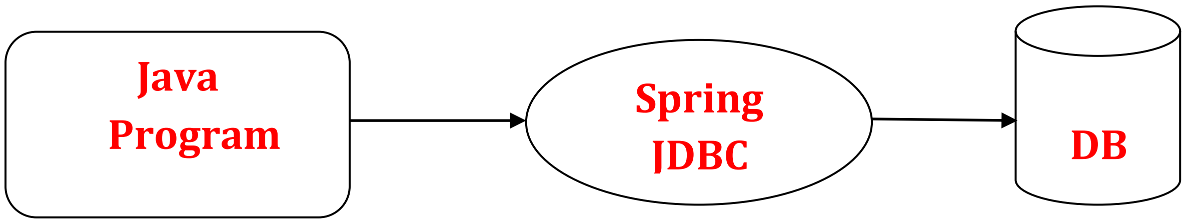
NAME : ROHIT KUMAR PANDEY

SUBJECT : SPRING JDBC

Spring JDBC is a powerful mechanism to connect to the database and execute SQL queries.



JDBC is an API to perform operation with database.

Problems of JDBC

1) Every time we need to use JDBC we need to
   - create connection,
   - create statement
   - create query
   - execute statement
   - close the connection

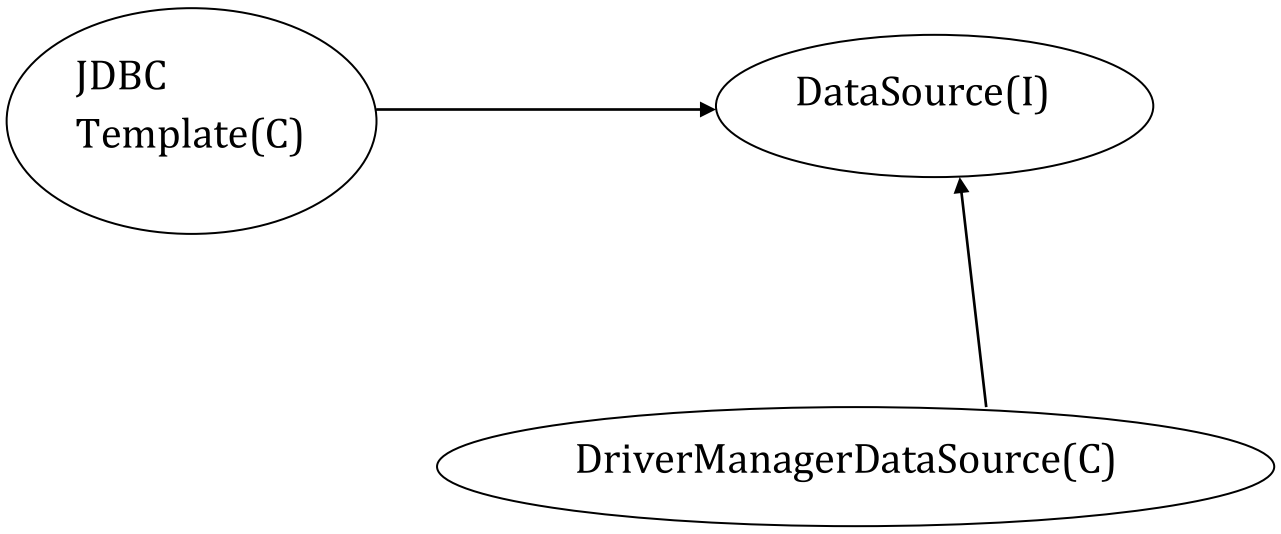
Doing this everytime,we need to write a lot of code.

2) Exception Handling problem: SQLException is a checked exception,so we need to write all the codes in try-catch block .
3) Repeating all these codes from one to another database logic is a time consuming task.

Solutions of above problem are provided by Spring JDBC.

Spring JDBC provides class *JdbcTemplate* which has all the important methods to perform operation with database.

- ✓ Jdbc template class allows us to fire queries for insert,delete,update,select etc.
- ✓ But Jdbc template class requires object of DataSource. Since DataSource is an Interface so we canot create object of it directly.
- ✓ DriverManagerDataSource is an implementation class of DataSource which will help us to get the object of DataSource.
- ✓ DriverManagerDataSource class has all the information about connection with DB like url,password,username etc.



Methods of JdbcTempltae Class

Let's see the methods of spring JdbcTemplate class.

- **public int update(String query) :** is used to insert, update and delete records.

- **public int update(String query,Object... args)** : is used to insert, update and delete records using PreparedStatement using given arguments.

- **public void execute(String query) :** is used to execute DDL query.

- **public List query(String sql, RowMapper rse) :** is used to fetch records using RowMapper.

RowMapper : **RowMapper** interface allows to map a row of the relations with the instance of user-defined class. It iterates the ResultSet internally and adds it into the collection. So we don't need to write a lot of code to fetch the records as ResultSet.

Method of RowMapper interface

It defines only one method mapRow that accepts ResultSet instance and int as the parameter list. Syntax of the method is given below:

public T mapRow(ResultSet rs, **int** rowNumber)**throws** SQLExcepti
on

Practical

- Create a maven project with following archtype

```
maven-archtype-quickstart
```

- Add the following dependencies in the pom.xml file

```xml
<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>5.2.3.RELEASE</version>   →  for spring core
</dependency>


<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.2.3.RELEASE</version>
</dependency>                               →  for spring context

<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.2.3.RELEASE</version>
</dependency>                               →  for spring jdbc


<dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.20</version>
</dependency>                         →  sql connector
```

Suppose we want to store the details of student in the DB using spring jdbc.

Student.java

```java
package com.Spring.jdbc.Entities;
public class Student {
      private int id;
      private String name;
      private String city;
      public Student(int id, String name, String city) {
            super();
            this.id = id;
            this.name = name;
            this.city = city;
      }
      public Student() {
            super();
      }
```

```java
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
}
```

We need the JdbcTemplate class object. Which require the DataSource which will be provided by DriverManagerDataSource. We can manually create object or we can use spring core to give bean.
Here we are configuring spring core to give us the object of JdbcTemplate class.

Config.java

```xml
<bean class="org.springframework.jdbc.datasource.DriverManagerDataSource" name="ds">
	<property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
	<property name="url"value="jdbc:mysql://localhost:3306/springjdbc"></property>
	<property name="username" value="root"></property>
	<property name="password" value="76448"></property>
</bean>

<bean class="org.springframework.jdbc.core.JdbcTemplate" name="jdbcTemplate"  >
	<property name="dataSource">
		<ref bean="ds"/>
	</property>
</bean>

<bean class="com.Spring.jdbc.Dao.StudentDao" name="studentDao">
	<property name="temp" ref="jdbcTemplate"/>
</bean>
```

Creating datasource object

Injecting datasource object in JdbcTemplate

Creating StudentDao object which takes JdbcTemplate object

StudentDao.java

```
package com.Spring.jdbc.Dao;
import java.util.List;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.RowMapperResultSetExtractor;

import com.Spring.jdbc.Entities.Student;
public class StudentDao
{
	private JdbcTemplate temp;
	public JdbcTemplate getTemp() {
		return temp;
	}
	public void setTemp(JdbcTemplate temp) {
		this.temp = temp;
	}
	public int insert(Student student)
	{
		String query="insert into student values(?,?,?)";
		int
res=this.temp.update(query,student.getId(),student.getName(),student.getCity(
));
		return res;
	}
	public int change(Student student)
	{
		String query="update student set name=?,city=? where id=?";
		int
res=this.temp.update(query,student.getName(),student.getCity(),student.getId(
));
		return res;
	}
	public int delete(int id)
	{
		String query="delete from student where id=?";
		int res=this.temp.update(query,id);
		return res;
	}
	public Student getStudent(int id)
	{
		String query="select * from student where id=?";
		RowMapper<Student> rowMapper=new RowMapperImpl();
		Student student=this.temp.queryForObject(query, rowMapper,id);
		return student;
	}
	public List<Student> getAllStudent()
	{
		String query="select * from student ";
		List<Student> allStudent=this.temp.query(query, new
RowMapperImpl());
		return allStudent;
	}
}
```

For getting single record we need to use queryForObject() method

For getting multiple record we need to use query () method

App.java

```
package com.Spring.jdbc;

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

import com.Spring.jdbc.Dao.StudentDao;
import com.Spring.jdbc.Entities.Student;

public class App {
	public static void main(String[] args) {
		System.out.println("My program started");

		ApplicationContext context = new
ClassPathXmlApplicationContext("com/Spring/jdbc/config.xml");
		StudentDao dao = context.getBean("studentDao", StudentDao.class);

//		Inserting an element into DB
//*****************************************
//		Student student = new Student();
//		student.setId(102);
//		student.setName("Md Ali");
//		student.setCity("Srinagar");
//          int result=dao.insert(student);
//          System.out.println(result);


//          Updating an element from DB
//****************************************
//          Student student2=new Student();
//          student2.setId(103);
//          student2.setName("Jocker");
//          student2.setCity("Itarsi");
//          int result = dao.change(student2);
//          System.out.println(result);

//          Deleting an element from DB
//****************************************

//          int result=dao.delete(101);
//          System.out.println(result);

//          Selecting single data from DB
//****************************************
//          Student student=dao.getStudent(104);
//          System.out.println(student);


//          Selecting multiple data(List) from DB
//****************************************

            List<Student> students=dao.getAllStudent();
            for(Student std:students)
            {
                  System.out.println(std);
            }
      }
}
```

Spring JDBC using Java Annotation

Till now we have declared all the bean in the configurartion file(xml file).

Beans declared by us in the xml file are

- DataSource
- JdbcTemplate
- StudentDao

Now instead of using xml file ,we can use java file for configuration

1. Create JdbcConfig.java file
2. Apply @Configuration annotation on the class.
3. Create three methods for one for each bean
4. Apply @Bean annotation on the methods.

JdbcConfig.java

```java
package com.Spring.jdbc;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

import com.Spring.jdbc.Dao.StudentDao;

@Configuration
public class JdbcConfig
{
	@Bean("ds")
	public DataSource getDataSource()
	{
		DriverManagerDataSource ds=new DriverManagerDataSource();
		ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
		ds.setUrl("jdbc:mysql://localhost:3306/springjdbc");
		ds.setUsername("root");
		ds.setPassword("76448");
		return ds;

	}
	@Bean("jdbcTemplate")
	public JdbcTemplate getTemplate()
	{
		JdbcTemplate template=new JdbcTemplate();
		template.setDataSource(getDataSource());
		return template;
	}
	@Bean("studentDao")
	public StudentDao  getStudentDao()
	{
		StudentDao dao =new StudentDao();
		dao.setTemp(getTemplate());
		return dao;
	}
}
```

Now we don’t need config.xml file.Rest all the things will be same .

AUTOWIRING IN SPRING JDBC

In the above example we are creating and returning the object manually by ourself in the methods creaed in JdbcConfig.java file. Instead of declaring StudentDao in the JdbcConfig.java file, we can use autowiring for injecting the JdbcTemplate object in StudentDa.

For that we need to declare the studentdao object with @Component annotation and property jdbcTemplate with annotation @Autowired. Also we need to declare the package using @ComponentScan annotaton.

JdbcConfig.java

```
package com.Spring.jdbc;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

import com.Spring.jdbc.Dao.StudentDao;

@Configuration
@ComponentScan(basePackages = "com.Spring.jdbc.Dao")
public class JdbcConfig
{
	@Bean("ds")
	public DataSource getDataSource()
	{
		DriverManagerDataSource ds=new DriverManagerDataSource();
		ds.setDriverClassName("com.mysql.cj.jdbc.Driver");
		ds.setUrl("jdbc:mysql://localhost:3306/springjdbc");
		ds.setUsername("root");
		ds.setPassword("76448");
		return ds;
```

```
        }
        @Bean("jdbcTemplate")
        public JdbcTemplate getTemplate()
        {
                JdbcTemplate template=new JdbcTemplate();
                template.setDataSource(getDataSource());
                return template;
        }
}
```

StudentDao.java

```
package com.Spring.jdbc.Dao;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.RowMapperResultSetExtractor;
import org.springframework.stereotype.Component;

import com.Spring.jdbc.Entities.Student;
@Component("studentDao")
public class StudentDao
{
        @Autowired
        private JdbcTemplate temp;
        public JdbcTemplate getTemp() {
                return temp;
        }
        public void setTemp(JdbcTemplate temp) {
                this.temp = temp;
        }
        public int insert(Student student)
        {
                String query="insert into student values(?,?,?)";
                int
res=this.temp.update(query,student.getId(),student.getName(),student.getCity());
                return res;
        }
        public int change(Student student)
        {
                String query="update student set name=?,city=? where id=?";
                int
res=this.temp.update(query,student.getName(),student.getCity(),student.getId());
                return res;
        }
        public int delete(int id)
        {
                String query="delete from student where id=?";
                int res=this.temp.update(query,id);
                return res;
        }
        public Student getStudent(int id)
        {
```

```
		String query="select * from student where id=?";
		RowMapper<Student> rowMapper=new RowMapperImpl();
		Student student=this.temp.queryForObject(query, rowMapper,id);
		return student;
	}
	public List<Student> getAllStudent()
	{
		String query="select * from student ";
		List<Student> allStudent=this.temp.query(query, new RowMapperImpl());

		return allStudent;
	}
}
```