SPRING PROGRAMMATIC TRANSACTION MANAGEMENT

Programmatic transaction management approach allows you to manage the transaction with the help of programming in your source code. That gives you extreme flexibility, but it is difficult to maintain.

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
);
```

Let us use *PlatformTransactionManager* directly to implement programmatic approach to implement transactions. To start a new transaction you need to have a instance of *TransactionDefinition* with the appropriate transaction attributes. For this example we will simply create an instance of *DefaultTransactionDefinition* to use the default transaction attributes.

Once the TransactionDefinition is created, you can start your transaction by calling *getTransaction* method, which returns an instance of *TransactionStatus*. The *TransactionStatus* objects helps in tracking the current status of the transaction and finally, if everything goes fine, you can use *commit* method of *PlatformTransactionManager* to commit the transaction, otherwise you can use *rollback* to rollback the complete operation.

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 Spring JDBC specific latest libraries **mysql-connector-java.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 give 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;
import org.springframework.transaction.PlatformTransactionManager;
import org.springframework.transaction.TransactionDefinition;
import org.springframework.transaction.TransactionStatus;
import org.springframework.transaction.support.DefaultTransactionDefinition;

public class StudentJDBCTemplate implements StudentDAO {
   private DataSource dataSource;
   private JdbcTemplate jdbcTemplateObject;
   private PlatformTransactionManager transactionManager;

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

```
   public void setTransactionManager(
      PlatformTransactionManager transactionManager) {
      this.transactionManager = transactionManager;
   }

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

      TransactionDefinition def = new DefaultTransactionDefinition();
      TransactionStatus status = transactionManager.getTransaction(def);

      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);
         transactionManager.commit(status);
      } catch (DataAccessException e) {
         System.out.println("Error in creating record, rolling back");
         transactionManager.rollback(status);
         throw e;
      }
      return;
   }

   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;
import com.tutorialspoint.StudentJDBCTemplate;

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

      StudentJDBCTemplate studentJDBCTemplate =
      (StudentJDBCTemplate)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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-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="password"/>
   </bean>

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

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

```
------Records creation--------
Created Name = Zara, Age = 11
Created Name = Nuha, Age = 20
Created Name = Ayan, Age = 25
------Listing all the records--------
ID : 1, Name : Zara, Marks : 99, Year : 2010, Age : 11
ID : 2, Name : Nuha, Marks : 97, Year : 2010, Age : 20
ID : 3, Name : Ayan, Marks : 100, Year : 2011, Age : 25
```