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 theMapping Related 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 see 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.HibernateException 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 exceptions in Hibernate.

Above Problems are solved as follows:

1) HibernateTemplate is provided which centralizes the Hibernate client code.

   Usage:

   a. hibernateTemp.save(cust);

2) In Spring Data Access, There is one root exception called DataAccessException 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 HibernateTemplate

1) Serializable save(Object)
2) void update(Object)
3) void update(Object ,LockMode)
4) void delete(Object)
5) void delete(Object ,LockMode)
6) void deleteAll(Collection)

7) Object load(Class , Serializable)
8) Object load(Class , Serializable ,LockMode)
9) List loadAll(Class)

10) List find(hql) - HQL
11) List find(hql,Object) - HQL
12) List find(hql,Object []) - HQL

13) List findByCriteria(DetachedCriteria) - QBC
14) List findByCriteria(DC,int ,int) - QBC

Lab60: Spring Data access with Hibernate

Working Steps:

1. Create the Java Project : Lab60

2. Add 21 Spring-5.2 Jars to Project Build Path.

3. Add 18 Hibernate 5.4 Jars to Project Build Path.

4. Add mysql-connector-java jar to Project Build Path.

5. Setup the Database

```
create database myspringdb;
use myspringdb;
```

6. Create a package called **com.coursecube.spring**

7. Write Spring Configuration Class and Enable with **@ComponentScan**

```
@Configuration
@ComponentScan(basePackages = {"com.coursecube.spring"})
public class JLCAppConfig {

}
```

8. Configure the DataSource Bean in Spring Configuration Class.

```
@Bean
public DriverManagerDataSource getDS() {

DriverManagerDataSource ds=new DriverManagerDataSource();

ds.setDriverClassName("com.mysql.jdbc.Driver");
ds.setUrl("jdbc:mysql://localhost:3306/myspringdb");
ds.setUsername("root");
ds.setPassword("srinivas");

return ds;
}
```

9. Configure SessionFactory Bean in Spring Configuration Class.

```
@Bean
public LocalSessionFactoryBean sessionFactory(DataSource dataSource) {

        LocalSessionFactoryBean factory = new LocalSessionFactoryBean();
        factory.setDataSource(dataSource); //1
        factory.setPackagesToScan("com.coursecube.spring"); //2

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

        factory.setHibernateProperties(props); //3

        return factory;
}
```

10. Configure HibernateTemplate Bean in Spring Configuration Class.

```
@Bean
public HibernateTemplate hTemp(SessionFactory sessionFactory) {
    return new HibernateTemplate(sessionFactory);
}
```

11. Configure SessionFactory Bean in Spring Configuration Class.

```
@Bean
public PlatformTransactionManager txManager(SessionFactory sessionFactory) {
        return new HibernateTransactionManager(sessionFactory);
}
```

12. Write the Persistence Entity called Customer for mycustomers table.

```
public class Customer {
private int cid;
private String cname;
private String email;
private long phone;
private String city;
...
}
```

13. Write the CustomerDAO interface.

```
public interface CustomerDAO {
    public void addCustomer(Customer cust);
    public void updateCustomer(Customer cust);
    public void deleteCustomer(int cid);
    public Customer getCustomerByCid(int cid);
}
```

14. Write the Implementation class for CustomerDAO interface and Override all the methods.
    Mark the CustomerDAOImpl class **@Repository**

```
@Repository("mycustDAO")
public class CustomerDAOImpl implements CustomerDAO {
...
@Autowired
HibernateTemplate hibernateTemp;
…
…
}
```

15. Write the Client Code :Lab60.java

Lab60: Files required

1. **Lab60.java**	2. **Customer.java**
3. **CustomerDAO.java**	4. **CustomerDAOImpl.java**
5. **JLCAppConfig.java**	

1. Lab60.java

```
package com.coursecube.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class Lab60 {
public static void main(String[] args) {
ApplicationContext ctx=new AnnotationConfigApplicationContext(JLCAppConfig.class);
System.out.println("Spring Container is Ready");

CustomerDAO custDAO=ctx.getBean("mycustDAO",CustomerDAO.class);

//1.Add Customer
Customer cust1=new Customer("SD","SD@jlc",123,"Blore");
custDAO.addCustomer(cust1);

//2.Update Customer
Customer cust2=new Customer(201,"hello","hello@jlc",55555,"Hyd");
custDAO.updateCustomer(cust2);

//3.Delete Customer
custDAO.deleteCustomer(103);

//4.getCustomer By Cid
Customer cust=custDAO.getCustomerByCid(1);
System.out.println(cust);

System.out.println("Done");
}
}
```

2. Customer.java

```
package com.coursecube.spring;

import javax.persistence.*;
/*
* @Author : Srinivas Dande
```

```
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Entity
@Table(name="mycustomers")
public class Customer {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(name="cid")
	private int cid;

	@Column(name="cname")
	private String cname;
	@Column(name="email")
	private String email;

	@Column(name="phone")
	private long phone;

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

	public Customer() {	}
	public Customer(String cname, String email, long phone, String city) {
		this.cname = cname;
		this.email = email;
		this.phone = phone;
		this.city = city;
	}
	public Customer(int cid, String cname, String email, long phone, String city) {
		this.cid = cid;
		this.cname = cname;
		this.email = email;
		this.phone = phone;
		this.city = city;
	}

	//Setters and Getters

	@Override
	public String toString() {
		return cid + ", " + cname + ", " + email + ", " + phone + ", " + city;
	}
}
```

3. CustomerDAO.java

```
package com.coursecube.spring;

/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public interface CustomerDAO {
		public void addCustomer(Customer cust);
		public void updateCustomer(Customer cust);
		public void deleteCustomer(int cid);
		public Customer getCustomerByCid(int cid);
}
```

4. CustomerDAOImpl.java

```
package com.coursecube.spring;

import org.hibernate.LockMode;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Repository("mycustDAO")
@Transactional
public class CustomerDAOImpl implements CustomerDAO {

	@Autowired
	HibernateTemplate hibernateTemp;

	@Override
	public void addCustomer(Customer cust) {
		hibernateTemp.save(cust);
	}
	@Override
	public void updateCustomer(Customer cust) {
		hibernateTemp.update(cust);
	}
	@Override
	public Customer getCustomerByCid(int cid) {
		Customer cust= hibernateTemp.load(Customer.class, cid, LockMode.READ);
		return cust;
	}
```

```
	@Override
	public void deleteCustomer(int cid) {
		Customer cust = hibernateTemp.get(Customer.class, cid);
		if (cust != null)
			hibernateTemp.delete(cust);
	}
}
```

5. JLCAppConfig.java

```
package com.coursecube.spring;

import java.util.Properties;
import javax.sql.DataSource;
import org.hibernate.SessionFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.orm.hibernate5.HibernateTransactionManager;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration
@ComponentScan(basePackages = {"com.coursecube.spring"})
@EnableTransactionManagement
public class JLCAppConfig {

@Bean
public DriverManagerDataSource getDS() {
DriverManagerDataSource ds=new DriverManagerDataSource();
//ds.setDriverClassName("com.mysql.jdbc.Driver");
ds.setUrl("jdbc:mysql://localhost:3306/myspringdb");
ds.setUsername("root");
ds.setPassword("srinivas");

return ds;
}
@Bean
public LocalSessionFactoryBean sessionFactory(DataSource dataSource) {

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

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

return factory;
}
@Bean
public HibernateTemplate hTemp(SessionFactory sessionFactory) {
return new HibernateTemplate(sessionFactory);
}
@Bean
public PlatformTransactionManager txManager(SessionFactory sessionFactory) {
return new HibernateTransactionManager(sessionFactory);
}
}
```

Lab61: Files required

1. Lab61.java	**Updated in Lab61**
2. Customer.java	Same as Lab60
3. CustomerDAO.java	**Updated in Lab61**
4. CustomerDAOImpl.java	**Updated in Lab61**
5. JLCAppConfig.java	Same as Lab60

1. Lab61.java

```
package com.coursecube.spring;

import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class Lab61 {
public static void main(String[] args) {
ApplicationContext ctx=new AnnotationConfigApplicationContext(JLCAppConfig.class);
System.out.println("Spring Container is Ready");
CustomerDAO custDAO=ctx.getBean("mycustDAO",CustomerDAO.class);

System.out.println("-----1. getAllCustomers----");
List<Customer> list=custDAO.getAllCustomers();
list.forEach(cust -> System.out.println(cust));

System.out.println("-----2. getCustomerByCity----");
list=custDAO.getCustomerByCity("Hyd");
list.forEach(cust -> System.out.println(cust));
```

```
System.out.println("-----3. getCustomerByEmail----");
list=custDAO.getCustomerByEmail("sri@jlc");
list.forEach(cust -> System.out.println(cust));

System.out.println("-----4. getCustomerByPhone----");
list=custDAO.getCustomerByPhone(123);
list.forEach(cust -> System.out.println(cust));

System.out.println("-----5. getCustomerByEmailAndPhone----");
list=custDAO.getCustomerByEmailAndPhone("sri@jlc",123);
list.forEach(cust -> System.out.println(cust));

System.out.println("-----6. getCustomerByEmailOrPhone----");
list=custDAO.getCustomerByEmailOrPhone("sri@jlc",123);
list.forEach(cust -> System.out.println(cust));

System.out.println("Done");
}
}
```

3. CustomerDAO.java

```
package com.coursecube.spring;

import java.util.List;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public interface CustomerDAO {
        public List<Customer> getAllCustomers();
        public List<Customer> getCustomerByCity(String city);
        public List<Customer> getCustomerByEmail(String email);
        public List<Customer> getCustomerByPhone(long phone);
        public List<Customer> getCustomerByEmailAndPhone(String email,long phone);
        public List<Customer>  getCustomerByEmailOrPhone(String email,long phone) ;
}
```

4. CustomerDAOImpl.java

```
package com.coursecube.spring;

import java.util.List;
import org.hibernate.criterion.DetachedCriteria;
import org.hibernate.criterion.Restrictions;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate5.HibernateTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
```

```
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Repository("mycustDAO")
@Transactional
public class CustomerDAOImpl implements CustomerDAO {

@Autowired
HibernateTemplate hibernateTemp;

@Override
public List<Customer> getAllCustomers() {
String hql="from Customer cust";
List<Customer> mylist= (List<Customer>) hibernateTemp.find(hql);
return mylist;
}
@Override
public List<Customer> getCustomerByCity(String city) {
DetachedCriteria dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.eq("city", city));
List<Customer> mylist= (List<Customer>) hibernateTemp.findByCriteria(dc);
return mylist;
}
@Override
public List<Customer> getCustomerByEmail(String email) {
DetachedCriteria dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.eq("email", email));
List<Customer> mylist= (List<Customer>) hibernateTemp.findByCriteria(dc);
return mylist;
}
@Override
public List<Customer> getCustomerByPhone(long phone) {
DetachedCriteria dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.eq("phone", phone));
List<Customer> mylist= (List<Customer>) hibernateTemp.findByCriteria(dc);
return mylist;
}

@Override
public List<Customer> getCustomerByEmailAndPhone(String email, long phone) {
DetachedCriteria dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.and(Restrictions.eq("email", email),Restrictions.eq("phone", phone)));
List<Customer> mylist= (List<Customer>) hibernateTemp.findByCriteria(dc);
return mylist;
}
```

```
@Override
public List<Customer> getCustomerByEmailOrPhone(String email, long phone) {
DetachedCriteria dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.or(Restrictions.eq("email", email),Restrictions.eq("phone", phone)));
List<Customer> mylist= (List<Customer>) hibernateTemp.findByCriteria(dc);
return mylist;
}
}
```

Lab62: Files required

1. Lab62.java	**Updated in Lab62**
2. Customer.java	Same as Lab60
3. CustomerDAO.java	**Updated in Lab62**
4. CustomerDAOImpl.java	**Updated in Lab62**
5. JLCAppConfig.java	Same as Lab60

1. Lab62.java

```
package com.coursecube.spring;

import java.math.BigInteger;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public class Lab62 {
public static void main(String[] args) {
ApplicationContext ctx=new AnnotationConfigApplicationContext(JLCAppConfig.class);
System.out.println("Spring Container is Ready");

CustomerDAO custDAO=ctx.getBean("mycustDAO",CustomerDAO.class);

BigInteger big=custDAO.getCustomerCount();
System.out.println("All Count : "+big.intValue());

big=custDAO.getCustomerCountByCity("Hyd");
System.out.println("Hyd Count : "+big.intValue());

big=custDAO.getCustomerCountByCity("Blore");
System.out.println("Blore Count : "+big.intValue());

String city=custDAO.getCityByEmail("hello@jlc");
System.out.println(city);
```

```
long phone=custDAO.getPhoneByEmail("hello@jlc");
System.out.println(phone);

int cid=custDAO.getCidByPhone(55555);
System.out.println(cid);
}
}
```

3. CustomerDAO.java

```
package com.coursecube.spring;

import java.math.BigInteger;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
public interface CustomerDAO {
        public BigInteger getCustomerCount();
        public BigInteger getCustomerCountByCity(String city);
        public String getCityByEmail(String email);
        public long  getPhoneByEmail(String email);
        public int  getCidByPhone(long phone);
}
```

4. CustomerDAOImpl.java

```
package com.coursecube.spring;

import java.math.BigInteger;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;
/*
* @Author : Srinivas Dande
* @Company : CourseCube
* @Website : www.coursecube.com
* */
@Repository("mycustDAO")
@Transactional
public class CustomerDAOImpl implements CustomerDAO {

        @Autowired
        SessionFactory sessionFactory;
```

```
	@Override
	public BigInteger getCustomerCount() {
		Session session=sessionFactory.openSession();
		String sql="select count(*) from mycustomers";

		BigInteger count=(BigInteger)session.createNativeQuery(sql).uniqueResult();
		return count;
	}

	@Override
	public BigInteger getCustomerCountByCity(String city) {
		String sql="select count(*) from mycustomers where city=?";

		return (BigInteger)sessionFactory.openSession()
		.createNativeQuery(sql)
		.setParameter(1,city)
		.uniqueResult();
	}
	@Override
	public String getCityByEmail(String email) {
		String sql="select city from mycustomers where email=?";

		return (String)sessionFactory.openSession()
		.createNativeQuery(sql)
		.setParameter(1,email)
		.uniqueResult();
	}
	@Override
	public long getPhoneByEmail(String email) {
		String sql="select phone from mycustomers where email=?";

		String str= (String)sessionFactory.openSession()
		.createNativeQuery(sql)
		.setParameter(1,email)
		.uniqueResult();
		return Long.parseLong(str);
	}
	@Override
	public int getCidByPhone(long phone) {
		String sql="select cid from mycustomers where phone=?";

		return (Integer)sessionFactory.openSession()
		.createNativeQuery(sql)
		.setParameter(1,phone)
		.uniqueResult();
	}
}
```

Exploring Query By Criteria(QBC):

A) Consider the following table:

```
create table mycustomers(
cid int primary key,
cname char(10),
email char(10),
phone char(10),
city char(10),
atype char(2),
bal sdouble,
status char(10)  );
```

B) Inset some Sample Records

C) Practice the Following Queries:

1) Display All the Customer Records

a. Without Pagination

```
dc=DetachedCriteria.forClass(Customer.class);
htemp.findByCriteria(dc);
```

b. With Pagination

```
dc=DetachedCriteria.forClass(Customer.class);
htemp.findByCriteria(dc,0,5);
```

2) Display Customers By Given CIty

```
dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.eq("city", city));
htemp.findByCriteria(dc);
```

3) Display Customers whose balance > 20,000.

```
dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.gt("bal", 20000));
htemp.findByCriteria(dc);
```

4) Display Customers whose balance <=5,000.

```
dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.le("bal", 500));
htemp.findByCriteria(dc);
```

5) Display Customers whose cname starts with 'Sri'.

```
dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.like("cname", "sri%"));
htemp.findByCriteria(dc);
```

6) Display Customers whose balance between 5,000 and 25,000

```
dc=DetachedCriteria.forClass(Customer.class);
dc.add(Restrictions.between("bal",5000,25,000));
htemp.findByCriteria(dc);
```

7) Display Customers whose bal between 5,000 and 25,000 and City is Blore.

```
dc=DetachedCriteria.forClass(Customer.class);
dc.add(
Restrictions.and(Restrictions.between("bal",50,100) ,
Restrictions.eq("city", city) )   )
htemp.findByCriteria(dc);
```

8) Display Customers whose bal between 5,000 and 25,000 and City is Blore and Status of Customer is Inactive.

```
dc=DetachedCriteria.forClass(Customer.class);
dc.add(
Restrictions.and(
Restrictions.eq("status", "Inactive") ,
Restrictions.and(
        Restrictions.between("bal",50,100) ,
        Restrictions.eq("city", city) )
)
htemp.findByCriteria(dc);
```