(No 1 in Training & Placement)

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)

1

(No 1 in Training & Placement)

- 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 `
 - o Spring version spring 3.0
 - o Spring 3.0 AOP libraries
 - o Spring 3.0 core Libraries
 - o Spring 3.0 persistence core Libraries
 - Spring 3.0 persistence JDBC Libraties
 - Click on Next
 - Provide spring configuration Document file name as itcindia. 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 itcindia;

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	

Spring PART 8

(No 1 in Training & Placement)

```
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	Public void updateCustomer (Custoemr TO cto) {
Package com. Jtcindia.spring.Hibernate;	Customer c=(customer) hibernate Temp
Import java.util.ArrayList;	Load(customer.class,cto.getCid());
Import java.util .list;	c.setCid(cto.getCname());
1 3	csetCname(cto.getCname());

3

Spring PART 8

(No 1 in Training & Placement)

```
c.setEmail(cto.getEmail());
Import org.hibernate.Lock mode;
Import org.springframework.beans.factory.
                                                       c.setPhone(cto.getPhone());
annotation. Autowired;
                                                       c.setcity(cto.getCity());
Import org.springframework.orm.hibernate3.
                                                       hibernateTemp.update(c, LockMode.NONE);
HibernateTemplate;
                                                       Public customerTO getCustomerByCid(int cid){
*@Author: Som Prakash Rai
                                                       Customer c=(customer)
*@Company: java Training Center
                                                       HibernateTemp.load (Customer.class, cid);
*@ Visit
               : www.jtcindia.org
                                                       customerTO cto=new customerTO (c.getCid();
                                                       c.getCname(), c.getEmail(), c.getPhone(), c.getCity());
Public class HibernateCustomerDAO implements
                                                       return cto:
customerDAO{
@Autowired
Hibernate Template Hibernate Temp;
Public void addCustomer(Customer TO cto){
                                                       Customer TO.java
Customer cust=new customer(cto.getCname(),
                                                       Package com.jtcindia.spring.hibernate;
Cto.getEmali(),
Cto.getPhone()lcto.getCity());
                                                       *@Author: Som Prakash Rai
hibernateTemp.save(cust);
                                                       *@Company: java Training Center
                                                       *@ Visit
                                                                      : www.jtcindia.org
Public void deleteCustomer(int cid) {
                                                       **/
Customer c = (customer)_{-}
                                                       Public class customerTO {
Hibernate Temp.load(customer.class,cid);
                                                                           private string cname;
                                                       Private int cid;
Hibernate temp.delete(c, lockMode.None);
                                                       Private string email.
                                                                              Private long phone;
                                                       Private string city;
Public list<customerTO>getAllCustomers(){
                                                       Public customer TO () {}
List<customerTO> ctoList=new
                                                       Public customerTO (int cid, string cname, string email,
ArrayList<customerTO>();
                                                       long phone, string city) {
String hql="from customer c";
                                                       This.cid=cid:
                                                                              this.cname=cname:
List<customer> list=hibernateTemp.find(hgl);
                                                       This.email=email;
                                                                             this.phone=phone
For(customer c: list){
                                                       This.city=city;
customerTO cto=new customerTO (c.getCid();
c.getCname()l
c.getEmail(),c.getPhone(),c.getCity()).
                                                       // setters and Getters
Ctolist.add(cto);
Return ctoList;
```

Customer.java	Customer.hbm.xml
Package com.jtcindia.spring.hibernate;	<hibernate-mapping< td=""></hibernate-mapping<>

(No 1 in Training & Placement)

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

```
Jtcindia.xml
<context:annotation-config/>
<br/>
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
com.mysql.jdbc.Driver"/>
cproperty name="url" value="jdbc:mysql://locall.ost/jtcindiadb"/>
property name="username" value="root"/>
cproperty name="password" value="somprakash"/>
</bean>
<bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
property name="mappingResources">
t>
<value>com/jtcindia/spring/hibernate/customer.hbm.xml</value>
</list></property>
property name="hibernateProperties">
<entry key="hibernate.dialect"/> ialect"/> ialect. 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"/>
```

Jtc50: files required

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

Java Training Center (No 1 in Training & Placement)

HibernateCustomerDAO.java	Public List <customerto> getAllcustomers(){</customerto>
Package com.jtcindia.spring.hibernate;	List <customerto> ctoList=new</customerto>
Import java.util.*;	ArrayList <customerto>();</customerto>
Import org.hibernate.criterion.*;	DetachedCriteria dc=
Import	DetachedCriteria.forClass(Customer.class);
org.springframework.beans.factory.annotation.	List <customer> list=hibernateTemp.findBy</customer>
Autowired;	criteria(dc);
Import	For (customer c: list) {
org.springframework.orm.hibernate3.Hibernate	CustomerTO cto =new CustomerTO (c.getCid();
Template;	c.getEmail(),c.getPhone(), c.getCity());
/*	ctoList.add(cto);
*@Author: Som Prakash Rai	}
*@Company: java Training Center	Return ctoList;
*@ Visit : www.jtcindia.org), AC MON
**/	
Public class HibernateCustomerDAO implements	Public Long get Customer PhoneBy Email(String
customerDAO {	email) {
@Autowired	DetachedCriteria dc=
HibernateTemplate hibernateTemp;	DetachedCriteria.forClass(Customer.class);
Public string getCustomerCityByEmail(String	dc.add(Expression.eq("email",email));
email){	List <customer></customer>
DetachedCriteria dc=	list=hibernateTemp.findByCriteria(dc);
DetachedCriteria.forClass(Customer.class);	Customer c=list.get(0);
dc.add(Expression.eq("email",email));	Return c.getPhone();
List <customer> list=hibernate GROW</customer>	HJUNBOUND 🛣
Temp.findByCriteria(dc);	\$
Customer c=list.get(0);	4
Return c.getCity();	49/x
	CONTAIN
Aal	VEO BILL
X	

Public int getCustomersCount() {	Public List <customerto></customerto>
detachedCriteria dc=	getCustomersByCity(String city) {
DetachedCriteria.forClass(Customer.class);	List <customerto> ctoList=new</customerto>
List <customer></customer>	ArrayList< customerTO>();
list=hibernateTemp.findByCriteria(dc);	DetachedCriteria dc=
Return list.size();	DetachedCriteria.for Class(Customer.class);
}	dc.add(Expression.eq("city",city));
Public customerTO getCustomerByEmail(String	hibernateTemp.setCacheQueries(true);
email){	hibernateTemp.setCacheQueries(true);
DetachedCriteria dc=	list <customer>list=hibernateTemp.findByCriteria(dc);</customer>

Spring PART 8

(No 1 in Training & Placement)

```
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.getPhone(),c.getPhone(),c.getCity());
c.getCname(), c.getEmail(), c.getPhone(),
c.getCity());
return cto;
} }

for (customer c: list) {
customerTO cto =new CustomerTO (c.getCid(),
c.getEmail(),c.getPhone(),c.getCity());
Return ctoList; }
```

Working with Hibernate DaoSupport:

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

```
Public Hibernate Template 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()");

Spring PART 8

(No 1 in Training & Placement)

```
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	

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");

A BUNDE CENTER

CustomerDAO.java	HibernateCustomer DAO.java
Package com.jtcindia.spring.hibernate;	Package com.jtcindia.spring.hibernate;
/*	Import java.sql.*;
@Author: Som Prakash Rai	Import org.hibernate.;
*@Company: java Training Center	Import org.springframework.orm.hibernate3.
*@ Visit : <u>www.jtcindia.org</u>	HibernatecCallback;
**/	Import
Public interface customerDAO{	org.springframework.orm.hibernat3.support.
Public void addCustomer(customerTO cto);	Hibernate Daosupport;
}	Public class HibernatCustomerDAO extends
	HibernateDaoSupport implements CustomerDAO {

Spring PART 8

(No 1 in Training & Placement)

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

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

Spring PART 8

Java Training Center (No 1 in Training & Placement)

|--|

TD. 7. 4.	YD LY OL
JPAJtc1.java	JPAJtc2.java
Package com.jtcindia.jpa;	Package com.jtcindia.jpa;
Import javax.persistence.*;	Import javax.persistence.*;
/*	/*
*@Author: Som Prakash Rai	*@Author: Som Prakash Rai
*@Company: java Training Center	*@Company: java Training Center
*@ Visit : <u>www.jtcindia.org</u>	*@ Visit : <u>www.jtcindia.org</u>
**/	**/
Public class JPAJtc1 {	Public class JPAJtc2 {
Public static void main(string[] args) {	Public static void main(string[] args) {
Entity Transaction tx=null;	Entity Transaction tx=null;
Try{	Try{
EntityManagerFactory	EntityManagerFactory factory=Persistence.create
factory=Persistence.create	EntityManagerFactory("jtcindia-pu);
EntityManagerFactory("jtcindia-pu);	EntityManager
EntityManager	Manager=factory.createEntity manager();
Manager=factory.createEntity manager();	Tx=manager.getTransaction();
Tx=manager.getTransaction();	Tx.begin();
Tx.begin();	Customer cust=(customer)
Customer cust=new	manager.getReference(Customer.class,1);
Customer("som", "som@jtc", "1111", "Noida");	Systim.out.println(cust.getCid()+"\t"+cust.getCname()
Manager.persist(cust);	+"\t"+cust.getEmail()+"\t"+cust.getPhone()+"\t"+cust.
Tx.commit();	Get city());
System.out.println("check Database");	Tx.commit();
} catch (Exception e){	}catch(Exception e){
e.printStackTrace();	e.printStackTrace();
tx.rollback();	tx.rollbcak();
}}}	}}}

Customer.java	@Column(name="email")
Package com.jtcindia.jpa;	Private string email;
Import javax.persistence.*;	@Column(name="phone")
@Entity	Private string phone;
@Table(name="customers")	@Column(name="city")
Public class Customer{	Private string city;
,	Public customer (){}
@id	Public customer (String cname, string email,
@GeneratedValue(strategy=GenerationType.AUTO)	string phone, string city) {
@Column(name="cid")	This.cname=cname;
Private int cid;	This.email=email;
	This.phone=phone;
@Column(name="cname")	This.city=city;

10

Spring PART 8

(No 1 in Training & Placement)

Private string cname;	}
	//setters and getters
	}

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 categorization of exceptions.

Above problems are solved as follows:

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

Spring PART 8

(No 1 in Training & Placement)

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 itcindia pu
- 5. Add spring capabilities as follows:
- Select the project and Right click
- Select my Eclipse > Add spring capabilities
- Select the following
 - o Spring version spring 3.0
 - o Spring 3.0 AOP libraries
 - o Spring 3.0 core Libraries
 - o Spring 3.0 persistence core Libraries
 - o Spring 3.0 persistence JDBC Libraties
- Click on Next

(No 1 in Training & Placement)

- Provide spring configuration Document file name as itcindia. 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.

10. Configure Jpa TransactionManager by injectiong LocalEntity managerFactory Bean.

```
<bean id="transactionManager"</pre>
```

Jtc52: files required

GROWTH LINBOLING

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

Jtc52.java

Package com.jtcindia.spring.jpa;

Import org.springframework.context.applicationContext;

 $Import\ org. spring framework. context. support. class PathXml Application Colntext;$

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);

13

(No 1 in Training & Placement)

```
//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
                                                   Public void update Customer (customer TO cto) {
                                                   TransactionDefinition TxDef=new
package com.jtcindia.spring.jpa;
                                                   defaultTransactionDefinition();
                                                   transactionstatus ts=txmanager.getTransaction(txDef);
*@Author: Som Prakash Rai
                                                   customer c=(customer) jpaTemp.find(Customer.class,
*@Company: java Training Center
                                                   cto.getCid());
               : www.jtcindia.org
*@ Visit
                                                   c.setEmail(cto.getEmail());
                                                   c.setphone(cto.getPhone());
Public interface customerDAO {
                                                   c.setCity(cto.getCity());
Public void addCustomer (customer TO cto);
                                                   jpaTemp.merge©;
Public void updateCustomer (customerTO cto);
                                                   txmanager.commit(ts);
Public void delect Customer (int cid);
Public customer To getCustomerByCid(int cid);
                                                   Public void delete Customer(int cid) {
                                                   TransactionDefinition txDef=new
jpaCustomerDAO.java
                                                   DefaultTransactionDefinition();
package com.jtcindia.spring.jpa;
                                                   TransactionStatus ts=txmanager.getTransaction(txDef);
import org.springframework.beans.factory.
                                                   Customer c=(customer) jpaTexp,find(customer.class,
annotation.autowired:
                                                   cid);
import org.springframework.orm.jpa.*;
                                                   jpaTemp.remove©;
import org.springframework.transaction.*;
                                                   txmanager.commit(ts);
import org.springframework.transaction.support.
DefaultTransa ctionDefinition;
public class jpaCustomerDAO implements
                                                   Public customerTO getCustomerByCid(int cid){
CustomerDAO{
                                                   Customer c=(customer)jpaTemp.find(customer.class, cid);
@Autowired
                                                   customerTO cto=new customerTO (c.getCity());
Jpa Template jpaTemp=null;
                                                   return cto;
@Autowired
                                                   }}
ipaTransactionnManager txmanager=null;
                                                   CustomerTO.java
public void addcustomer(customerTO cto){
                                                   Package com.jtcindia.spring.hibernate;
customer cust=new customer(cto.getCid();
                                                   Public class customer TO{
cto.getCname(),
                                                   Private int cid;
                                                                       private string cname;
cto.getEmail(),cto.getPhone(), cto.getCity());
                                                   Private string email;
                                                                           private long phone;
Transaction Definition txDef=new
```

Spring PART 8

(No 1 in Training & Placement)

DefaultTransactionDefinition();p	Private string city;
TransactionStatus ts=	Public customerTO() {}
Txmanager.getTransaction(txDef);	Public customerTO (string cname, string email, long
jpaTemp.persist(cust);	phone, string city) {
txmanager.commit(ts);	This.cname=cname; this.email=email;
}	This.phone=phone; this.city=city;
	}
	//setters and getters
	}

```
Persistence.xml
Customer.java
Package com.jtcindia.jpa;
                                                <persistence...>
Import javax.persistence.*;
                                                 <persistence unit name="JTCINDIA PU"</pre>
                                                transaction-Type="RESOURCE-LOCAL">
@Entity
@Table(name="customers")
                                                Public class Customer{
                                                provider>
                                                <class.com.jtcindia.spring.jpa.customer</class>
@Entity
                                                cproperties>
@Table(name="customers")
                                                <peeperties>
Public class customer{
                                                property name="hibernate.connection.driver
@id
                                                class" value=com.mysql.jdbc.driver"/>
@ @Column(name="cid")
                                                property name="hibernat.connection.url"value=
Private int cid:
                                                jdbc:mysql://localhost:3306/jtcindiadb"/>
                                                property name="hibernate.connection.password"
@Column(name=''cname'')
                                                value="somprakash"/>
Private string cname;
                                                </properties>
@Column(name="email")
                                                </persistence-unit>
Private string email;
                                                </persistence>
@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
```

Jtcindia.xml

(No 1 in Training & Placement)

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("itcindia.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);
```

Spring PART 8

(No 1 in Training & Placement)

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

Public string getCustomerCityByEmail(String email);

Public Long getCustomerPhoneByEmail(String email);

Public Long getCustomerPhoneByEmail(String email);
```

JPA CustomerDAO.java	Public list <customerto> getCustomersByCity(String</customerto>
Package com.jtcidia.spring.jpa;	city) {
Import java.util.*;	List <customerto> ctoList=new</customerto>
Import org.springframework.beans.factory.	ArrayList <customerto>();</customerto>
annotation. Autowired;	String jpaql="from customer c where c. city=?";
Import org.spring framework.orm.jpa.jpaTemplate;	List <customer>list=jpaTemp.lfind(jpaql, city);</customer>
Import org.springframework.orm.jpa.jpaTransaction	For(customer c:list){

Spring PART 8

(No 1 in Training & Placement)

```
customerTO cto=new customerTO(c.getCid();
Manager;
                                                     c.getCname();
*@Author: Som Prakash Rai
                                                     c.getEmail(), c.getPhone(), c.getCity());
*@Company: java Training Center
                                                     ctoList. Add(cto);
               : www.jtcindia.org
*@ Visit
                                                     Return ctoList;
Public class jpaCustomerDAO implements
CustomerDAO {
@Autowired
ipaTemplate ipaTemp = null;
```

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

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

Spring PART 8

(No 1 in Training & Placement)

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:
<br/>
Configuration file:
<br/>
<
```

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;	

Spring PART 8

(No 1 in Training & Placement)

```
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 JpaCustomerDAO.java Package com.jtcindia.spring.jpa; Package com.jtcindia.spring.jpa; Import javax.persistence.*; *@Author: Som Prakash Rai Import org.springframework.beans.factory.annotation. *@Company: java Training Center Autowired: *@ Visit : www.jtcindia.org Import org.springframework.orm.jpa.*; Import Public interface customerDAO{ org.springframework.orm.jpa.support.jpaDaoSupport; Public void addCustomer(customerTO cto); Import org.springframework.transaction.*; Import org.springframework.transaction.support. default Transa ction Definition: Jtcindia.xml *@Author: Som Prakash Rai *@Company: java Training Center <context:annotation-config/> *@ Visit : www.itcindia.org <bean id="entityManagerFactory"</pre> Class="org.springframework.orm.jpa.LocalEntity Public class jpaCustomerDAO extends jpaDaoSupport manager factory Bean"> property name="persistenceUnitName" value =" implements CustomerDAO{ @Autowired JTCINDIA-PA"/> ipaTransactionManagerTxm anager=null; </bean> public void addCustomer(CustomerTO cto){ <bean id="transactionManager" class=k"org.</pre> final customer cust=new customer (cto.getCid(); springframework.org.jpa.jpaTransactionManager"> cto.getCname(), cto.getEmail(), cto.getPhone(), cproperty name="entityManagerFactory" ref=" cto.getCity()); entityManagerFactory"/> jpaCallback<EntityManager> jc=new </bean> jpaCallback<EntityManager>() { <bean id="ipaTemp" class="org.springframework.</pre> public EntityManager doinjpa(EntityManager Manager) orm.jpa.jpaTemplate" autowire=constructor"/> Throws persistenceException{ <bean id="cdao" class="com.jtcindia.</pre>

Spring PART 8

(No 1 in Training & Placement)

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

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:

(No 1 in Training & Placement)

Package org.springframework.transaction Interface transactionDefinition

	interface transactions emitted		
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

Class Hoernate Hansaction Wanager			
Hibernate Transaction manager()		*	
Hibernate TransactionManager(sessionFactory sessionFactory)	/		

dataSource	getDataSource() UNBOUND
sessionFactory	getSessionFactory()
Void	setDataSource(DataSource dataSource)
Void	setSession factory(sessionFactory sessionFactory)
	X X X X X X X X X X X X X X X X X X X
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)

(No 1 in Training & Placement)

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

```
Programation Transaction with JDBC
1. Configure the Data Source
       <bean id="dataSource"</pre>
               Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
       </bean>
2. Configure the Jdbc Template by injection DataSource.
       <br/>bean id="idbcTemp"
               Class="org.springframework.jdbc.core.jdbcTemplate"
               Autowire="construstoer"/>
3. Configure the DataSource TransactionManager by injecting DataSource
       <bean id="txManager"</pre>
               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');
```

(No 1 in Training & Placement)

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));
System.out.printin(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));
\{\bar{\}}

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

(No 1 in Training & Placement)

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

RuntimeException{}

RuntimeException{}
}

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

Spring PART 8

(No 1 in Training & Placement)

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

Programmatic Transactions with Hibernate

```
1. . Configure the Data Source
```


dataSource

Class="org.springframework.jdbc.datasource.DriverManagerDataSource">

....

</bean>

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

<bean id="sessionFactory"</pre>

Class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

</bean>

3. Configure the Hibernate Template by injecting sessingFactory

<bean id="hibernateTemp"</pre>

Class="org.springframework.orm.hibernate3.HibernateTemplate"

Autowire="constructor"/>

4. Configure the Hibernate TransactionManager by injecting DataSource

<bean id="txManager"</pre>

Class="org.springframework.orm.hibernate3.HibernateTransactionManager"

Autowire="constructor"/>

5. Configure the JdbcCustomerDAO.

<bean id="cdao" class="com.jtcindia.spring.hibernate.HibernateCustomerDAO'/>

26

Spring PART 8

(No 1 in Training & Placement)

- 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 GROWT	AccountDAO.java 💮
HibernateAccountDAO.java	Account.java
Account.hbm.xml	inSufficientFundsException.jave
Jtcindia.xml	* *

10 CENTER

Account.java	Account.hbm.xml
Package com.jtcindia.spring.hibernate;	<a com.jtcindia."="" href="chibernate-mapping package=">hibernate-mapping package="com.jtcindia.
Public class Account {	spring. Hibernate">
Private int accno;	<class name='Account"table="accounts"'></class>
Private string atype;	<id column="accno" name="accno" type="int"></id>
Private double bal;	<pre><generator class="increment"></generator></pre>
Public Account()[]	
Public Account(String atype, double bal) {	<pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre>
This.atype=atype;	<pre><pre><pre><pre>cproperty name="bal"type="double"/></pre></pre></pre></pre>
This.bal=bal;	
}	
//setters and getters	
}	

27

Java Training Center (No 1 in Training & Placement)

HibernateAccountDAO.java	@override
package com.jtcindia.spring.jdbc;	Public void funds Transfer(int saccno, int daccno,
import org.hibernate.LockMode;	double amt){
import	TransactionDefinition txDef=new
org.springframework.beans.factory.annotation.Autowired;	DefaultTransactionDefinition();
import org.springframework.orm.hibernate3.hibernate	Transaction Status
Template;	ts=txManager.getTransaction(txDef);
import org.springframework.	Try{
orm.hibernate3.HibernateTransac tionManager;	Account
import org.springframework.transaction.*;	acc1=hibernateTemp.load(Account.class,
import org.springframework.transaction.	daccno,LockMode.Read);
org.springframework.transaction.support.DefaultTransa	Acc1.setbal(acc1.getbal()+amt);
ctionDefinition;	Hiber nate Temp.update(acc1);
/*	Account
*@Author: Som Prakash Rai	acc2=hibernateTemp.load(Account.class, saccno,
	LockMode.Read);
*@Company : java Training Center *@ Visit : www.jtcindia.org	Double scbal=acc2.getBal();
*@ Visit : www.jtcindia.org	If(scbal>=5000+ amt) {
	Acc2. Setbal(scbal amt);
Public class Hibernate Account DAO implements	}else{
AccountDAO(Throw new inSufficientFundsException();
@Autowired	Throw new insufficient runds exception(),
HibernateTemplate hibernate Temp=null;	ty Managar agmmit(ta)
@Autowired	txManager.commit(ts);
Hibernate TransactionManager txManager=null;	}catch(Exception e) {
@override Public void deposit(int accno. double amt){ GROWTH	txManager.rollback(ts);
Tubile void deposit(int deeno, double dint)	ONBOOND
TransactionDefinition txDef=new	
DefaultTransactionDefinition();	@i1-
TransactionStatus ts=	@override
txManager.getTransaction(txDef);	Public void withdraw9int accno, double amt) {
Account acc=hibernateTemp.load(Account.class,	TransactionDefinition txDef=new
accno,LockMode.Read);	DefaultTransactionDefinition();
Acc.setBal9acc.getBal()+amt);	TransactionStatus
hibernateTemp.update(acc);	ts=txManager.getTransaction(txDef);
txManager.commit(ts);	Account acc=hibernateTemp.load(Account.class,
}	accno,
	LockMode.Read);
@override	Double cbal=acc.getbal();
Public double getBalance(int accno) {	$If(cbal >= 5000 + amt) \{$
System.out.println(hibernateTemp);	Acc.setBal(cbal-amt);
Account acc=hibernate Temp.load(Account.Class,	hibernateTemp.update(acc);
accno,LockMode.Read);	}else{
Double cbal=acc.getBal();	Throw new inSufficientFundsException();
Return cbal;	}
}	txManager.commit(ts);
· ·	

28

Spring PART 8

(No 1 in Training & Placement)

```
Programmatic Transactions with JPA
1. Configure the EntityManagerFactory
       <bean id="entityManagerFactory"</pre>
               Class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean">
       property name="persistence unitName" value="JTCINDIA PU"/>
</bean>
2. Configure the jpaTemplate by injecting EntityManagerFactory
       <bean id="ipaTemp"
               Class="org.springframework.orm.jpa.jpaTemplate"
               Autowire="constructor"/>
3. Configure the JpaTransactionManager by injecting EntityManagerFactory
       <bean id="txManager"</pre>
               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
       ipaTemplate ipaTemp=null;
       @Autowired
       jpaTransactionManager txmanager=null; INTO ON
       public void add customer (Customer cust) {
       transactionStatus ts=null:
               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
```

29

Spring PART 8 Author: Som Prakash Rai

Java Training Center (No 1 in Training & Placement)

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

Account.java	Persistence.xml
Package com.jtcindia.spring.jpa;	<pre><persistence></persistence></pre>
Import javax.persistence.*;	<pre><persistence <="" name="JTCINDIA-PU" pre="" unit=""></persistence></pre>
@Entity	transaction type="RESOURCE LOCAL">
@Table(name="accounts");	<pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>
Public class Account(
@Id	<class> com.jtcindia.spring.jpa.Account</class>
@Column(name="accno")	<pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>
Private int accno;	<pre><pre><pre><pre>property name="hibernate.connection.driver</pre></pre></pre></pre>
@column(name="atype")	class" value="com.mysql.jdbc.Driver"/>
Private string atype;	<pre><pre><pre><pre>property name="hibernate.connection.driver</pre></pre></pre></pre>
@column(name='"bal")	class" jdbc:mysql://localhost:3306/jtcindiadb"/>
Private double bal;	<pre><pre><pre><pre>property name="hibernate.connection.username"</pre></pre></pre></pre>
Public Account() {}	value="root"/>
Public Account(String atype, double bal) {	<pre><pre><pre><pre>property name="hibernate.connection.password"</pre></pre></pre></pre>
This.atype=atype;	value="somprakash"/>
This.bal=bal;	
}	

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

Spring PART 8

(No 1 in Training & Placement)

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

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

Jtc 58: Example using spring programmatic Transactions with JDBC.

Jtc 58: Files required

(No 1 in Training & Placement)

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

```
jdbcAccountDAO.java
                                                        @Transactional
                                                        (propagation=propagation.TEQUIRED. iso
package com.jtcindia.spring.jpa;
                                                        lation = lsolation.REPATABLE READ)
import org.springframework.beans.factory.annotation.
                                                        Public void funds Transfer(int saccno, int
Autowired:
                                                        daccno, double amt) {
import org.springframework.jdbc.core.jdbcTemplate;
                                                        String sql1="select bal from accounts where
import
                                                        accno=?;
org.springframework.transaction.annotation.lsolation:
                                                        String sql2="update accounts set bal=? where
import org.springframework.transaction.annotation.
                                                        accno=?:
propagation;
                                                        Int y=idbcTemp.queryForint(sql1, daccno);
import org.springframework.transaction.annotation.
                                                        Double dcbal=new integer(y).doubleValue();
Transactional;
                                                        System.out.println("Before deposit"+ dcbal);
                                                        Double dnbal=dcbal+amt;
*@Author: Som Prakash Rai
                                                        jdbcTemp.update(sql2, dnbal,daccno);
*@Company: java Training Center
                                                        v=idbcTemp.query(v).doubleValue();
               : www.jtcindia.org
*@ Visit
                                                        system.out.println("after deposit"+dcbal);
**/
                                                        int x=jdbc Temp.queryForint(sql1 saccno);
@Transactional
                                                        double scbal=new Integer(x).doubleValue();
Public class jdbcAccountDAO implements AccountDAO
                                                        if(scbal > = 5000 + amt){
@Autowired
                                                        double snbal=scbal amt;
                                                        jdbcTemp.update(sql2, snbal, saccno);
jdbcTemplate jdbcTemp=null;
                                                        }else{
@Transactional(propagation=propagation.REQUIRED,is
                                                        Throw new InSufficientFundsException();
Olation=Isolation.REPEATABLE READ)
Public void deposit(int accno, douvble amt) {
                                                        @Transactional(propagation=propagation.
String sql="select bal from accounts where accno=?";
                                                        REQUIRED, iso
Int X=jdbcTemp.queryForInt(sql,accno);
                                                        lation=lsolation.REPEATABLE READ)
Double cbal=new Integer(x).double Value();
                                                        Public void withdraw(int accno, double amt){
Double nbal=cbal+amt:
                                                        String sql="select bal from accounts where
String sql1="update accounts set bal=? where accno=?;
                                                        accno=?;
Jdbc Temp.update(sql1,nbal,accno);
                                                        Int x="idbcTemp.queryForint(sql,accno);
                                                        Double cbal=new Integer(x).doubleValue();
@Transactional(propagation=propagation.REQUIRED,is
                                                        If(cbal > = 5000 + amt)
olation=lsolation.REPEATABLE READ)
                                                        Double nbal=cbal amt;
Public double getBalance(int accno) {
                                                        String sql1="update accounts set bal=? where
String sql="select bal from accounts where accno=?;
Int X= idbc Temp.queryForInt(sql, accno);
                                                        accno=?;
                                                        jdbcTemp.update(sql1,nbal,accno);
Double cbal=new Integer(X).double Value();
Return cbal:
                                                        Throw new InSufficientFundsException();
```

(No 1 in Training & Placement)

```
Jtcindia.xml
<br/>beans....>
     <context:annotation-config/>
     < tx:annotation-driver transaction manager="txManager"/>
     <bean id="dataSource"</pre>
              Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
               cproperty name="driverClassName" value="com.mysql.jdbc.Driver"/>
               cproperty name="url" value="jdbc:mysql://localhost/jtcindiadb"/>
               property name="username" value="root"/>
               cproperty name="password" value="somprakash"/>
     </bean>
     <bean id="txManager" class="org.springframework.orm.jpa.jpaTransactionManager"</p>
           autowire="by Type"/>
     <bean id="txManager"</pre>
           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"</pre>
               Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        </bean>
2. Configure the JpaTransactionManager by injecting EntityManagerFactory
        <br/>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"</pre>

Class="org.springframework.orm.hibernate3.LocalSessionFactoryBean"> </bean>

4. Configure the HibernateTransactionManager by injecting sessionFactory.

<bean id="txManager"</pre> 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.

33

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

<tx:annotation-driven/>

8. Inject HibernateTemplate into HibernateCustomerDAO

(No 1 in Training & Placement)

- 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;

 $@Transactional (propagation=propagation. REQUIRED\ NEW,\ isolation=lsolation. READ\ and\ respect to the propagation of the pr$

UNCOMMITED)

Public void addCustomer(customer Cust){

Hibernate Temp.save(cust);

}

@Transactional(propagation=propagation.REQUIRED, isolation=lsolation.READ COMMITED)

Public void update Customer (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;	*@Author: Som Prakash Rai
	*@Company: java Training Center
import org.hibenrate.LockMode;	*@ Visit : <u>www.jtcindia.org</u>
import org.springframework.beans.factory.	**/
accotation. Autowired;	@Transactional(propagation=propagation.REQUIRED,iso
import org.sprinsfamework.orm.hibernate3.Hibernate	lation=lsolation.REPEATABLE READ)
Template;	Public double getBalance(int accno) {
import org.springfreamework.transaction.annotation.*;	Account acc=htemp.load(Account.class,accno, lock
	mode.READ);
	Double cbal=acc.getbal():
X	Returen cbal;
	}}

@Transaction1	Htemp.update(acc);
Publil class HibernateAccountDAO implements	}else{
AccountDAO {	Throw new inSufficientFundsException();
@Autowired	}}}
HibenateTemplate htemp=null;	@Transactional(propagation=Propagation.
@Transactional)propagation=propagation.REQUIRES	REQUIRES NEW isolation=isolation.
NEW, isolation=lsolation.REPEATABLE READ)	REPEATABLE READ)

Spring PART 8

(No 1 in Training & Placement)

```
Public void funds Transfer(int saccno, int
Public void deposit(int accno, double amt) {
Account acc=htemp.load(Account.class, accno,
                                                     daccno, double amt) {
lockMode.READ);
                                                     Account
                                                     acc1=htemp.load(Account.class,daccno,
Acc.setBal(acc.getBal()+amt);
Htemp.update(acc);
                                                     LockMode.READ);
                                                     Acc1.setBal(acc1.getBal()+amt);
@Transactional(propagation=Propagation.REQUIRES
                                                     Htemp.update(acc1);
NEW, isolation=lsolation.REPEATABLE READ)
                                                     Account
Public void withdraw(int accno, double amt) {
                                                     acc2=htemp.load(Account.class,saccno,
Account acc=htemp.load(Account.class,accno, Lock
                                                     LockMode.READ):
mode.READ);
                                                     Double scbal=acc2.getBal();
Double cbal=acc.getBal():
                                                     If(scbal > = 5000 + amt)
If(cbal > = 5000 + amt)
                                                     Acc2.setbal(scbal-amt);
Acc.setBal(cbal-amt);
                                                     Htemp.update(acc2);
                                                     }else{
                                                     Throw new InSufficientFundsException();
```

```
Jtcindia.xml
<br/>beans....>
     <context:annotation-config/>
     < tx:annotation-driver transaction manager="txManager"/>
     <bean id="dataSource"</pre>
              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"</pre>
               Class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
               property name="dataSource" ref="dataSource"/>
               property name="mappingResources">
                    t>
                            <value>com/jtcindia/spring/hibernate/account.hbm.xml</value>
                    </list>
               </property>
               property name="hibernateProperties">
props >
                prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect
               prop key="hibernate.show sql>true</prop>
              </props>
        </props>
</bean>
       <bean id="hibernateTemp" class="org.springframework.orm.hibernate3.hibernateTemplate"</p>
Sutowire="constructor"/>
```

(No 1 in Training & Placement)

Using Annotation Support with jpa

1. Configure the EntityManagerFactory

<bean id="EntityManagerFactory"</pre>

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 ipaCustomerDAO.

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

5. Enable tx namespace in the spring configuration Document.

GROWTH LINBOLIND

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

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

<tx:annotation-driven/>

ABTNAS OENTER

- 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){

ipaTemp.persist(cust);

36

Spring PART 8

(No 1 in Training & Placement)

@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	@Transactional(propagation=propagation.
Package com.jtcindia.spring.jpa;	REQUIRES NEW isolation. REPEATABLE READ)
Import org.springframework.beans.factory. annotation.	Public double getBalance (int accno) {
Autowired;	Account
Import org.springframework.orm.jpa.jpaTemplate;	acc=jtemp.getReference(Account.class,accno);
Import org.springframework.transaction. annotation.	Double cbal=acc.getBal();
	Return cbal;

Spring PART 8

(No 1 in Training & Placement)

```
Propagation;
Import org.springframework.transaction.
annotation. Transactional;
                                                       @Transactionl(propagation=propagation.REZUIRES
                                                       NEW ISOLATION=isolation.REPEATABAL
*@Author: Som Prakash Rai
                                                       READ)
*@Company: java Training Center
                                                       Public void withdraw (int accno, double amt) {
*@ Visit
              : www.jtcindia.org
                                                       Account
**/
                                                       acc=jtemp.getReference(Account.class,accno);
@Transactional
                                                       Double cbal=acc.getBal();
Public class jpaAcconutDAO implements Account
                                                       If(cbal > = 5000 + amt)
DAO {
                                                       Acc.setbal(cbal-amt);
@Autowired
                                                       Jtemp.merge(acc);
jpaTemplate jtemp=null;
                                                       }else(
@Transactional(Propagation=Propagation.
                                                       Throw new insufficientFundsException();
REQUIRES NEW isolation = Isolation.REPEATABLE
READ)
Public void deposit(int accno, double amt) {
Account acc=itemp.getReference(Account.class,
Acc.setBal(acc.getBal()+amt);
Jtemp.merge(acc);
@Transactional(propagation=propagation, REQUIRES NI
Isolation=isolation.REPEATABLE READ)
Public void funds Transfer(int saccno, int daccno, double
Amt) {
Account acc1=jtemp.getReference (Account.class, daccno TH UNBOUND
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();
```

```
Jtcindia.xml

<br/>
<br/>
<br/>
<br/>
<context:annotation-config/>
<br/>
<tx:annotation-driver transaction manager="txManager"/>
<br/>
<br
```

(No 1 in Training & Placement)

```
Class="org.springframework.jpa.localEntityManagerFactoryBean">
              property name="persistenceUnitName" value="JTCINDIA PU"/>
              property name="ipaDialect">
                   <pean class="com.jtcindia.spring.jpa.JTCJpaDialect"/>
              </bean>
    <bean id="txManager" class="org.springframework.orm.jpa.jpaTransactionManager"</p>
          autowire="by Type"/>
    <bean id="jpaTemp" class="org.springframework.orm.jpa.jpaTemplate"</pre>
    <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
       <br/>bean id="DataSource"
               Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
       </bean>
2. Configure the idbcTransaction by injecting DataSource
       <br/>bean id="idbcTemp"
               Class="org.springframework.jdbc.core.jdbcTemplate"
              Autowire="constructor"/>
```

3. Configure the DataSourceTemplateManager by injecting, DataSource.

```
<bean id="txManager"</pre>
```

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>

39

Spring PART 8

(No 1 in Training & Placement)

```
</tx:advice>
```

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

```
<aop:config>
```

<aop:pointcut id="txPointcut"</pre>

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	

GROWTH UNBOUND

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();

l

Public double getBalance(int accno){

String sql="select bal from accounts where

accno=?

Int x=jdbc Temp.queryForint(sql, accno);

Double cbal=new Integer(x). doubvleValue();

Return cbal:

}

Public void withdraw (int accno, double amt) {

String sql="select bal from accounts where

accno=?;

Int x=idbcTemp.queryForint9sql, accno);

Double cbal=new integer(x). doubleValue();

If(cbal>=5000+amt){

Double nbal=cbal-amt;

String sql1="update accounts set bal=? where

Spring PART 8

(No 1 in Training & Placement)

```
accno=?":
Int x=idbcTemp.queryForInt(sql, accno);
Double cbal=new Integer(x).double value();
                                                   Jdbc temp.update(sql1, nbal,accno);
Double nbal=new integer(x).double value();
                                                   }else{
                                                   Throw newf insufficientFundsException();
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.queryForInt(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.queryForInt(sql1, daccno);
Dcbal=new Integer(y) doubaleValue();
System.out.println("after deposit"+dcbal);
Int x=idbcTemp.queryForln(sql1, saccno);
Double scbal=new lnteger(x). double value();
If (scbal > = 5000 + amt)
double snbal=scbal amt;
jdbcTemp.update(sql2,snbla, saccno);
Jtcindia.xml
<br/>beans....>
     <context:annotation-config/>
     < bean id="dataSource"
              Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
               cyroperty name="driverClassName" value="com.mysql.jdbc.Driver"/>
               cproperty name="url" value="jdbc:mysql://localhost/jtcindiadb"/>
               cproperty name="username" value="root"/>
               competty name="password" value="somprakash"/>
     </bean>
     <bean id="simplejdbcTemp" class="org.springframework.jdbc.core.jdbcTemplate"</p>
autowire="contructor"/>
     <bean id="txdManager"</pre>
Class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
Autowire="constructor"/>
     <bean id="adao" class="com.jtcindia.spring.jdbc.jdbcAccountDAO"/>
<tx:advice id="txAdvice"transaction manager="txManager">
```

Spring PART 8

Author: Som Prakash Rai

<tx:attributes>

(No 1 in Training & Placement)

Using schema support with Hibernate:

```
1. Configure the DataSource
       <bean id="DataSource"</pre>
               Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
       </bean>
2. Configure the sessionFactory by injecting DataSource, Hibernate properties, mapping Resource,
       <bean id="sessionFactory"</pre>
               Class="org.springframework.orm.hibernat3.LocalSessionFactoryBean">
         </bean>
3. Configure the HibernateTemplate by injecting, sessionFactory.
       <bean id="hibernateTemp"</pre>
               Class="org.springframework.orm.hibernate3.HibernateHibernateManager"
               Autowire="constructor"/>
4. configure the Hibernate Transaction Manager by injecting session Factory.
       <bean id="txManager"</pre>
                   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">
```

42

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

Author: Som Prakash Rai

Ex:

(No 1 in Training & Placement)

```
<tx: advice id="txAdvice" transaction-manager="txManager ">
       <tx:attributes>
                      name="addCustomer"
       <tx:method
                      Propagation="REQUIRES NEW"
                      Isolation="READ COMMITTED"/>
                      name="updateCustomer*;
       <tx:method
                      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 Hibernate Customer DAO.
Class hibernate CustomerDAO imp customerDAO.
       @Autowired
       hibernate Template hibernate Temp=null;
       Public void addCustomer(customer cust){
       hibernate Temp.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.nom.java	insufficient funds Exception. Java
Jtcindia.xml	
Hibernate AccountDAO.java	Public void funds Transfer(int saccno, int daccno, double
Package com.jtcindia.spring.hibernate;	amt){
Import org.hibernate.LockMode;	Account acc1=htemp.load(Account.class,daccno,
Import org.springframework.beans.factory.	LockModeREAD);
annotation Autowired	Acc1.setBal(acc1.getBal()+amt):

Import org.springframework.orm.hibernate3. HibernateTe

Public class Hibernate AccountDAO implements

AccountDAO { @Autowired

HibernateTemplate htemp=null;

A account hhm ious

in Cufficient Funds Expontion invo

Htemp.update9acc1);

Account acc2=htemp.load(account.class.saccno,

lockMode.READ);

Double scbal=acc2.getBal();

If(scbal > = 5000 + amt)Acc2.setBal(scbal amt);

Spring PART 8

(No 1 in Training & Placement)

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

```
Jtcindia.xml
<context:annotation-config/>
     < bean id="dataSource"
             Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
              property name="driverClassName" value="com.mysql.jdbc.Driver"/>
              cyroperty name="url" value="jdbc:mysql://localhost/jtcindiadb"/>
              roperty name="username" value="root"/>
              property name="password" value="somprakash"/>
    </bean>
    <bean id="sessionFactory"</pre>
               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>
                     st>
                </property>
                property name="hibernateProperties">
                   props>
                          <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                         prop key="hibernate.show sql">true
                          prop key="hibernate.hbm2ddl.auto">update
                   </bean>
<bean id="hibernateTemp" class="org.springframework.orm.hibernate3.HibernateTemplate"</p>
     Autowire="constructor"/>
<bean id="txManager"</pre>
     Class="org.springframework.orm.hibernate3.hibernateTransactionManager"
       Autowire="constructo"/>
<bean id="adao" class="com.itcindia.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"/>
```

44

(No 1 in Training & Placement)

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

Using schema Support with jpa

1. Configure the EntityManagerFactory

<bean id="EntityManagerFactory"</pre>

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"</pre>

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 />

45

Spring PART 8

(No 1 in Training & Placement)

<tx: advice id="txAdvice" transaction-manager="txManager"> <tx:attributes> <tx:method name="addCustomer" Propagation="REQUIRES NEW" Isolation="READ COMMITTED"/> name="updateCustomer*; <tx:method Propagation="REQUIRED" Isolation="READ UNCOMMITTED"/> </tx:attributes> </tx:advice> 8. Enable app 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.

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

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;	Double cbal=acc.getBal();
import org.springframework.beans.factory.	Return cbal;
annotation.Autowired;	Public void funds Transfer(int saccno, int daccno,

Spring PART 8

(No 1 in Training & Placement)

```
double amt) {
import
org.sprinfframework.jdbc.core.jdbc.Template;
                                                  Account
                                                  Acc1=jtemp.getReference(account.class,daccno);
                                                  Acc1.setBal(acc1.getBal()+amt);
*@Author: Som Prakash Rai
*@Company: java Training Center
                                                  Jtemp.merge9acc1);
               : www.jtcindia.org
*@ Visit
                                                  Account
**/
                                                  Acc2=jtemp.getReference9Account.class,saccno);
Public class jdbcAccountDAO implements
                                                 Double scbal=acc2.getBal();
AccountDAO {
                                                 If(scbal > = 5000 + amt)
                                                  Acc2.setBal9scbal-amt):
@Autowired
                                                  Ditemp.merge(acc2);
Jdbc Template jdbc Temp=null;
                                                  }else{
                                                  Throw new insufficient FundsException();
Public void deposit(int accno, double amt) {
Account
                                                 Public void withraw(int accno, double amt) {
acc=jtemp.getReference(Account.class,accno);
                                                  Account
                                                  acc=itemp.getReference(Account.class,accno);
Acc.setBal9acc.getBal()+amt);
Jtemp.merge(acc);
                                                 Double cbal=acc.getBal();
                                                 If 9cbal > = 5000 + amt
Public double getBalace(intaccno){
                                                  Acc.setBal(cbal-amt);
                                                 Jtemp.merge(acc);
Account
acc=jtemp.getReference(Account.class,accno);
                                                 }else{
Double cbal=itemp.getReference(Account.class,
                                                 Throw new inSufficientFundsException();
Jtcindia.xml
<br/>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"/>
           </bean>
     <bean id="tmManager" class="org.springframework.orm.jpa.jpaTransactionManager"</p>
                autowire="byType"/>
     <bean id="jpaTemp"Class="org.springframework.orm.jpa.jpaTemplate"</p>
                Autowire="constructor"/>
     <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="REOUIRED"/>
```

47

(No 1 in Training & Placement)

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	- 6

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	-