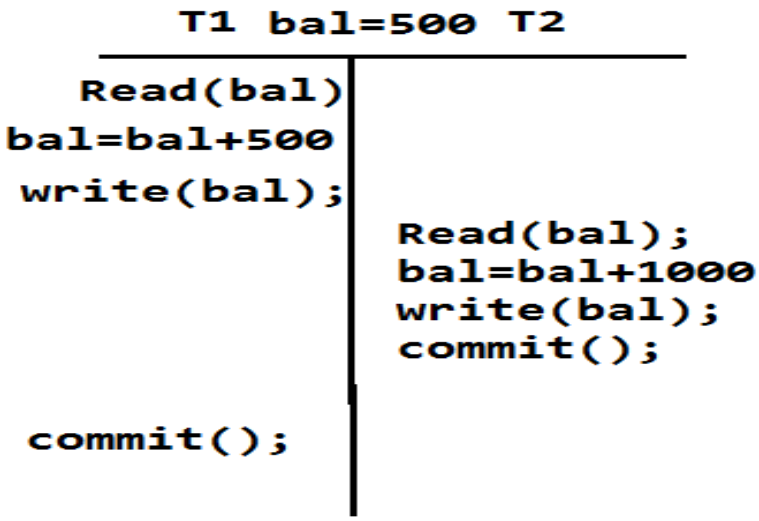
In Transactions concurrency we are able to get the following data consistency problems while executing more than one transaction at a time.

1. Lost Update Problem
2. Dirty Read Problem
3. Non Repeatable Read Problem
4. Phanthom Read Problem

CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

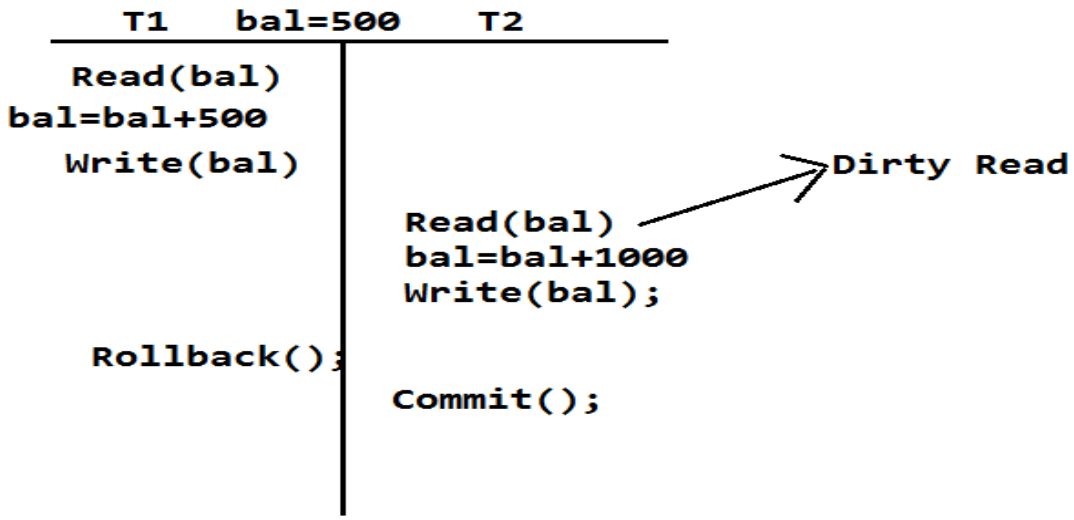
1. Lost Update Problem

In Transactions concurrency, if one transaction perform updations over the data with out commit operation , mean while, other transactions perform updations with commit operation then the first transaction updations are lost, this data consistency problem is called as Lost Update problem.



2. Dirty Read Problem:

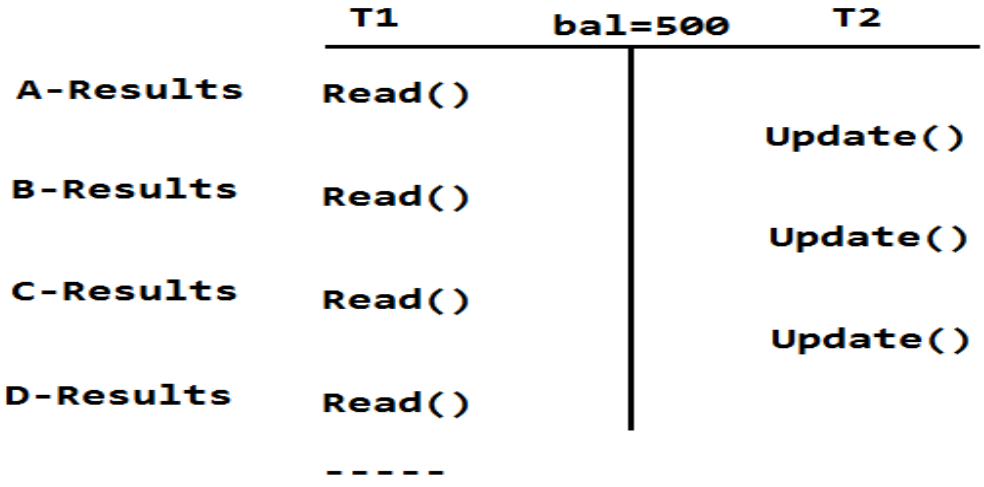
In Transactions concurrency, if one transaction perform updations over data with out performing commit / rollback, mean while if other Transaction perform Read operation over the uncommitted data with out performing commit/rollback operations, in this context, if first transaction perform Rollback operation then the read operation performed by second transaction is Dirty Read, this problem is called as Dirty Read problem.



CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

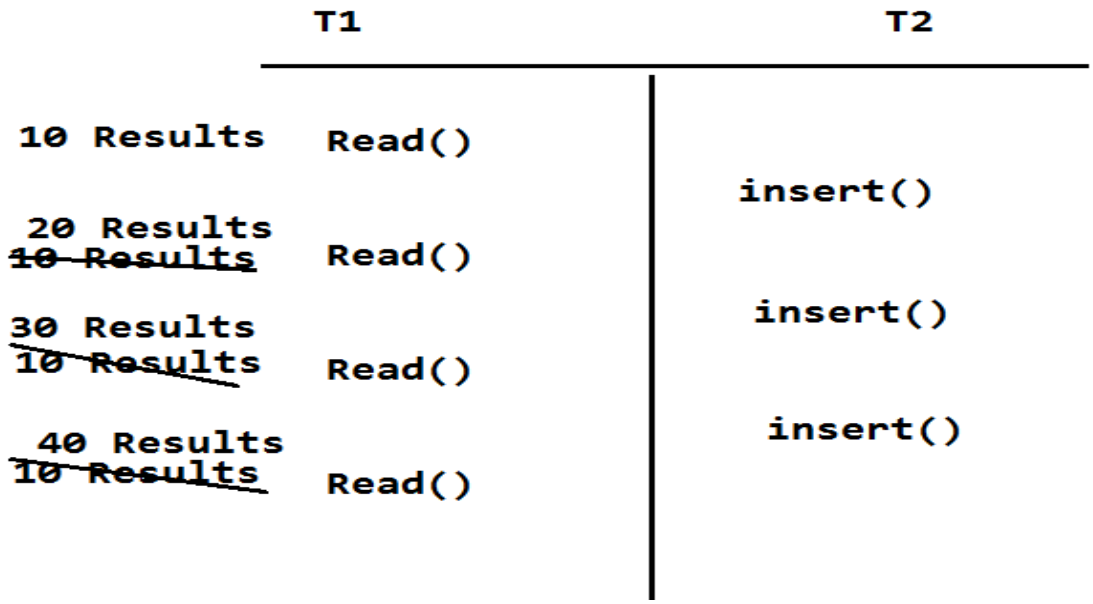
3. Non Repeatable Read Problem

In Transactions concurrency, One transaction perform continous read operations to get same results, mean while, between two read operations another transaction performs update operation over the same data, in this context, in the next read operation performed by first transaction may not get same repeatable results, this problem is called as Non Repeatable Read Problem.



4. Phantom Read Problem

In Transactions concurrency, one transaction perform read operation continously to get same no of results at each and every read operation , mean while, other transactions may perform insert operations between two read operations performed by first transactions, in this context, in the next read operation performed by first transaction may not generate the same no of results, this problem is called as "Panthom Read" Problem, here the extra records inserted by second transaction are called as "Panthom Records".



CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

1. public sttaic final int TRANSCATION_NONE = 0;
2. public sttaic final int TRANSCATION_READ_UNCOMMITTED = 1;
3. public sttaic final int TRANSCATION_READ_COMMITTED = 2;
4. public sttaic final int TRANSCATION_REPETABLE_READ = 4;
5. public sttaic final int TRANSCATION_SERIALIZABLE = 8;

public void setTransactionIsolation(int isolation_Level)
EX: con.setTransactionIsolation(Connection.TRANSACTION_READ_
COMMITTED);

<property name="hibernate.connection.isolation>val</property>
Where value may be either of the following Constants
NONE = 0;
READ_UNCOMMITTED = 1;
READ_COMMITTED = 2;
REPEATABLE_READ = 4;
SERIALIZABLE = 8;

EX:

```
1. <hibernate-configuration>
2.  <session-factory>
3.    ----
4.    <property name="hibernate.connection.isolation>
5.          SERIALIZABLE
6.    </property>
7.    -----
8.  </session-factory>
9. </hibernate-configuration>
```

There are two types of Transactions
1. Local Transaction
2. Global Transaction

1. Local Transaction

Local transactions are specific to a single transactional resource like a JDBC connection

Local transaction management can be useful in a centralized computing environment where application components and resources are located at a single site, and transaction management only involves a local data manager running on a single machine. Local transactions are easier to be implemented

2. Global Transaction

Global transactions can span multiple transactional resources like transaction in a distributed system

Global transaction management is required in a distributed computing environment where all the resources are distributed across multiple systems

A distributed or a global transaction is executed across multiple systems, and its execution requires coordination between the global transaction management system and all the local data managers of all the involved systems.

Transaction Support in Spring

Spring provides extensive support for transaction management and help developers to focus more on business logic rather than worrying about the integrity of data incase of any system failures.

Spring Transaction Management is providing the following advantages in enterprise applications:

1. Spring Supports Declarative Transaction Management. In this model, Spring uses AOP over the transactional methods to provide data integrity. This is the preferred approach and works in most of the cases.
2. Spring Supports most of the transaction APIs such as JDBC, Hibernate, JPA, JDO, JTA etc. All we need to do is use proper transaction manager implementation class.

EX:

1. org.springframework.jdbc.datasource.DriverManagerDataSource for JDBC transaction management
2. org.springframework.orm.hibernate3.HibernateTransactionManager for Hibernate as ORM tool.
3. Support for programmatic transaction management by using TransactionTemplate or PlatformTransactionManager implementation.

To represent ISOLATION levels Spring Framework has provided the following Constants from "org.springframework.transaction .TransactionDefinition".

1. **ISOLATION_DEFAULT:** It is default isolation level, it will use the underlying database provided default Isolation level.
2. **ISOLATION_READ_UNCOMMITTED:** It will not resolve any ISOLATION problem, It represents dirty read problem, non-repeatable read problem, phantom read problem .
3. **ISOLATION_READ_COMMITTED:** It will resolve dirty read problem,but, it will represent non-repeatable read problem and phantom read problem.

CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

4. **ISOLATION_REPEATABLE_READ:** It will resolve dirty read problem and non-repeatable read problem , but, It will represent phantom read problem.
5. **ISOLATION_SERIALIZABLE:** It will resolve all ISOLATION problems like dirty reads, non-repeatable read, and phantom read Problems.

To set the above Isolation level to TransactionTemplate then we have to use the following method .

public void setIsolationLevel(int value)
EX: txTemplate.setIsolationLevel(TransactionDefinition.ISOLATION_DEFAULT);

Transaction Attributes

In Enterprise applications, if we a method begins a transaction and access another method, in this context, another method execution is going on in the same Transaction or in any new Transaction is completely depending on the Transaction Propagation behaviour what we are using.

Spring Framewrk has provided very good suport for Propagation Behaviour in the form of the following constants from "org.springframework.transaction .TransactionDefinition".

1. PROPAGATION_REQUIRED
2. PROPAGATION_REQUIRES_NEW
3. PROPAGATION_SUPPORTS
4. PROPAGATION_NOT_SUPPORTED
5. PROPAGATION_MANDATORY
6. PROPAGATION_NEVER
7. PROPAGATION_NESTED

1. PROPAGATION_REQUIRED:

If a method1 is running in a transaction and invokes method2 then method2 will be executed in the same transaction. If method1 is not associated with any Transaction then Container will create new Transaction to execute Method2.

2. PROPAGATION_REQUIRES_NEW:

If a method1 is running in a transaction and invokes method2 then container will suspend the current Transaction temporarily and creates new Transaction for method2. After executing method2 transaction then method1 transaction will continue.If Method1 is not associated with any transaction then container will start a new transaction before starts new method.

3. PROPAGATION_MANDATORY:

If a method1 is running in a transaction and invokes method2 then method2 will be executed in the same transaction. If method1 is not associated with any Transaction then Container will raise an exception like "TransactionThrowsException".

4. PROPAGATION_SUPPORTS:

If a method1 is running in a transaction and invokes method2 then method2 will be executed in the same transaction. If method1 is not associated with any Transaction then Container does not start new Transaction before running method2.

5 .PROPAGATION_NOT_SUPPORTED:

If a method1 is running in a transaction and invokes method2 then Container Suspends the Mehod1 transaction before invoking Method2. When Method2 has completed , container resumes Method1 transaction. If Mehod1 is not associated with Tramsaction then Container does not start new Tansaction before executing Method2.

6. PROPAGATION_NEVER:

If a method1 is running in a transaction and invokes method2 then Container throws an Exception like RemoteException. I mthod1 is not associated with any transaction then Container will not start new Transaction for Method2.

7. PROPAGATION_NESTED:

Indicates that the method should be run with in a nested transaction if an existed transaction is in progress.

To set the above Propagation Behaviout to TransactionTemplate then we have to use the following method.

public void setPropagationBehaviour(int value)
EX:
txTemplate.setPropagationBehaviour(TransactionDefinition.PROPAGATION_EQUIRES_NEW);

CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

BY NAGOOR BABU

Transactions Approaches in Spring

Spring Framework supports Transactions in two ways.
1.Programmatic Approach
2.Declarative Approach

Programmetic Approach:

In Programatic Approach of transactions we have to declare the transaction and we have to perform transactions commit or rollback operations explicitly by using JAVA code.

In Programetic approach if we want to provide transactions then we have to use the following predefined Library.

1.Transaction Manager:

The main intention of TransactionManager is is able to define a Transaction Strategy in spring application.

Spring Framework has provided Transaction manager in the form of a predefined interfaces
1. org.springframework.jdbc.datasource.DataSourceTransactionManager
2. org.springframework.transaction.PlatformTransactionManager
3. org.springframework.orm.hibernate4.HibernateTransactionManager
4. org.springframework.transaction.jta.JtaTransactionManager

In Spring Transaction based applicatins , We must configure either of the above TransactionManager in configuration file and we must inject TransactionManager in DAO implementation class.

DataSourceTransactionManager includes the following methods to manage transaction.
1. public TransactionStatus getTransaction(TransactionDefinition tx_Def)
2. public void commit(TransactionStatus tx_Status)
3. public void rollback(TransactionStatus tx_Status)

2. TransactionDefinition:

org.springframework.transaction.TransactionDefinition is able to specify ISOLATION levels, Propagation behaviours, Transactions Time out and Transactions Read Only status,.....

TransactionDefinition includes the following Constants to manage Transactions ISOLATION Levels in Spring applications.
1. ISOLATION_READ_UNCOMMITTED

2. ISOLATION_READ_COMMITTED
3. ISOLATION_REPEATABLE_READ
4. ISOLATION_SERIALIZABLE

TransactionDefinition includes the following Constants to manage Transactions Propagation Behaviours in Spring applications.

1. PROPAGATION_REQUIRED
2. PROPAGATION_REQUIRES_NEW
3. PROPAGATION_SUPPORTS
4. PROPAGATION_NOT_SUPPORTED
5. PROPAGATION_MANDATORY
6. PROPAGATION_NESTED
7. PROPAGATION_NEVER

3. TransactionStatus:

org.springframework.transaction.TransactionStatus interface provides a simple way for transactional code to control transaction execution and query transaction status.

TransactionStatus includes the following methods to manage Transactions in Spring applications.

1. public boolean isNewTransaction()
2. boolean hasSavepoint()
3. public void setRollbackOnly()
4. public boolean isRollbackOnly()
5. public void flush()
6. public boolean isCompleted()

Example on Spring Transactions in Programmetic Approach:

TransactionDao.java

```
1. package com.durgasoft.dao;
2. public interface TransactionDao {
3.     public String transferFunds(String fromAccount, String toAccount, int transfer_Amt);
4. }
```

TransactionDaoImpl.java

```
1. package com.durgasoft.dao;
2. import org.springframework.jdbc.core.JdbcTemplate;
3. import org.springframework.jdbc.datasource.DataSourceTransactionManager;
4. import org.springframework.transaction.TransactionDefinition;
```


```
5.  import org.springframework.transaction.TransactionStatus;
6.  import org.springframework.transaction.support.DefaultTransactionDefinition;
7.
8.  public class TransactionDaoImpl implements TransactionDao {
9.
10.    private JdbcTemplate jdbcTemplate;
11.    private DataSourceTransactionManager transactionManager;
12.
13.    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
14.        this.jdbcTemplate = jdbcTemplate;
15.    }
16.    public void setTransactionManager(DataSourceTransactionManager transactionManag
    er) {
17.        this.transactionManager = transactionManager;
18.    }
19.
20.    @Override
21.    public String transferFunds(String fromAccount, String toAccount, int transfer_Amt) {
22.        String status = "";
23.        TransactionDefinition tx_Def = new DefaultTransactionDefinition();
24.        TransactionStatus tx_Status = transactionManager.getTransaction(tx_Def);
25.        try {
26.            withdraw(fromAccount, transfer_Amt);
27.            deposit(toAccount, transfer_Amt);
28.            transactionManager.commit(tx_Status);
29.            status = "Transaction Success";
30.
31.        } catch (Exception e) {
32.            transactionManager.rollback(tx_Status);
33.            status = "Transaction Failure";
34.            e.printStackTrace();
35.        }
36.        return status;
37.    }
38.    public void withdraw(String acc, int wd_Amt) {
39.
40.            jdbcTemplate.execute("update account set BALANCE = BALANCE -
    "+wd_Amt+" where ACCNO = '"+acc+"'");
41.
42.
43.    }
44.    public void deposit(String acc, int dep_Amt)throws Exception {
45.
```


CONTACT US:

Mobile: +91- **8885 25 26 27**

+91- **7207 21 24 27/28**

US NUM: **4433326786**

Mail ID: **durgasoftonlinetraining@gmail.com**

WEBSITE: **www.durgasoftonline.com**

FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

```
46.         float f = 100/0;
47.         jdbcTemplate.execute("update account set BALANCE = BALANCE + "+dep_Amt+"
    where ACCNO = '"+acc+"'");
48.
49.     }
50.}
```

applicationContext.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:context="http://www.springframework.org/schema/context"
5.     xsi:schemaLocation="
6.        http://www.springframework.org/schema/beans
7.        http://www.springframework.org/schema/beans/spring-beans.xsd
8.        http://www.springframework.org/schema/context
9.        http://www.springframework.org/schema/context/spring-context.xsd">
10.    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerData
    Source">
11.       <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
12.       <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
13.       <property name="username" value="system"/>
14.       <property name="password" value="durga"/>
15.    </bean>
16.    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
17.       <property name="dataSource" ref="dataSource"/>
18.    </bean>
19.    <bean id="transactionDao" class="com.durgasoft.dao.TransactionDaoImpl">
20.       <property name="jdbcTemplate" ref="jdbcTemplate"/>
21.       <property name="transactionManager" ref="transactionManager"/>
22.    </bean>
23.    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourc
    eTransactionManager">
24.       <property name="dataSource" ref="dataSource"/>
25.    </bean>
26.
27. </beans>
```

Test.java

```
1.  package com.durgasoft.test;
2.
```


CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,


```
3.  import org.springframework.context.ApplicationContext;
4.  import org.springframework.context.support.ClassPathXmlApplicationContext;
5.
6.  import com.durgasoft.dao.TransactionDao;
7.
8.  public class Test {
9.
10.    public static void main(String[] args) {
11.        ApplicationContext context = new ClassPathXmlApplicationContext("applicationConte
    xt.xml");
12.        TransactionDao tx_Dao = (TransactionDao)context.getBean("transactionDao");
13.        String status = tx_Dao.transferFunds("abc123", "xyz123", 5000);
14.        System.out.println(status);
15.    }
16.}
```

Drawbacks with Transactions Programatic Approach:

1. Programatic Approach is suggestible when we have less no of operations in Transactions, it is not suggestible when we have more no of operations in Transactions.
2. Programatic Approach is some what tightly coupled approach , because, it includes both Business logic and Transactions service code as a single unit.
3. Programatic approach is not convenient to use in enterprise Applications.

Declarative Approach:

In Declarative approach, we will seperate Transaction Service code and Business Logic in Spring applications , so that, we are able to get loosly coupled design in Spring applications.

Declarative approach ios suggestible when we have more no of operations in Transactions.

Declarative approach is not convenient to use in enterprise Applications because of AOP.

There are two ways to provide Transaction management in declarative approach.

1. Using Transactions Namespace Tags with AOP implementation
2. Using Annotations.

1. Using Transactions Namespace Tags with AOP implementation

To manage Transactions in declarative approach, Spring Transaction module has provided the following AOP implemented tags.

1. <tx:advice>

It represent Transaction Advice, it is the implementation of Transaction Service.

Syntax:

<tx:advice id="---" transaction-manager="----">

2. <tx:attributes>

It will take Transactional methods inorder to apply Isolation levels and Propagation behaviours,..... by using <tx:method> tag.

Syntax:

<tx:attributes>

<tx:attributes>

3. <tx:method>

It will define Transactional method and its propagation Behaviours, Isolation levels, Timeout statuses,....

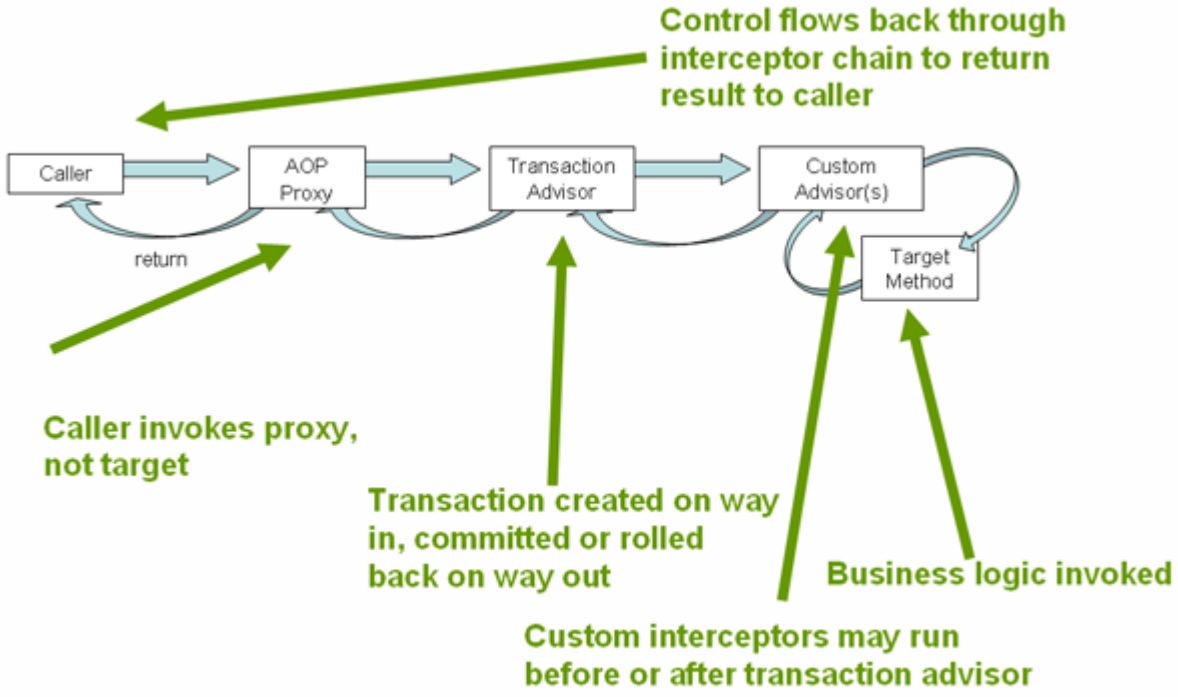
Syntax:

<tx:method name="---" propagation="---" isolation="-------" />

With the above tags, we must define Transaction Advice and it must be configured with a particular Pointcut expression by using <aop:advisor> tag.

EX:

```
1. <tx:advice id="txAdvice" transaction-manager="transactionManager">
2.   <tx:attributes>
3.     <tx:method name="transferFunds"/>
4.   </tx:attributes>
5. </tx:advice>
6. <aop:config>
7.   <aop:pointcut expression="execution(* com.durgasoft.dao.TransactionDao.transferFunds(..))" id="transfer"/>
8.   <aop:advisor pointcut-ref="transfer" advice-ref="txAdvice"/>
9. </aop:config>
```



Example on Spring Transactions in Declarative Approach with XML Configuration:

TransactionDao.java

```
1.  package com.durgasoft.dao;
2.  public interface TransactionDao {
3.      public String transferFunds(String fromAccount, String toAccount, int transfer_Amt);
4.  }
```

TransactionDaoImpl.java

```
1.  package com.durgasoft.dao;
2.  import org.springframework.jdbc.core.JdbcTemplate;
3.  import org.springframework.jdbc.datasource.DataSourceTransactionManager;
4.  import org.springframework.transaction.TransactionDefinition;
5.  import org.springframework.transaction.TransactionStatus;
6.  import org.springframework.transaction.support.DefaultTransactionDefinition;
7.
8.
9.  public class TransactionDaoImpl implements TransactionDao {
10.
11.     private JdbcTemplate jdbcTemplate;
12.     public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
```


CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,


```
13.         this.jdbcTemplate = jdbcTemplate;
14.     }
15.     @Override
16.     public String transferFunds(String fromAccount, String toAccount, int transfer_Amt) {
17.         String status = "";
18.             int val1 = jdbcTemplate.update("update account set BALANCE = BALANCE -
    "+transfer_Amt+" where ACCNO = '"+fromAccount+"'");
19.             float f = 100/0;
20.             int val2 = jdbcTemplate.update("update account set BALANCE = BALANCE + "+tra
    nsfer_Amt+" where ACCNO = '"+toAccount+"'");
21.             if(val1 ==1 && val2 ==1) {
22.                     status = "Transaction Success";
23.             }else {
24.                 status = "Transaction Failure";
25.             }
26.         return status;
27.     }
28.}
```

ApplicationContext.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <beans xmlns="http://www.springframework.org/schema/beans"
3.      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.      xmlns:aop="http://www.springframework.org/schema/aop"
5.      xmlns:tx="http://www.springframework.org/schema/tx"
6.      xsi:schemaLocation="
7.          http://www.springframework.org/schema/beans
8.          http://www.springframework.org/schema/beans/spring-beans.xsd
9.          http://www.springframework.org/schema/tx
10.         http://www.springframework.org/schema/tx/spring-tx.xsd
11.         http://www.springframework.org/schema/aop
12.         http://www.springframework.org/schema/aop/spring-aop.xsd">
13.
14.
15.     <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerData
    Source">
16.         <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
17.         <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
18.         <property name="username" value="system"/>
19.         <property name="password" value="durga"/>
20.     </bean>
21.     <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
```


CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,


```
22.       <property name="dataSource" ref="dataSource"/>
23.    </bean>
24.    <bean id="transactionDao" class="com.durgasoft.dao.TransactionDaoImpl">
25.       <property name="jdbcTemplate" ref="jdbcTemplate"/>
26.    </bean>
27.    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourc
    eTransactionManager">
28.       <property name="dataSource" ref="dataSource"/>
29.    </bean>
30.
31.    <tx:advice id="txAdvice" transaction-manager="transactionManager">
32.       <tx:attributes>
33.          <tx:method name="transferFunds"/>
34.       </tx:attributes>
35.    </tx:advice>
36.    <aop:config>
37.       <aop:pointcut expression="execution(* com.durgasoft.dao.TransactionDao.transferF
    unds(..))" id="transfer"/>
38.       <aop:advisor pointcut-ref="transfer" advice-ref="txAdvice"/>
39.    </aop:config>
40.
41.</beans>
```

Test.java

```
1.  package com.durgasoft.test;
2.  import org.springframework.context.ApplicationContext;
3.  import org.springframework.context.support.ClassPathXmlApplicationContext;
4.
5.  import com.durgasoft.dao.TransactionDao;
6.
7.  public class Test {
8.
9.     public static void main(String[] args) {
10.       ApplicationContext context = new ClassPathXmlApplicationContext("applicationConte
    xt.xml");
11.       TransactionDao tx_Dao = (TransactionDao)context.getBean("transactionDao");
12.       String status = tx_Dao.transferFunds("abc123", "xyz123", 100);
13.       System.out.println(status);
14.    }
15.}
```


CONTACT US:
Mobile: +91- **8885 25 26 27**
+91- **7207 21 24 27/28**
US NUM: **4433326786**
Mail ID: **durgasoftonlinetraining@gmail.com**
WEBSITE: **www.durgasoftonline.com**
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

2. Using Annotations:

This approach is very simple to use for transactions in Spriong applications. In this approach, we will use @Transactional annotation just before the transactional methods in DAO implementation classes, but, to use this annotation we must activate @Transactional annotation in Spring configuration file vy using the following tag.

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

Example on Spring Transactions in Declarative Approach with Annotations:

TransactionDao.java

```
1.  package com.durgasoft.dao;
2.  public interface TransactionDao {
3.      public String transferFunds(String fromAccount, String toAccount, int transfer_Amt);
4.  }
```

TransactionDaoImpl.java

```
1.  package com.durgasoft.dao;
2.  import org.springframework.jdbc.core.JdbcTemplate;
3.  import org.springframework.jdbc.datasource.DataSourceTransactionManager;
4.  import org.springframework.transaction.TransactionDefinition;
5.  import org.springframework.transaction.TransactionStatus;
6.  import org.springframework.transaction.annotation.Transactional;
7.  import org.springframework.transaction.support.DefaultTransactionDefinition;
8.
9.
10. public class TransactionDaoImpl implements TransactionDao {
11.
12.     private JdbcTemplate jdbcTemplate;
13.     public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
14.         this.jdbcTemplate = jdbcTemplate;
15.     }
16.
17.     @Transactional
18.     @Override
19.     public String transferFunds(String fromAccount, String toAccount, int transfer_Amt) {
20.         String status = "";
```


CONTACT US:
Mobile: +91- 8885 25 26 27
+91- 7207 21 24 27/28
US NUM: 4433326786
Mail ID: durgasoftonlinetraining@gmail.com
WEBSITE: www.durgasoftonline.com
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

```
21.          int val1 = jdbcTemplate.update("update account set BALANCE = BALANCE -
    "+transfer_Amt+" where ACCNO = '"+fromAccount+"'");
22.          float f = 100/0;
23.          int val2 = jdbcTemplate.update("update account set BALANCE = BALANCE + "+tra
    nsfer_Amt+" where ACCNO = '"+toAccount+"'");
24.          if(val1 ==1 && val2 ==1) {
25.                status = "Transaction Success";
26.          }else {
27.              status = "Transaction Failure";
28.          }
29.      return status;
30.   }
31.}
```

ApplicationContext.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <beans xmlns="http://www.springframework.org/schema/beans"
3.     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4.     xmlns:aop="http://www.springframework.org/schema/aop"
5.     xmlns:tx="http://www.springframework.org/schema/tx"
6.     xsi:schemaLocation="
7.        http://www.springframework.org/schema/beans
8.        http://www.springframework.org/schema/beans/spring-beans.xsd
9.        http://www.springframework.org/schema/tx
10.       http://www.springframework.org/schema/tx/spring-tx.xsd
11.       http://www.springframework.org/schema/aop
12.       http://www.springframework.org/schema/aop/spring-aop.xsd">
13.
14.    <tx:annotation-driven transaction-manager="transactionManager"/>
15.    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerData
    Source">
16.       <property name="driverClassName" value="oracle.jdbc.OracleDriver"/>
17.       <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
18.       <property name="username" value="system"/>
19.       <property name="password" value="durga"/>
20.    </bean>
21.    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
22.       <property name="dataSource" ref="dataSource"/>
23.    </bean>
24.    <bean id="transactionDao" class="com.durgasoft.dao.TransactionDaoImpl">
25.       <property name="jdbcTemplate" ref="jdbcTemplate"/>
26.    </bean>
```


CONTACT US:
Mobile: +91- **8885 25 26 27**
+91- **7207 21 24 27/28**
US NUM: **4433326786**
Mail ID: **durgasoftonlinetraining@gmail.com**
WEBSITE: **www.durgasoftonline.com**
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,

```
27.    <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
28.        <property name="dataSource" ref="dataSource"/>
29.    </bean>
30.
31.
32. </beans>
```

Test.java

```
1.  package com.durgasoft.test;
2.
3.  import org.springframework.context.ApplicationContext;
4.  import org.springframework.context.support.ClassPathXmlApplicationContext;
5.
6.  import com.durgasoft.dao.TransactionDao;
7.
8.  public class Test {
9.
10.    public static void main(String[] args) {
11.        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
12.        TransactionDao tx_Dao = (TransactionDao)context.getBean("transactionDao");
13.        String status = tx_Dao.transferFunds("abc123", "xyz123", 100);
14.        System.out.println(status);
15.    }
16.}
```


CONTACT US:
Mobile: +91- **8885 25 26 27**
+91- **7207 21 24 27/28**
US NUM: **4433326786**
Mail ID: **durgasoftonlinetraining@gmail.com**
WEBSITE: **www.durgasoftonline.com**
FLAT NO: 202, HMDA MYTRIVANUM, AMEERPET, HYDERABAD.,