Spring JDBC

Why there is need of Spring JDBC?

It simplifes the exisiting jdbc code and reduces the boilerplate code for various operation performed on db.It acts as a between JavaProgram and DB

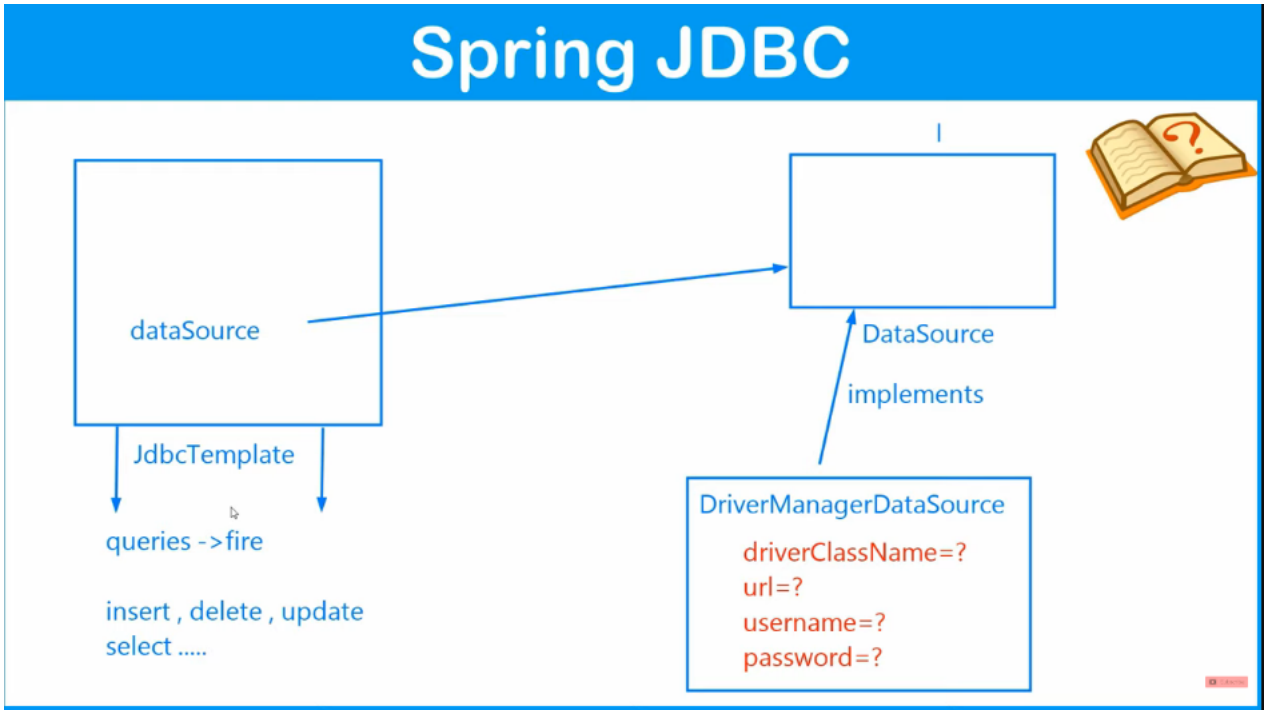
Disadvantages of StandAlone JDBC :

1) Too much , for large projects.
2) Extracting info from
3) , forget to close connection .
4) and

StandAlone JDBC throws Checked Exceptions(Compile Time) which needed to be handeled everytime , whereas Spring JDBC throws all unchecked Exceptions(Runtime Exceptions) which can be ignored

---

Spring JDBC Provides Jdbc Template



Jdbc Template -> Central class in spring jdbc (simplifies the use of jdbc) .
provides methods to execute SQL queries , and creation and release of resources.This jdbc template class contains info of dataSource.

DataSource-> Interface that abstracts info of data base.

DriverManagerDataSource->Class that implements DS ,and contains 4 things (Driver class Name) , (url) , (userName) , (password). Bean is created for this class.

Steps :

1) Configure Bean of DMDS class with all 4 properties.
2) Inject the dataSource into jdbc Temp.
3) Now u can use Jdbc Temp. class for performing oper.

Practical Steps:

1) Create a project using maven

2) Open POM.xml and add dependencies of Spring core and Spring context
3) Then add spring Jdbc dependencies from google (same as spring core above)
4) Add mysql connector

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.2.3.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.2.3.RELEASE</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdb
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.3.0</version>
    </dependency>
```

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.2.3.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-core</artifactId>
        <version>5.2.3.RELEASE</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdb
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>5.3.0</version>
    </dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.33</version>
</dependency>
```

2) Setup Database

1) Open MySql commandline client
2) Create a DataBase / Sql code typed on Com

```
create database springJdcb;

create table student(id int primary key, name varchar(100) not null , city va
```

"To find package of any class in intellij use "

3) Create 2 beans in config.xml , of driverManagarDataSource and jdbcTemplate class

```
While setting DriverManagerDataSource Class u need to set 4 properties :
1)            = easily find online (like for mysql = "com.mysql.cj.jdcb.Driver"          2)     =  te
3)          = root
4)          = Defined by u
```

Now after setting up beans use an IOC container to access jdbcTemplate

But this is not the best practice , that you should follow

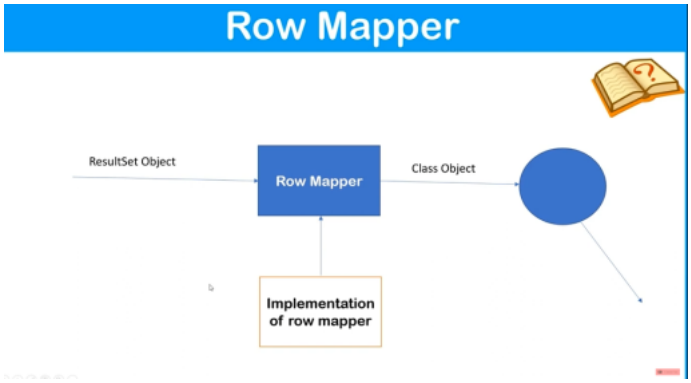
How it's really done ? (Desing Pattern) DAO Pattern (Data Access Object)

First u need an interface for SQL Operation (for loose Coupling) And a class implementing that (where u'll perform query containing jdbcTemplate) Inject this jdbcTemplate using config beans now in driver class make an IOC container using container get the implemented class pass the entity in the sql method
So if you want to perform any operation like update , insert , delete on DB , just create it as a method in that interface and override it in the implementation class.

Student (entity class) -> **DAO interface (having query methods)** -> **DAOImplement (override query methods using JDBC template)** -> ***Main Driver Class***

The process is same for all types of queries but for select queries : **Select Query**

RowMapper : It is an interface used to . It converts each row of the resultSet into a desired Java Object.



if you want to get a single row then use

```
public T queryForObject( String sql, RowMapper<T> rowMapper,Object args)
```

and if you want many rows then use:

```
public List<T> query(String sql, RowMapper<T> rowMapper)
```

Practical Steps :

First Create the Select Method and define the return type of it (Obejct) in that interface -> Then override the method in the implementation class :
1) using queryForObject (sql , RowMapperImplementationClass , id)

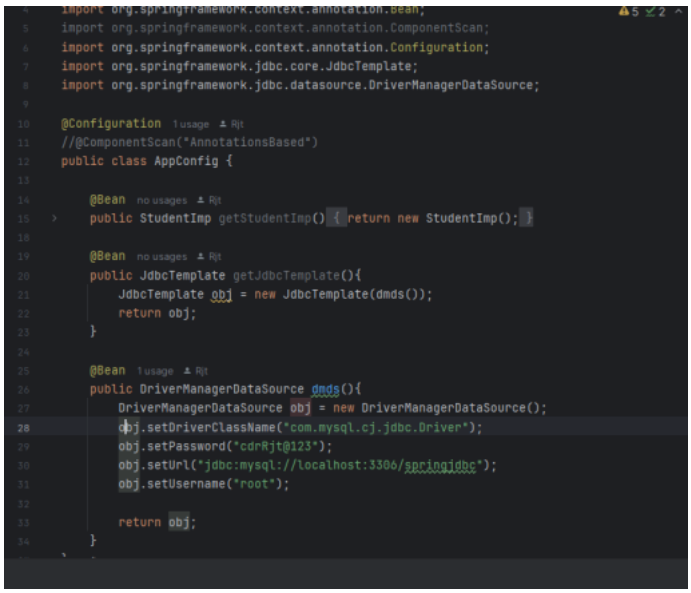
```
Remember to create an implementation class of rowMapper Interface
Using an Implementation Class:

public class RowMapperImp implements RowMapper<Student> {
    @Override
    public Student mapRow(ResultSet set, int i) throws SQLException {
        Student student = new Student(set.getInt(1),set.getString(2),set.getString(3));
        return student;
    }
}
Using an Anonymous Clas:
Student obj = jdbcTemplate.queryForObject( sqlQuery ,new RowMapper(){
 public Object mapRow( ResultSet rs,int rowNum) throws SQLException{
     Student stu = new Student(rs.getInt(1),rs.getString(2),rs.getString(3));
  reutrn stu;
}
```

}

Sometimes using annotations with spring might give you "Unsupported major class File version 65" that due to spring downgrade version and mismatch of spring and java jdk version.

Annotation Based Configuration

Create 3 beans DataSourceDriverManager ,
JDBCTemplate and
StudentImplementation Class



```
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;

@Configuration 1 usage  Rjt
//@ComponentScan("AnnotationsBased")
public class AppConfig {

    @Bean  no usages  Rjt
    public StudentImp getStudentImp() { return new StudentImp(); }

    @Bean  no usages  Rjt
    public JdbcTemplate getJdbcTemplate(){
        JdbcTemplate obj = new JdbcTemplate(dmds());
        return obj;
    }

    @Bean  1 usage  Rjt
    public DriverManagerDataSource dmds(){
        DriverManagerDataSource obj = new DriverManagerDataSource();
        obj.setDriverClassName("com.mysql.cj.jdbc.Driver");
        obj.setPassword("cdrRjt@123");
        obj.setUrl("jdbc:mysql://localhost:3306/springjdbc");
        obj.setUsername("root");

        return obj;
    }
```

JdbcTemplate Methods

2 main methods :

update() -> insert, update , delete..
query() , queryForObject() -> select queries..