INDEX

Spring Tx Mgmt ----------

Spring Tx Mgmt

Introduction

Transaction Management:

- It is the process of combining related operations into single unit and executing them by applying do everything or nothing principle.
- We should always execute sensitive logics by applying TxMgmt support.

e.g.

1. Transfer Money (combination of withdraw money from source account + deposit money into destination account).
2. Employee registration (inserting record into HR DB table + inserting record into Finance DB table).
3. Student Registration (insert record into Academics DB table + insert record into Course DB table)
4. Online shopping (deducting product count from inventory DB table+ payment)

Tx: Transaction
Tx Mgmt: Transaction Management

- Tx Mgmt is Around Advice + ThrowsAdvice together.

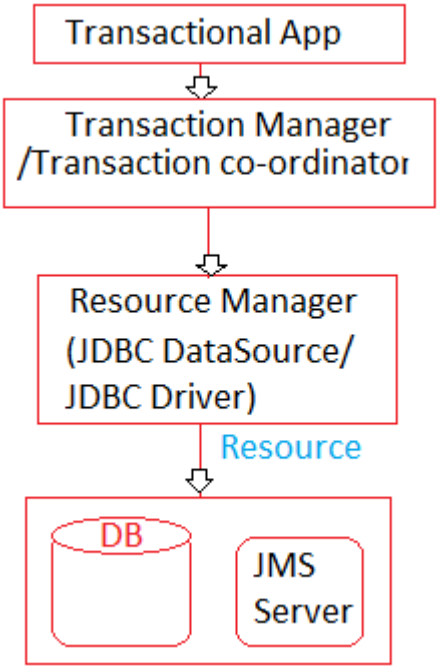
```
public void m1() {
        try {
                //begin the Tx
        JDBC -  con.setAutoCommit(false)
        HB -    Transaction tx = ses.
                        beginTransaction();
                //execute logic
                .......
                ......
                //commit Tx
        JDBC -  con.commit();
        HB -    tx.commit();
        }
        catch (Exception e) {
                //rollback the Tx
        JDBC -  con.rollback();
        HB -    tx.rollback();
        }
}
```

Can be implemented as Around Advice

Can be implemented as Throws Advice

Tx Mgmt: around advice +throws advice

Tx Mgmt architecture:



- JMS: Java Message Server (Capable of storing messages in Messages based communication).
- Transaction Manager takes the instructions like begin Tx, commit Tx, rollback Tx from Transactional application and performs those operations on the Resource(s) with the support of Resource Manager.
- We cannot take files as the Transactional resource because it does not support rollback activity i.e., we can take only DB software/ JMS server as the Transactional resources because they support commit and rollback activities.

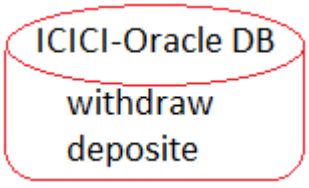
Type of Transaction:

- Based on the number of resources that are involved in a Transaction boundary there are two types of Transactions
  a. Local Tx Mgmt
  b. Global/ distributed/ XA TxMgmt

Local Tx Mgmt:

All the operations of a Transaction/ Task take place in a single a Resource i.e. In a Transaction boundary only, one resources will be there maintaining the data.

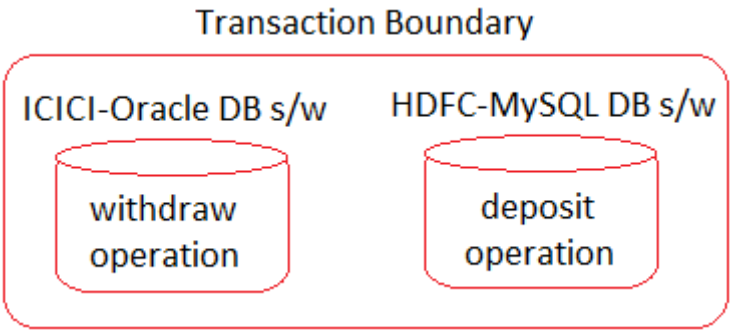
e.g.: Transfer money operation b/w two account of same bank.



Global/ Distributed/ XA Tx Mgmt:

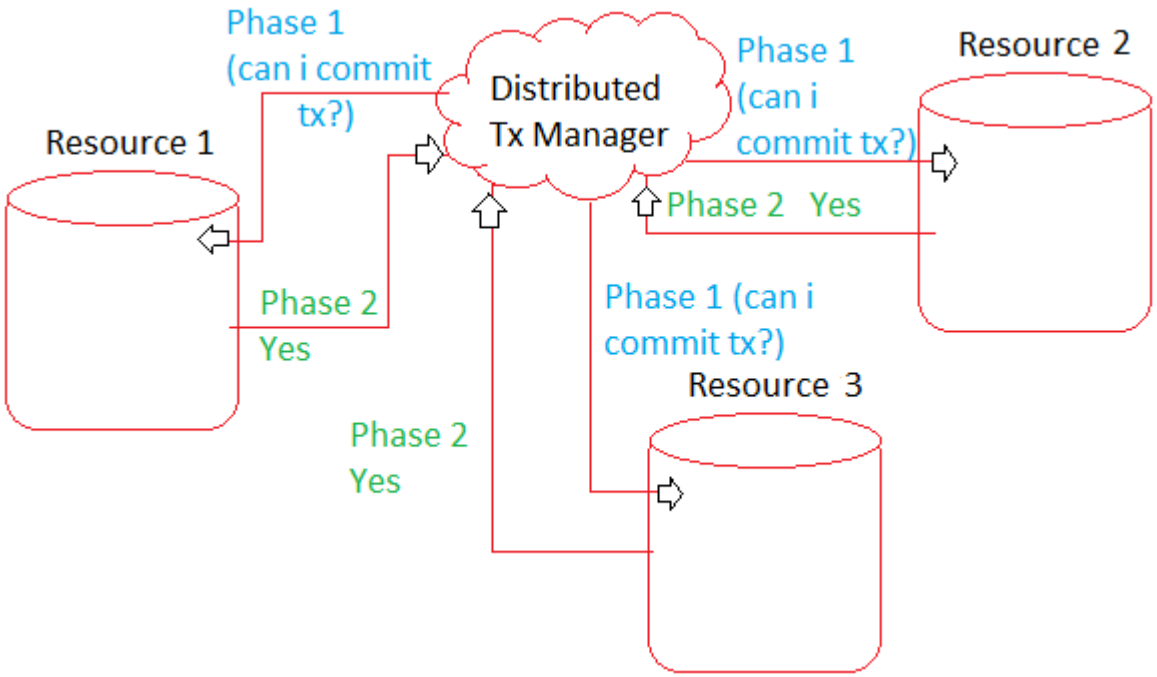
The operations of a Transaction take place on multiple resources i.e. In a Tx Boundary multiple resources will be there maintaining the data.

e.g.: Transfer Money operations b/w two different accounts of two different banks.



Note:

- ✓ Global/ Distributed / XA Transaction Mgmt takes place based on 2 PC/ protocol/ principle.
- ✓ 2PC: Two phase commit protocol/ principle.



Note:

- ✓ At end the business method execution on which Global Tx Mgmt is enabled, the Distributed Tx manager checks the weather data in all resources effected properly or not (Phase1 activity). If resource (DB softwares) give positive reply then then commits the data of resources

(DB softwares) in single short otherwise rollbacks the data of all the resources in single short (phase2).

- ✓ To work with Distributed TxMgmt we need XA DB softwares, XA JDBC drivers and XA Data Sources to support 2pc protocol.
- ✓ XA: Extended Architecture
- ✓ Oracle 10g+, MySQL 5.x+, PostgreSQL all versions are XA DB softwares.
- ✓ ojdbc14.jar, ojdbc6/ 7/ 8/ 9/ 10.jar files are XA oracle jdbc drivers which can be used for Local and Distributed Tx Mgmt.
- ✓ JDBC, Hibernate, EJB, Spring supports Local Tx Mgmt.
- ✓ Using JDBC, hibernate we can bring effect of Global/ Distributed Tx Mgmt but it is not safe.
- ✓ So, prefer using EJB or Spring or JTA (Java Transaction API).

Q. Why using JDBC, hibernate we can bring effect of Global/ Distributed Tx Mgmt but it is not safe?

Ans.

```
try {
        con1.setAutoCommit(false);
        con2.setAutoCommit(false);
        //execute queries using both con1, con2
        ……
        con1.commit();
        con2.commit();
}
catch (Exception e) {
        con1.rollback();
        con2.rollback();
}
```

If con1.commit() or con1.rollback() throws Exception then con2.commit() or con2.rollback() will not be executed so it is not safe to write Global Tx code in JDBC or Hibernate.

Note:

- ✓ We generally use JDBC or Hibernate for Local Tx Mgmt in Programmatic Approach (writing java code).
- ✓ Writing TxMgmt code along with business logic code is called Programmatic Tx Mgmt.
- ✓ Writing TxMgmt instructions as xml file-based configurations is called Declarative Tx Mgmt.

JDBC code for Local Tx Mgmt (programmatic):

```
public void bm1() {
        try {
                con.setAutoCommit(false); //begin Tx
                …….
                ……. //queries to execute
                …….
                con.commit(); //commit Tx
        }
        catch (Exception e) {
                con.rollback(); // rollback Tx
        }
} //bm1()
```

Hibernate code for Local Tx Mgmt (programmatic):

```
public void bm1() {
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try {
                tx=ses.beginTransaction();
                ……
                ……    //hb persisternce logic
                ……
                tx.commit();
        }
        catch (Exception e) {
                tx.rollback();
        }
} //bm1()
```

Note:

- ✓ JDBC, hibernate do not support Declarative Tx Mgmt.
- ✓ We generally write JTA for programmatic Global Tx Mgmt and EJB for Declarative Global Tx Mgmt.
- ✓ To perform 2PC based Global Tx Mgmt we need global/distributed Tx Manager and we can get it from Application server like WebLogic or from third party vendors like Atomikos or narayana.
- ✓ In WebLogic server the Global/Distributed Tx Manager is placed in Jndi registry having the Jndi name javax/ transaction/ User Transaction and we can access it and use it by using Jndi lookup operation.

Global Tx Mgmt in Programmatic Approach using JTA:

```
public void bm1() {
        //get global Tx manager from WebLogic Jndi registry
        InitialContext ic=new InitialContext();
        UserTransaction ut=(UserTransaction)ic.
                                lookup("javax/transaction/UserTransanction");
        try {
                ut.begin();
                …….
                …… //execute queries in multiple DB software
                ……
                ut.commit(); //commits the data of multiple DB s/w in single
                                                                        short
        }
        catch (Exception e) {
                ut.rollback(); //rollbacks the data of multiple Db s/w in single
                                                                        short
        }
} //bm1()
```

Global Tx Mgmt in Declarative Approach using EJB:

EJB component class:

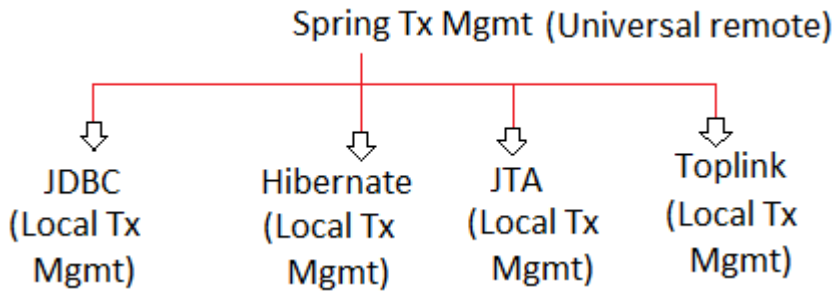
```
public void bm1() {
        …….
        …….. //only business logic (No Tx Mgmt code)
}
```

ejb-jar.xml:

- EJB component class configuration
- Tx Mgmt configuration

Problem: We are using different technologies and different styles of coding for different modes of TxMgmt. So, switching from Local TxMgmt to global Tx Mgmt or switching one technology and another technology is complex.

Solution: Use Spring Transaction Mgmt Spring provides Unified model for Transaction Mgmt i.e. it internally can multiple technologies to generate different modes of Tx Mgmt code but makes the programmer to follow standard process for Tx Mgmt. So, switching from Local Tx to global Tx and One Tx technology to another Tx Technology becomes easy by using the standard process (Universal remote).



Different Approaches to write Spring Tx Mgmt code

a. Spring Tx Mgmt in Programmatic Approach (not good against of AOP)
b. Spring AOP Declarative Tx Mgmt (outdated)
c. AspectJ AOP Declarative Tx Mgmt
d. AspectJ AOP Annotation Tx Mgmt
e. AspectJ AOP 100%code Tx Mgmt
f. AspectJ AOP Boot Tx Mgmt (new projects)

(form maintenance and small-scale projects)

- Based on the Transaction Manager that we configure the spring framework decides the mode and technology that it should use to perform Transaction Management.
- DataSourceTransactionManager --> for jdbc based Local TxMgmt
- HibernateTransactionManager --> for Hibernate based Local TxMgmt
- JTATransactionManager --> for JTA based Global TxMgmt
  and etc.
- All these classes are implementation classes of PlatformTransactionMamager (I)

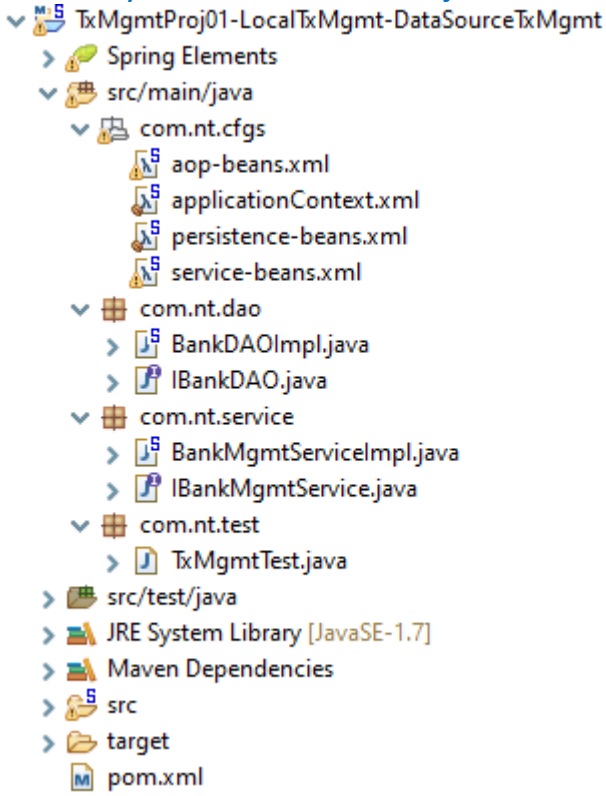
Declarative driven AspectJ AOP style Local Tx Mgmt

- Local TxMgmt using DataSourceTransactionManager (JDBC style Local Transaction Mgmt)
- Example App is transfer Money Operation b/w two accounts of same bank (same DB s/w)

Note:

✓ Always prefer to Tx Mgmt service on the service class methods because service class methods internally call DAO methods and Tx Mgmt will also be applied on DAO methods.
✓ In real time standalone application instead of client application they write JUnit Testcases to the test the output the application.

Directory Structure of IOCProj01-SetterInjection-POC:



- Develop the above directory structure and package, class, XML file and add the jar dependencies in pom.xml file then use the following code with in their respective file.

pom.xml

```
        <dependencies>
                <!--
https://mvnrepository.com/artifact/org.springframework/spring-context-
support -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context-support</artifactId>
                        <version>5.3.1</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-jdbc</artifactId>
                        <version>5.3.1</version>
                </dependency>
```

```
		</dependency>
		<!-- https://mvnrepository.com/artifact/com.zaxxer/HikariCP --
>
		<dependency>
			<groupId>com.zaxxer</groupId>
			<artifactId>HikariCP</artifactId>
			<version>3.4.5</version>
		</dependency>
		<!-- https://mvnrepository.com/artifact/org.aspectj/aspectjrt --
>
		<dependency>
			<groupId>org.aspectj</groupId>
			<artifactId>aspectjrt</artifactId>
			<version>1.9.6</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
		<dependency>
			<groupId>org.aspectj</groupId>
			<artifactId>aspectjweaver</artifactId>
			<version>1.9.6</version>
			<scope>runtime</scope>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.projectlombok/lombok -->
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<version>1.18.16</version>
			<scope>provided</scope>
		</dependency>
		<!--
https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc8 -->
		<dependency>
			<groupId>com.oracle.database.jdbc</groupId>
			<artifactId>ojdbc8</artifactId>
			<version>19.8.0.0</version>
		</dependency>
	</dependencies>
```

IBankDAO.java

```
package com.nt.dao;
public interface IBankDAO {
	public int withdraw(long accNo, double amount);
	public int deposit(long accNo, double amount);
}
```

IBankDAO.java

```
package com.nt.dao;

import org.springframework.jdbc.core.JdbcTemplate;

public class BankDAOImpl implements IBankDAO {
	private static final String WITHDRAW_QUERY = "UPDATE
BANKACCOUNT SET BALANCE=BALANCE-? WHERE ACCOUNTNO=?";
	private static final String DEPOSIT_QUERY = "UPDATE
BANKACCOUNT SET BALANCE=BALANCE+? WHERE ACCOUNTNO=?";

	private JdbcTemplate jt;

	public BankDAOImpl(JdbcTemplate jt) {
		this.jt = jt;
	}
	@Override
	public int deposit(long accNo, double amount) {
		int count = jt.update(DEPOSIT_QUERY, amount, accNo);
		return count;
	}
	@Override
	public int withdraw(long accNo, double amount) {
		int count = jt.update(WITHDRAW_QUERY, amount, accNo);
		return count;
	}
}
```

IBankMgmtService.java

```
package com.nt.service;

public interface IBankMgmtService {
	public boolean transferMoney(long srcAccNo, long desAccNo, double
amount);
}
```

BankMgmtServiceImpl.java

```
package com.nt.service;

import com.nt.dao.IBankDAO;

public class BankMgmtServiceImpl implements IBankMgmtService {

	private IBankDAO dao;

	public BankMgmtServiceImpl(IBankDAO dao) {
		this.dao = dao;
	}

	@Override
	public boolean transferMoney(long srcAccNo, long desAccNo, double
amount) {
		boolean flag = false;
		int count1 = dao.withdraw(srcAccNo, amount);
		int count2 = dao.deposit(desAccNo, amount);
		if (count1==0||count2==0)
			throw new RuntimeException("Problem in Money
transfering");
		else
			flag = true;
		return flag;
	}

}
```

service-beans.xml

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

	<!-- Configure service class -->
	<bean id="bankService"
class="com.nt.service.BankMgmtServiceImpl">
		<constructor-arg ref="bankDAO"/>
	</bean>

</beans>
```

persistence-beans.xml

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

	<!-- Data Source -->
	<bean id="hkDS" class="com.zaxxer.hikari.HikariDataSource">
		<property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"/>
		<property name="jdbcUrl"
value="jdbc:oracle:thin:@localhost:1521:xe"/>
		<property name="username"  value="system"/>
		<property name="password" value="manager"/>
		<property name="maximumPoolSize" value="10"/>
		<property name="minimumIdle" value="10000"/>
	</bean>

	<!-- JdbcTemplate configuration -->
	<bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
		<property name="dataSource" ref="hkDS"/>
	</bean>

	<!-- DAO class configuration -->
	<bean id="bankDAO" class="com.nt.dao.BankDAOImpl">
		<constructor-arg ref="jt"/>
	</bean>

</beans>
```

applicationContext.xml

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

	<import resource="persistence-beans.xml"/>
	<import resource="service-beans.xml"/>
	<import resource="aop-beans.xml"/>

</beans>
```

aop-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:aop="http://www.springframework.org/schema/aop"
	xmlns:tx="http://www.springframework.org/schema/tx"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
		http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">

	<!-- Configure Tx Manager -->
	<bean id="dsTxMgmr"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
		<constructor-arg ref="hkDS"/>
	</bean>

	<!-- Make Tx Manager as Tx Service -->
	<tx:advice id="txAdvice" transaction-manager="dsTxMgmr">
		<!-- Apply transaction attributes on the business method -->
		<tx:attributes>
			<!-- Specifies how Tx service should applied on given
transferMoney mehtod -->
			<tx:method name="transferMoney"
propagation="REQUIRED" read-only="false"/>
		</tx:attributes>
	</tx:advice>

	<aop:config>
		<!-- Pointcut -->
		<aop:pointcut expression="execution(boolean
com.nt.service.BankMgmtServiceImpl.transferMoney(..))" id="ptc"/>
		<!-- Making Tx service as AspectJ AOP advice -->
		<aop:advisor advice-ref="txAdvice" pointcut-ref="ptc"/>
	</aop:config>

</beans>
```

TxMgmtTest.java

```
package com.nt.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.service.IBankMgmtService;

public class TxMgmtTest {

	public static void main(String[] args) {
		//create container
		ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
		//get service object
		IBankMgmtService service = ctx.getBean("bankService",
IBankMgmtService.class);
		//invoke business method
		try {
			if (service.transferMoney(1003, 1000, 800))
				System.out.println("Money transfered");
			else
				System.out.println("Problem in Money
transfered");
		}
		catch (Exception e) {
			System.out.println("Money not transfered");
			e.printStackTrace();
		}
		//close container
		((AbstractApplicationContext) ctx).close();
	}

}
```

Note:

- ✓ Tx service will be applied on the transferMoney method that is given by pointcut expression. On that method Tx will be applied "REQUIRED" propagation mode.
- ✓ The Spring declarative Tx, the Transaction service will be started

automatically before getting into business method and will be committed at the end of business method if no Unchecked Exception raised. The Tx will be rolled back if unchecked exception raised (By default it will rollback only for unchecked exception, not for checked exception).

- To make Tx Service, rollbacking the Transaction for our choice checked Exceptions the we need specify that list of exceptions as comma separated list of values in "rollback-for" attribute of <tx:method> tag as shown below.
  <tx:method name="transferMoney" propagation="REQUIRED" read-only="false" rollback-for="java.sql.SQLException"/>

- The sub classes of RuntimeException are called Unchecked Exception classes whereas the immediate sub classes of java.lang.Exception are called Checked Exception classes.

- Tx Mgmt IoC container always uses JDK Libraries to generated the Tx Mgmt based Proxy class irrespective of proxy interface name is specified or not in the pointcut expression. But this can be changed to CGLIB Libraries based Proxy class generation by using <aop:config proxy-target-class="true">.

- We can specify Tx timeout period for business method using "timeout" attribute of <tx:method> tag, if the Tx started in the business method is not completed within timeout then we get org.springframework.transaction.TransactionTimedOutException: Transaction timed out: deadline was Thu Dec 10 12:46:15 IST 2020

  aop-benas.xml
  <tx:method name="transferMoney" propagation="REQUIRED" read-only="false" rollback-for="java.sql.SQLException" timeout="10"/>

Business method with Programmatic Tx Mgmt code:

```
public void bm1() {
        try {
                //begin Tx
                con.setAutoCommit(false);
                //execute other methods/ queries to complete the task
                ……
```

```
                ……
                ……
                con.commit();
        }
        catch (Exception e) {
                con.rollback();
        }
}
```

Note:

- ✓ In Programmatic Tx Mgmt, every time new Tx will begin for business method call and that Tx will be committed or rolled back at the end of business method execution.
- ✓ In Programmatic Tx Mgmt, we cannot propagate the Tx from one business method to another business method or from client application to business method because we begin Tx inside business method. So, business method runs always in new Tx.

Declarative Tx Mgmt:

```
public void bm1() {
        //logic or queries to complete task
        …….
        …….
        …….
}
```

Note: In Declarative Tx Mgmt, Tx Propagation is possible i.e. Tx can be passed from client application to business method or from 1 business method to another business method. Here, business method can run in new Tx or no Tx or client/ other business method supplied Tx based on the Tx attribute that is configured on the business method.

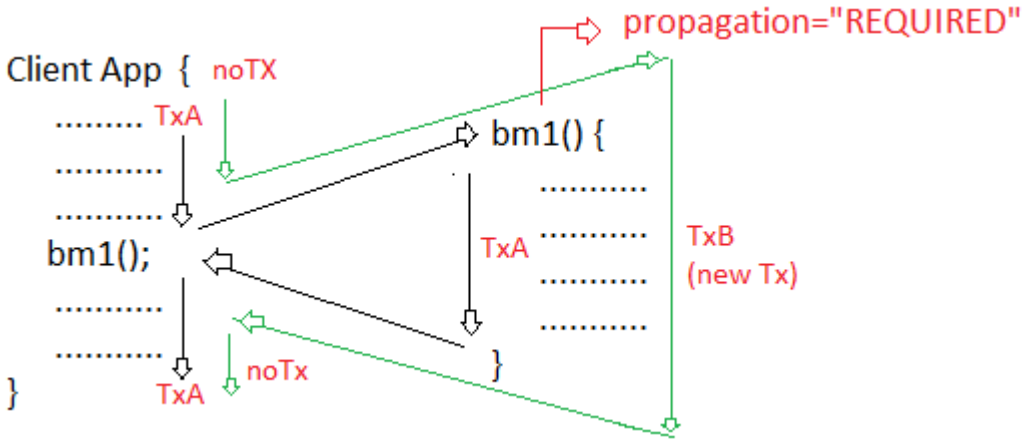
Transaction Attributes:

These are the following Tx attributes,

- REQUIRED (default) (important)
- REQUIRES_NEW
- SUPPORTS
- NOT_SUPPORTED
- MANDATORY
- NEVER

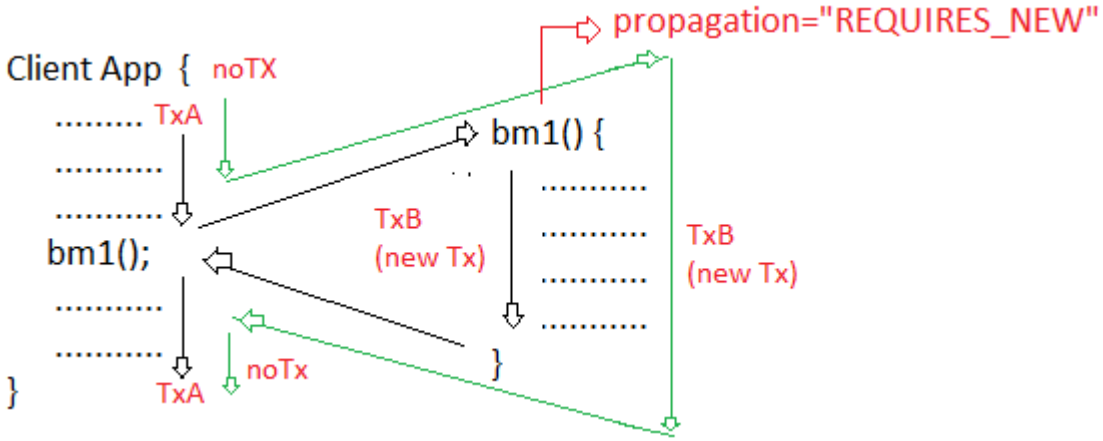
propagation="REQUIRED":

- If Caller (Client application or other business method) calls the business method having Tx then business method executes having same Tx otherwise business method executes with new Tx.



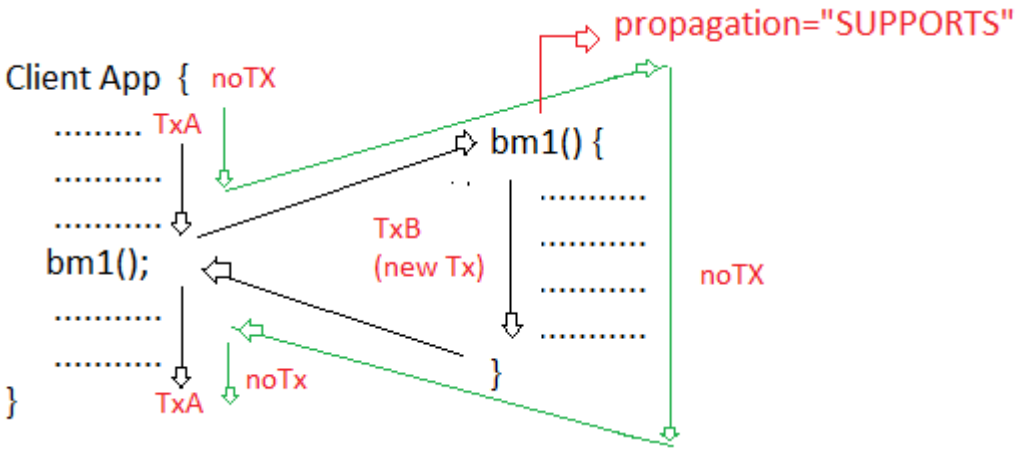
propagation="REQUIRES_NEW":

- Makes the business method to run in new Tx always irrespective of whether caller calls business method with or without Tx.



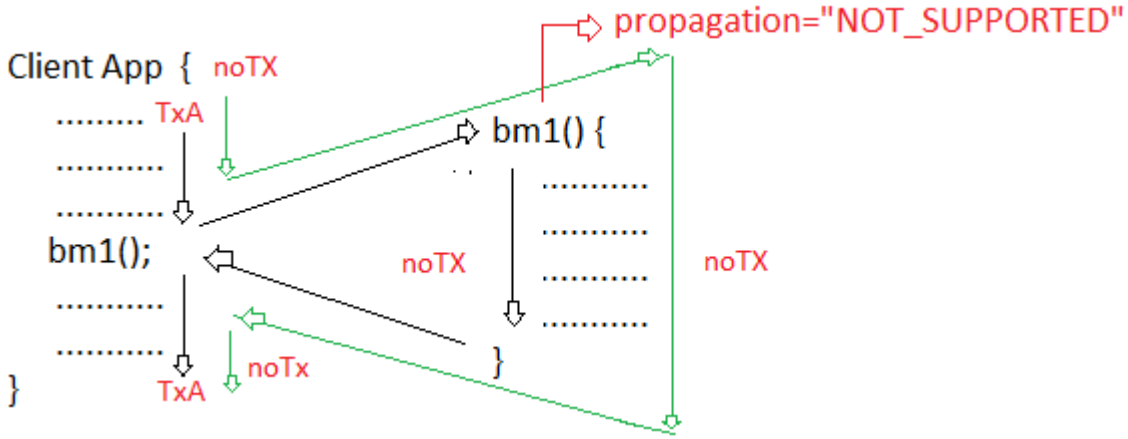
propagation="SUPPORTS":

- If the caller calls the business method with Tx, then business method runs in the same Tx other business method runs in no Tx.



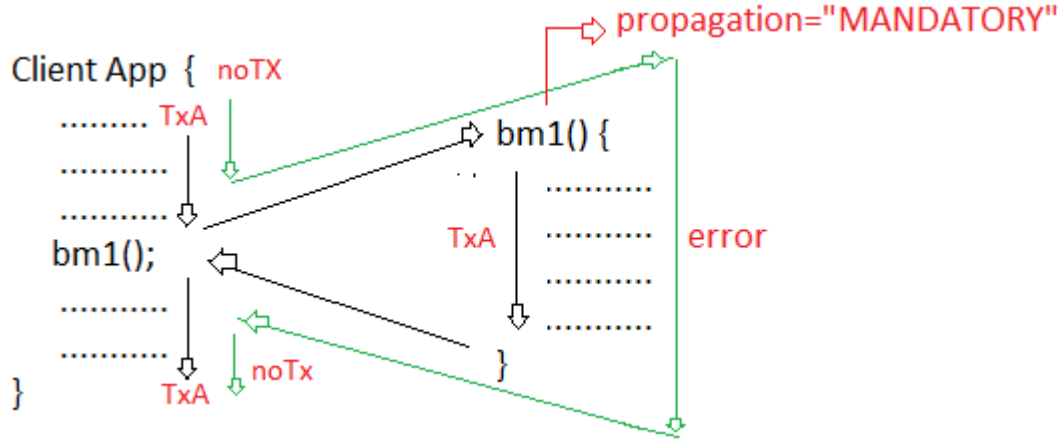
propagation="NOT_SUPPORTED":

- Business method always runs without Tx irrespective of whether caller is calling business method with or without Tx.



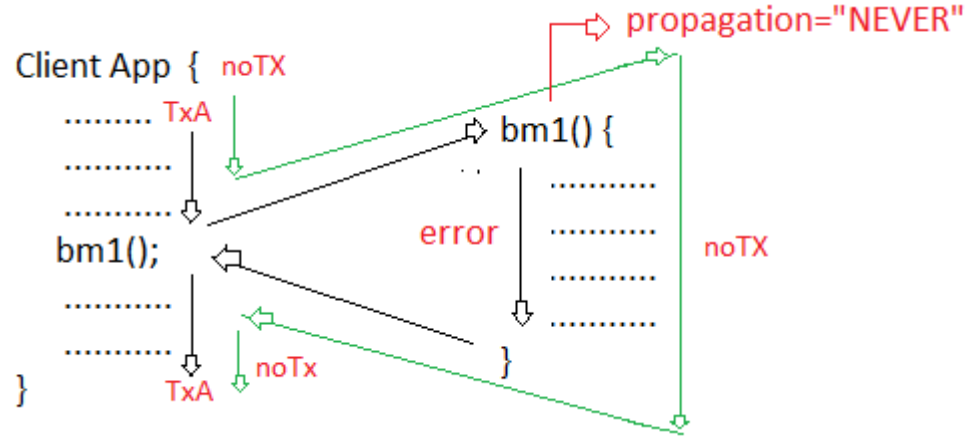
propagation="MANDATORY":

- Caller must call business method with Tx otherwise exception (org.springframework.transaction.IllegalTransactionStateException) will be raised in the execution of business method.



propagation="NEVER":

- Caller must not call business method with Tx, if called then exception will be raised.



Summary table on Tx Attributes:

Tx Attribute	Caller Tx	Business Method Tx
REQUIRED	Tx A No Tx	Tx A Tx B (new Tx)
REQUIRES_NEW	Tx A No Tx	Tx B (new Tx) Tx B (new Tx)
SUPPORTS	Tx A No Tx	Tx A No Tx
MANDATORY	Tx A No Tx	Tx A Error
NEVER	Tx A No Tx	Error No Tx
NOT_SUPPORTED	Tx A No Tx	No Tx No Tx

Q. How to make the Transaction Manager not rollbacking the Tx for certain unchecked exception raised in business method?

Ans. Use "no-rollback-for" attribute of <tx:method> tag as shown below

```
<tx:method name="transferMoney" propagation="REQUIRED" read-only="false" rollback-for="java.sql.SQLException" timeout="10" no-rollback-for="java.lang.NullPointerException"/>
```

Annotation driven AspectJ AOP Style Local Tx Mgmt

a. Apply @Transactional on the top business method(s) or on the top of service class specifying the Tx Attribute.

   Note: In order to apply same Tx attribute on all business methods use @Transactional on the top of the class otherwise use on the top of business methods.

b. Place <tx:annotation-driven> tag in aop-beans.xml file specifying the Transaction manager bean id

Directory Structure of TxMgmtAnnoProj01-LocalTxMgmt-DataSourceTxMgmt:

- Copy paste the TxMgmtProj01-LocalTxMgmt-DataSourceTxMgmt application.
- Need not to add any new file same structure as TxMgmtProj01-LocalTxMgmt-DataSourceTxMgmt
- Add the following code in their respective file.
- Rest of code same as previous project.

aop-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:aop="http://www.springframework.org/schema/aop"
	xmlns:tx="http://www.springframework.org/schema/tx"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
		http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">

	<!-- Configure Tx Manager -->
	<bean id="dsTxMgmr"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
		<constructor-arg ref="hkDS"/>
	</bean>

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

</beans>
```

persistence-beans.xml

```
	<context:component-scan base-package="com.nt.dao"/>
```

service-beans.xml

```
	<context:component-scan base-package="com.nt.service"/>
```

BankDAOImpl.java

```
@Repository("bankDAO")
public class BankDAOImpl implements IBankDAO {

	private static final String WITHDRAW_QUERY = "UPDATE
BANKACCOUNT SET BALANCE=BALANCE-? WHERE ACCOUNTNO=?";
	private static final String DEPOSIT_QUERY = "UPDATE
BANKACCOUNT SET BALANCE=BALANCE+? WHERE ACCOUNTNO=?";

	@Autowired
	private JdbcTemplate jt;
```

BankMgmtServiceImpl.java

```
@Service("bankService")
public class BankMgmtServiceImpl implements IBankMgmtService {

	@Autowired
	private IBankDAO dao;

	@Override
	@Transactional(propagation = Propagation.REQUIRED, readOnly =
false, timeout = 10)
	public boolean transferMoney(long srcAccNo, long desAccNo, double
amount) {
```

Q. Can we mark service and DAO classes with @Component?
Ans. Yes, possible but not recommended. @Service, @Repository are recommended to use, these annotations are built on the top of @Component.

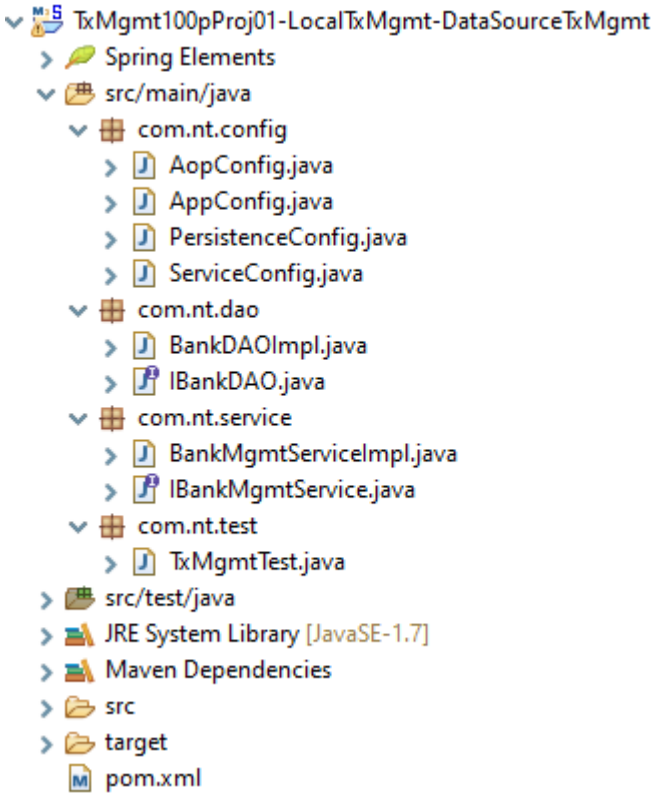
@Service = @Component + Tx Mgmt support
@Repository = @Component + Exception translation

100% Code driven AspectJ AOP Style Local Tx Mgmt

Step 1: Configure pre-defined classes as spring beans using @Bean methods of @Configuration class.
Step 2: Configure user-defined classes as spring beans using stereo type annotations of and link them with @Configuration class using @ComponentScan annotations.
Step 3: Place @Transactionsactional on the top of service class or its business methods to specify Tx details.
Step 4: Place @EnableTransactionManagement on t e top of AopConfig.java @Configuration class as alternate to <tx:annotation-driven> tag.
Step 5: Use AnnotationConfigApplicationContext class to create IoC container in the Client App.

Note: Every @Configuration class is Spring bean. So, we can do @Autowired based injections on that class. If @Bean method-based spring bean class object of one @Configuration class-based injection in another @Configuration class is required in another @Confiugation class then we need to use @Autowired annotation-based injection in other @Configuration class.

Directory Structure of TxMgmt100pProj01-LocalTxMgmt-DataSourceTxMgmt:



- Develop the above directory structure and package, class, XML file and add the jar dependencies in pom.xml file then use the following code with in their respective file.
- Rest of copy from Previous project.

AopConfig.java

```
package com.nt.config;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.jdbc.datasource.DataSourceTransactionManager;
import
org.springframework.transaction.annotation.EnableTransactionManageme
nt;

@Configuration
@EnableTransactionManagement
public class AopConfig {
```

```
	@Autowired
	private DataSource ds;

	@Bean("dsTxMgmt")
	public DataSourceTransactionManager	createDTxMgmt() {
		return new DataSourceTransactionManager(ds);
	}

}
```

AppConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.Import;
import org.springframework.stereotype.Component;

@Component
@Import(value = {AopConfig.class, PersistenceConfig.class,
ServiceConfig.class})
public class AppConfig {

}
```

PersistenceConfig.java

```
package com.nt.config;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;

import com.zaxxer.hikari.HikariDataSource;

@Configuration
@ComponentScan(basePackages = "com.nt.dao")
public class PersistenceConfig {

	@Bean(name = "hkDS")
	public DataSource createHkDS() {
		HikariDataSource hkDs = new HikariDataSource();
		hkDs.setDriverClassName("oracle.jdbc.driver.OracleDriver");
		hkDs.setJdbcUrl("jdbc:oracle:thin:@localhost:1521:xe");
		hkDs.setUsername("system");
```

```
                hkDs.setPassword("manager");
                hkDs.setMinimumIdle(10);
                hkDs.setMaximumPoolSize(100);
                return hkDs;
        }

        @Bean(name = "jt")
        public JdbcTemplate createJT() {
                return new JdbcTemplate(createHkDS());
        }

}
```

ServiceConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.nt.service")
public class ServiceConfig {

}
```

TxMgmtTest.java

```
public class TxMgmtTest {

        public static void main(String[] args) {
                //create container
                ApplicationContext ctx = new
AnnotationConfigApplicationContext(AppConfig.class);
```

Spring Boot driven AspectJ AOP Style Local Tx Mgmt

Step 1: Configure pre-defined classes as spring beans using @Bean methods of @Configuration class, if they are not coming through AutoConfiguration. Add entries in application.properties or application.yml file as inputs to AutoConfiguration.

Note: If we add spring jdbc starter to spring boot project the following spring beans will through AutoConfiguration.

- HikariCP based DataSource gathering JDBC properties from properties file
- JdbcTemplate injected with above HikariCP DataSource
- DataSourceTransactionManager injected with above HikariCP DataSource with default bean id "transactionManager"
  and etc.

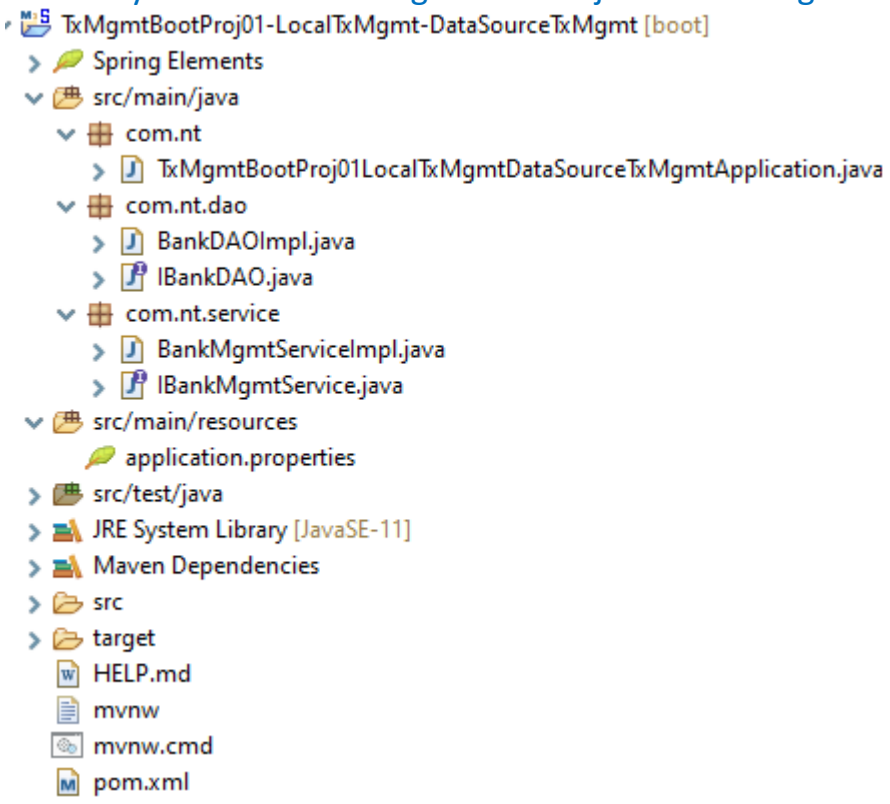
Step 2: Configure user-defined classes as spring beans using stereo type annotations. (These classes will be linked with @SpringBootApplication based @Configuration class. If keep all the classes in the sub packages of @SpringBootapplication class root package).

Step 3: Place @Transactionsactional on the top of service class or its business methods to specify Tx details.

Step 4: Get IoC container from SpringApplicaiton.run(...) method and use to get Proxy object and invoke the business method.

Note: Adding @EnableTransactionManagement is not required in spring boot application, based on spring boot starter jars that are added, it automatically enables it.

Directory Structure of TxMgmtBootProj01-LocalTxMgmt-DataSourceTxMgmt:



- Develop the above directory structure using Spring Starter Project option and package and classes also.
- While developing Spring Starter Project we choose some jars.
  - JDBC API
  - Oracle Driver
- Then use the following code with in their respective file.
- And rest of copy from previous project.

application.properties

```
#DataSource Configuration
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
```

application.properties

```
package com.nt;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;

import com.nt.service.IBankMgmtService;

@SpringBootApplication
public class TxMgmtBootProj01LocalTxMgmtDataSourceTxMgmtApplication
{

	public static void main(String[] args) {
		//get Container
		ApplicationContext ctx =
SpringApplication.run(TxMgmtBootProj01LocalTxMgmtDataSourceTxMgmt
Application.class, args);
		// get service object
		IBankMgmtService service = ctx.getBean("bankService",
IBankMgmtService.class);
		// invoke business method
		try {
			if (service.transferMoney(1001, 1000, 800))
				System.out.println("Money transfered");
```

```
                    else
                            System.out.println("Problem in Money
transfered");
                } catch (Exception e) {
                        System.out.println("Money not transfered");
                        e.printStackTrace();
                }
                // close container
                ((AbstractApplicationContext) ctx).close();
        }

}
```

Distributed Tx Mgmt/ Global Tx Mgmt/ XA Tx Mgmt

- The operations of task will be taking place multiple resources of Transaction boundary.
- This run based on 2PC (2 phase commit) protocol/ principle.
- E.g.: Transfer money operation b/w two accounts of two different banks (Two Different banks use two different DBS i.e. two resources are involved in the Tx Boundary).
- To work this TxMgmt we need Distributed Tx service and Distributed Tx Manager either given by underlying server or Third-party API (atomikos or narayana).
- To enable this Tx Mgmt, we need to configure JtaTransactionManager injecting with Server Managed or Third-party API supplied Distributed Tx service and Distributed Tx Manager.

Using Atomikos:

aop-beans.xml

```
<!-- Configure Distributed Tx Service of Atomikos -->
<bean id="dTx" class="com.atomikos.icatch.jta.UserTransactionlmp"/>

<!-- Configure Distributed Tx Manager of Atomikos -->
<bean id="dTxMgmr"
class="com.atomikos.icatch.jta.UserTransactionManager"/>

<!-- Configure Jta TransactionManager
<bean id="jtaTxMgmr"
class="org.springframework.transaction.jta.JtaTransactionManager">
```

```
        <property name="userTransaction" ref="dTx"/>
        <property name="transactionManager" ref="dTxMgmr"/>
</bean>
```

Q. Why configure JtaTransactionManager separately, when we have already got Third-party API or server Manager Distributed Transaction manager directly (like UserTransactionManager of Atomikos API)?

Ans. Spring decides the mode of TxMgmt, based on TransactionManager we configure, all these TransactionManager must implement spring supplied PlatformTransactionManager (I) directly or indirectly.

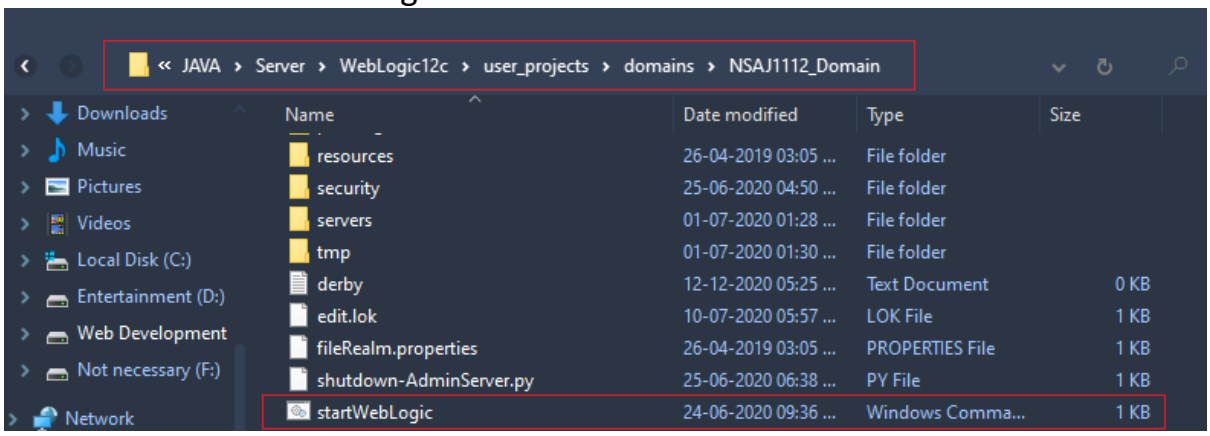
- DataSourceTransactionManager For Local Tx Mgmt using JDBC
- HibernateTransactionManager For Local Tx Mgmt using HB
- JtaTransactionManager For Global Tx Mgmt using the Injected Distributed Tx service and Distributed Tx Manager

Note: Since the Third-party API/ server supplied UserTransactionManager does not implement PlatformTransactionManager. So, we cannot configure them directly.

Using WebLogic Server Managed

Step 1: Start WebLogic domain server

Go to E:\JAVA\Server\WebLogic12c\user_projects\domains\NSAJ1112_Domain folder and use startWeblogic.cmd file



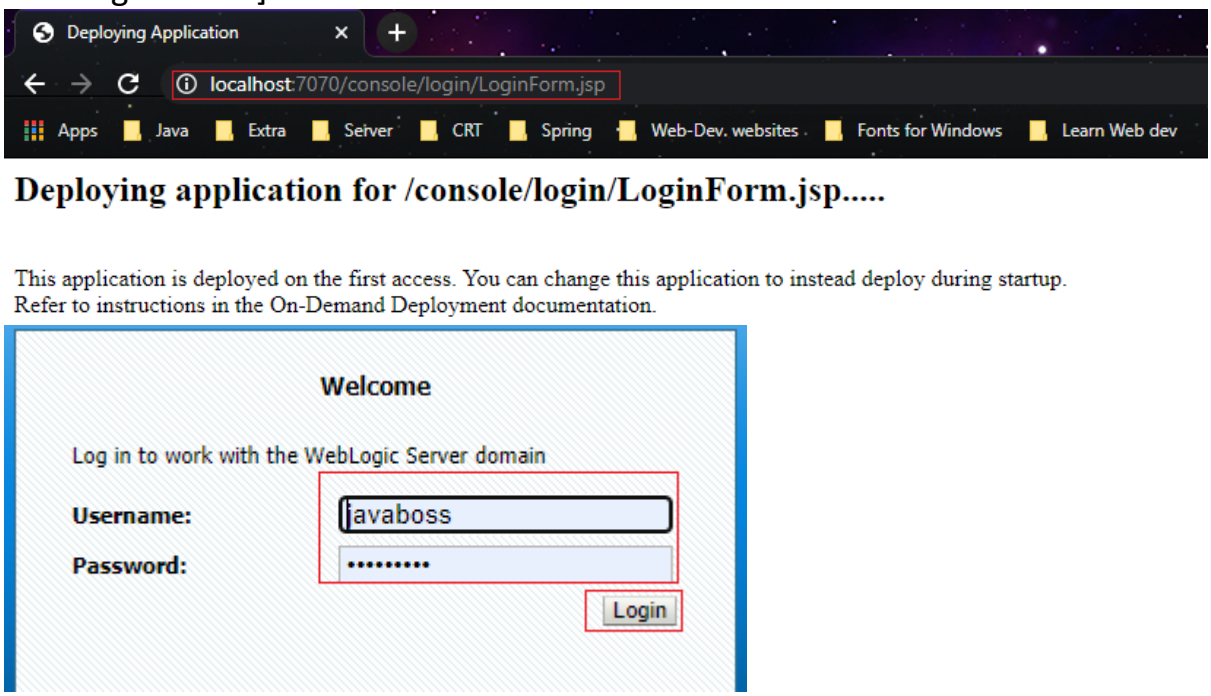
Step 2: Open admin console of WebLogic domain server.

http://localhost:7070/console

submit username: javaboss

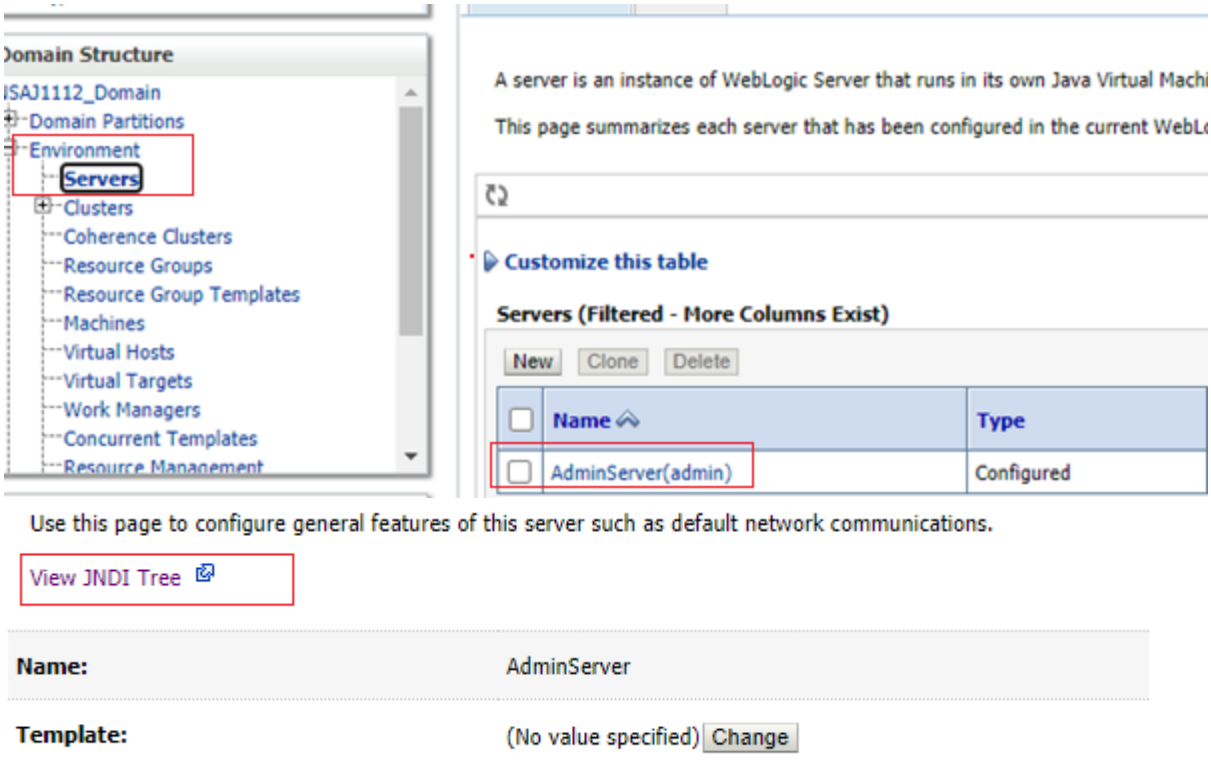
password: javaboss1

[Choose this username, password during the domain server creation in WebLogic server]

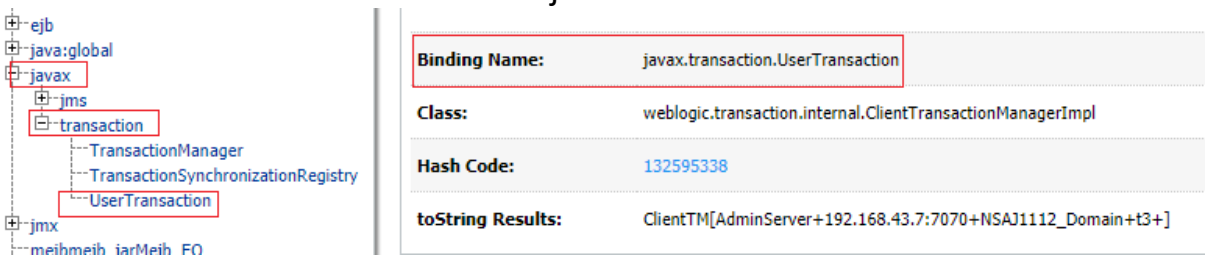


Step 3: Collect Jndi names of Distributed Tx Manager and Distributed Tx service from the Jndi registry of WebLogic server.

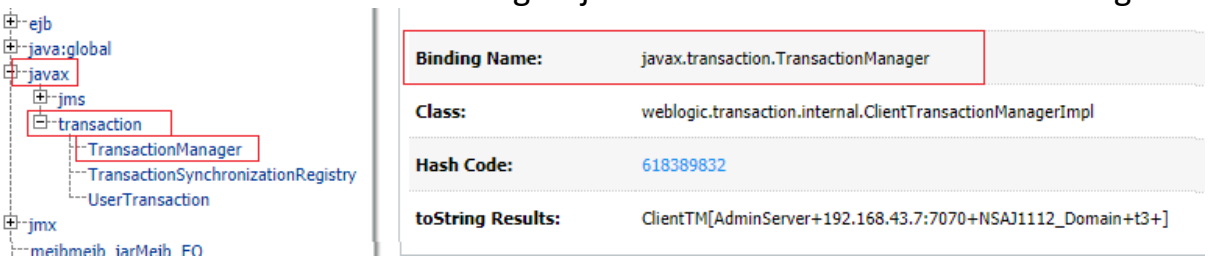
WebLogic domain server's admin console --> Environment --> servers AdminServer --> view JNDI Tree --> collect the info



Jndi name of Distributed Tx service: javax.transaction.UserTransaction



Jndi name of Distributed Tx Manager: javax.transaction.TransactionManager



Step 4: Write following entries in aop-beans.xml file

aop-beans.xml (import beans, jee namespaces)

```
<!-- Configure server managed Distributed Tx service -->
<jee:jndi-lookup id="dTxService" jndi-name="
javax.transaction.UserTransaction"/>
```

Takes the given Jndi name contacts underlying server's Jndi registry and gets given Jndi name-based object (User Transaction object) and makes it as spring bean having the bean id "dTxService".

```
<!-- Configure server Managed Distributed Tx Manager -->
<jee:jndi-lookup id="dTxManager" jndi-name="
javax.transaction.TransactionManager”/>

<!-- Configure JtaTransactionManager -->
<bean id="jtaTxMgmr"
class="org.springframework.transaction.jta.JtaTransactionManager">
        <property name="userTransaction" ref="dTxService"/>
        <property name="transactionManager" ref="dTxManager"/>
</bean>
```

Example Application:

BankAccount (DB table in Oracle) (ICICI)

acno (pk)	holdername (vc2)	balance (float)
1001	raja	9000

BankAccount (DB table in MySQL) (HDFC)

acno (pk)	holdername (vc)	balance (float)
1002	suresh	5000

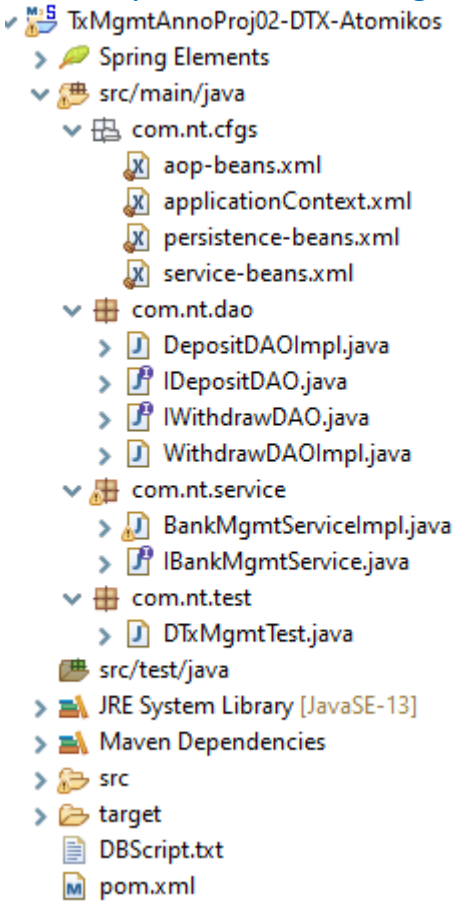
TransferMoney operations b/w two different DB softwares (Oracle, MySQL).

Using Atomikos API

- Atomikos API provides
  - Distributed Tx service (com.atomikos.icatch.jta.UserTransactionImp)
  - Distributed Tx Manager (com.atomikos.icatch.jta.UserTransactionManager)
  - XA DataSource (com.atomikos.jdbc.AtomikosDataSourceBean)

Example application: transferMoney between two accounts of two different banks (two different DB softwares)

Directory Structure of TxMgmtAnnoProj02-DTX-Atomikos:



- Develop the above directory structure and package, class, XML file and add the jar dependencies in pom.xml file then use the following code within their respective file.

pom.xml

```
        <dependencies>
                <!--
https://mvnrepository.com/artifact/org.springframework/spring-context-
support -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context-support</artifactId>
                        <version>5.3.1</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-jdbc</artifactId>
                        <version>5.3.1</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/com.zaxxer/HikariCP --
>
                <dependency>
                        <groupId>com.zaxxer</groupId>
                        <artifactId>HikariCP</artifactId>
                        <version>3.4.5</version>
                </dependency>
                <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjrt --
>
                <dependency>
                        <groupId>org.aspectj</groupId>
                        <artifactId>aspectjrt</artifactId>
                        <version>1.9.6</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
                <dependency>
                        <groupId>org.aspectj</groupId>
                        <artifactId>aspectjweaver</artifactId>
                        <version>1.9.6</version>
                        <scope>runtime</scope>
                </dependency>
                <!--
```

```
https://mvnrepository.com/artifact/org.projectlombok/lombok -->
		<dependency>
			<groupId>org.projectlombok</groupId>
			<artifactId>lombok</artifactId>
			<version>1.18.16</version>
			<scope>provided</scope>
		</dependency>
		<!--
https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc8 -->
		<dependency>
			<groupId>com.oracle.database.jdbc</groupId>
			<artifactId>ojdbc8</artifactId>
			<version>19.8.0.0</version>
		</dependency>
		<!-- https://mvnrepository.com/artifact/mysql/mysql-
connector-java -->
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>8.0.22</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/com.atomikos/atomikos-util -->
		<dependency>
			<groupId>com.atomikos</groupId>
			<artifactId>atomikos-util</artifactId>
			<version>5.0.8</version>
		</dependency>
		<!-- https://mvnrepository.com/artifact/javax.transaction/jta --
>
		<dependency>
			<groupId>javax.transaction</groupId>
			<artifactId>jta</artifactId>
			<version>1.1</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/com.atomikos/transactions-jta -->
		<dependency>
			<groupId>com.atomikos</groupId>
			<artifactId>transactions-jta</artifactId>
```

```
                <version>5.0.8</version>
            </dependency>
            <!--
https://mvnrepository.com/artifact/com.atomikos/transactions-jdbc -->
            <dependency>
                <groupId>com.atomikos</groupId>
                <artifactId>transactions-jdbc</artifactId>
                <version>5.0.8</version>
            </dependency>
        </dependencies>
```

DBScript.txt

```
DTX_BANKACCOUNT (Both Oracle and MySQL DB s/w)
        |-> accno (pk)
        |-> holdername
        |-> balance
```

aop-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:aop="http://www.springframework.org/schema/aop"
        xmlns:tx="http://www.springframework.org/schema/tx"
        xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd
                http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-4.3.xsd">

        <!-- Configure Atomikos Distributed Tx Service -->
        <bean id="dTxService"
class="com.atomikos.icatch.jta.UserTransactionImp"/>

        <!-- Configure Atomikos Distributed Tx Manager -->
        <bean id="dTxMgmr"
class="com.atomikos.icatch.jta.UserTransactionManager"/>

        <!-- Configure JtaTransactionManager -->
        <bean id="jtaTxMgmr"
```

```
class="org.springframework.transaction.jta.JtaTransactionManager">
                <property name="userTransaction" ref="dTxService"/>
                <property name="transactionManager" ref="dTxMgmr"/>
        </bean>

        <!-- Enable Annotation driven Tx Mgmt -->
        <tx:annotation-driven transaction-manager="jtaTxMgmr"/>

</beans>
```

applicationContext.xml

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

        <import resource="persistence-beans.xml"/>
        <import resource="service-beans.xml"/>
        <import resource="aop-beans.xml"/>

</beans>
```

persistence-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">


        <!-- XA Data Source configuration for Oracle given by Atomikos-->
        <bean id="oracleXADS"
class="com.atomikos.jdbc.AtomikosDataSourceBean" init-method="init"
destroy-method="close">
                <property name="uniqueResourceName" value="XAOracle"/>
                <property name="xaDataSourceClassName"
value="oracle.jdbc.xa.client.OracleXADataSource"/>
                <property name="xaProperties">
                        <props>
```

```
                        <prop key="databaseName">xe</prop>
                        <prop key="user">system</prop>
                        <prop key="password">manager</prop>
                        <prop
key="URL">jdbc:oracle:thin:@localhost:1521:xe</prop>
                    </props>
                </property>
                <property name="poolSize" value="10"/>
        </bean>

        <!-- XA Data Source configuration for MySQL given by Atomikos-->
        <bean id="mysqlXADS"
class="com.atomikos.jdbc.AtomikosDataSourceBean" init-method="init"
destroy-method="close">
                <property name="uniqueResourceName" value="XAMysql"/>
                <property name="xaDataSourceClassName"
value="com.mysql.cj.jdbc.MysqlXADataSource"/>
                <property name="xaProperties">
                    <props>
                        <prop key="databaseName">nssp713db</prop>
                        <prop key="user">root</prop>
                        <prop key="password">root</prop>
                        <prop
key="URL">jdbc:mysql:///nssp713db</prop>
                    </props>
                </property>
                <property name="poolSize" value="10"/>
        </bean>

        <!-- JdbcTemplate configuration -->
        <bean id="oracleTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
                <constructor-arg ref="oracleXADS"/>
        </bean>

        <bean id="mysqlTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
                <constructor-arg ref="mysqlXADS"/>
        </bean>

        <context:component-scan base-package="com.nt.dao"/>

</beans>
```

service-beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd">

	<context:component-scan base-package="com.nt.service"/>

</beans>
```

IDepositeDAO.java

```
package com.nt.dao;

public interface IDepositDAO {
	public int deposit(long accno, double amount);
}
```

DepositeDAOImpl.java

```
package com.nt.dao;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

@Repository("depositeDAO")
public class DepositDAOImpl implements IDepositDAO {

	private static final String DEPOSIT_QUERY = "UPDATE
DTX_BANKACCOUNT SET BALANCE=BALANCE+? WHERE ACCNO=?";

	@Autowired
	@Qualifier("mysqlTemplate")
	private JdbcTemplate mysqlJT;

	@Override
	public int deposit(long accno, double amount) {
		int count=0;
		count = mysqlJT.update(DEPOSIT_QUERY, amount, accno);
```

```
            return count;
        }

}
```

IWithdrawDAO.java

```
package com.nt.dao;

public interface IWithdrawDAO {
        public int withdraw(long accno, double amount);

}
```

WithdrawDAOImpl.java

```
package com.nt.dao;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

@Repository("withdrawDAO")
public class WithdrawDAOImpl implements IWithdrawDAO {

        private static final String WITHDRAW_QUERY = "UPDATE
DTX_BANKACCOUNT SET BALANCE=BALANCE-? WHERE ACCNO=?";

        @Autowired
        @Qualifier("oracleTemplate")
        private JdbcTemplate oraJT;

        @Override
        public int withdraw(long accno, double amount) {
                int count=0;
                count = oraJT.update(WITHDRAW_QUERY, amount, accno);
                return count;
        }

}
```

IBankMgmtService.java

```
package com.nt.service;

public interface IBankMgmtService {
	public String transferMoney(long srcAccno, long destAccno, double
amount);
}
```

BankMgmtServiceImpl.java

```
package com.nt.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

import com.nt.dao.IDepositDAO;
import com.nt.dao.IWithdrawDAO;

@Service("bankService")
public class BankMgmtServiceImpl implements IBankMgmtService {

	@Autowired
	private IWithdrawDAO withdrawDAO;

	@Autowired
	private IDepositDAO depositDAO;

	@Override
	@Transactional(propagation = Propagation.REQUIRED, readOnly =
false)
	public String transferMoney(long srcAccno, long destAccno, double
amount) {
		int count1 = withdrawDAO.withdraw(srcAccno, amount);
		int count2 = depositDAO.deposit(destAccno, amount);
		if (count1==0||count2==0)
			throw new RuntimeException("Problem in Money
transfering");
		else
			return "Money Transffered from "+ srcAccno+" to
"+destAccno;
	}
}
```

DTxMgmtTest.java

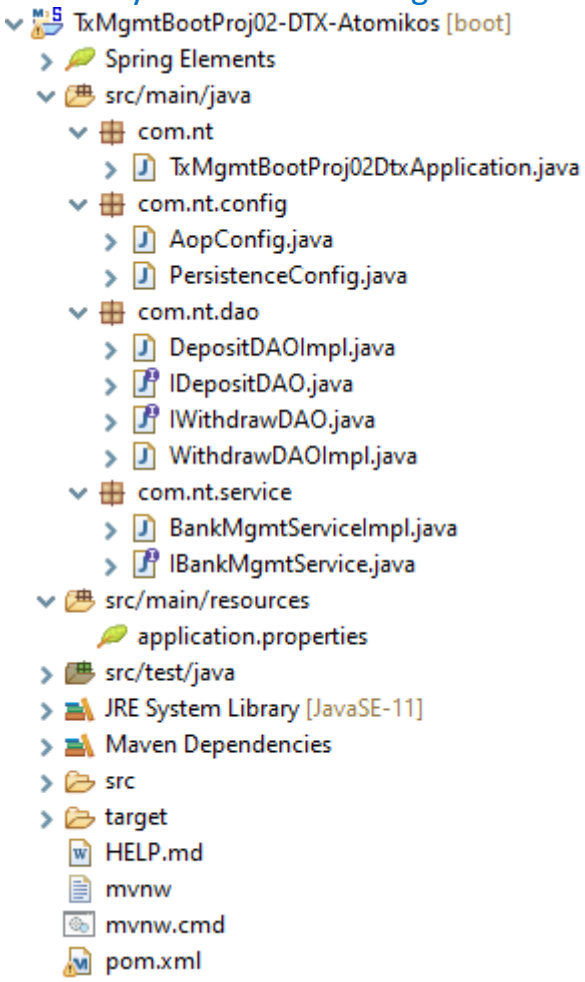
```
package com.nt.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;

import com.nt.service.IBankMgmtService;

public class DTxMgmtTest {

	public static void main(String[] args) {
		//create container
		ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
		//get service object
		IBankMgmtService service = ctx.getBean("bankService",
IBankMgmtService.class);
		//invoke business method
		try {
			System.out.println(service.transferMoney(1001, 1002,
800));
		}
		catch (Exception e) {
			System.out.println("Money not transfered");
			e.printStackTrace();
		}
		//close container
		((AbstractApplicationContext) ctx).close();
	}

}
```

Spring Boot Distributed Tx Management:

- AtomikosDataSourceBean, UserTransactionImp, UserTransactionManager classes as spring beans will not come through AutoConfiguration. So, we need to configure them explicitly by using @Bean methods @Configuration class.
- We get JdbcTemplate having HikariCP DataSource object through AutoConfiguration, but that is not our requirement. So, we need to configure JdbcTemplate having AtomikosDataSourceBean object

as the dependent object using @Bean.

Directory Structure of TxMgmtBootProj02-DTX-Atomikos:



- Develop the above directory structure using Spring Starter Project option and package and classes also.
- While developing Spring Starter Project we choose some jars.
    - JDBC API
    - Oracle Driver
    - MySQL Driver
- Then use the following code with in their respective file.
- And rest of copy from previous project and Jars also.

BankMgmtServiceImpl.java

```
	@Override
	@Transactional(propagation = Propagation.REQUIRED, readOnly = false, transactionManager = "jtaTxMgmr")
	public String transferMoney(long srcAccno, long destAccno, double amount) {
```

AopConfig.java

```
package com.nt.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.transaction.jta.JtaTransactionManager;

import com.atomikos.icatch.jta.UserTransactionImp;
import com.atomikos.icatch.jta.UserTransactionManager;

@Configuration
public class AopConfig {

	@Bean(name ="aDTxService")
	public UserTransactionImp createADTX() {
		return new UserTransactionImp();
	}

	@Bean(name ="aDTxMgmr")
	public UserTransactionManager createADTXMgmr() {
		return new UserTransactionManager();
	}

	@Bean("jtaTxMgmr")
	public JtaTransactionManager createJtaTxMgmr() {
		JtaTransactionManager jtaTxMgmr = new
JtaTransactionManager();
		jtaTxMgmr.setUserTransaction(createADTX());
		jtaTxMgmr.setTransactionManager(createADTXMgmr());
		return jtaTxMgmr;
	}

}
```

PersistenceConfig.java

```
package com.nt.config;

import java.util.Properties;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;

import com.atomikos.jdbc.AtomikosDataSourceBean;
```

```
@Configuration
@ComponentScan(basePackages = "com.nt.dao")
public class PersistenceConfig {

	@Bean(name = "oracleADSB")
	public AtomikosDataSourceBean createADSForOracle() {
		AtomikosDataSourceBean adsOracle = new
AtomikosDataSourceBean();
		adsOracle.setUniqueResourceName("oracleXADS");

	adsOracle.setXaDataSourceClassName("oracle.jdbc.xa.client.OracleX
ADataSource");
		Properties props = new Properties();
		props.setProperty("databaseName", "xe");
		props.setProperty("user", "system");
		props.setProperty("password", "manager");
		props.setProperty("URL",
"jdbc:oracle:thin:@localhost:1521:xe");
		adsOracle.setXaProperties(props);
		return adsOracle;
	}

	@Bean(name = "mysqlADBS")
	public AtomikosDataSourceBean createADSForMySQL() {
		AtomikosDataSourceBean adsMySQL = new
AtomikosDataSourceBean();
		adsMySQL.setUniqueResourceName("mysqlXADS");

	adsMySQL.setXaDataSourceClassName("com.mysql.cj.jdbc.MysqlXAD
ataSource");
		Properties props = new Properties();
		props.setProperty("databaseName", "nssp713db");
		props.setProperty("user", "root");
		props.setProperty("password", "root");
		props.setProperty("URL", "jdbc:mysql:///nssp713db");
		adsMySQL.setXaProperties(props);
		return adsMySQL;
	}

	@Bean(name = "oracleTemplate")
	public JdbcTemplate createJTForOracle() {
```

```
		return new JdbcTemplate(createADSForOracle());
	}

	@Bean(name = "mysqlTemplate")
	public JdbcTemplate createJTForMySQL() {
		return new JdbcTemplate(createADSForMySQL());
	}

}
```

TxMgmtBootProj02DtxApplication.java

```
package com.nt;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;

import com.nt.service.IBankMgmtService;

@SpringBootApplication
public class TxMgmtBootProj02DtxApplication {

	public static void main(String[] args) {
		//create container
		ApplicationContext ctx = 
SpringApplication.run(TxMgmtBootProj02DtxApplication.class, args);
		//get service object
		IBankMgmtService service = ctx.getBean("bankService", 
IBankMgmtService.class);
		//invoke business method
		try {
			System.out.println(service.transferMoney(1001, 1002, 
800));
		}
		catch (Exception e) {
			System.out.println("Money not transfered");
			e.printStackTrace();
		}
	}
}
```

---------- The END ----------