Spring JDBC Example

To understand the concepts related to Spring JDBC framework with JdbcTemplate class, let us write a simple example, which will implement all the CRUD operations on the following Student table.

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

Before proceeding, let us have a working Eclipse IDE in place and take the following steps to create a Spring application −

Steps	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 *Student*, *StudentMapper*, *StudentJDBCTemplate* and *MainApp* under the *com.tutorialspoint* package.
6	Make sure you already created **Student** table 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 table.
   */
   public void create(String name, Integer age);
```

```
   /**
      * This is the method to be used to list down
      * a record from the Student table corresponding
      * to a passed student id.
   */
   public Student getStudent(Integer id);

   /**
      * This is the method to be used to list down
      * all the records from the Student table.
   */
   public List<Student> listStudents();

   /**
      * This is the method to be used to delete
      * a record from the Student table corresponding
      * to a passed student id.
   */
   public void delete(Integer id);

   /**
      * This is the method to be used to update
      * a record into the Student table.
   */
   public void update(Integer id, Integer age);
}
```

Following is the content of the **Student.java** file

```
package com.tutorialspoint;

public class Student {
   private Integer age;
   private String name;
   private Integer id;

   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;
   }
}
```

Following is the content of the **StudentMapper.java** file

```
package com.tutorialspoint;

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

public class StudentMapper implements RowMapper<Student> {
   public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
      Student student = new Student();
      student.setId(rs.getInt("id"));
      student.setName(rs.getString("name"));
      student.setAge(rs.getInt("age"));

      return student;
   }
}
```

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.jdbc.core.JdbcTemplate;

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

   public void setDataSource(DataSource dataSource) {
      this.dataSource = dataSource;
```

```
      this.jdbcTemplateObject = new JdbcTemplate(dataSource);
   }
   public void create(String name, Integer age) {
      String SQL = "insert into Student (name, age) values (?, ?)";
      jdbcTemplateObject.update( SQL, name, age);
      System.out.println("Created Record Name = " + name + " Age = " + age)
      return;
   }
   public Student getStudent(Integer id) {
      String SQL = "select * from Student where id = ?";
      Student student = jdbcTemplateObject.queryForObject(SQL,
         new Object[]{id}, new StudentMapper());

      return student;
   }
   public List<Student> listStudents() {
      String SQL = "select * from Student";
      List <Student> students = jdbcTemplateObject.query(SQL, new StudentMa
      return students;
   }
   public void delete(Integer id) {
      String SQL = "delete from Student where id = ?";
      jdbcTemplateObject.update(SQL, id);
      System.out.println("Deleted Record with ID = " + id );
      return;
   }
   public void update(Integer id, Integer age){
      String SQL = "update Student set age = ? where id = ?";
      jdbcTemplateObject.update(SQL, age, id);
      System.out.println("Updated Record with ID = " + id );
      return;
   }
}
```

Following is the content of the **MainApp.java** file

```
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) {
```

```
   public static void main(String[] args) {
      ApplicationContext context = new ClassPathXmlApplicationContext("Bean

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

      System.out.println("------Records Creation--------" );
      studentJDBCTemplate.create("Zara", 11);
      studentJDBCTemplate.create("Nuha", 2);
      studentJDBCTemplate.create("Ayan", 15);


      System.out.println("------Listing Multiple Records--------" );
      List<Student> students = studentJDBCTemplate.listStudents();

      for (Student record : students) {
         System.out.print("ID : " + record.getId() );
         System.out.print(", Name : " + record.getName() );
         System.out.println(", Age : " + record.getAge());
      }

      System.out.println("----Updating Record with ID = 2 -----" );
      studentJDBCTemplate.update(2, 20);

      System.out.println("----Listing Record with ID = 2 -----" );
      Student student = studentJDBCTemplate.getStudent(2);
      System.out.print("ID : " + student.getId() );
      System.out.print(", Name : " + student.getName() );
      System.out.println(", Age : " + student.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 id="dataSource"
      class = "org.springframework.jdbc.datasource.DriverManagerDataSource"
      <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>

   <!-- Definition for studentJDBCTemplate bean -->
   <bean id = "studentJDBCTemplate"
      class = "com.tutorialspoint.StudentJDBCTemplate">
      <property name = "dataSource" ref = "dataSource" />
   </bean>

</beans>
```

Once you are done creating the source and bean configuration files, let us run the application. If everything is fine with your application, it will print the following message −

```
------Records Creation--------
Created Record Name = Zara Age = 11
Created Record Name = Nuha Age = 2
Created Record Name = Ayan Age = 15
------Listing Multiple Records--------
ID : 1, Name : Zara, Age : 11
ID : 2, Name : Nuha, Age : 2
ID : 3, Name : Ayan, Age : 15
----Updating Record with ID = 2 -----
Updated Record with ID = 2
----Listing Record with ID = 2 -----
ID : 2, Name : Nuha, Age : 20
```

You can try and delete the operation yourself, which we have not used in the example, but now you have one working application based on Spring JDBC framework, which you can extend to add sophisticated functionality based on your project requirements. There are other approaches to access the database where you will use **NamedParameterJdbcTemplate** and **SimpleJdbcTemplate** classes, so if you are interested in learning these classes then kindly check the reference manual for Spring Framework.