Spring With Hibernate & Spring with JPA:

Spring Data access with Hibernate:

When you want to perform any persistent operation with Hibernate then you need to write the Hibernate Code with the following steps:

1. Write the persistence class
2. Write the Mapping document or use annotations with persistence class
3. Client code

```
Try{                              //1
//take the SessionFactory //2
//Take session              //3
//Begin transaction         //4
//Perform persistent operation //5
//End transaction           //6
//Close session             //7
} catch (Exception e) {}
```

You can Visit the following problems with above code:

1. All the above statements other than 5 are common for all the persistent operations. Writing same client code multiple times repeatedly gives you code duplication problem.
2. All the methods in Hibernate API are throwing one common exception called org. Hibernate. Hibernate Exception which is checked exception. Because of checked exception, you need to write try/ catch blocks for every program.
3. There is no clear categorization of exception is Hibernate.

Above problems are solved as follows:

1. Hibernate Template is provided which centralizes the Hibernate client code.
   Usage
   ```
   HibernateTemp.save (cust);
   ```
2. In spring Data Access, there is one root exception called Data Access Exception which unchecked or runtime. Because of unchecked exceptions, you no need to write try catch blocks for every program.
3. There is clear categorization of exceptions in spring Data Access.

Important methods of Hibernate Template:

1. Serializable save (Object)
2. Void update (object)
3. Void update (object,LockMode)
4. Void delete(object)
5. Void delete(object, LockMode)
6. Void deleteAll (collection)
7. Object load (class, Serializable)
8. Object load (class, Serializable, lockMode)

9. List loadAll(class)
10. List find(hpl)
11. List find(hql, object)
12. List find(hql, object [])
13. List findByCriteria (DetachedCriteria)
14. List findByCriteria (DC, int, int)

Steps in my Eclipse: Spring Data access with Hibernate

1. Create the Java project with the project name Jtc 48.
2. Add Hibernate capabilities as follows:
    - Select the Project and Right click
    - Select My Eclipse Add Hibernate Capabilities
    - Select Hibernate Specification as Hibernate 3.1
    - Click on Next.
    - Click on Next.
    - Uncheck the checkbox for specifying the database connection details
    - Click on Next.
    - Uncheck the checkbox for create Session Factory class click on Finish.
3. Add Spring capabilities as follows:
    - Select the project and Right click
    - Select the following `
        - Spring version spring 3.0
        - Spring 3.0 AOP libraries
        - Spring 3.0 core Libraries
        - Spring 3.0 persistence core Libraries
        - Spring 3.0 persistence JDBC Libraties
    - Click on Next
    - Provide spring configuration Document file name as jtcindia. Xml
    - Click on next button.
    - Click on Finish button.
4. Enable context namespace in the spring configuration Document.
5. Add mysql.Jar file the project build path.
6. Database steps:

```
CREATE DATABASE Jtcindiadb;
USE jtcindia;
CREATE TABLE customers {
cid int PRIMARY KEY, cname CHAR (10), email CHAR (20)
        phone Long, city char (20);
```

Jtc48: files required

Jtc48.java	customerDAO.java
HibernateCustomerDAO.java	Customer TO.java
Customer.java	Customer.hbm.xml
Jtcindia.xml	

Jtc 48.java

```
Package com.jtc india.spring;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXmlApplicationContext;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class Jtc48{
Public static void main (string[] args) {
ApplicationContext ctx=new classpathXmlApplicationContext("jtcindia.xml");
customerDAO cdao=(customerDAO) ctx.getBean("cdao");
//1. addCustomer
customerTO cto=new CustomerTO (401, som som1@jtc, 3333, Noida")
cdao.addCustomer(cto);
//2. UpdateCustomer
Customer to cto1=new CustomerTO(203, pra pra@jtc" 8888, Noida");
Cdao.updateCustomer(cto1);
//3. deleteCustomer
Cdao.deleteCustomer(206);
System.out.printl("Check your Database");
//4. getCustomersByCid
System.out.println("getCustomersByCid");
Cto=cdao.getCustomerByCid(301);
System.out.println(cto);
}
}
```

CustomerDAO.java

```
Package com.jtcindia.spring.hibernate;
Public interface customerDAO{
Public void addCustomer (Customer TO cto);
Public void updateCustomer (Customer TO cto);
Public void deleteCustomer(int cid);
Public customer TO getCustomerByCid(int cid);
}
```

HibernateCustomerDAO.java

```
Package com. Jtcindia.spring.Hibernate;
Import java.util.ArrayList;
Import java.util .list;
```

```
Public void updateCustomer (Custoemr TO cto) {
Customer c=(customer) hibernate Temp
Load(customer.class,cto.getCid());
c.setCid(cto.getCname());
csetCname(cto.getCname());
```

```
Import org.hibernate.Lock mode;
Import org.springframework.beans.factory.
annotation.Autowired;
Import org.springframework.orm.hibernate3.
HibernateTemplate;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class HibernateCustomerDAO implements
customerDAO{
@Autowired
Hibernate Template HibernateTemp;
Public void addCustomer(Customer TO cto){
Customer cust=new customer(cto.getCname(),
Cto.getEmali(),
Cto.getPhone()lcto.getCity());
hibernateTemp.save(cust);
}
Public void deleteCustomer(int cid) {
Customer c = (customer)_
Hibernate Temp.load(customer.class,cid);
Hibernate temp.delete(c, lockMode.None);
}
Public list<customerTO>getAllCustomers(){
List<customerTO> ctoList=new
ArrayList<customerTO>();
String hql="from customer c";
List<customer> list=hibernateTemp.find(hql);
For(customer c: list){
customerTO cto=new customerTO (c.getCid();
c.getCname()l
c.getEmail(),c.getPhone(),c.getCity()).
Ctolist.add(cto);
}
Return ctoList;
}
```

```
c.setEmail(cto.getEmail());
c.setPhone(cto.getPhone());
c.setcity(cto.getCity());
hibernateTemp.update(c, LockMode.NONE);
}
Public customerTO getCustomerByCid(int cid){
Customer c=(customer)
HibernateTemp.load (Customer.class, cid);
customerTO cto=new customerTO (c.getCid();
c.getCname(), c.getEmail(), c.getPhone(), c.getCity());
return cto;
}
}
```

Customer TO.java

```
Package com.jtcindia.spring.hibernate;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class customerTO {
Private int cid;          private string cname;
Private string email.     Private long phone;
Private string city;
Public customer TO () {}
Public customerTO (int cid, string cname, string email,
long phone, string city) {
This.cid=cid;             this.cname=cname;
This.email=email;         this.phone=phone
This.city=city;
}

// setters and Getters

}
```

Customer.java	Customer.hbm.xml
Package com.jtcindia.spring.hibernate;	<hibernate-mapping

Public class customer{
Private int cid; private string cname;
Private string email; private long phone;
Private string city;
Public customer() {}
Public customer (string cname, string email, long phone, string city) {
This.cname=cname; this.email=email;
This.phone=phone; this.city=city;
}
//setters and getters
}
	package=”com.jtcindia.spring.hibernate”>
	<class name=”customer” table=”customers” Lazy=”false”>
	<id name=”cid” column=”cid” type =int”>
	<generator class=increment”/>
	</id>
	<property name=”cname”/>
	<property name=”email/>
	<property name=”phone”type=”long”/>
	<property name=”city”/>
	</class>
	</hibernate-mapping>

Jtcindia.xml

```
<beans….>
<context:annotation-config/>
<bean id=”dataSource” class=”org.springframework.jdbc.datasource.DriverManagerDataSource”>
<property name=”driverClassName” value=”com.mysql.jdbc.Driver”/>
<property name=”url” value=”jdbc:mysql://locall.ost/jtcindiadb”/>
<property name=”username” value=”root”/>
<property name=”password” value=”somprakash”/>
</bean>
<bean id=”sessionFactory” class=”org.springframework.orm.hibernate3.LocalSessionFactoryBean”>
<property name=”mappingResources”>
<list>
<value>com/jtcindia/spring/hibernate/customer.hbm.xml</value>
</list></property>
<property name=”hibernateProperties”>
<map><entry key=”hibernate.dialect”value=”org.hibernate.dialect.MySQLDialect”/>
<entey key=”hibernate.show sql” value=”true”/>
<entry key=”hibernate.hbm2ddl.auto”value=”update”/>
</map></property>
</bean><bean id=”hibernateTemp” class=”org.springframework.orm.hibernate3.HibernateTemplate”
autowire=”constructor”/>
<bean id=”cdao” class=”com.jtcindia. spring.hibernate.HibernateCustomerDAO”/>
</beans>
```

Jtc50: files required

Jtc50.java	customerDAO.java
HibernateCustomerDAO.java	Customer TO.java
Customer.java	Customer.hbm.xml
Jtcindia.xml	

HibernateCustomerDAO.java

```
Package com.jtcindia.spring.hibernate;
Import java.util.*;
Import org.hibernate.criterion.*;
Import
org.springframework.beans.factory.annotation.
Autowired;
Import
org.springframework.orm.hibernate3.Hibernate
Template;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class HibernateCustomerDAO implements
customerDAO {
@Autowired
HibernateTemplate hibernateTemp;
Public string getCustomerCityByEmail(String
email){
DetachedCriteria dc=
DetachedCriteria.forClass(Customer.class);
dc.add(Expression.eq("email",email));
List<Customer> list=hibernate
Temp.findByCriteria(dc);
Customer c=list.get(0);
Return c.getCity();
```

```
Public List<customerTO> getAllcustomers(){
List<customerTO> ctoList=new
ArrayList<CustomerTO>();
DetachedCriteria dc=
DetachedCriteria.forClass(Customer.class);
List<customer> list=hibernateTemp.findBy
criteria(dc);
For (customer c: list) {
CustomerTO cto =new CustomerTO (c.getCid();
c.getEmail(),c.getPhone(), c.getCity());
ctoList.add(cto);
}
Return ctoList;
}

Public Long get Customer PhoneBy Email(String
email) {
DetachedCriteria dc=
DetachedCriteria.forClass(Customer.class);
dc.add(Expression.eq("email",email));
List<customer>
list=hibernateTemp.findByCriteria(dc);
Customer c=list.get(0);
Return c.getPhone();
}
```

```
Public int getCustomersCount() {
detachedCriteria dc=
DetachedCriteria.forClass(Customer.class);
List<customer>
list=hibernateTemp.findByCriteria(dc);
Return list.size();
}
Public customerTO getCustomerByEmail(String
email){
DetachedCriteria dc=
```

```
Public List<customerTO>
getCustomersByCity(String city) {
List<customerTO> ctoList=new
ArrayList< customerTO>();
DetachedCriteria dc=
DetachedCriteria.for Class(Customer.class);
dc.add(Expression.eq("city",city));
hibernateTemp.setCacheQueries(true);
hibernateTemp.setCacheQueries(true);
list<customer>list=hibernateTemp.findByCriteria(dc);
```

Detached Criteria.for class(customer.class);
dc.add(Expression.eq("email",email));
List<customer> list=hibernate
temp.findByCriteria(dc);
Customer c=list.get(0);
CustomerTO cto=new CustomerTO(c.getCid(),
c.getCname(), c.getEmail(), c.getPhone(),
c.getCity());
return cto;
} }
	for (customer c: list) {
	customerTO cto =new CustomerTO (c.getCid(),
	c.getCname(),
	c.getEmail(),c.getPhone(),c.getCity());
	ctoList.add(cto);
	}
	Return ctoList; }

Working with Hibernate DaoSupport:

Public class HibernateDaoSupport extends DaoSupport implements initializing Bean{
HibernateTemplate HibernateTemplate;

Public HibernateTemplate getHiberate Template(){
Return hibernateTemplate;
}
Public void setHibernate Template (Hibernate Template hibernate Template){
This.hibernate Template=hibernate Template;
}
…..
Public void after properties Set() {
If(hibernateTemplate==null){
Throw some Exception.
}}}
Public class Hibernate Customer DAO extends Hibernate DaoSupport implements customer DAO{
Public void add customer(Customer TO cto) {
Customer cust=new customer (cto.getCname (),cto.getEmail(). Cto.getPhone().cto.getCity();
getHibernateTemplate().save(cust);
}
}

In spring Configuration File:

<bean id="cdao" class="com.jtcindia.spring.HibernateCustomerDAO" autowire="byType"/>

Working with hibernate Callback

Hibernate Callback allows you to access the hibernate session object directly. Public class
hibernateCustomer DAO extends Hibernate DaoSupport implements customerDAO {
Public void addCustomer (customer TO cto) {
Final customer cust=new customer (cto.getCname(),cto.getEmail().cto.getPhone().cto.getCity());

HibernateCallback<session> hc=new HibernateCallback<session>(){
Public session dolnHibernate(session session)thows hibernateException, SQLException{
System.out.println(*doInHibernate()");

```
Session.save(cust);
}
};
getHibernateTemplate().execute(hc);
}}
```

In spring configuration file;

```
<bean id="cdao"class="com.jtcindia.spring.HibernateCustomerDAO"autowire="byType"/>
```

Jtc51: files required

Jtc51.java	customerDAO.java
HibernateCustomerDAO.java	Customer TO.java
Customer.java	Customer.hbm.xml
Jtcindia.xml	

Jtc51.java

```
Package com.jtcindia.spring.hibernate;
Import org.springframework.context.applicationContext;
Import org.springframework.context.support.classPathXmlApplicationColntext;
Public class Jtc51{
Public static void main(String[] args){
ApplicationContext ctx=new classPathXmlApplicationContext("jtcindia.xml");
customerDAO cdao=(customerDAO) ctx.getBean("cdao");
//1. addCustomer
customerTO cto=new Customer TO (119, "som","som@",3333,"Noida");
cdao.addCustomer(cto);
system.out.println("check your database");
} }
```

CustomerDAO.java

```
Package com.jtcindia.spring.hibernate;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public interface customerDAO{
Public void addCustomer(customerTO cto);
}
```

HibernateCustomer DAO.java

```
Package com.jtcindia.spring.hibernate;
Import java.sql.*;
Import org.hibernate.*;
Import org.springframework.orm.hibernate3.
HibernatecCallback;
Import
org.springframework.orm.hibernat3.support.
Hibernate Daosupport;
Public class HibernatCustomerDAO extends
HibernateDaoSupport implements CustomerDAO {
```

Jtcindia.xml	Public void addCustomer(CustomerTO cto) {

<beans…..>
//same as Jtc50
<bean id="cdao"
Class="com.jtcindia.spring.hibernate.Hibernate
CustomerDAO"
Autowire="byType"/>
</beans>
	Final customer cust=new
	Customer (cto.getCname(),cto.getEmail(),cto.get
	phone(),
	Cto.getCity());
	HibernateCallback<session> hc=new
	HibernateCallback<session>(){
	Public session dolnHibernate(Session session)
	Throws hibernateException, SQLException{
	Session.save(cust);
	Return session;
	} }
	getHibernateTemplate().execute(hc);
	} }

Steps in to develop First jpa Jtc with My Eclipse

1. Create the java project with the name : JPAJtc
2. Add the jpa capabilities as follows:
   - Driver template : MySQL connection/J
   - Driver name : jtcmysqldriver
   - URL : jdbc : mysql://localhost/jtcindiadb
   - Username : root
   - Password : somprakash
   - Add the mysql.jar
   - Check save password option and connect to MyEclipse at startup option and click Test Driver button.
   - Click Finish button
3. Open persistence.xml under src/META-INF.
4. Change <persistence-unit> name to JTCINDIA PU
5. Add the mysql.jar file to project build path.
6. Create the table called customers under jtcindiadb.
   - CREATE TABLE customers(
   - cid int primary key auto increment,
   - cname VARCHAR(15),
   - email VARCHAR(15),
   - phone VARCHAR(15),
   - city VARCHAR(15),
7. write the JPA Entity class Customer.java under package called com.jtcindia.jpa
8. write the following JPA Client code under package com.jtcindia.jpa
   A. JPAJtc1.java
   B. JPAJtc2.java

JPAJtc: Files required

JPAJtc1.java	JPAJtc2.java	Customer.java

Persistence.xml		

JPAJtc1.java

```
Package com.jtcindia.jpa;
Import javax.persistence.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class JPAJtc1 {
Public static void main(string[] args) {
Entity Transaction tx=null;
Try{
EntityManagerFactory
factory=Persistence.create
EntityManagerFactory("jtcindia-pu);
EntityManager
Manager=factory.createEntity manager();
Tx=manager.getTransaction();
Tx.begin();
Customer cust=new
Customer("som","som@jtc","1111","Noida");
Manager.persist(cust);
Tx.commit();
System.out.println("check Database");
} catch (Exception e){
e.printStackTrace();
tx.rollback();
} } }
```

JPAJtc2.java

```
Package com.jtcindia.jpa;
Import javax.persistence.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class JPAJtc2 {
Public static void main(string[] args) {
Entity Transaction tx=null;
Try{
EntityManagerFactory factory=Persistence.create
EntityManagerFactory("jtcindia-pu);
EntityManager
Manager=factory.createEntity manager();
Tx=manager.getTransaction();
Tx.begin();
Customer cust=(customer)
manager.getReference(Customer.class,1);
Systim.out.println(cust.getCid()+"\t"+cust.getCname()
+"\t"+cust.getEmail()+"\t"+cust.getPhone()+"\t"+cust.
Get city());
Tx.commit();
}catch(Exception e){
e.printStackTrace();
tx.rollbcak();
} } }
```

Customer.java

```
Package com.jtcindia.jpa;
Import javax.persistence.*;
@Entity
@Table(name="customers")
Public class Customer{

@id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name="cid")
Private int cid;

@Column(name="cname")
@Column(name="email")
Private string email;
@Column(name="phone")
Private string phone;
@Column(name="city")
Private string city;
Public customer (){}
Public customer (String cname, string email,
string phone, string city) {
This.cname=cname;
This.email=email;
This.phone=phone;
This.city=city;
```

Private string cname;	}
	//setters and getters
	}

Persistence.xml

```
<persistence…>
<persistence unit name=”JTCINDIA-PU” transaction-type=”RESOURCE LOCAL”>
<peovider>org.hibernate.ejb.HibernatePersistence</provider>
<class>com.jtcindia.jpa.customer</class>
<properties>
<property name=”hibernate.connection.driver, class” value=”com.mysql.jdbc.Driver”/>
<property name=”hibernate.connection.driver, url” value=”jdbc:mysql://localhost:3306/jtcindiadb”/>
<property name=”hibernate.connection.driver, username” value=”root”/>
<property name=”hibernate.connection.driver, password” value=”somprakash”/>
</properties>
</persistence-unit>
</persistence>
```

Spring Data access with JPA:

When you want to pearform any persistent operation with jpa then you need to write the jpa code with the following steps:

Write the Entity class with Annotations
Client code
Cty{

```
//Take the Entity ManagerFactory //2
//Take Entity Manager           //3
//Take Entity Transaction       //4
//Begin Transaction             //5
//Perform persistent operation  //6
//End transaction               //7
//Close Entity Manager          //8
} catch (Exception e) { }
```

You can Visit the following problems with above code:

1. All the above statements other then 6 are common for all the persistent operations. Writing same client code multiple times repeatedly gives you code duplication problem.
2. All the methods in JPA API are throwing one common exceptions related to persistence provider (Ex with Hibernate, you will get org.hibernate.HibernateException which is checked exception). Because of checked exception, you need to write try/ catch blocks for every peogram
3. There is no clear categorixation of exceptions.

Above problems are solved as follows:

1. jpaTemplate is provided which centralizes the JPA client code.

Usage

jpaTemp persist (cust);

2. In spring Data Access, there is one root exception called Data Access Exception which unchecked or runtime. Because of unchecked exceptions, you no need to write try catch blocks for every program.
3. There is clear categorization of exceptions.

Important methods of jpa Template:

1. Void persist(object)
2. Void merge(object)
3. Void remove(object)
4. Object getReference(Class, object)
5. List find(jpaql)
6. List find(jpaql,object…args)
7. Object execute(jpaCallback jc)
8. List findByNamedQuery(jpaql)
9. List findByNamedQuery(jpaql,object….args)

Steps in my Eclipse: Spring Data access with JPA:

1. Create the Java project with the name Jtc52.
2. Add the JPA capabilities as follows:

- Right click the project and then select My Eclipse > Add JPA capabilities
- Select the persistence provider: Hibernate 3.X.
- Hibernate version 3.2 and click on Next button
- Use the PlatForm: Generic 1.0
- Select connection
- Jtcmysqldriver from the list
- Click on finish button

3. Open persistence.xml under src/META-INF.
4. Change<persistence unit> name to jtcindia pu
5. Add spring capabilities as follows:

- Select the project and Right click
- Select my Eclipse > Add spring capabilities
- Select the following
  - Spring version spring 3.0
  - Spring 3.0 AOP libraries
  - Spring 3.0 core Libraries
  - Spring 3.0 persistence core Libraries
  - Spring 3.0 persistence JDBC Libraties
- Click on Next

- Provide spring configuration Document file name as jtcindia. Xml
- Click on next button.
- Click on Finish button.

6. Enable context namespace in the spring configuration Document.
7. Add mysql.Jar file the project build path.
8. Database steps:

```
CREATE DATABASE Jtcindiadb;
USE jtcindia;
CREATE TABLE customers {
        cid int PRIMARY KEY, cname CHAR (10), email CHAR (20)
        phone Long, city char (20);
```

9. Configure LocalEntityManagerFactoryBean With persistenceUnitName.

```
<bean id=”entity ManagerFactory”
        Class=”org.springframework.orm.jpa.LocalEntityManagerFactoryBean>
<property name=”persistenceUnitName” value=”JTCINDIA-pu”/>
</bean>
```

10. Configure Jpa TransactionManager by injectiong LocalEntity managerFactory Bean.

```
<bean id=”transactionManager”
        Class=”org.springframework.orm.jpa.jpaTransactionManager”>
        <property name=”entityManagerFactory” ref=”entityManagerFactory”/>
</bean>
```

Jtc52: files required

Jtc52.java	customerDAO.java
HibernateCustomerDAO.java	Customer TO.java
Customer.java	persistence.xml
Jtcindia.xml	

Jtc52.java

```
Package com.jtcindia.spring.jpa;
Import org.springframework.context.applicationContext;
Import org.springframework.context.support.classPathXmlApplicationColntext;
Public class Jtc52{
Public static void main(String[] args){
ApplicationContext ctx=new classPathXmlApplicationContext(“jtcindia.xml”);
customerDAO cdao=(customerDAO) ctx.getBean(“cdao”);
//1. addCustomer
customerTO cto=new Customer TO (202, “som”,”som@”,3333,”Noida”);
cdao.addCustomer(cto);
```

```
//2. updateCustomer
Customer TO cto1=new customerTO(2,"ds","ds@jtc", 8888, "Noida");
Cdao.updateCustomer(cto1);
//3. deleteCustomer
Cdao.deleteCustomer(106);
//4. getCustomersByCid
System.out.println("getCustomersByCid");
Cto=cdao.getCustomerByCid(2);
System.out.println(cto);
}
}
```

customerDAO.java

```
package com.jtcindia.spring.jpa;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public interface customerDAO {
Public void addCustomer (customer TO cto);
Public void updateCustomer (customerTO cto);
Public void delect Customer (int cid);
Public customer To getCustomerByCid(int cid);
}
```

jpaCustomerDAO.java

```
package com.jtcindia.spring.jpa;
import org.springframework.beans.factory.
annotation.autowired;
import org.springframework.orm.jpa.*;
import org.springframework.transaction.*;
import org.springframework.transaction.support.
DefaultTransa ctionDefinition;
public class jpaCustomerDAO implements
CustomerDAO{
@Autowired
Jpa Template jpaTemp=null;
@Autowired
jpaTransactionnManager txmanager=null;
public void addcustomer(customerTO cto){
customer cust=new customer(cto.getCid();
cto.getCname(),
cto.getEmail(),cto.getPhone(), cto.getCity());
Transaction Definition txDef=new
Public void update Customer (customer TO cto) {
TransactionDefinition TxDef=new
defaultTransactionDefinition();
transactionstatus ts=txmanager.getTransaction(txDef);
customer c=(customer) jpaTemp.find(Customer.class,
cto.getCid());
c.setEmail(cto.getEmail());
c.setphone(cto.getPhone());
c.setCity(cto.getCity());
jpaTemp.merge©;
txmanager.commit(ts);
}

Public void delete Customer(int cid) {
TransactionDefinition txDef=new
DefaultTransactionDefinition();
TransactionStatus ts=txmanager.getTransaction(txDef);
Customer c=(customer) jpaTexp,find(customer.class,
cid);
jpaTemp.remove©;
txmanager.commit(ts);
}

Public customerTO getCustomerByCid(int cid){
Customer c=(customer)jpaTemp.find(customer.class, cid);
customerTO cto=new customerTO (c.getCity());
return cto;
}}
```

CustomerTO.java

```
Package com.jtcindia.spring.hibernate;
Public class customer TO{
Private int cid;          private string cname;
Private string email;       private long phone;
```

```
DefaultTransactionDefinition();p
TransactionStatus ts=
Txmanager.getTransaction(txDef);
jpaTemp.persist(cust);
txmanager.commit(ts);
}
```

```
Private string city;
Public customerTO() {}
Public customerTO (string cname, string email, long
phone, string city) {
This.cname=cname;          this.email=email;
This.phone=phone;          this.city=city;
}
//setters and getters
}
```

Customer.java

```
Package com.jtcindia.jpa;
Import javax.persistence.*;
@Entity
@Table(name="customers")
Public class Customer{

@Entity
@Table(name="customers")
Public class customer{
@id
@ @Column(name="cid")
Private int cid;
@Column(name="cname")
Private string cname;

@Column(name="email")
Private string email;
@Column(name="phone")
Private string phone;
@Column(name="city")
Private string city;
Public customer (){}
Public customer (String cname, string email, string
phone, string city) {
This.cname=cname;
This.email=email;
This.phone=phone;
This.city=city;
}
//setters and getters
}
```

Persistence.xml

```
<persistence….>
<persistence unit name="JTCINDIA PU"
transaction-Type="RESOURCE-LOCAL">
<provider>org.hibernat.ejb.hibernatePersistence</
provider>
<class.com.jtcindia.spring.jpa.customer</class>
<properties>
<peoperties>
<property name="hibernate.connection.driver
class" value=com.mysql.jdbc.driver"/>
<property name="hibernat.connection.url"value=
jdbc:mysql:// localhost:3306/jtcindiadb"/>
<property name="hibernate.connection.password"
value="somprakash"/>
</properties>
</persistence-unit>
</persistence>
```

Jtcindia.xml

```
<beans
<context:annotation-config/>
<bean id=”entity ManagerFactory” class=”org.springframework.orm.jpa.LocalEntityManagerFactoryBean”>
<property name=”persistenceUnitName” value=”JTCINDIA-PU”/>
</bean>
<bean id=”transaction Manager” class=”org.springframework.orm.jpa.jpaTransactionManager”>
<property name=”entityManagerFactory” ref=”entityManagerFactory”/>
</bean>
<bean id=”jpaTemp” class=”org.springframework.orm.jpa.jpaTemplate” autowire=”constructor”/>
<bean id=”cdao” class=”com.jtcindia.spring.jpa.jpaCustomerDAO”/>
</beans>
```

Jtc53: files required:

Jtc53.java	customerDAO.java
HibernateCustomerDAO.java	Customer TO.java
Customer.java	persistence.xml
Jtcindia.xml	

Jtc53.java

```
Package com.jtcindia.spring.jpa;
Import java.util.list;
Import org.springframework.context.applicationContext;
Import org.springframework.context.support.classPathXmlApplicationColntext;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class Jtc53{
Public static void main(String[] args){
ApplicationContext ctx=new classPathXmlApplicationContext(“jtcindia.xml”);
customerDAO cdao=(customerDAO) ctx.getBean(“cdao”);

//1. getAllCustomers
System.out.println(“getAllCustomers”);
List<CustomerTO> list=cdao.getAllCustomers();
For (CustomerTO ct: list) {
System.out.println(ct);
}

//2. getCustomerByEmail
System.out.prinntln(“getCustomerByEmail”);
customerTO cto=cdao.getCustomerByEmail(“som@jtc”);
system.out.println(cto);
```

```
//3. getCustomersByCity
System.out.println("getCustomersByCity");
List=cdao.getCustomerByCity("Noida");
For (CustomerTO ct:list) {
System.out.println(ct);
}
//4. getCustomerCount
System.out.println ("getCustomerCount");
Int cuunt=cdao.getCustomersCount();
System.out.println("no of cust:"+count);
//5. getCustomerCityByEmail
System.out.println("getCustomerCityByEmail");
String ci=cdao.getCustomerCityByEmail("som@jtc");
System.out.println(ci);
//6. getCustomerPhoneByEmail
System.out.println("getCustomerPhoneByEmail");
Long ph=cdao.getCustomerPhoneByEmail("som@jtc");
System.out.println(ph);
}
}
```

CustomersDAO.java

```
Package com.jtcindia.spring.jpa.
Import java.util.List;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public interface customerDAO {
Public List<customerTO> getAllCustomers();
Public customerTO getCustomerByEmail(String email);
Public List<customerTO> getCustomersByCity(String city);
Public int getCustomerCount();
Public string getCustomerCityByEmail(String email);
Public Long getCustomerPhoneByEmail(String email);
}
```

JPA CustomerDAO.java

```
Package com.jtcidia.spring.jpa;
Import java.util.*;
Import org.springframework.beans.factory.
annotation.Autowired;
Import org.spring framework.orm.jpa.jpaTemplate;
Import org.springframework.orm.jpa.jpaTransaction
```

```
Public list<customerTO> getCustomersByCity(String
city) {
List<customerTO> ctoList=new
ArrayList<customerTO>();
String jpaql="from customer c where c. city=?";
List<customer>list=jpaTemp.lfind(jpaql, city);
For(customer c:list){
```

```
Manager;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class jpaCustomerDAO implements
CustomerDAO {
@Autowired
jpaTemplate jpaTemp = null;
```

```
customerTO cto=new customerTO(c.getCid();
c.getCname();
c.getEmail(), c.getPhone(), c.getCity());
ctoList. Add(cto);
}
Return ctoList;
}
```

```
Public List<customerTO> getAllCustomers() {
List<customerTO> ctoList=new
ArrayList<customerTO> ();
String jpaql=from customer c";
List<customer> list=jpaTemp.find(jpaql);
For (Customer c: list) {
customerTO cto=new customerTO(c.getCid(),
c.getCname(), c.getEmail(), c.getPhone(),
c.getCity());
ctoList.add(cto);
}
Return ctoList;
}
Public string getCustomerCityByEmail(string
email) {
String jpaql=from customer c where c.email=?";
List<customer> list=jpaTemp.find(jpaql, email);
Customer c=(customer) list.get(0);
Return c.getCity();
}
Public int getCustomersCount() {
String jpaql="from customer c";
List<customer> list=jpaTemp.find(jpaql);
Return list.size();
}
```

```
Public Long getCustomerPhoneByEmail (string
email) {
String jpaql="from customer c where c.email=?";
List<customer> list = jpaTemp.find(jpaql, email);
Customer c=(customer)list.get(0);
Return c.getPhone();
}

Public CustomerTO getCustomerByEmail(String
email){
String jpaql="from customer c where c.email=?";
List<customer> list=jpaTemp.find(jpawl,email);
Customer c=(customer) list.get(0);
customerTO cto=new customerTO(c.getCid();
c.getCname(), c.getEmail();
c.getphone(), c.getCity());
return cto;
}
}
```

Working with JpaDaoSupport

```
Public class JpaDaoSupport extends DaoSupport implements initializingBean{
Jpa Template jpa Template;
Public jpaTemplate getjpa Template(){
Return jpaTemplate;
}
```

```
Public void setjpaTemplate (jpaTemplate jpaTemplate){
This.jpaTemplate=jpaTemplate;
}
…..
Public void afterPropertiesSet(){
If(jpaTemplate==null){
Throw some Exception.
}
}
}
Public class jpaCustomerDAO extends jpaDaoSupport implements CustomerDAO {
Public void add Customer(CustomerTO cto){
Customer Cust=new
Customer (cto.getCid(),cto.getCname(),cto.getEmail(),cto.getPhone(),cto.getCity());
Getjpa Template().persiste(cust);
}
```

In spring Configuration File:

```
<bean id="cdao" class="com.jtcindia.spring.jpa.jpaCustomerDAO" autowire="byType"/>
```

Working with jpaCallback

- Jpa callback allows you to access the entity manager object directly.

```
Public class jpa CustomerDAO extends jpaDaoSupport implements customerDAO{
Public void addCustomer(CustomerTO cto){
Final customer cust=new
Customer(cto.getCid(),cto.getCname(),cto.getEmail(),cto.getPhone(),cto.getCity());
jpaCallback<Entitymanager> jc=newjpaCallback<EntityManager>() {
public EntityManagerDoInJpa()"0;
manager.persist(cust);
return manager;
}
}
getjpaTemplate().execute(jc);
}
```

In spring Configuration file:

```
<bean id="cdao" class="com.jtcindia.spring.jpa.jpaCustomerDAO" autowire="byType"/>
```

Jtc54: files required

Jtc54.java	customerDAO.java
HibernateCustomerDAO.java	Customer TO.java
Customer.java	persistence.xml
Jtcindia.xml	

Jtc54.java
Package com.jtcindia.spring.jpa;

```
Import org.springframework.context.applicationContext;
Import org.springframework.context.support.classPathXmlApplicationColntext;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class Jtc54{
Public static void main(String[] args){
ApplicationContext ctx=new classPathXmlApplicationContext("jtcindia.xml");
customerDAO cdao=(customerDAO) ctx.getBean("cdao");

//1. addCustomer
CustomerTlO cto=new CustomerTO(302,"som","som@jtc",3333,"Noida");
Cdao.addCustomer(cto);
System.out.println("check Database");
}  }
```

CustomerDAO.java

```
Package com.jtcindia.spring.jpa;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public interface customerDAO{
Public void addCustomer(customerTO cto);
}
```

Jtcindia.xml

```
<beans….>
<context:annotation-config/>
<bean id="entityManagerFactory"
Class="org.springframework.orm.jpa.LocalEntity
manager factory Bean">
<property name="persistenceUnitName" value ="
JTCINDIA-PA"/>
</bean>
<bean id="transactionManager" class=k"org.
springframework.org.jpa.jpaTransactionManager">
<property name="entityManagerFactory" ref="
entityManagerFactory"/>
</bean>
<bean id="jpaTemp" class="org.springframework.
orm.jpa.jpaTemplate" autowire=constructor"/>
<bean id="cdao" class="com.jtcindia.
```

JpaCustomerDAO.java

```
Package com.jtcindia.spring.jpa;
Import javax.persistence.*;
Import org.springframework.beans.factory.annotation.
Autowired;
Import org.springframework.orm.jpa.*;
Import
org.springframework.orm.jpa.support.jpaDaoSupport;
Import org.springframework.transaction.*;
Import org.springframework.transaction.support. default
Transa ction Definition;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class jpaCustomerDAO extends jpaDaoSupport
implements CustomerDAO{
@Autowired
jpaTransactionManagerTxm anager=null;
public void addCustomer(CustomerTO cto){
final customer cust=new customer (cto.getCid();
cto.getCname(), cto.getEmail(), cto.getPhone(),
cto.getCity());
jpaCallback<EntityManager> jc=new
jpaCallback<EntityManager>() {
public EntityManager doinjpa(EntityManager Manager)
Throws persistenceException{
```

spring.jpa.jpaCustomerDAO”autowire=”byType”/>
</beans>
	Manager.persist(cust);
	Return manager;
	}
	};
	TransactionDefinition txDef =new
	DefaultTransactionDefinition();
	transactionStatus ts=txmanager.getTransaction(txDef);
	getJpa Template().execute(jc);
	txmanager.commit(ts);
	} }

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.
- Platform transaction manager is the root for all the Transaction managers in spring.
- Following are the methods provided in platform transaction manager:

TransactionStatus	getTransaction(TransactionDefinition definition)
Void	Commit(TransactionStatus status)
Void	Rollback(TransactionStatus status)

- Following are various Transaction managers provided which are sub classis of platform transaction Manager.
    1. DataSourceTransactionManager
    2. HibernateTransactionManager
    3. JpaTransactionManager
- Spring supports the following ways to manage the Transactions:
    1. Programmatic Transactions
    2. Declarative Transactions with AOP using Annotation support
    3. Declarative Transactions with AOP using schema support
    4. Declarative Transactions with AOP using TransactionProxyFactoryBean

Programmatic Transaction:

Package org.springframework.transaction
Interface transactionDefinition

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.orm.hibernate3
Class Hibernate Transaction Manager

Hibernate Transaction manager()
Hibernate TransactionManager(sessionFactory sessionFactory)

dataSource	getDataSource()
sessionFactory	getSessionFactory()
Void	setDataSource(DataSource dataSource)
Void	setSession factory(sessionFactory sessionFactory)
TransactionStatus	getTransaction(TransactionDefinition definition)
Void	Commit(TransactionStatus status)
Void	Rollback(transactionStatus status)

Package org.spring framework.orm.hibernate3
Class jpaTransactionmanager

jpaTransactionManager()
jpaTransactionManager(EntityManagerFactory emFactory)

dataSource	getDataSource()
EntityManagerFactory	getEntityManagerFactory()
Void	setDataSource(DataSource dataSource)
Void	setEntityManagerFactory (EntityManagerFactory emFactory)

TransactionStatus	getTransaction(TransactionDefinition definition)
Void	Commit(TransactionStatus status)
Void	Rollback(transactionStatus status)

Programation Transaction with JDBC

1. Configure the Data Source

```
<bean id="dataSource"
        Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        …..
</bean>
```

2. Configure the Jdbc Template by injection DataSource.

```
<bean id="jdbcTemp"
        Class="org.springframework.jdbc.core.jdbcTemplate"
        Autowire="construstoer"/>
```

3. Configure the DataSource TransactionManager by injecting DataSource

```
<bean id="txManager"
        Class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
        Autowire="constructor"/>
```

4. Configure the JdbcCustomerDAO.

```
<bean id="cdao" class="com.jtcindia.spring.jdbc.jdbcCustomerDAO'/>
```

5. Inject jdbc Template and DataSourceTransactionManager into jdbc Customer DAO.
6. Write the Transactional begin and end inside the methods of jdbc customerDAO

```
Class jdbc Customer DAO imp CustomerDAO {
@autowired
jdbcTemplate jdbcTemp=null;
@Autowired
DataSourceTransactionManagerTxmanager=null;
Public void addCustomer(CustomerTO cto){
TransactionStaturs ts=null;
        Try{
        TransactionDefinition txDef=new DefaultTransactionDefinition();
        Ts=rxmanager.getTransaction(txDef);
        String sql="insert into customers value(?????);
        jdbcTemp.update(sql,cto.getCid(),cto.getCname(),cto.getEmail(),cto.getPhone());
        txmanager.commit(ts);
        }catch(Exception e){
        Txmanager.rollback(ts);
        }
}
```

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');
```

Jtc 55: Example using spring programmatic Transactions with JDBC.
Jtc 55: Files required

Jtc55.java	AccountDAO.java
jdbcAccountDAO.java	inSufficientFundsException.jave
Jtcindia.xml	

Jtc55.java

```
Package com.jtcindia.spring.jdbc;
Import org.springframework.context.applicationContext;
Import org.springframework.context.support.classPathXmlApplicationColntext;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class Jtc55{
Public static void main(String[] args){
ApplicationContext ctx=new classPathXmlApplicationContext("jtcindia.xml");
customerDAO adao=(customerDAO) ctx.getBean("adao");
System.out.println(adao.getBalance(101));
System.out.println(adao.getBalance(102));

//1. Deposit()
Adao.deposit(101, 2000.0);
//2.withdraw()
Adao.withdraw(102, 5000.0);
//3.getBalance()
System.out.println(adao.getBalance(101));
System.out.println(adao.getBalance(102));
//3. fundsTransfer()
System.out.println(adao.getbalance(103));
System.out.println(adao.getbalance(101));
Adao.fundsTransfer(103, 101, 30000.0);
System.out.println(adao.getbalance(103));
System.out.println(adao.getbalance(101));
} }
```

AccountDAO.java

```
Package com.jtcindia.spring.jdbc;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public interface AccountDAO {
```

inSufficientFundsException.java

```
package com.jtcindia.spring.jdbc;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class insufficientFunds Exception extends
```

Public void deposit(int accno, double amt)l Public void withdraw (int accno,double amt); Public void funds Transfer(int saccno, int daccno, double amt); Public double getBalance(int accno); }	RuntimeException{}

jdbcAccountDAO.java

```
package com.jtcindia.spring.jdbc;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.jdbcTemplate;
import org.springframework.jdbc.datasource.DataSource
Trans action Manager;
import org.springframework.transaction.*;
import
org.springframework.transaction.support.DefaultTransa
ctionDefinition;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class jdbcAccountDAO implements AccountDAO
{
@Autowired
jdbcTemplate jdbc Temp=null;
@Autowired
DataSource TransactionManager TxManager=null;
Public void deposit(int accno, double amt) {
TransactionDefinition txDef=new
DefaultTransactionDefinition();
TransactionStatus ts=rxManager.getTransaction(txDef);
String sql=”select bal from accounts where acne=?”;
Int x=jdbcTemp.queryForlnt(sql,accno);
Double cbal=new lnteger(X). doubleValue();
Double nbal=cbal+amt;
String sql1=”update accounts set bal=? where accno=?”;
jdbcTemp.update(sql1, nbal,accno);
txManager.commit(ts);
}
@Override
Public double getBalance (int accno) {
String sql=”select bal from accounts where accno=?;
Int x=jdbcTemp.queryForlnt(sql, accno);
Double cbal=new lnteger(x).doubleValue();
Public void fundsTransfer(int saccno, int daccno, double
amt) {
TransactionDefinition txDef=new
defaultTransactionDefinition();
transactionStatus ts=txManager.getTransaction(txDef);
try{
string sql1=”select bal from accounts where accno=?;
string sql2=”update accounts set bal=? where accno=?
Int y=jdbcTemp.queryForlnt(sql1, daccno);
Double dcbal=new Integer(y). doubleValue();
System.out.println(“Before deposit”+dcbal);
Double dnbal=dcbal+amt;
jdbcTemp.update(sql2,dnbal,daccno);
y=jdbcTemp.queryForlnt(sql1,daccno);
dcbal=new lnteger(y). doubleValue();
system.out.println(“After deposit” + dcbal);
int x=jdbcTemp.queryForlnt(sql1, saccno);
double scbal=new lnteger(x).doubleValue();
if(scbal>=5000+amt){
double snbal=scbal-amt;
jdbcTemp.update(sql2,snbal,saccno);
}else{
Throw new insufficientFundsException();
}
txManager.commit9ts);
}catch(Exception e){
txManager. Rollback(ts);
e.printStackTrace();
}}
Public void withdraw(int accno, double amt){
TransactionDefinitionDef=new
defaultTransactionDefinition();
TransactionStatus ts=txManager.getTransaction(txDef);
String sql=”select bal from accounts where accno=?;
Int x=jdbcTemp.queryForlnt(sql,accno);
Double cbal=new lnteger(X).doubleValue();
If(cbal>=50000+amt){
Double nbal=cbal amt;
String sql1=”update accounts set bal=? where accno=?;
```

Return cbal;
}
	jdbcTemp.update(sql1,nbal,accno);
	}else{
	Throw new lnSufficientFundsException();
	}
	txManager.Commit(ts);
	} }

Jtcindia.xml

```
<beans
<context:annotation-config/>
<bean id=”dataSource” class=”org.springframework.jdbc.datasource.driverManagerDataSource”>
<property name=”driverClassName”value=com.mysql.jdbc.Driver”/>
<property name=”url”value”jdbc:mysql://localhost/jtcindiadb”/>
<property name=”username”value=”root”/>
<property name=”password” value=somprakash”/>
</bean>
<bean id=”jdbcTemp”class=”org.springframework.jdbc.core.jdbcTemplate” autowire=”constructor”/>
<bean id=”txManager”
Class=”org.springframework.jdbc.datasource.DataSource TransactionManager”
Autowire=”constructor”/>
<bean id=”adao” class=”com.jtcindia.spring.jdbc.jdbcAccountDAO”/>
</beans>
```

Programmatic Transactionsf with Hibernate

1. . Configure the Data Source

```
<bean id=”dataSource”
        Class=”org.springframework.jdbc.datasource.DriverManagerDataSource”>
        …..
</bean>
```

2. Configure the SessionFactory by injectiong DataSource, Hibernate Properties, mapping Resources.

```
<bean id=”sessionFactory”
        Class=”org.springframework.orm.hibernate3.LocalSessionFactoryBean”>
        …..
</bean>
```

3. Configure the Hibernate Template by injecting sessingFactory

```
<bean id=”hibernateTemp”
        Class=”org.springframework.orm.hibernate3.HibernateTemplate”
        Autowire=”constructor”/>
```

4. Configure the Hibernate TransactionManager by injecting DataSource

```
<bean id=”txManager”
        Class=”org.springframework.orm.hibernate3.HibernateTransactionManager”
        Autowire=”constructor”/>
```

5. Configure the JdbcCustomerDAO.

```
<bean id=”cdao” class=”com.jtcindia.spring.hibernate.HibernateCustomerDAO’/>
```

6. Inject HibernateTemplate and Hibernate TransactionManager into HibernateCustomerDAO.

7. Write the Transactional begin and end inside the methods of Hibernate customerDAO

```
Class Hibernate Customer DAO imp CustomerDAO {
@autowired
Hibernate Template Hibernate Temp=null;
@Autowired
DataSourceTransactionManagerTxmanager=null;
Public void addCustomer(CustomerTO cto){
TransactionStaturs ts=null;
        Try{
        TransactionDefinition txDef=new DefaultTransactionDefinition();
        Ts=rxmanager.getTransaction(txDef);
        hibernateTemp.save(cust);
        txmanager.commit(ts);
        }catch(Exception e){
        Txmanager.rollback(ts);
        }
}
}
}
```

Jtc 56: Example using spring programmatic Transactions with JDBC.

Jtc 56: Files required

Jtc56.java	AccountDAO.java
HibernateAccountDAO.java	Account.java
Account.hbm.xml	inSufficientFundsException.jave
Jtcindia.xml	

Account.java	Account.hbm.xml

Package com.jtcindia.spring.hibernate;
Public class Account {
Private int accno;
Private string atype;
Private double bal;
Public Account()[]
Public Account(String atype, double bal) {
This.atype=atype;
This.bal=bal;
}
//setters and getters
}
	<hibernate-mapping package=”com.jtcindia.
	spring. Hibernate”>
	<class name=Account”table=”accounts”>
	<id name=”accno” column=”accno” type=”int”>
	<generator class=” increment”/>
	</id>
	<property name=”atype”/>
	<property name=”bal”type=”double”/>
	</class>
	</hibernate-mapping>

HibernateAccountDAO.java

```
package com.jtcindia.spring.jdbc;
import org.hibernate.LockMode;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate3.hibernate
Template;
import org.springframework.
orm.hibernate3.HibernateTransac tionManager;
import org.springframework.transaction.*;
import
org.springframework.transaction.support.DefaultTransa
ctionDefinition;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class HibernateAccountDAO implements
AccountDAO(
@Autowired
HibernateTemplate hibernate Temp=null;
@Autowired
Hibernate TransactionManager txManager=null;
@override
Public void deposit(int accno, double amt){
TransactionDefinition txDef=new
DefaultTransactionDefinition();
TransactionStatus ts=
txManager.getTransaction(txDef);
Account acc=hibernateTemp.load(Account.class,
accno,LockMode.Read);
Acc.setBal9acc.getBal()+amt);
hibernateTemp.update(acc);
txManager.commit(ts);
}

@override
Public double getBalance(int accno) {
System.out.println(hibernateTemp);
Account acc=hibernate Temp.load(Account.Class,
accno,LockMode.Read);
Double cbal=acc.getBal();
Return cbal;
}
@override
Public void funds Transfer(int saccno, int daccno,
double amt){
TransactionDefinition txDef=new
DefaultTransactionDefinition();
Transaction Status
ts=txManager.getTransaction(txDef);
Try{
Account
acc1=hibernateTemp.load(Account.class,
daccno,LockMode.Read);
Acc1.setbal(acc1.getbal()+amt);
Hiber nate Temp.update(acc1);
Account
acc2=hibernateTemp.load(Account.class, saccno,
LockMode.Read);
Double scbal=acc2.getBal();
If(scbal>=5000+ amt) {
Acc2. Setbal(scbal amt);
}else{
Throw new inSufficientFundsException();
}
txManager.commit(ts);
}catch(Exception e) {
txManager.rollback(ts);
}
}

@override
Public void withdraw9int accno, double amt) {
TransactionDefinition txDef=new
DefaultTransactionDefinition();
TransactionStatus
ts=txManager.getTransaction(txDef);
Account acc=hibernateTemp.load(Account.class,
accno,
LockMode.Read);
Double cbal=acc.getbal();
If(cbal>=5000+ amt){
Acc.setBal(cbal-amt);
hibernateTemp.update(acc);
}else{
Throw new inSufficientFundsException();
}
txManager.commit(ts);
```

	} }

Programmatic Transactions with JPA

1. Configure the EntityManagerFactory

```
<bean id="entityManagerFactory"
        Class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
<property name="persistence unitName" value="JTCINDIA PU"/>
</bean>
```

2. Configure the jpaTemplate by injecting EntityManagerFactory

```
<bean id="jpaTemp"
        Class="org.springframework.orm.jpa.jpaTemplate"
        Autowire="constructor"/>
```

3. Configure the JpaTransactionManager by injecting EntityManagerFactory

```
<bean id="txManager"
        Class="org.springframework.orm.jpa.jpaTransactionManager"
        Autowire="by Type"/>
```

4. configure the jpaCustomerDAO.

```
<bean id="cdao" class="com.jtcindia.spring.jpa.jpaCustomerDAO"/>
```

5. Inject jpaTemplate and jpa TransactionManager into jpaCustomerDAO.

6. Write the Transactional begin and end inside the methods of japCustomerDAO.

```
Class jpaCustomerDAO imp customerDAO{
@Autowired
jpaTemplate jpaTemp=null;
@Autowired
jpaTransactionManager txmanager=null;
public void add customer (Customer cust) {
transactionStatus ts=null;
        try{
        transactionDefinition txDef=new DefaultTransactionDefinition();
        ts=txmanager.getTransaction(txDef);
        jpaTemp.persist(cust);
        txmanager.commit(ts);
        }catch(Exception e){
        Txmanager.rollback(ts);
        }
}
```

Jtc 57: Example using spring programmatic Transactions with JDBC.

Jtc 57: Files required

Jtc57.java	AccountDAO.java
jpaAccountDAO.java	Account.java
InSufficientFundsException.java	Persistence.xml
Jtcindia.xml	

Account.java	Persistence.xml

Package com.jtcindia.spring.jpa;
Import javax.persistence.*;
@Entity
@Table(name=”accounts”);
Public class Account(
@Id
@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;
}
	<persistence….>
	<persistence unit name=”JTCINDIA-PU
	transaction type=”RESOURCE LOCAL”>
	<provider> org.hibernate.ejb.HibernatePersistence
	</provider>
	<class> com.jtcindia.spring.jpa.Account</class>
	<properties>
	<property name=”hibernate.connection.driver
	class” value=”com.mysql.jdbc.Driver”/>
	<property name=”hibernate.connection.driver
	class” jdbc:mysql://localhost:3306/jtcindiadb”/>
	<property name=”hibernate.connection.username”
	value=”root”/>
	<property name=”hibernate.connection.password”
	value=”somprakash”/>
	</properties>
	</persistence-unit>
	</persistence>

jpaAccountDAO.java	

package com.jtcindia.spring.jpa;
import
org.springframework.beans.factory.annotation.
Autowired;
import org.springframework.orm.jpa.*;
import org.springframework.transaction.*;
import org.springframework.transaction.support.
DefaultTransactionDefinition;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit : www.jtcindia.org
**/
Public class jpaAccountDAO implements
AccountDAO {
@Autowired
jpaTemplate jtemp=null;
@Autowired
	TransactionStatus ts=txManager.getTransaction
	(txDef);
	Account
	acc=jtemp.getReference(Account.class.accno);
	Double cbal=acc.getBal();
	txManager.commit(ts);
	return cbal;
	}
	@Override
	Public void withdrow (int accno, double amt) {
	TransactionDefinition txDef=new
	DefaultTransactionDefinition();
	transactionStatus ts=txManager.getTransaction
	(txDef);
	Account acc=jtemp.getReference
	(Account.class.accno);
	Double cbal=acc.getBal();
	If(cbal>=5000+amt){
	Acc.setBal(cbal-amt);
	Jtemp.merge(acc);

JpaTransactionManager txManager=null;
@Override
Public double getBalance (int accno){
TransactionDefinition txDef=new
DefaultTransactionDefinition();
	}else{
	Throw new lnSufficientFundsException();
	}
	txManager.commit(ts);

@override
Public void funds Transfer(int saccno, int daccno,
double amt) {
TransactionDefinition txDef=new
DefaultTransactionDefinition();
transactionStatus ts=txManager.getTransaction
(txDef):
try{
account
Acc1=jtemp.getReference(Account.class,daccno);
Double scbal=acc2.getBal();
If(scbal>=5000+amt){
Acc2.setBal(scbal-amt);
Jtemp.merge(acc2);
}else{
Throw new lnSufficientFundsException();
}
txManager.commit(ts);
}catch(Exception e){
txManager.rollback(ts);
}
}
	@Override
	Public void deposit (lin accno, double amt){
	transactionDefinition txDef=new
	DefaultTransactionDefinition();
	TransactionStatus ts=txManager.getTransaction
	(txDef);
	Account
	acc=jtemp.getReference(Account.class,accno);
	Acc.setBal(acc.getBal()+amt);
	Jtemp.merge(acc);
	Txmanager.commit(ts);
	}

Jtcindia.xml

```
<beans…..>
<context:annotation-config/>
<bean id=”emfactory”
Class=”org.springframework.orm.jpa.localEntityManagerFactoryBean”>
<property name=”persistence UnitName” value=”JTCINDIA PU”/>
</bean>
<bean id=”txManager” class=”org.springframework.orm.jpa.jpaTransactionManager” autowire=”by
Type”/>
<bean id=”jpaTemp” class=”org.springframework.orm.jpa.jpaTemplate” autowire=”constructor”/>
<bean id=”adao” class=”com.jtcindia.spring.jpa.jpaAccountDAO”/>
</beans>
```

Jtc 58: Example using spring programmatic Transactions with JDBC.
Jtc 58: Files required

Jtc58:java	AccountDAO.java
jpaAccountDAO.java	inSufficientFundsException.java
Jtcindia.xml	

jdbcAccountDAO.java

```
package com.jtcindia.spring.jpa;
import org.springframework.beans.factory.annotation.
Autowired;
import org.springframework.jdbc.core.jdbcTemplate;
import
org.springframework.transaction.annotation.lsolation;
import org.springframework.transaction.annotation.
propagation;
import org.springframework.transaction.annotation.
Transactional;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
@Transactional
Public class jdbcAccountDAO implements AccountDAO
{
@Autowired
jdbcTemplate jdbcTemp=null;
@Transactional(propagation=propagation.REQUIRED,is
Olation=lsolation.REPEATABLE READ)
Public void deposit(int accno, douvble amt) {
String sql=”select bal from accounts where accno=?”;
Int X=jdbcTemp.queryForlnt(sql,accno);
Double cbal=new lnteger(x).double Value();
Double nbal=cbal+amt;
String sql1=”update accounts set bal=? where accno=?;
Jdbc Temp.update(sql1,nbal,accno);
}
@Transactional(propagation=propagation.REQUIRED,is
olation=lsolation.REPEATABLE READ)
Public double getBalance(int accno) {
String sql=”select bal from accounts where accno=?;
Int X= jdbc Temp.queryForlnt(sql, accno);
Double cbal=new lnteger(X).double Value();
Return cbal;
}
```

```
@Transactional
(propagation=propagation.TEQUIRED. iso
lation =lsolation.REPATABLE READ)
Public void funds Transfer(int saccno, int
daccno, double amt) {
String sql1=”select bal from accounts where
accno=?;
String sql2=”update accounts set bal=? where
accno=?;
Int y=jdbcTemp.queryForint(sql1, daccno);
Double dcbal=new integer(y).doubleValue();
System.out.println(“Before deposit”+ dcbal);
Double dnbal=dcbal+amt;
jdbcTemp.update(sql2, dnbal,daccno);
y=jdbcTemp.query(y).doubleValue();
system.out.println(“after deposit”+dcbal);
int x=jdbc Temp.queryForint(sql1 saccno);
double scbal=new lnteger(x).doubleValue();
if(scbal>=5000+amt){
double snbal=scbal amt;
jdbcTemp.update(sql2, snbal, saccno);
}else{
Throw new lnSufficientFundsException();
} }
@Transactional(propagation=propagation.
REQUIRED, iso
lation=lsolation.REPEATABLE READ)
Public void withdraw(int accno, double amt){
String sql=”select bal from accounts where
accno=?;
Int x=”jdbcTemp.queryForint(sql,accno);
Double cbal=new lnteger(x).doubleValue();
If(cbal>=5000+amt){
Double nbal=cbal amt;
String sql1=”update accounts set bal=? where
accno=?;
jdbcTemp.update(sql1,nbal,accno);
}else{
Throw new lnSufficientFundsException();
} }
```

Jtcindia.xml

```
<beans…..>
        <context:annotation-config/>
        < tx:annotation-driver transaction manager=”txManager”/>
       <bean id=”dataSource”
                    Class=”org.springframework.jdbc.datasource.DriverManagerDataSource”>
                     <property name=”driverClassName” value=”com.mysql.jdbc.Driver”/>
                     <property name=”url” value=”jdbc:mysql://localhost/jtcindiadb”/>
                     <property name=”username” value=”root”/>
                     <property name=”password” value=”somprakash”/>
       </bean>
       <bean id=”txManager” class=”org.springframework.orm.jpa.jpaTransactionManager”
              autowire=”by Type”/>
       <bean id=”txManager”
              class=”org.springframework.orm .jpa.jpaTemplate”
              autowire=”constructor”/>
       <bean id=”adao” class=”com.jtcindia.spring.jpa.jpaAccountDAO”/>
</beans>
```

Using Annotation Support with Hibernate

1. Configure the DataSource

```
<bean id=” DataSource”
        Class=”org.springframework.jdbc.datasource.DriverManagerDataSource”>
…..
</bean>
```

2. Configure the JpaTransactionManager by injecting EntityManagerFactory

```
<bean id=”txManager”
        Class=”org.springframework.orm.jpa.jpaTransactionManager”
        Autowire=”by Type”/>
```

3. Configure the sessionFactory by injecting DataSource, Hibernate properties, mapping resources.

```
<bean id=”sessionFactory”
        Class=”org.springframework.orm.hibernate3.LocalSessionFactoryBean”>
        …..
        </bean>
```

4. Configure the HibernateTransactionManager by injecting sessionFactory.

```
<bean id=”txManager”
        Class=”org.springframework.orm.hibernate3.HibernateTransactionManager”
        Autowire=”constructor”/>
```

5. configure the HibernateCustomerDAO.

```
<bean id=”cdao” class=”com.jtcindia.spring.HibernateCustomerDAO”/>
```

6. Enable tx namespace in the spring configuration Document.
7. Enable the annotation based transaction by writing the following in spring ConfigurationDocument.

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

8. Inject HibernateTemplate into HibernateCustomerDAO

9. Use @Transactional annotation for class and methods as per the requirement.
   a. mark the HibernateCustomerDAO class with@Transactional annotation.
   b. Mark the required method with @Transactional and specify the required propagation and isolation.

```
@Transactional
Class HIbernateCustomerDAO imp customerDAO{
@Autowired
HibernateTemplate hibernateTemp=null;
@Transactional(propagation=propagation.REQUIRED NEW, isolation=lsolation.READ
UNCOMMITED)
Public void addCustomer(customer Cust){
Hibernate Temp.save(cust);
}
@Transactional(propagation=propagation.REQUIRED, isolation=lsolation.READ COMMITED)
Public void updateCustomer(customer Cust){
Hibernate Temp. update(cust);
}
```

Jtc 59: Example using spring programmatic Transactions with JDBC.
Jtc 59: Files required

Jtc59:java	AccountDAO.java
hibernateAccountDAO.java	Account.java
Account.hbm.xml	inSufficientFundsException.java
Jtcindia.xml	

hibernateAccountDAO.java

```
package com.jtcindia.spring.hibernate;

import org.hibenrate.LockMode;
import org.springframework.beans.factory.
accotation.Autowired;
import org.sprinsfamework.orm.hibernate3.Hibernate
Template;
import org.springfreamework.transaction.annotation.*;
```

```
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
@Transactional(propagation=propagation.REQUIRED,iso
lation=lsolation.REPEATABLE READ)
Public double getBalance(int accno) {
Account acc=htemp.load(Account.class,accno, lock
mode.READ);
Double cbal=acc.getbal():
Returen cbal;
} }
```

```
@Transactionl
Publil class HibernateAccountDAO implements
AccountDAO {
@Autowired
HibenateTemplate htemp=null;
@Transactional)propagation=propagation.REQUIRES
NEW, isolation=lsolation.REPEATABLE READ)
```

```
Htemp.update(acc);
}else{
Throw new inSufficientFundsException();
}}}
@Transactional(propagation=Propagation.
REQUIRES NEW isolation=isolation.
REPEATABLE READ)
```

```
Public void deposit(int accno, double amt) {
Account acc=htemp.load(Account.class, accno,
lockMode.READ);
Acc.setBal(acc.getBal()+amt);
Htemp.update(acc);
}
@Transactional(propagation=Propagation.REQUIRES
NEW, isolation=lsolation.REPEATABLE READ)
Public void withdraw(int accno, double amt) {
Account acc=htemp.load(Account.class,accno, Lock
mode.READ);
Double cbal=acc.getBal():
If(cbal>=5000+amt){
Acc.setBal(cbal-amt);
```

```
Public void funds Transfer(int saccno, int
daccno, double amt) {
Account
acc1=htemp.load(Account.class,daccno,
LockMode.READ);
Acc1.setBal(acc1.getBal()+amt);
Htemp.update(acc1);
Account
acc2=htemp.load(Account.class,saccno,
LockMode.READ);
Double scbal=acc2.getBal();
If(scbal>=5000+amt){
Acc2.setbal(scbal-amt);
Htemp.update(acc2);
}else{
Throw new lnSufficientFundsException();
}
```

Jtcindia.xml

```
<beans…..>
        <context:annotation-config/>
        < tx:annotation-driver transaction manager=”txManager”/>
       <bean id=”dataSource”
                    Class=”org.springframework.jdbc.datasource.DriverManagerDataSource”>
                     <property name=”driverClassName” value=”com.mysql.jdbc.Driver”/>
                     <property name=”url” value=”jdbc:mysql://localhost/jtcindiadb”/>
                     <property name=”username” value=”root”/>
                     <property name=”password” value=”somprakash”/>
       </bean>
       <bean id=”session factory”
                     Class=”org.springframework.orm.hibernate3.LocalSessionFactoryBean”>
                     <property name=”dataSource” ref=”dataSource”/>
                     <property name=”mappingResources”>
                            <list>
                                    <value>com/jtcindia/spring/hibernate/account.hbm.xml</value>
                            </list>
                      </property>
                      <property name=”hibernateProperties”>
<props >
                      <prop key=”hibernate.dialect”>org.hibernate.dialect.MySQLDialect</prop>
                      <prop key=”hibernate.show sql>true</prop>
                     </props>
             </props>
</bean>
          <bean id=”hibernateTemp” class=”org.springframework.orm.hibernate3.hibernateTemplate”
Sutowire=”constructor”/>
```

```
        <bean id="txManager"
        Class="org.springframework.orm.hibernate3.HibernateTransactionManager"
autowire="constructor"/>
        <bean id="adao" class="com.jtcindia.spring.hibernate.hibernateAccountDAO"/>
</beans>
```

Using Annotation Support with jpa

1. Configure the EntityManagerFactory

```
<bean id=" EntityManagerFactory"
        Class="org.springframework.orm.jap.LocalEntityManagerFactoryBean">
        <property name="persistenceUnitName" value="JTCINDIA PU"/>
</bean>
```

2. Configure the JpaTransaction by injecting EntityManagerFactory

```
<bean id="txTemp"
        Class="org.springframework.orm.jpa.jpaTemplate"
        Autowire="constructor"/>
```

3. Configure the jpaTemplateManager by injecting, EntityManagerFactory .

```
<bean id="txManager"
        Class="org.springframework.orm.jpa.jpaTransactionManager"
        Autowire="byType"/>
```

4. configure the jpaCustomerDAO.

```
<bean id="cdao" class="com.jtcindia.spring.jpa.jpaCustomerDAO"/>
```

5. Enable tx namespace in the spring configuration Document.

6. Enable the annotation based transaction by writing the following in spring ConfigurationDocument.

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

7. Use @Transactional annotation for class and methods as per the requirement.

c. Mark the jpaCustomerDAO class with @Transactional annotation.

d. Mark the required method with @Transactional and specify the required propagation and isolation.

```
@Transactional
Class jpaCustomerDAO imp customerDAO{
@Autowired
jpaTemplate jpaTemp=null;

@Transactional(propagation=propagation.REQUIRED NEW, isolation=lsolation.READ
UNCOMMITED)
Public void addCustomer(customer Cust){
jpaTemp.persist(cust);
```

```
}

@Transactional(propagation=propagation.REQUIRED, isolation=lsolation.READ COMMITED)
Public void updateCustomer(customer Cust){
jpaTemp.merge(cust);
}
```

Jtc 60: Example using spring programmatic Transactions with JDBC.
Jtc 60: Files required

Jtc60:java	AccountDAO.java
jpaAccountDAO.java	Account.java
inSufficientFundsException.java	JTCJpaDialect.java
Jtcindia.xml	Persistence.xml

JTCJpaDialect.java

```
Package com.jtcindia.spring.jpa;
Import java.sql.SQLException;
Import javax.persistence.*;
Import org.hibernate.*;
Import org.springframework.jdbc.datasource.DataSourceUtils;
Import org.springframework.orm.jpa.vendor.HibernateJpaDialect;
Import org.Springframework.transaction.TransactionDefinition;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class JTClpaDialect extends HibernateJpaDialect {
@Override
Public object begin Transaction(EntityManager entityManager, TransactionDefinition definition throws
persistenceException, SQLException, transactionException {
Session session = (session) entityManager.getDelegate();
DataSourceUtils.PrepareConnectionForTransaction(Session.connection(), definition);
entityManager.getTransaction().begin();
return prepare Transaction(entityManager, definition. Is ReadOnly(), definition. getName());
}
}
```

JpaAccountDAO.java

```
Package com.jtcindia.spring.jpa;
Import org.springframework.beans.factory. annotation.
Autowired;
Import org.springframework.orm.jpa.jpaTemplate;
Import org.springframework.transaction. annotation.
```

```
@Transactional(propagation=propagation.
REQUIRES NEW isolation. REPEATABLE READ)
Public double getBalance (int accno) {
Account
acc=jtemp.getReference(Account.class,accno);
Double cbal=acc.getBal();
Return cbal;
```

```
Propagation;
Import org.springframework.transaction.
annotation.Transactional;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
@Transactional
Public class jpaAcconutDAO implements Account
DAO {
@Autowired
jpaTemplate jtemp=null;
@Transactional(Propagation=Propagation.
REQUIRES NEW isolation = lsolation.REPEATABLE
READ)
Public void deposit(int accno, double amt) {
Account acc=jtemp.getReference(Account.class,
Accno);
Acc.setBal(acc.getBal()+amt);
Jtemp.merge(acc);
}
@Transactional(propagation=propagation. REQUIRES NE
Isolation=isolation.REPEATABLE READ)
Public void funds Transfer(int saccno, int daccno, double
Amt) {
Account acc1=jtemp.getReference (Account.class, daccno)
Jtemp.merage(acc1);
Account acc2= jtemp.getReference(Account.class,saccno)
Double scbal=acc2.getBal();
If(scbal>=5000+amt){
Acc2.setBal(scbal-amt);
Jtemp.merge(acc2);
}else{
Throw new insufficientFunds Exception();
}
}

@Transactionl(propagation=propagation.REZUIRES
NEW ISOLATION=isolation.REPEATABAL
READ)
Public void withdraw (int accno, double amt) {
Account
acc=jtemp.getReference(Account.class,accno);
Double cbal=acc.getBal();
If(cbal>=5000+amt){
Acc.setbal(cbal-amt);
Jtemp.merge(acc);
}else(
Throw new insufficientFundsException();
}
}
```

Jtcindia.xml

```
<beans…..>
        <context:annotation-config/>
        < tx:annotation-driver transaction manager=”txManager”/>
       <bean id=”emfactory”
```

```
                Class="org.springframework.jpa.localEntityManagerFactoryBean">
                <property name="persistenceUnitName" value="JTCINDIA PU"/>
                <property name="jpaDialect">
                        <pean class="com.jtcindia.spring.jpa.JTCJpaDialect"/>
                </property>
        </bean>
        <bean id="txManager" class="org.springframework.orm.jpa.jpaTransactionManager"
                autowire="by Type"/>
        <bean id="jpaTemp" class="org.springframework.orm.jpa.jpaTemplate"
        <bean id="adao" class="com.jtcindia.spring.jpa.jpaAccountDAO"/>
</beans>
```

Declarative Transactions with AOP using schema schema support:

Using schema support with JDBC

1. Configure the DataSource
```
        <bean id=" DataSource"
                Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
                ……
        </bean>
```
2. Configure the jdbcTransaction by injecting DataSource
```
        <bean id="jdbcTemp"
                Class="org.springframework.jdbc.core.jdbcTemplate"
                Autowire="constructor"/>
```
3. Configure the DataSourceTemplateManager by injecting, DataSource.
```
        <bean id="txManager"
                Class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
                Autowire="constructor"/>
```
4. configure the jpaCustomerDAO.
```
        <bean id="cdao" class="com.jtcindia.spring.jdbc.jdbcCustomerDAO"/>
```
5. Enable tx namespace in the spring configuration Document.
6. Configure Tx advice
```
        <tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:advice id="txAdvice">
```
7. Use<tx:method/> by spcifying the method name with the required propagation and isolation.

Ex:
```
        <tx: advice id="txAdvice" transaction-manager="txManager ">
        <tx:attributes>
        <tx:method      name="addCustomer"
                        Propagation="REQUIRES NEW"
                        Isolation="READ COMMITTED"/>
        <tx:method      name="updateCustomer*;
                        Propagation="REQUIRED"
                        Isolation="READ UNCOMMITTED"/>
        </tx:attributes>
```

</tx:advice>

8. Enable aop namespace in the spring configuration Document.
9. Configure pointcut and advisor

```
<aop:config>
        <aop:pointcut id="txPointcut"
                Expression="execution(*com.jtcindia.spring.jdbc.*DAO.add*(…))"/>
        <aop:advisor pointsor ref="txPointcut" advice-ref="txAdvice"/>
</aop:config>
```

10. Inject jdbc Template into jdbcCustomer DAO.

Class jdbc CustomerDAO imp customerDAO.
@Autowired
JdbcTemplate jdbcTemp=null;
Public void addCustomer(cto){
String sql="insert…";
jdbcTemp.update(sql,cto.getCid(),….);
}
}

Jtc 61: Example using schema support with JDBC.
Jtc 61: Files required

Jtc61:java	AccountDAO.java
jdbcAccountDAO.java	inSufficientFundsException.java
Jtcindia.xml	

jdbcAccountDAO.java

```
package com.jtcindia.spring.jdbc;
import org.springframework.beans.factory.
annotation.Autowired;
import
org.sprinfframework.jdbc.core.jdbc.Template;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class jdbcAccountDAO implements
AccountDAO {
@Autowired
Jdbc Template jdbc Temp=null;
Public void deposit(int accno, double amt) {
String sql="select bal from accounts where
accno=?;
}else{
Throw new inSufficientFundsException();
}
Public double getBalance(int accno){
String sql="select bal from accounts where
accno=?
Int x=jdbc Temp.queryForint(sql, accno);
Double cbal=new lnteger(x). doubvleValue();
Return cbal;
}
Public void withdraw (int accno, double amt) {
String sql="select bal from accounts where
accno=?;
Int x=jdbcTemp.queryForint9sql, accno);
Double cbal=new integer(x). doubleValue();
If(cbal>=5000+amt){
Double nbal=cbal-amt;
String sql1="update accounts set bal=? where
```

```
Int x=jdbcTemp.queryForlnt(sql, accno);
Double cbal=new lnteger(x).double value();
Double nbal=new integer(x).double value();
Double nbgal=cbal+amt;
String sql1=”update accounts set bal=? where
accno=?
jdbcTemp.update(sql1,nbal,accno);
}

Public void dunds Transfer(int saccno, int daccno,
double amt) {
String sql1=”select bal from accounts where
accno=?”;
String sql2=”update accounts set bal=? where
accno=?”;
Int y=jdbc Temp.queryForlnt(sql1, daccno);
Double dcabl=new integer(Y).doubleValue();
System.out.println(“Before deposit”+dcbal);
Double dnbal=dcbal+amt;
Jdbc Temp.update(sql2, dnbal,daccno);
Y=jdbc temp.queryForlnt(sql1, daccno);
Dcbal=new lnteger(y) doubaleValue();
System.out.println(“after deposit”+dcbal);
Int x=jdbcTemp.queryForln(sql1, saccno);
Double scbal=new lnteger(x). double value();
If (scbal>=5000+amt){
double snbal=scbal amt;
jdbcTemp.update(sql2,snbla, saccno);
accno=?”;
Jdbc temp.update(sql1, nbal,accno);
}else{
Throw newf insufficientFundsException();
}
}
```

Jtcindia.xml

```
<beans…..>
        <context:annotation-config/>
        < bean id=”dataSource”
                Class=”org.springframework.jdbc.datasource.DriverManagerDataSource”>
                 <property name=”driverClassName” value=”com.mysql.jdbc.Driver”/>
                 <property name=”url” value=”jdbc:mysql://localhost/jtcindiadb”/>
                 <property name=”username” value=”root”/>
                 <property name=”password” value=”somprakash”/>
       </bean>
       <bean id=”simplejdbcTemp” class=”org.springframework.jdbc.core.jdbcTemplate”
autowire=”contructor”/>
       <bean id=”txdManager”
Class=”org.springframework.jdbc.datasource.DataSourceTransactionManager”
Autowire=”construstor”/>
       <bean id=”adao” class=”com.jtcindia.spring.jdbc.jdbcAccountDAO”/>
<tx:advice id=”txAdvice”transaction manager=”txManager”>
         <tx:attributes>
```

```
                    <tx:method name=”deposit*”isolation=”READ COMMITTED”
                                Propagation=”REQUIRED”/>
                    <tx:method name=”withdraw” isolation=”READ COMMITTED”
                                PROPAGATION=”REQUIRED”/>
                    <tx:method name=”fundsTransfer*” isolation=”REPEATABLE READ”
                                Propagation=REQUIRED”/>
                </tx:attributes>
            </tx:attributes>
        </tx:advice>

        <aop:config>
                <aop:pointcut id=”txPointcut”
                        Expression=”execution(*com.jtcindia.spring.jdbc.*DAO.*(…))”/>
                <aop:advisor pointcut-ref=”txPointcut” advice-ref=”txAdvice”/>
        </aop:config>
</beans>
```

Using schema support with Hibernate:

1. Configure the DataSource
```
<bean id=” DataSource”
        Class=”org.springframework.jdbc.datasource.DriverManagerDataSource”>
        ……
</bean>
```
2. Configure the sessionFactory by injecting DataSource, Hibernate properties, mapping Resoures,
```
<bean id=”sessionFactory”
        Class=”org.springframework.orm.hibernat3.LocalSessionFactoryBean”>
        ……
 </bean>
```

3. Configure the HibernateTemplate by injecting, sessionFactory.
```
<bean id=”hibernateTemp”
        Class=”org.springframework.orm.hibernate3.HibernateHibernateManager”
        Autowire=”constructor”/>
```
4. configure the Hibernate Transaction Manager by injecting session Factory.
```
<bean id=”txManager”
          Class=”org.springframework.orm.hibernate3.HibernateTransactionManager”
        Autowire=”constructor”/>
```
5. Configure the HibernateCustomerDAO.
```
<bean id=”cdao” class=”com.jtcindia.spring.hibenrte.HibernateCustomerDAO”/>
```
6. Enable tx namespace in the spring configuration Document.
7. Configure Tx advice
```
<tx:advice id=”txAdvice” transaction-manager=”txManager”>
<tx:advice id=”txAdvice”>
```
8. Use<tx:method/> by sprcifying the method name with the required propagation and isolation.
Ex:

```
<tx: advice id="txAdvice" transaction-manager="txManager ">
<tx:attributes>
<tx:method      name="addCustomer"
                Propagation="REQUIRES NEW"
                Isolation="READ COMMITTED"/>
<tx:method      name="updateCustomer*;
                Propagation="REQUIRED"
                Isolation="READ UNCOMMITTED"/>
</tx:attributes>
</tx:advice>
```

9. Enable aop namespace in the spring Configuration Document.
10. Configure pointcut and advisor

```
<aop:config>
        <aop:pointcut id="txPointcut"
                Expression="execution(*com.jtcindia.spring.hibernate.*DAO.add*(…))"/>
        <aop:advisor pointsor ref="txPointcut" advice-ref="txAdvice"/>
</aop:config>
```

11. Inject hibernate Template into HibernateCustomer DAO.

Class hibernate CustomerDAO imp customerDAO.

```
@Autowired
hibernate Template hibernate Temp=null;
Public void addCustomer(customer cust){
hibernateTemp.save(cust);
}
Public void updateCustomer(customer cust){
hibernateTemp.update(cust);
}
}
```

Jtc 62: Example using schema support with JDBC.

Jtc 62: Files required

Jtc62:java	AccountDAO.java
HibernateAccountDAO.java	Account.java
Account.hbm.java	inSufficientFundsException.java
Jtcindia.xml	

Hibernate AccountDAO.java

```
Package com.jtcindia.spring.hibernate;
Import org.hibernate.LockMode;
Import org.springframework.beans.factory.
annotation.Autowired;
Import org.springframework.orm.hibernate3. HibernateTe
mplate;
Public class Hibernate AccountDAO implements
AccountDAO {
@Autowired
HibernateTemplate htemp=null;
```

```
Public void funds Transfer(int saccno, int daccno, double
amt){
Account acc1=htemp.load(Account.class,daccno,
LockModeREAD);
Acc1.setBal(acc1.getBal()+amt);
Htemp.update9acc1);
Account acc2=htemp.load(account.class.saccno,
lockMode.READ);
Double scbal=acc2.getBal();
If(scbal>=5000+amt){
Acc2.setBal(scbal amt);
```

```
Public void deposit(int accno, double amt) {
Account acc=htemp.load(Account.class, accno,
LockMode.READ);
Acc.setBal()+amt);
Htemp.update(acc);
}
Public double get Balance (int accno) {
Account acc=htemp.load(Account.class, accno,
LockMode.READ);
```

```
}else{ throw new lnSufficientFundsException(); }
Public void withdraw (int accno, double amt) {
Acccount acc=htemp.load (Account.class,
accno,LockMode.READ);
Double cbal=acc.getBal();
If(cbal>=5000+amt);
Acc.setbal(cbal-amt);
Htemp.update(acc);
}else{ throw new inSufficientFundsException();}
```

Jtcindia.xml

```
<beans…..>
        <context:annotation-config/>
        < bean id=”dataSource”
                Class=”org.springframework.jdbc.datasource.DriverManagerDataSource”>
                 <property name=”driverClassName” value=”com.mysql.jdbc.Driver”/>
                 <property name=”url” value=”jdbc:mysql://localhost/jtcindiadb”/>
                 <property name=”username” value=”root”/>
                 <property name=”password” value=”somprakash”/>
       </bean>
       <bean id=”sessionFactory”
                  class=”org.springframework.orm.hibernate3.LocalSessionFactoryBean”/>
                  <property name=”dataSource” ref=”dataSource”/>
                  <property name=”mappingResources”>
                  <property name=”mapping Resources”>
                        <list>
                              <value>com/jtcindia/spring/hibernate/Account.hbm.xml</value>
                        <list>
                   </property>
                   <property name=”hibernateProperties”>
                        <props>
                              <prop key=”hibernate.dialect”>org.hibernate.dialect.MySQLDialect</prop>
                              <prop key=”hibernate.show sql”>true</prop>
                              <prop key=”hibernate.hbm2ddl.auto”>update</prop>
                        </props
             </property>
</bean>
<bean id=”hibernateTemp” class=”org.springframework.orm.hibernate3.HibernateTemplate”
        Autowire=”constructor”/>
<bean id=”txManager”
        Class=”org.springframework.orm.hibernate3.hibernateTransactionManager”
           Autowire=”constructo”/>
<bean id=”adao” class=”com.jtcindia.spring.hibernate.HibernateAccountDAO”/>
<tx:advice id=”txAdvice” transaction manager=”txManager”>
<tx:attributes>
<tx:method name=”deposit*” isolation=”READ COMMITTED” propagation=”REQUIRED”/>
<tx:method name=”withdraw*” isolation=”READ COMMITTED” propagation=”REQUIRED”/>
```

```
<tx:method name="fundsTransfer*" isolation="READ COMMITTED" propagation="REQUIRED"/>
<tx:method name="getBalance*" isolation="READ COMMITTED" propagation="REQUIRED"/>
</tx:attributes>
</tx:advice>
<aop:aonfig>
<aop:pointcut id="txPointcut" expression="execution(* com.jtcindia.spring.hibernate.*DAO.*(…))/>
<aop:advisor pointcut-ref="txPointcut" advice-ref="txAdvice"/>
</aop:config>
</beans>
```

Using schema Support with jpa

1. Configure the EntityManagerFactory

```
<bean id=" EntityManagerFactory"
        Class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
        <property name="persistenceUnitName" value="JTCINDIA PU"/>
</bean>
```

2. Configure the JpaTransaction by injecting EntityManagerFactory

```
<bean id="txTemp"
        Class="org.springframework.orm.jpa.jpaTemplate"
        Autowire="constructor"/>
```

3. Configure the jpaTemplateManager by injecting, EntityManagerFactory .

```
<bean id="txManager"
        Class="org.springframework.orm.jpa.jpaTransactionManager"
        Autowire="byType"/>
```

4. configure the jpaCustomerDAO.

```
<bean id="cdao" class="com.jtcindia.spring.jpa.jpaCustomerDAO"/>
```

5. Enable tx namespace in the spring configuration Document.

6. Enable Tx advice.

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

7. Use<tx:method/> by sprcifying the method name with the required propagation and isolation.
Ex:

```
<tx: advice id=”txAdvice” transaction-manager=”txManager “>
<tx:attributes>
<tx:method      name=”addCustomer”
                Propagation=”REQUIRES NEW”
                Isolation=”READ COMMITTED”/>
<tx:method      name=”updateCustomer*;
                Propagation=”REQUIRED”
                Isolation=”READ UNCOMMITTED”/>
</tx:attributes>
</tx:advice>
```

8. Enable aop namespace in the spring Configuration Document.
9. Configure pointcut and advisor

```
<aop:config>
        <aop:pointcut id=”txPointcut”
            Expression=”execution(*com.jtcindia.spring.jpa.*DAO.add*(…))”/>
        <aop:advisor pointsor ref=”txPointcut” advice-ref=”txAdvice”/>
  </aop:config>
```

11. Inject jpa Template into jpaCustomer DAO.

```
Class jpa CustomerDAO imp customerDAO.
        @Autowired
        jpa Template jpa Temp=null;
        Public void addCustomer(customer cust){
        jpa Temp.persist(cust);
        }
        Public void updateCustomer(customer cust){
        jpaTemp.merge(cust);
        }
}
```

Jtc 63: Example using schema support with JDBC.
Jtc 63: Files required

Jtc63:java	AccountDAO.java
jpaAccountDAO.java	Account.java
insufficientFundsException.java	inSufficientFundsException.java
Jtcindia.xml	Persistence.xml

jdbcAccountDAO.java	Accno);
package com.jtcindia.spring.jdbc; import org.springframework.beans.factory. annotation.Autowired;	Double cbal=acc.getBal(); Return cbal; Public void funds Transfer(int saccno, int daccno,

```
import
org.sprinfframework.jdbc.core.jdbc.Template;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ Visit          : www.jtcindia.org
**/
Public class jdbcAccountDAO implements
AccountDAO {

@Autowired
Jdbc Template jdbc Temp=null;

Public void deposit(int accno, double amt) {
Account
acc=jtemp.getReference(Account.class,accno);
Acc.setBal9acc.getBal()+amt);
Jtemp.merge(acc);
}
Public double getBalace(intaccno){
Account
acc=jtemp.getReference(Account.class,accno);
Double cbal=jtemp.getReference(Account.class,
double amt) {
Account
Acc1=jtemp.getReference(account.class,daccno);
Acc1.setBal(acc1.getBal()+amt);
Jtemp.merge9acc1);
Account
Acc2=jtemp.getReference9Account.class,saccno);
Double scbal=acc2.getBal();
If(scbal>=5000+amt){
Acc2.setBal9scbal-amt);
Djtemp.merge(acc2);
}else{
Throw new insufficient FundsException();
}
Public void   withraw(int accno, double amt) {
Account
acc=jtemp.getReference(Account.class,accno);
Double cbal=acc.getBal();
If9cbal>=5000+amt){
Acc.setBal(cbal-amt);
Jtemp.merge(acc);
}else{
Throw new inSufficientFundsException();
```

Jtcindia.xml

```
<beans…..>
        <context:annotation-config/>
        < bean id=”emfactory”
                Class=”org.springframework.orm.jpa.LocalEntityManagerFactoryBean”>
                 <property name=”persistenceUnitName” value=”JTCINDIA PU”/>
                 <property name=”jpaDialect”>
                 <pean class=”com.jtcindia.spring.jpa.JTCJpaDsialect”/>
              </property>
       </bean>
       <bean id=”tmManager” class=”org.springframework.orm.jpa.jpaTransactionManager”
                  autowire=”byType”/>
       <bean id=”jpaTemp”Class=”org.springframework.orm.jpa.jpaTemplate”
                   Autowire=”construstor”/>
       <bean id=”adao” class=”com.jtcindia.spring.jpa.jpaAccountDAO”/>

       <tx:advice id=”txAdvice”transaction manager=”txManager”>
                  <tx:attributes>

                     <tx:method name=”deposit*”isolation=”READ COMMITTED”
                              Propagation=”REQUIRED”/>
                    <tx:method name=”withdraw” isolation=”READ COMMITTED”
                              PROPAGATION=”REQUIRED”/>
```

```
                    <tx:method name=”fundsTransfer*” isolation=”REPEATABLE READ”
                                Propagation=REQUIRED”/>
                    <tx:method name=”getBalance*” isolation=” READ COMMITTED”
                                Propagation=REQUIRED”/>
                </tx:attributes>
            </tx:advice>
        <aop:config >
                <aop:pointcut id=”txPointcut”
                        Expression=”execution (*com.jtcindia.spring.jdbc.*DAO.*(…))”/>
                <aop:advisor pointcut-ref=”txPointcut” advice-ref=”txAdvice”/>
        </aop:config>
</beans>
```

Following are the attributes of @ Transactional

SNO	Attribute name	Possible values	Default
1	Propagation	Propagation enum	REQUIRED
2	Isolation	Isolation enum constant	DB vendor
3	readOnly	True / false	-
4	Timeout	Integer	-
5	rollbackFor	Exception array	-
6	noRollbackFor	Exception array	-

Following are the attributes of <txd:method>

SNO	Attribute name	Possible values	Default
1	Name	-	-
2	Propagation	Propagateion level	REQUIRED
3	Isolation	Isolation level	DB vendor
4	Read-only	True / false	False
5	Timeout	Integer	-
6	Rollback-for	Exception array	-
7	No-rollback-for	Exception array	-