CHAPTER#2 SPRING JDBC

Spring-JDBC:-

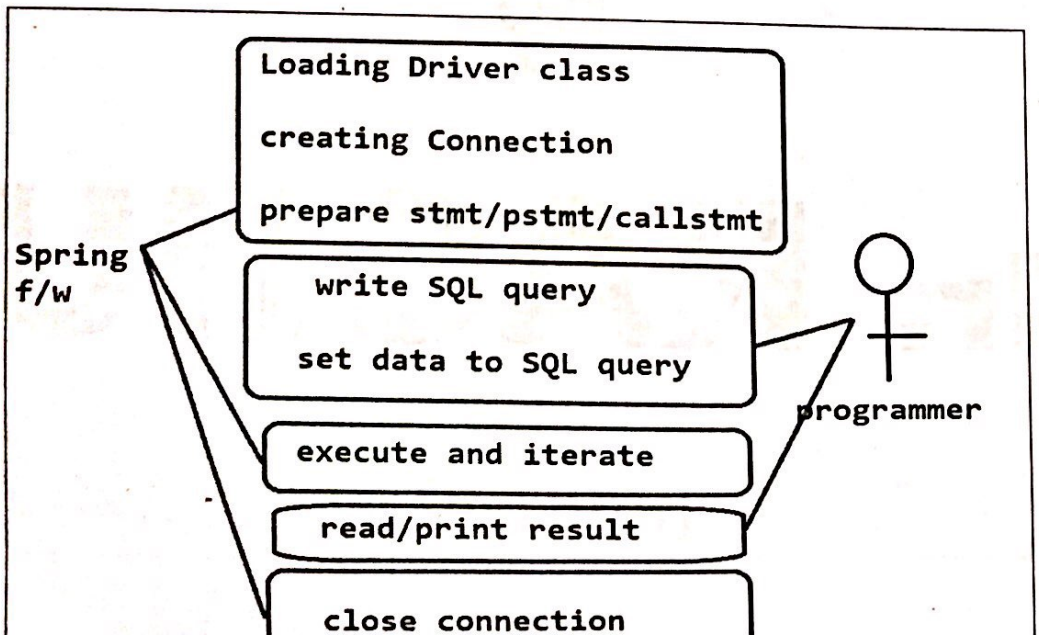
This is used to perform database operations in less lines by removing common lines of code (redundant code/boilerplate) .

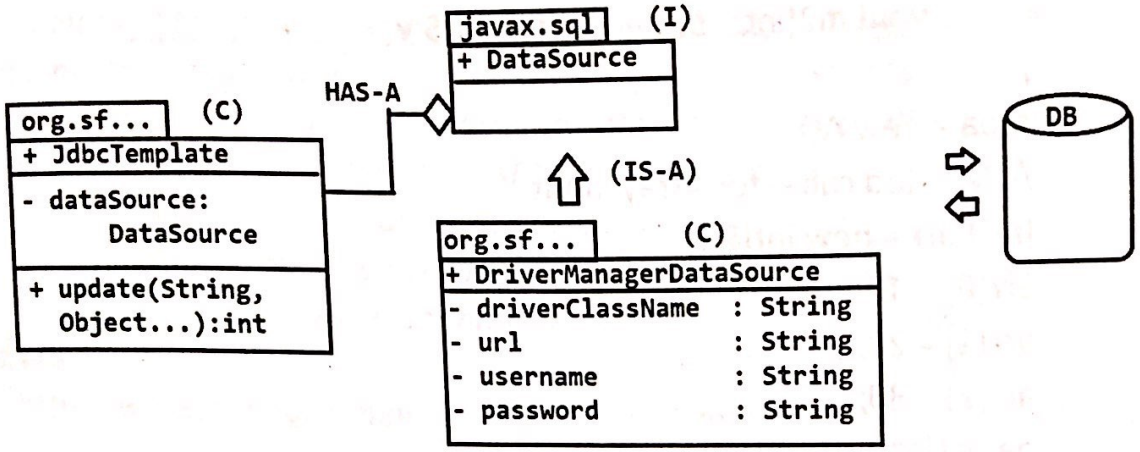
That is, if we consider two (2) JDBC programs common lines are connection, Driver class loading, statement etc... known as duplicate code.

These lines can be written one time and used multiple time done using Template Design Pattern.

Jdbc	**+**	**Template**	**=**	**JdbcTemplate**
Technology		**Design Pattern**		**(class)**

- JdbcTemplate (c) created by Spring Framework reduces common lines of code, by writing then only one time and use multiple times.
- Here Spring Container performs common operation steps, Programmer should write SQL, set data, get result and print code.





#JdbcTemplate (c) :- (org.springframework.jdbc.core)

➔ This class is given by Spring Framework, to perform database operations **(CURD).**

➔ For this we need to provide DataSource (I) (javax.sql) object. Here DataSource is an interface. So,we have to pass it's any one implementation class object.
Ex: DriverManagerDataSource, BasicDataSource etc...

➔ JdbcTemplate provide method "update" which is used to perform insert, update and delete operations.
API:-
update(String sql, Object... args) : int

 # Var-args : Varing length argument:-

➔ This concept also similar to Array. we can pass multiple values comma seprated in mrthod call instead of creating Array and providing data.

➔ Var-args applicable to any datatype.

➔ It is introduced in JDK 1.5 version.

➔ Var-args must be last parameter of a method that is after var-args parameter no other parameter is allowed.
Syntex is DataType... VariableName
Ex:
Int...marks
String...subject
Double...avegs

Example: Array and var-args

```
class A {
        void m1(int[] a) {  }
```

```
        void m2(int... b) {    }   // JDK1.5 v
    }
    A oa = new A();
    // Method call -- for Array input
    Int[] arr = new int[3];
    arr[0] = 10;
    arr[1] = 20;
    arr[2] = 30;
    oa.m1(arr);
    // Method call – using var-args.
    oa.m2(10,20,30,40);
```

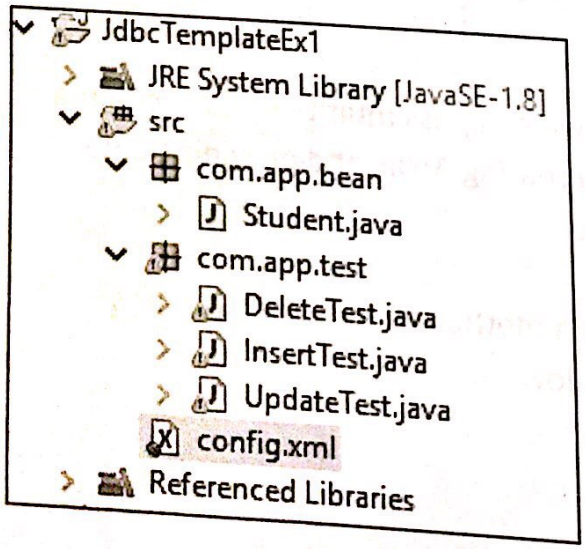
 # Code -- Spring JDBC Using XML Configuration:-

➔ JdbcTemplate class having update() method which perform "insert, update and delete" operation by taking two inputs.

String -- SQL Query
Object... -- inputs to SQL Query

Example Code #1 Using Spring XML Configuration:---

Folder Structure:



#Insert Operation

1. **Config.xml file code**

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<beans xmlns=http://www.springframework.org/schema/beans
xmlns:context="http://www.springframework.org/schema/context"
      xmlns:util="http://www.springframework.org/schema/util"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
                    http://www.springframework.org/schema/beans/spring-
beans.xsd
      http://www.springframework.org/schema/context

       http://www.springframework.org/schema/context/spring-context.xsd
                          http://www.springframework.org/schema/util

       http://www.springframework.org/schema/util/spring-util.xsd  ">

       <bean
       class="org.springframework.jdbc.datasource.DriverManagerDataSource"
                    name="dmdsObj"
                    p:driverClassName="oracle.jdbc.driver.OracleDriver"
                    p:url="jdbc:oracle:thin:@localhost:1521:xe"
                    p:username="system"
                    p:password="root"
                    />
        <beanclass="org.springframework.jdbc.core.JdbcTemplate"
                    name="jtObj"
                    p:dataSource-ref="dmdsObj"
                    />

   </beans>
```

2. Spring Bean Code:

```
package com.app.bean;

publicclass Student {
     privateintstdId;
     private String stdName;
     private String course;
     privatedoublestdFee;

      public Student() {
```

```
        super();
    }

    publicint getStdId() {
        returnstdId;
    }

    publicvoid setStdId(intstdId) {
        this.stdId = stdId;
    }

    public String getStdName() {
        returnstdName;
    }

    publicvoid setStdName(String stdName) {
        this.stdName = stdName;
    }

    public String getCourse() {
        returncourse;
    }

    publicvoid setCourse(String course) {
        this.course = course;
    }

    publicdouble getStdFee() {
        returnstdFee;
    }

    publicvoid setStdFee(doublestdFee) {
        this.stdFee = stdFee;
    }

    @Override
    public String toString() {
        return"Student [stdId=" + stdId + ", stdName=" +
stdName + ", course=" + course + ", stdFee=" + stdFee +
"]";
    }
```

```
}
```

3. Test class code:

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

publicclass InsertTest {
        publicstaticvoid main(String[] args) {
                ApplicationContext c = new
ClassPathXmlApplicationContext("config.xml");

                JdbcTemplate jt = (JdbcTemplate) c.getBean("jtObj");

                String sql = "insert into student1 values(?,?,?,?)";
                intcount = jt.update(sql, 1005,"Ashutosh","Spring",49999);

//              DriverManagerDataSource  ds= null;
//              ds.setDriverClassName(arg0);
//              ds.setUrl(url);
//              ds.setUsername(username);
//              ds.setPassword(password);

                System.out.println("Student saved successfully :: "+count);

        }
}
```

OUTPUT:

Student saved successfully :: 1

Update Operation

Spring Config File Code and Spring Bean Code Same as before

Test class:

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
```

```
publicclass UpdateTest {

        publicstaticvoid main(String[] args) {
                ApplicationContext ac = new
ClassPathXmlApplicationContext("config.xml");

                JdbcTemplate jt = (JdbcTemplate) ac.getBean("jtObj");

                String sql = "update student1 set stdname=?, stdcourse=?,
stdfee=? where stdid=?";
                intcount = jt.update(sql, "Pooja","Adv.ja",8000, 111 );

                System.out.println("Student Updated Successfully :: "+count);
        }
}
```

OUTPUT:

Student Updated Successfully :: 1

Delete Operation

Spring Config File Code and Spring Bean Code Same as before

Test class:

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

publicclass DeleteTest {

        publicstaticvoid main(String[] args) {
                ApplicationContext ac = new
ClassPathXmlApplicationContext("config.xml");

                JdbcTemplate jt = (JdbcTemplate) ac.getBean("jtObj");

                String sql = "delete from student1 where stdid=?";
```

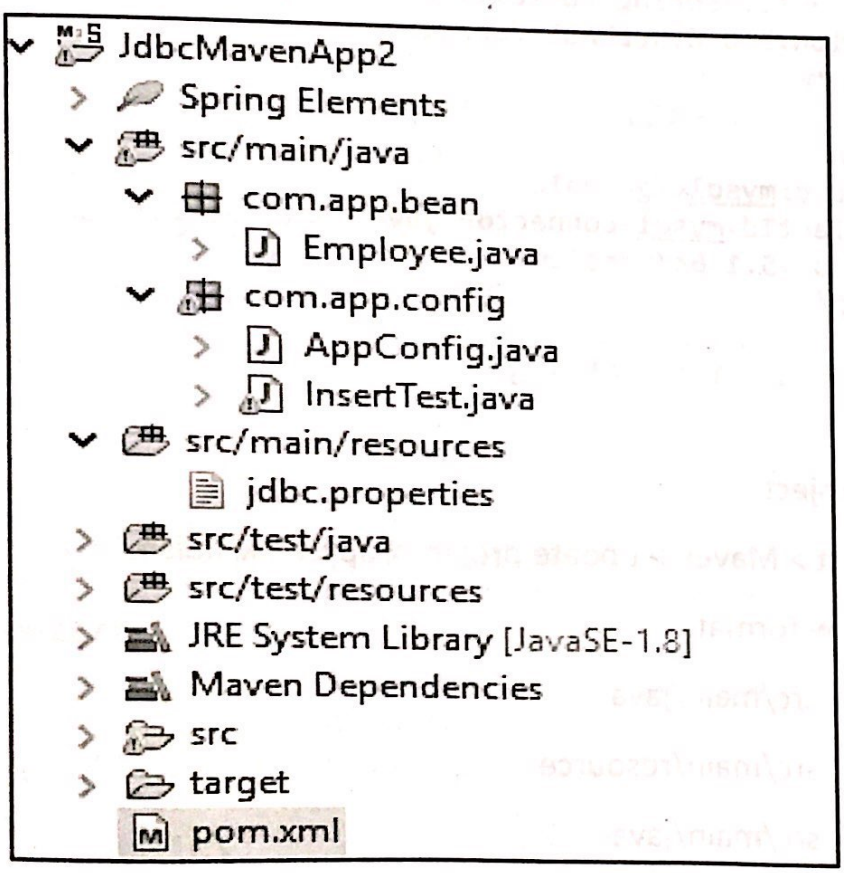
```
        intcount = jt.update(sql,113);

        System.out.println("Student deleted Successfully :: "+count);
    }
}
```

OUTPUT:

Student deleted Successfully :: 1

Spring JDBC Using Java Configuration and Maven Tool

Folder Structure- Maven Tool



Steps To Create Maven Project For Core Application:

Step#1: Create Maven Project in Eclipse or STS.

- File > New > Maven Project > Click Check Box
- LVT create simple maven project (skip.....)
- Next button > Enter Details like

  groupId : org.sathyatech
  artifactId : SpringJDBCMavenApp
  Version : 1.0
- Click on finish button.

Step#2: open pom.xml and add below dependencies(jar details) and build plugins (compiler details)

```
<dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>5.0.6.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.0.6.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6</version>
        </dependency>

    </dependencies>
```

step#3: Update Maven Project

- Right click on project > Maven > update project > apply/ok/finish.

step#4: Write code in below format.

1. Spring Bean : src/main/java
2. Spring Config : src/main/resource
3. Test Class : src/main/java

step#5: Run Test class to see output.

- .java files must be placed under src/main/java folder.
- Non-java files like xml/properties etc... Must be placed under src/main/resource folder.

Example:

1. Pom.xml file code:-

```
<projectxmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi
="http://www.w3.org/2001/XMLSchema-
instance"xsi:schemaLocation="http://maven.apache.org/POM/4
.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.sathyatech</groupId>
<artifactId>SpringMavenApp2</artifactId>
<version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>5.0.6.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.0.6.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-
java</artifactId>
            <version>5.1.6</version>
        </dependency>
    </dependencies>

    <build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.8.0</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
</plugins>
```

```
</build>
</project>
```

2.Java config code:

```
package com.app.config;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

@PropertySource("jdbc.properties")
@Configuration
publicclass AppConfig {
	@Autowired
	private Environment env;

	@Bean
	public DriverManagerDataSource dsObj() {
		DriverManagerDataSource ds = new
DriverManagerDataSource();
		ds.setDriverClassName(env.getProperty("dc"));
		ds.setUrl(env.getProperty("url"));
		ds.setUsername(env.getProperty("un"));
		ds.setPassword(env.getProperty("pwd"));
		returnds;
	}
	@Bean
	public JdbcTemplate jtObj() {
		JdbcTemplate jt = new JdbcTemplate();
		jt.setDataSource(dsObj());
		returnjt;
	}

}
```

3.Test class code:

```
package com.app.config;
```

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;

publicclass InsertTest {

      publicstaticvoid main(String[] args) {
            ApplicationContext ac = new
AnnotationConfigApplicationContext(AppConfig.class);
            JdbcTemplate jt = (JdbcTemplate) ac.getBean("jtObj");
            String sql = "insert into emptab2 values(?,?,?,?)";
            intcount = jt.update(sql, 11,"Ashu","JS",999);
            System.out.println("Data Inserted :: "+count);
      }
}
```

4.db.properties file code:

```
dc=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/test
un=root
pwd=root
```

5.Spring Bean code:

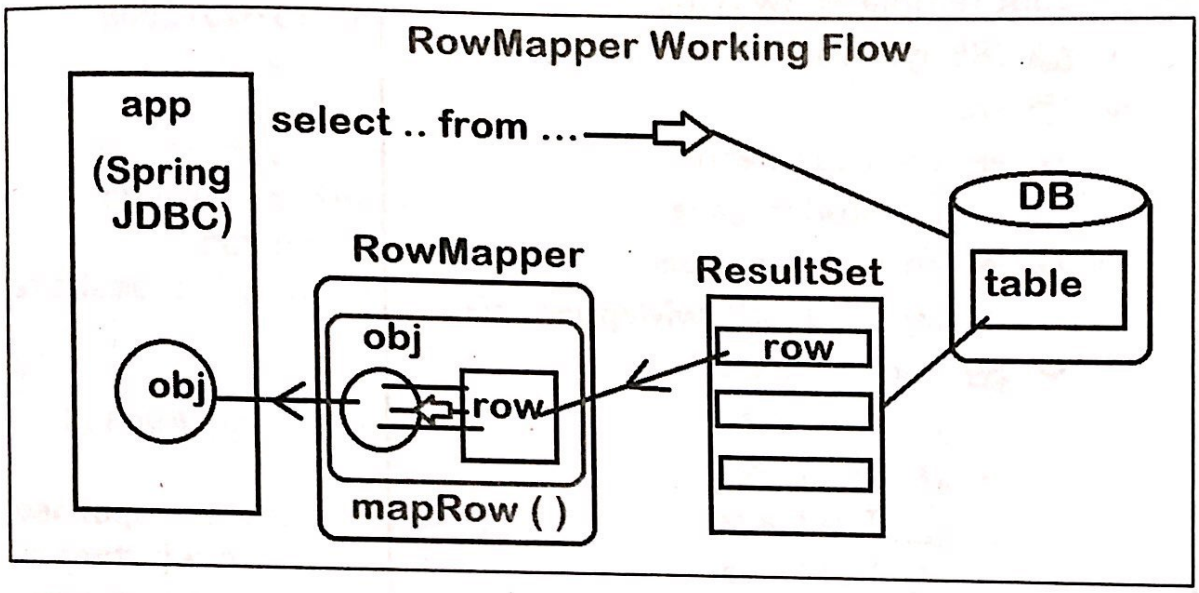
```
package com.app.bean;

publicclass Employee {
     privateintempId;
     private String empName;
     private String desig;
     privatedoubleempSal;

     public Employee() {
          super();
     }

     publicint getEmpId() {
          returnempId;
     }

     publicvoid setEmpId(intempId) {
```

```
        this.empId = empId;
    }

    public String getEmpName() {
        returnempName;
    }

    publicvoid setEmpName(String empName) {
        this.empName = empName;
    }

    public String getDesig() {
        returndesig;
    }

    publicvoid setDesig(String desig) {
        this.desig = desig;
    }

    publicdouble getEmpSal() {
        returnempSal;
    }

    publicvoid setEmpSal(doubleempSal) {
        this.empSal = empSal;
    }

    @Override
    public String toString() {
        return"Employee [empId=" + empId + ", empName=" +
empName + ", desig=" + desig + ", empSal=" + empSal + "]";
    }

}
```

Output:

```
Data Inserted :: 1
```

RowMapperin Spring JDBC:



#RowMapper(I) in Spring JDBC:- (package: org.springframework.jdbc.core)

➔ To fetch data from database using select query, JdbcTemplate has provided special method like:

a) queryForObject(Single row)
b) query(Multiple row)

both methods perform select operations

But these methods will fetch data from database table in ResultSet format (having rows) after excuting SQL query.

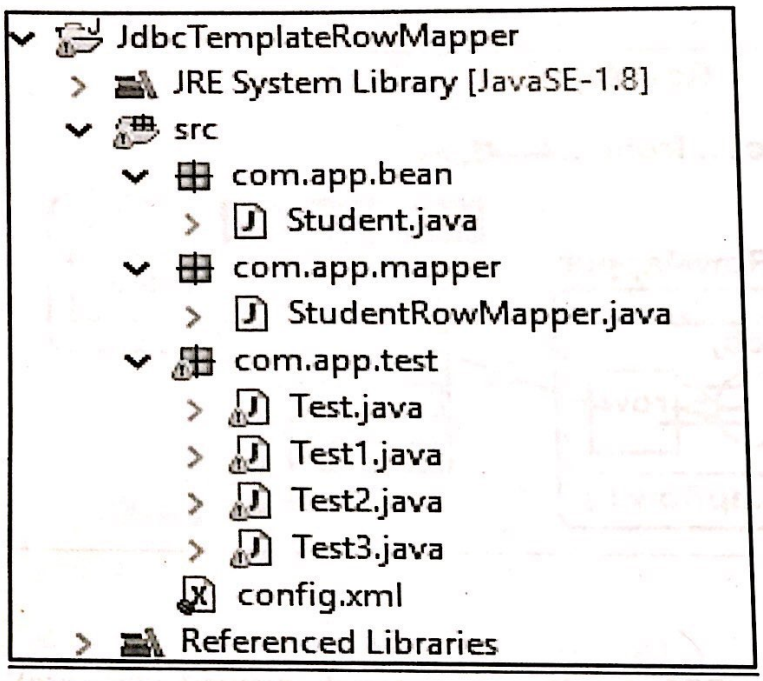
➔ This ResultSet data cab be converted to Objects format using RowMapper(I).

➔ RowMapper will not get data form database. If only converts ResultSet rows to model class Objects.

➔ It is having MapRow() method which contains conversion logic (row -> object).

Example of RowMapper:

Folder Structure:



1. Spring Bean

```
package com.app.bean;

publicclass Student {
    privateintstdId;
    private String stdName;
    privatedoublestdFee;

    public Student() {

    }
    publicint getStdId() {
        returnstdId;
    }
    publicvoid setStdId(intstdId) {
        this.stdId = stdId;
    }
    public String getStdName() {
        returnstdName;
    }
    publicvoid setStdName(String stdName) {
        this.stdName = stdName;
    }
    publicdouble getStdFee() {
        returnstdFee;
```

```
    }
    publicvoid setStdFee(doublestdFee) {
        this.stdFee = stdFee;
    }
    @Override
    public String toString() {
        return"Student [stdId=" + stdId + ", stdName=" +
stdName + ", stdFee=" + stdFee + "]";
    }
}
```

2. RowMapper Code:

```
package com.app.mapper;
import java.sql.ResultSet;
import java.sql.SQLException;
import org.springframework.jdbc.core.RowMapper;
import com.app.bean.Student;

publicclass StudentRowMapper implements RowMapper<Student>
{

    public Student mapRow(ResultSet rs, intcount) throws
SQLException {
        Student s = new Student();
        s.setStdId(rs.getInt(1));
        s.setStdName(rs.getString(2));
        s.setStdFee(rs.getDouble(3));
        returns;

    }
}
```

3. Spring config file:

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

    http://www.springframework.org/schema/beans/spring-beans.xsd
```

```
        http://www.springframework.org/schema/context

        http://www.springframework.org/schema/context/spring-context.xsd

        http://www.springframework.org/schema/util

        http://www.springframework.org/schema/util/spring-util.xsd">

        <bean
        class="org.springframework.jdbc.datasource.DriverManagerDataSource"
                    name="dmdsObj"
                    p:driverClassName="oracle.jdbc.driver.OracleDriver"
                    p:url="jdbc:oracle:thin:@localhost:1521:xe"
                    p:username="system"
                    p:password="root"
                    />
        <beanclass="org.springframework.jdbc.core.JdbcTemplate"
                    name="jtObj"
                    p:dataSource-ref="dmdsObj"
                    />
</beans>
```

4. Test Class:

```
package com.app.test;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.jdbc.core.JdbcTemplate;
import com.app.bean.Student;
import com.app.mapper.StudentRowMapper;

publicclass Test {
    publicstaticvoid main(String[] args) {
        ApplicationContext c = new
ClassPathXmlApplicationContext("config.xml");
        JdbcTemplate jt = (JdbcTemplate)
c.getBean("jtObj");
        String sql = "select * from stdtab4 where sid=?";
        StudentRowMapper srm = new StudentRowMapper();
        Student s = jt.queryForObject(sql, srm, 8);
```

```
        System.out.println(s);
    }
}
```

Output:

Student [stdId=6, stdName=AAA, stdFee=50.0]

Student [stdId=8, stdName=ASJK, stdFee=444.0]

Student [stdId=5, stdName=AAA, stdFee=50.0]

 # query() method:-

➔ This method is used to get multiple rows from database table using SQL and RowMapper(I).

API: query(String sql,RowMapper<T> rm) : List<T>

Final data is returned in List format.

 # queryForObject() method:-

➔ This method is used to get one row data from database table using SQl and RowMapper(I).

API:

queryForObject(String sql, RowMapper<T> rm, Object... inputs) : T

Here Object... (var-args) are used to pass data to SQL at runtime in place of '?' symbols (place holder).

##RowMapper<T> (I) is a functional interface, so we can write above code without writing StudentRowMapper class, directly using lambda expression.

EX#1. (Test class code)

```
String  sql = "select * from student where sid=?";
RowMapper<Student> rm = (rs, count)-> {
Student s = new Student();
s.setStdId(rs.getInt("sid"))
s.setStdName(rs.getString("sname"))
s.setStdFee(rs.getDouble("sfee"))
```

```
return s;
};
Student s = jt.queryForObject(sql, rm, 45);
Sysout(s);
```

EX#2: Generate 3 param constructor in Student Then lambda expression is:

```
RowMapper<Student> rm = (rs, count) ->
                    new Student (rs.getInt(1),rs.getString(2),rs.getDouble(3));
```