**Spring JDBC**

* First lets start by creating a maven project. Will be quickstart type.
* GroupId will be com and and artifactID will be spring.jdbc.demo
* Since this is a simple Spring Maven Project. We are going to need Spring Context, MySQL, Java Connector, and Spring JDBC dependencies imported in the POM.XML file.
* POM.XML file will look like this:

A screenshot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

* **Note**: Since this is a maven project, JUnit for JUnit testing is a dependency that we already have. We also already have an App.java in our src/main/java folder and have a App.test in our src/test folder. These come with maven project. If you first use dynamic web **and convert TO MAVEN**, then this might not pop up which is completely okay.
* Now lets create a model class for our database. So go to your **src/main/java** folder then create a new package called **com.spring.jdbc.demo.model** and in that package create a java class called Dish.
* At the **top of Dish class**, define some properties. Then create a normal constructor first and pass the parameter values with “this” keyword, after create a constructor with fields, then getter/setter methods, and then toString method.
* **As of now Dish class looks like this**:

A screenshot of a computer program

Description automatically generated

* Next **go to MySQL** and create a new table in our eStore database that has already been created. This table will be named Dish, create it like this:

A screen shot of a computer

Description automatically generated

* Now we want to Map the Dish object we create in Java with the Dish table inside of our MySQL database.
* Lets start by creating a new package in **src/main/java** called **com.spring.jdbc.demo.db** and in this package add a new class called DB.
* In **DB class** we are going to start by defining DataSource and JDBC Template.
* After defining that, lets create a default constructor for DB. Followed by getter/setters **ONLY** for datasource **NOT** for Template.
* Then in the datasource setter we would like to initialize our JdbcTemplate by writing **jdbcTemplate = new JdbcTemplate(dataSource);** so we are passing our data source through Jdbc Template.
* **Note**: That JdbcTemplate is apart of jdbc core.
* And then below our jdbcTemplate lets give a print statement for setter/injection that will print when this is being used as a setter injection.
* So far DB class looks like this:

A screenshot of a computer program

Description automatically generated

* Now we need a context.xml file. You can copy an xml file from previous projects. I am going to copy it from out estore project. Now go to your src/main/java folder and paste it in there.
* And remove all beans that you created so your XML looks like this at the moment:

A screenshot of a computer

Description automatically generated

* Now we want to start in XML file by defining a bean for our dataSource. Now go to your maven dependency, click on **org.springframework.jdbc.datasource** and then go the class that is **DriverManagerDataSource.class**. Click on that and copy the full package name and then go back to your context.xml file and paste it into your class. Then go back to the class and scroll all the way down and copy the class name and paste it in your class after the package name.
* So in web.xml in the class it should be the full package name plus the class name of datasource from Driver from spring framework.
* Now within this bean in xml, we need to create 4 properties with values. These properties that we are creating will be for driverClassName, url, username, and password. The value for driverClassName will be jdbc Driver and the URL value will be the URL for your database with the extension of your database name that you would like to connect to.
* **Note**: So essentially, like we did before when creating configuration for our database details inside of class. We are doing it here in xml file inside a bean that the Spring Container will be taking care of.
* So as of now in XML we have defined our data source configuration. And we need to provide these configurations to our DB class.
* So after that bean, lets create a new bean in XML file with the id name is db for our db class and our class is the class name that we created for db. Which is the full package name along with the class name as the extension.
* Then create properties, our first property name is dataSource and our ref will be ds, the ref here is the same as the id we gave for out first bean.
* As of now, context.xml looks like this:

A screenshot of a computer

Description automatically generated

* **Recap what weve done so far**: In DB class, we have given two attributes that is dataSource and jdbcTemplate that we defined above our constructor. We have given dataSource a value. And this dataSource contains all the details from the first bean which contains 4 properties.
* And our dataSource is used to create a new JdbcTemplate. So if you go to your JdbcTemplate class that is apart of your packages that are already created by default, you can see that we are passing dataSource with all these properties. And in the set dataSource there we are setting them.
* Now lets go to our main java class that is App.java and here we need to define the ApplicationContext by using ClassPathXmlApplicationContext()
* First you can start by wring Spring JDBC in the print statement as a reminder.
* Now below the print statement start defining the Application context and in the class path pass the name of the file of where your configuration details are, which in our case is context.xml
* Next below that we are going to use our ApplicationContext to get our DB details by creating new object od DB, importing it and then passing db as our id in the context get bean. So now our application context (IOC container) is able to receive the information/properties/values that we created in our xml file.
* As of now, App.java class looks like this:

A screenshot of a computer program

Description automatically generated

* Next lets create some CRUD operations inside our DB class.
* After the setDataSource lets create a function for inserting values into the Dish, and in the parameters pass the object of Dish and import Dish from Dish class. Then lets create some SQL queries for inserting into MySQL database.
* **So we haven’t performed** any explicit steps of registering the driver or creating the connection or creating the statement/prepared statement/ or the result set because we are directly using JdbcTemplate to call out the update function with the result.
* In this update function we would like to pass the sql, and then get whatever dish property we want to get, for example name and price.
* So basically we are updating two columns name and price, id we DO NOT need to add although that is a third property we have, it is auto-incremented so we don’t need to add here. Although In the String sql you would pass the value of null for id.
* And after the int result in app.java just create a print statement that will print the dish name that we create and stating that it has been entered in to the database.
* DB.java now looks like this:

A screenshot of a computer

Description automatically generated

* Now go to app.java and lets create some dishes. Start by creating a new dish object and create 3 dishes. Use 0 for the first column, id auto increment so it does not matter.
* Then use the insertDish function for all 3 dishes you created.
* App.java now looks like this:

A screenshot of a computer program

Description automatically generated

* Run the application and console will look like this:

A screenshot of a computer

Description automatically generated

* And those dishes have been successfully stored in your Dish table within your estore database in MySQL.
* **Note**: Spring JdbcTemplate takes care of a lot of code.
* Now go to DB.java and create a function to get all of the database details. Then pass the sql statement to retrieve records from database.
* Next create a List for Dish, dishList as the name of the list. Then call the jdbcTemplate, pass query and in the () pass sql and then pass a new **BeanPropertyRowMapper** and then pass the Dish class.
* And then use a print statement to print the dishList details in your console.
* DB.java now looks like this:

A screenshot of a computer program

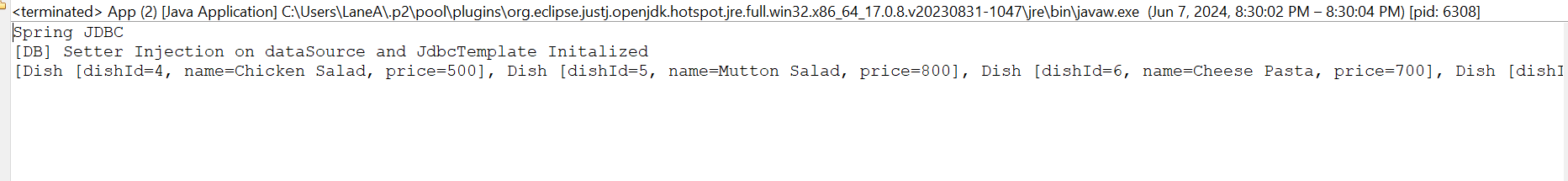
Description automatically generated

* Then go to App.java, comment out the dishes and then use method db.getAllDishes(); to get all the records from dish table.
* App.java now looks like this:

A screenshot of a computer program

Description automatically generated

* Run on application, and in console you will see all you records from Dish table printed.  
  Console will now look like this:



* **JdbcTemplate**: Saves a lot of code/time. JdbcTemplate is able to replace the use of the iterator and result Set. Which looked like this:

A computer screen shot of a program

Description automatically generated

* In this picture above, we created a function for getting all the customers. We had to create a sql statement, then use a prepared statement, then execute the query, then we had to iterate over the List and create the customer object one by one and set all the details one by one. And then add each customer object into the customer List.
* And now with spring JdbcTemplate, all of that is replaced by this

A computer code on a white background

Description automatically generated

* **With JdbcTemplate** we have created a SQL statement, then use jdbcTemplate.query. This is internally querying the database, getting all the rows and mapping them to the Dish objects, and then creating a List of Dish objects for us. Everything is done In an abstract manner.