**3. Annotation Tx management mechanism:**

|-Working with Default Exception in tx's

|-Throwing user Checked Exception in transactionality

|-Throwing user Unchecked Exception in transactionality

**Working with Default Exception in tx's:**

public class VehicleService {

public void register(VehicleRegistrationVO vehicleRegistrationVO) {

....

ownerId = ownerDao.saveOwnerAndReturnId(ownerBO);

vehicleId = vehicleDao.findVehicleId(

vehicleRegistrationVO.getMake(), vehicleRegistrationVO.getModel());

}

}

Here vehicleId = vehicleDao.findVehicleId() will throws spring specific exception hence whenever vehicleId not found spring will understands and applies the tx without any problem bcz we are writing the try-catch and we are nor converting spring exception to our user exception so tx will be applied successfully.

Here we are going to work with Spring Annotation based transactionality hence instead of writing config for transactionality we need to use annotations.

As we are not writing the Tx Aspect class hence we need to put as it <tx:advice> tag bcz this will internally invokes the Tx Aspect class to transaction and along with this we need to put <aop:config> as well bcz this will talks about the pointcut and which classes we need to apply the cross-cutting logic.

But within <tx:advice> we are specifying the <tx:attribute> which will applies the transaction with the help of transaction manager on the <tx:method> hence this logic we apply using annotations hence we can remove <tx:attribute> tag and we can configure this by using annotations with the help of @transactional annotation on which method we wanted to apply the tx that will becomes transactional method but by default IOC will not recognizes the tx-annotations hence we need to enable the <tx:annotation-driven /> tag within the aop-beans.xml as follows.

public class VehicleService {

// Annotation based tx

@Transactional(readOnly = false)

public void register(VehicleRegistrationVO vehicleRegistrationVO) {

....

ownerId = ownerDao.saveOwnerAndReturnId(ownerBO);

vehicleId = vehicleDao.findVehicleId(

vehicleRegistrationVO.getMake(), vehicleRegistrationVO.getModel());

}

}

<beans>

<tx:advice id="txAdvice" transaction-manager="transactionManager"/>

<aop:config>

<aop:pointcut expression="within(com.dt.service.\*)" id="txPointcut" />

<aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut" />

</aop:config>

<tx:annotation-driven />

</beans>

**Throwing user Checked Exception in transactionality:**

If we are throwing User-defined checked exception then we have to specify for which exception the transaction need to be rollback hence we need to specify the exception in the @Transactional annotation. **Same as like Declarative Tx’s.**

**If we want we can specify multiple exceptions in the @Transactional attribute if our register() method is throwing multiple exceptions.**

class VehicleService {

.....

// Annotation based tx

/\*\*

\* @param vehicleRegistrationVO

\* @throws VehicleNotFoundException

\*/

// Test: 1

// @Transactional(readOnly = false)

// Output: Observation

// Test: 1

/\*\*

\* Observation: As our service class method register() is throwing checked

\* VehicleNotFoundException which is user defined exception hence for which

\* exception the tx should be rollback we have to specify otherwise tx will

\* become inconsistence. For example in the Test: 1 we didn't specified the

\* exception but we throwing our exception hence tx will become

\* in-consistence bcz spring will not know for which exception it has

\* rollback hence it looks for default spring exception but we didn't throw

\* default exception of spring as it is rather we thrown our exception hence

\* tx becomes inconsistence.

\*

\* So if we throw our exception we have to specify for which exception it

\* has to roll-back so that tx will become consistence which has been shown

\* in Test: 2

\*/

// Test: 2

@Transactional(readOnly = false,

rollbackForClassName = { "VehicleNotFoundException" })

public void register(VehicleRegistrationVO vehicleRegistrationVO)

try {

vehicleId = vehicleDao.findVehicleId(

vehicleRegistrationVO.getMake(),

vehicleRegistrationVO.getModel());

} catch (DataAccessException e) {

throw new VehicleNotFoundException("Vehicle make and model are not valid", e);

}

}

}

If we wanted to throw Our own Exception then we need to write exception in the catch block by catching with Spring DataAccessException then we can throw our own exception.

aop-beans.xml

<beans>

<tx:advice id="txAdvice" transaction-manager="transactionManager"/>

<aop:config>

<aop:pointcut expression="within(com.dt.service.\*)" id="txPointcut" />

<aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut" />

</aop:config>

<tx:annotation-driven />

</beans>

For Test:2 the transaction will consistence bcz we specified for which exception transaction should be rollback bcz we are not throwing as it is default exception rather we are throwing user exception.

**Throwing user Unchecked Exception in transactionality**

If we are throwing Unchecked-Exception then we no need to specify the exception for which it has to roll back explanation same as like Declarative Tx’s.