SPRING DECLARATIVE TRANSACTION MANAGEMENT

Declarative transaction management approach allows you to manage the transaction with the help of configuration instead of hard coding in your source code. This means that you can separate transaction management from the business code. You only use annotations or XML based configuration to manage the transactions. The bean configuration will specify the methods to be transactional. Here are the steps associated with declarative transaction:

- We use <tx:advice /> tag, which creates a transaction-handling advice and same time we define a pointcut that matches all methods we wish to make transactional and reference the transactional advice.
- If a method name has been included in the transactional configuration then created advice will begin the transaction before calling the method.
- Target method will be executed in a *try / catch* block.
- If the method finishes normally, the AOP advice commits the transaction successfully otherwise it performs a rollback.

Let us see how above mentioned steps work but before we begin, it is important to have at least two database tables on which we can perform various CRUD operations with the help of transactions. Let us take **Student** table, which can be created in MySQL TEST database with the following DDL:

```
CREATE TABLE Student(
   ID   INT NOT NULL AUTO_INCREMENT,
   NAME VARCHAR(20) NOT NULL,
   AGE  INT NOT NULL,
   PRIMARY KEY (ID)
);
```

Second table is **Marks** in which we will maintain marks for students based on years. Here **SID** is the foreign key for Student table.

```
CREATE TABLE Marks(
   SID INT NOT NULL,
   MARKS  INT NOT NULL,
   YEAR   INT NOT NULL
);
```

Now let us write our Spring JDBC application which will implement simple operations on Student and Marks tables. Let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step	Description
1	Create a project with a name *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project.
2	Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter.
3	Add other required libraries *mysql-connector-java.jar*, *aopalliance-x.y.jar*, *org.springframework.jdbc.jar*, and *org.springframework.transaction.jar* in the project. You can download required libraries if you do not have them already.
4	Create DAO interface *StudentDAO* and list down all the required methods. Though it is not required and you can directly write *StudentJDBCTemplate* class, but as a good practice, let's do it.

5	Create other required Java classes *StudentMarks*, *StudentMarksMapper*, *StudentJDBCTemplate* and *MainApp* under the *com.tutorialspoint* package. You can create rest of the POJO classes if required.
6	Make sure you already created **Student** and **Marks** tables in TEST database. Also make sure your MySQL server is working fine and you have read/write access on the database using the given username and password.
7	Create Beans configuration file *Beans.xml* under the **src** folder.
8	The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Following is the content of the Data Access Object interface file **StudentDAO.java**:

```
package com.tutorialspoint;

import java.util.List;
import javax.sql.DataSource;

public interface StudentDAO {
   /**
    * This is the method to be used to initialize
    * database resources ie. connection.
    */
   public void setDataSource(DataSource ds);
   /**
    * This is the method to be used to create
    * a record in the Student and Marks tables.
    */
   public void create(String name, Integer age, Integer marks, Integer year);
   /**
    * This is the method to be used to list down
    * all the records from the Student and Marks tables.
    */
   public List<StudentMarks> listStudents();
}
```

Following is the content of the **StudentMarks.java** file:

```
package com.tutorialspoint;

public class StudentMarks {
   private Integer age;
   private String name;
   private Integer id;
   private Integer marks;
   private Integer year;
   private Integer sid;

   public void setAge(Integer age) {
      this.age = age;
   }
   public Integer getAge() {
      return age;
   }

   public void setName(String name) {
      this.name = name;
   }
   public String getName() {
      return name;
   }

   public void setId(Integer id) {
      this.id = id;
   }
```

```
   public Integer getId() {
      return id;
   }
   public void setMarks(Integer marks) {
      this.marks = marks;
   }
   public Integer getMarks() {
      return marks;
   }

   public void setYear(Integer year) {
      this.year = year;
   }
   public Integer getYear() {
      return year;
   }

   public void setSid(Integer sid) {
      this.sid = sid;
   }
   public Integer getSid() {
      return sid;
   }
}
```

Following is the content of the **StudentMarksMapper.java** file:

```
package com.tutorialspoint;

import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;

public class StudentMarksMapper implements RowMapper<StudentMarks> {
   public StudentMarks mapRow(ResultSet rs, int rowNum) throws SQLException {

      StudentMarks studentMarks = new StudentMarks();

      studentMarks.setId(rs.getInt("id"));
      studentMarks.setName(rs.getString("name"));
      studentMarks.setAge(rs.getInt("age"));
      studentMarks.setSid(rs.getInt("sid"));
      studentMarks.setMarks(rs.getInt("marks"));
      studentMarks.setYear(rs.getInt("year"));

      return studentMarks;
   }
}
```

Following is the implementation class file **StudentJDBCTemplate.java** for the defined DAO interface StudentDAO:

```
package com.tutorialspoint;

import java.util.List;
import javax.sql.DataSource;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.JdbcTemplate;

public class StudentJDBCTemplate implements StudentDAO{
   private JdbcTemplate jdbcTemplateObject;

   public void setDataSource(DataSource dataSource) {
      this.jdbcTemplateObject = new JdbcTemplate(dataSource);
   }

   public void create(String name, Integer age, Integer marks, Integer year){
```

```
      try {
         String SQL1 = "insert into Student (name, age) values (?, ?)";
         jdbcTemplateObject.update( SQL1, name, age);

         // Get the latest student id to be used in Marks table
         String SQL2 = "select max(id) from Student";
         int sid = jdbcTemplateObject.queryForInt( SQL2 );

         String SQL3 = "insert into Marks(sid, marks, year) " +
                       "values (?, ?, ?)";
         jdbcTemplateObject.update( SQL3, sid, marks, year);

         System.out.println("Created Name = " + name + ", Age = " + age);
         // to simulate the exception.
         throw new RuntimeException("simulate Error condition") ;
      } catch (DataAccessException e) {
         System.out.println("Error in creating record, rolling back");
         throw e;
      }
   }

   public List<StudentMarks> listStudents() {
      String SQL = "select * from Student, Marks where Student.id=Marks.sid";

      List <StudentMarks> studentMarks=jdbcTemplateObject.query(SQL,
      new StudentMarksMapper());
      return studentMarks;
   }
}
```

Now let us move with the main application file **MainApp.java**, which is as follows:

```
package com.tutorialspoint;
import java.util.List;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");

      StudentDAO studentJDBCTemplate =
      (StudentDAO)context.getBean("studentJDBCTemplate");

      System.out.println("------Records creation--------" );
      studentJDBCTemplate.create("Zara", 11, 99, 2010);
      studentJDBCTemplate.create("Nuha", 20, 97, 2010);
      studentJDBCTemplate.create("Ayan", 25, 100, 2011);

      System.out.println("------Listing all the records--------" );
      List<StudentMarks> studentMarks = studentJDBCTemplate.listStudents();
      for (StudentMarks record : studentMarks) {
         System.out.print("ID : " + record.getId() );
         System.out.print(", Name : " + record.getName() );
         System.out.print(", Marks : " + record.getMarks());
         System.out.print(", Year : " + record.getYear());
         System.out.println(", Age : " + record.getAge());
      }
   }
}
```

Following is the configuration file **Beans.xml**:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:tx="http://www.springframework.org/schema/tx"
```

```
   xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/tx
   http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
   http://www.springframework.org/schema/aop
   http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

   <!-- Initialization for data source -->
   <bean
      >
      <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
      <property name="url" value="jdbc:mysql://localhost:3306/TEST"/>
      <property name="username" value="root"/>
      <property name="password" value="cohondob"/>
   </bean>

   <tx:advice >
      <tx:attributes>
      <tx:method name="create"/>
      </tx:attributes>
   </tx:advice>

   <aop:config>
      <aop:pointcut
      expression="execution(* com.tutorialspoint.StudentJDBCTemplate.create(..))"/>
      <aop:advisor advice-ref="txAdvice" pointcut-ref="createOperation"/>
   </aop:config>

   <!-- Initialization for TransactionManager -->
   <bean
   >
      <property name="dataSource"  ref="dataSource" />
   </bean>

   <!-- Definition for studentJDBCTemplate bean -->
   <bean
   >
      <property name="dataSource"  ref="dataSource" />
   </bean>

</beans>
```

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following exception will be raised. In this case transaction will be rolled back and no record will be created in the database table.

```
------Records creation--------
Created Name = Zara, Age = 11
Exception in thread "main" java.lang.RuntimeException: simulate Error condition
```

You can try above example after removing exception, and in this case it should commit the transaction and you should see a record in the database.