Spring Transaction Management

- Spring has its own Transactional model which is common for all the persistence implementations.
- Without spring, you need to use persistence provider specific API to manage the transactions but with spring, you can use uniform API to manage the transactions for various persistence providers.
- Various Transaction managers are provided for various persistence providers.
- PlatformTransactionManager is the root for all the Transaction managers in spring.
- Following are the methods provided in PlatformTransactionManager:

TransactionStatus	getTransaction(TransactionDefinition definition)
void	commit(TransactionStatus status)
void	rollback(TransactionStatus status)

- Following are various Transaction managers provided which are sub classes of PlatformTransactionManager.

1)	DataSourceTransactionManager	(For JDBC)
2)	HibernateTransactionManager	(For Hibernate)
3)	JpaTransactionManager	(For JPA)

- Spring supports the following ways to manage the Transactions:
    1) Programmatic Transactions
    2) Declarative Transactions (AOP Style)

1) Programmatic Transactions:

package org.springframework.transaction

interface TransactionDefinition

static int	ISOLATION_READ_COMMITTED
static int	ISOLATION_READ_UNCOMMITTED
static int	ISOLATION_REPEATABLE_READ
static int	ISOLATION_SERIALIZABLE
static int	PROPAGATION_REQUIRED
static int	PROPAGATION_REQUIRES_NEW
static int	PROPAGATION_SUPPORTS
static int	PROPAGATION_NOT_SUPPORTED
static int	PROPAGATION_MANDATORY
static int	PROPAGATION_NESTED
static int	PROPAGATION_NEVER
static int	TIMEOUT_DEFAULT
int	getIsolationLevel()
String	getName()
int	getPropagationBehavior()
int	getTimeout()
boolean	isReadOnly()

package org.springframework.transaction.support

class DefaultTransactionDefinition implements TransactionDefinition

DefaultTransactionDefinition()	
DefaultTransactionDefinition(int propagation)	
DefaultTransactionDefinition(TransactionDefinition def)	
static int	**ISOLATION_DEFAULT**
static int	**ISOLATION_READ_COMMITTED**
static int	**ISOLATION_READ_UNCOMMITTED**
static int	**ISOLATION_REPEATABLE_READ**
static int	**ISOLATION_SERIALIZABLE**
static int	**PROPAGATION_REQUIRED**
static int	**PROPAGATION_REQUIRES_NEW**
static int	**PROPAGATION_SUPPORTS**
static int	**PROPAGATION_NOT_SUPPORTED**
static int	**PROPAGATION_MANDATORY**
static int	**PROPAGATION_NESTED**
static int	**PROPAGATION_NEVER**
static int	**TIMEOUT_DEFAULT**
int	**getIsolationLevel()**
String	**getName()**
int	**getPropagationBehavior()**
int	**getTimeout()**
boolean	**isReadOnly()**

package org.springframework.jdbc.datasource

class DataSourceTransactionManager

DataSourceTransactionManager()	
DataSourceTransactionManager(DataSource dataSource)	
void	**afterPropertiesSet()**
DataSource	**getDataSource()**
void	**setDataSource(DataSource dataSource)**
TransactionStatus	**getTransaction(TransactionDefinition definition)**
void	**commit(TransactionStatus status)**
void	**rollback(TransactionStatus status)**

package org.springframework.orm.hibernate5

class HibernateTransactionManager

HibernateTransactionManager()	
HibernateTransactionManager(SessionFactory sessionFactory)	
DataSource	getDataSource()
SessionFactory	getSessionFactory()
void	setDataSource(DataSource dataSource)
void	setSessionFactory(SessionFactory sessionFactory)
TransactionStatus	getTransaction(TransactionDefinition definition)
void	commit(TransactionStatus status)
void	rollback(TransactionStatus status)

Attributes of @Transactional

SNO	attribute name	Possible values	default
1	propagation	Propagation enum constant	REQUIRED
2	isolation	Isolation enum constant	DB vendor
3	readOnly	true / false	false
4	timeout	integer	-
5	rollbackFor	exception array	-
6	noRollbackFor	exception array	-

Table Required:

```
Create table accounts (
accno int primary key,
bal double,
atype char(2)
);

INSERT INTO accounts values (101, 50000,'SA');
INSERT INTO accounts values (102, 35000,'CA');
INSERT INTO accounts values (103, 45500,'SA');
```

Lab63: Spring Programatic Transactions with JDBC

Lab63: Files required

1. Lab63.java	2. AccountDAO.java
3. JdbcAccountDAO.java	4. InSufficientFundsException.java
5. JLCConfig.java	

1. Lab63.java

```
package com.coursecube.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @company : CourseCube
* @see : www.coursecube.com
* */
public class Lab63 {
public static void main(String[] args) {
ApplicationContext ctx= new AnnotationConfigApplicationContext(JLCConfig.class);
AccountDAO adao = (AccountDAO) ctx.getBean("adao");

// 1.deposit()
System.out.println(adao.getBalance(101));
adao.deposit(101, 2000.0);
System.out.println(adao.getBalance(101));

// 2.withdraw()
System.out.println(adao.getBalance(102));
adao.withdraw(102, 5000.0);
System.out.println(adao.getBalance(102));

// 3. fundsTransfer()
System.out.println(adao.getBalance(103));
System.out.println(adao.getBalance(101));
try {
adao.fundsTransfer(103, 101, 30000.0);
}catch(Exception ex) {
System.out.println("In Main : "+ex);
}
System.out.println(adao.getBalance(103));
System.out.println(adao.getBalance(101));

}
}
```

2. AccountDAO.java

```
package com.coursecube.spring;
/*
* @Author : Srinivas Dande
* @company : CourseCube
* @see : www.coursecube.com
* */
public interface AccountDAO {
public double getBalance(int accno);
public void deposit(int accno,double amt);
public void withdraw(int accno,double amt);
public void fundsTransfer(int saccno,int daccno,double amt);
}
```

3. JdbcAccountDAO.java

```
package com.coursecube.spring;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;
/*
* @Author : Srinivas Dande
* @company : CourseCube
* @see : www.coursecube.com
* */
@Repository("adao")
public class JdbcAccountDAO implements AccountDAO {

@Autowired
JdbcTemplate jtemp;

@Autowired
PlatformTransactionManager txManager;

public double getBalance(int accno) {
String sql = "select bal from accounts where accno=?";
double cbal = jtemp.queryForObject(sql, Double.class, accno);
return cbal;
}
```

```
public void deposit(int accno, double amt) {
TransactionStatus ts=null;
try{
TransactionDefinition txDef=new
DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED);
ts=txManager.getTransaction(txDef);

String sql1="select bal from accounts where accno=?";
double cbal=jtemp.queryForObject(sql1, Double.class, accno);
cbal=cbal+amt;
String sql2="update accounts set bal=? where accno=?";
jtemp.update(sql2, cbal,accno);

txManager.commit(ts);
}catch(Exception ex) {
txManager.rollback(ts);
}
}


public void withdraw(int accno, double amt) {
TransactionStatus ts = null;
try {
TransactionDefinition txDef = new
DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED);
ts = txManager.getTransaction(txDef);

String sql1 = "select bal from accounts where accno=?";
double cbal = jtemp.queryForObject(sql1, Double.class, accno);
if (cbal >= 10000 + amt) {
cbal = cbal - amt;
String sql2 = "update accounts set bal=? where accno=?";
jtemp.update(sql2, cbal, accno);
} else {
throw new InSufficientFundsException();
}

txManager.commit(ts);
} catch (Exception ex) {
txManager.rollback(ts);
}
}
```

```
public void fundsTransfer(int saccno, int daccno, double amt) {
TransactionStatus ts=null;
try{
TransactionDefinition txDef=new
DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRES_NEW);
ts=txManager.getTransaction(txDef);  //New Tx

deposit(daccno,amt);
withdraw(saccno, amt);

txManager.commit(ts);
}catch(Exception ex){
txManager.rollback(ts);
}
}
}
```

4. InSufficientFundsException.java

```
package com.jlcindia.spring;
/*
* @Author : Srinivas Dande
* @company : CourseCube
* @see : www.coursecube.com
* */
public class InSufficientFundsException extends RuntimeException { }
```

5. JLCConfig.java

```
package com.coursecube.spring;

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 org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
/*
* @Author : Srinivas Dande
* @company : CourseCube
* @see : www.coursecube.com
* */
@Configuration
@ComponentScan(basePackages="com.coursecube.spring" )
public class JLCConfig {
```

```
@Bean
public DataSource mysqlDS() {
DriverManagerDataSource ds = new DriverManagerDataSource();
ds.setDriverClassName("com.mysql.jdbc.Driver");
ds.setUrl("jdbc:mysql://localhost/jlcdb");
ds.setUsername("root");
ds.setPassword("srinivas");
return ds;
}

@Bean
public JdbcTemplate getJdbcTemp(DataSource dataSource) {
return new JdbcTemplate(dataSource);
}

@Bean
public DataSourceTransactionManager getTxManager(DataSource dataSource) {
return new DataSourceTransactionManager(dataSource);
}
}
```

Lab 64: Spring Declarative Transactions with JDBC.

Lab64: Files required

1. Lab64.java	Same as Lab63
2. AccountDAO.java	Same as Lab63
3. JdbcAccountDAO.java	**Updated in Lab64**
4. InSufficientFundsException.java	Same as Lab63
5. JLCConfig.java	Same as Lab63

3. JdbcAccountDAO.java

```
package com.coursecube.spring;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.annotation.Isolation;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.transaction.support.DefaultTransactionDefinition;
/*
* @Author : Srinivas Dande
* @company : CourseCube
* @see : www.coursecube.com
* */
@Repository("adao")
@Transactional
public class JdbcAccountDAO implements AccountDAO {
@Autowired
JdbcTemplate jdbcTemp;

public double getBalance(int accno) {
String sql = "select bal from accounts where accno=?";
double cbal = jdbcTemp.queryForObject(sql, Double.class, accno);
return cbal;
}

@Transactional(propagation=Propagation.REQUIRED,isolation=Isolation.READ_COMMITTED)
public void deposit(int accno, double amt) {
String sql1="select bal from accounts where accno=?";
double cbal=jdbcTemp.queryForObject(sql1, Double.class, accno);
cbal=cbal+amt;
String sql2="update accounts set bal=? where accno=?";
jdbcTemp.update(sql2, cbal,accno);
}

@Transactional(propagation=Propagation.REQUIRED,isolation=Isolation.READ_COMMITTED)
public void withdraw(int accno, double amt) {
String sql1 = "select bal from accounts where accno=?";
double cbal = jdbcTemp.queryForObject(sql1, Double.class, accno);
if (cbal >= 10000 + amt) {
cbal = cbal - amt;
String sql2 = "update accounts set bal=? where accno=?";
jdbcTemp.update(sql2, cbal, accno);
} else {
throw new InSufficientFundsException();
}
}
@Transactional(propagation=Propagation.REQUIRES_NEW,isolation=Isolation.READ_COMMITTED)
public void fundsTransfer(int saccno, int daccno, double amt) {
deposit(daccno,amt);
withdraw(saccno, amt);
}
}
```

Lab 65: Spring Programmatic Transactions with Hibernate.

Lab65: Files required

1. **Lab65.java**	Same as Lab63
2. **Account.java**	**New in Lab65**
3. **AccountDAO.java**	Same as Lab63
4. **HibernateAccountDAO.java**	**Updated in Lab65**
5. **InSufficientFundsException.java**	Same as Lab63
6. **JLCConfig.java**	**Updated in Lab65**

2. Account.java

```
package com.coursecube.spring;

import javax.persistence.*;
/*
* @Author : Srinivas Dande
* @company : CourseCube
* @see         : www.coursecube.com
* */
@Entity
@Table(name = "accounts")
public class Account {
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
@Column(name="accno")
private int accno;

@Column(name="atype")
private String atype;

@Column(name="bal")
private double bal;

public Account() { }
public Account(String atype, double bal) {
this.atype = atype;
this.bal = bal;
}
public Account(int accno,String atype, double bal) {
this.accno=accno;
this.atype = atype;
this.bal = bal;
}
//Setters and Getters
}
```

5. HibernateAccountDAO.java

```
package com.coursecube.spring;

import org.hibernate.LockMode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;

@Repository("adao")
public class HibernateAccountDAO implements AccountDAO {

@Autowired
HibernateTemplate htemp;

@Autowired
PlatformTransactionManager txManager;

public double getBalance(int accno) {
Account acc = htemp.load(Account.class, accno,LockMode.READ);
return acc.getBal();
}

public void deposit(int accno, double amt) {
TransactionStatus ts = null;
try {
TransactionDefinition txDef = new
DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED);
ts = txManager.getTransaction(txDef);

Account acc = htemp.load(Account.class, accno);
acc.setBal(acc.getBal() + amt);
htemp.update(acc);

txManager.commit(ts);
} catch (Exception e) {
e.printStackTrace();
txManager.rollback(ts);
}
}
public void withdraw(int accno, double amt) {
TransactionStatus ts = null;
try {
TransactionDefinition txDef = new
DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRED);
ts = txManager.getTransaction(txDef);

Account acc = htemp.load(Account.class, accno);
double cbal = acc.getBal();
if (cbal >= 10000 + amt) {
acc.setBal(cbal - amt);
htemp.update(acc);
} else {
throw new InSufficientFundsException();
}

txManager.commit(ts);
} catch (Exception e) {
e.printStackTrace();
txManager.rollback(ts);
}
}


public void fundsTransfer(int saccno, int daccno, double amt) {
TransactionStatus ts = null;
try {
TransactionDefinition txDef = new
DefaultTransactionDefinition(TransactionDefinition.PROPAGATION_REQUIRES_NEW);
ts = txManager.getTransaction(txDef); // New Tx

deposit(daccno, amt);
withdraw(saccno, amt);

txManager.commit(ts);
} catch (Exception e) {
e.printStackTrace();
txManager.rollback(ts);
}
}
}
```

6. JLCConfig.java

```
package com.coursecube.spring;

import java.util.Properties;
import javax.sql.DataSource;
import org.hibernate.SessionFactory;
import org.springframework.context.annotation.*;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.orm.hibernate5.HibernateTransactionManager;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;
/*
* @Author : Srinivas Dande
* @company : CourseCube
* @see : www.coursecube.com
* */
@Configuration
@ComponentScan(basePackages="com.coursecube.spring" )
public class JLCConfig {

@Bean(name="jlcDataSource")
public DataSource dataSource() {
DriverManagerDataSource ds = new DriverManagerDataSource();
ds.setDriverClassName("com.mysql.jdbc.Driver");
ds.setUrl("jdbc:mysql://localhost:3306/jlcdb");
ds.setUsername("root");
ds.setPassword("srinivas");
return ds;
}

@Bean(name="jlcSessionFactory")
public LocalSessionFactoryBean sessionFactory(DataSource dataSource) {
LocalSessionFactoryBean factoryBean = new LocalSessionFactoryBean();

Properties props = new Properties();
props.put("hibernate.show_sql", "true");
props.put("hibernate.hbm2ddl.auto", "update");
props.put("hibernate.transaction.factory_class",
"org.hibernate.transaction.JDBCTransactionFactory");

factoryBean.setHibernateProperties(props); //1
factoryBean.setDataSource(dataSource); //2
factoryBean.setPackagesToScan("com.coursecube.spring"); //3

return factoryBean;
}
@Bean(name="jlcHibernateTemplate")
public HibernateTemplate hibernateTemplate(SessionFactory sessionFactory) {
HibernateTemplate htemp=new HibernateTemplate();
htemp.setSessionFactory(sessionFactory);
return htemp;
}

@Bean(name="jlcHibernateTransactionManager")
public PlatformTransactionManager transactionManager(SessionFactory sessionFactory) {
return new HibernateTransactionManager(sessionFactory);
}
}
```

Lab 66: Spring Declarative Transactions with Hibernate.

Lab66: Files required

1. Lab66.java	Same as Lab65
2. Account.java	Same as Lab65
3. AccountDAO.java	Same as Lab65
4. HibernateAccountDAO.java	**Updated in Lab65**
5. InSufficientFundsException.java	Same as Lab65
6. JLCConfig.java	Same as Lab65

4. HibernateAccountDAO.java

```
package com.coursecube.spring;

import org.hibernate.LockMode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.*;

@Repository("adao")
@Transactional
public class HibernateAccountDAO implements AccountDAO {

@Autowired
HibernateTemplate htemp;

public double getBalance(int accno) {
Account acc = htemp.load(Account.class, accno,LockMode.READ);
return acc.getBal();
}
@Transactional(propagation=Propagation.REQUIRED,isolation=Isolation.READ_COMMITTED)
public void deposit(int accno, double amt) {

Account acc = htemp.load(Account.class, accno);
acc.setBal(acc.getBal() + amt);
htemp.update(acc);
}


@Transactional(propagation=Propagation.REQUIRED,isolation=Isolation.READ_COMMITTED)
public void withdraw(int accno, double amt) {
Account acc = htemp.load(Account.class, accno);
double cbal = acc.getBal();
if (cbal >= 10000 + amt) {
acc.setBal(cbal - amt);
htemp.update(acc);
} else {
throw new InSufficientFundsException();
}
}


@Transactional(propagation=Propagation.REQUIRES_NEW,isolation=Isolation.READ_COMMITTED)
public void fundsTransfer(int saccno, int daccno, double amt) {
deposit(daccno, amt);
withdraw(saccno, amt);
}
}
```