

Sep 1st Lecture and Project spring-mvc-with-orm-demo

Create a Maven MVC project and deploy on an IDE Tomcat server. This document describes creating a MVC project in the SpringTool IDE. To begin the spring-mvc-demo project is copied.

The Spring Framework is an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE.

Object Relational Mapping (ORM) data access.

The Spring Framework provides integration with Hibernate, JDO, Oracle TopLink, iBATIS SQL Maps and JPA: in terms of resource management, DAO implementation support, and transaction strategies. For example for Hibernate, there is first-class support with lots of IoC convenience features, addressing many typical Hibernate integration issues. All of these support packages for O/R (Object Relational) mappers comply with Spring's generic transaction and DAO exception hierarchies. There are usually two integration styles: either using Spring's DAO 'templates' or coding DAOs against plain Hibernate/JDO/TopLink/etc APIs. In both cases, DAOs can be configured through Dependency Injection and participate in Spring's resource and transaction management.

The order of topics is not specifically the order of the lecture recording. In the recording, some items are presented out of order of how they should go. Like when creating a database bean class before checking your environment variables. Also like starting to create packages and classes before updating dependencies in the pom.xml file. There are other areas as well.

Table of Contents

Create new Maven project to use ORM	5
Create the new project based on a previous project	5
Update the POM.xml file	6
Update the project name.....	6
Add database related dependencies to POM.xml	6
Update the WEB.xml file	8
Update the display name	8
Create configuration package & class.....	9
Create configuration package	9
Create configuration class.....	10
Add class annotations to configuration class.....	10

Add three beans from the lecture	12
Add a data source bean	12
Verify data source environment variables	13
Add a session factory bean	14
Add a transaction manager bean	15
Setup a database for the demo	17
Change the system environment variable	17
Create the demo database	18
Create DAO and Model packages	19
Create DAO package	19
Create model package	19
Create model class	20
Complete the model class Ship	21
Update the base Ship class code generated when created	21
Test the database connection	22
Add this project to the IDE Tomcat server	22
Verify the sessionFactory() method in ORMConfig class is set to "create"	23
Start the Tomcat server	24
Create DAO class	25
Complete the DAO class ShipDao	25
Update the base ShipDao class code generated when created	25
Create DTO class	27
Complete the DTO class ShipDTO	28
Update the base ShipDTO class code generated when created	28
Create Service package and class	30
Create Service package	30
Create service class	31
Complete the service class ShipService	31
Update the base ShipService class code generated when created	31
Create Exception package and classes	34
Create Service package	34
Create exception classes	35
Create bad parameter exception	35

Create not found exception	38
Complete the service class ShipNotFoundException	38
Update Controller package and classes	40
Update TestController class	40
Create ShipController class	40
Complete the controller class ShipController	40
Update the base ShipController class code generated when created.....	41
Update the Message.Dto class to the following:	45
Continue Updating the base ShipController class	46
Test the POST API.....	49
Testing the POST API.....	49
Possible Testing Issues	50
Postman web (cloud) based issue.....	50
Project URL issue.....	50
Update ORMConfig sessionFactory() method	52
Test the GET API.....	53
Testing the GET API.....	53
Create an Angular Web Application for the Demo	54
Generate an Angular folder “display-ships-app”	54
Open the default project with VS Code	55
Create a ship Model	56
Update the model file	56
Generate a ship service.....	57
Update ship service.....	57
First add import to the application	57
Next update ship service.....	58
Add service to application.....	60
Modify file "app.component.html"	62
Run the Angular Application	63
Command ng serve	63
Command ng serve – with port number.....	64
Additional Angular prompts that might be encountered	64
Appendix: Alternative to db_url environment variable.....	65

Example for hosted is to use:.....	65
Example for localhost is to use:	65
Example for database name is to use:	66
Update DataSource bean in ORMConfig file.....	66
Appendix: Correct Issue with Project URL	67

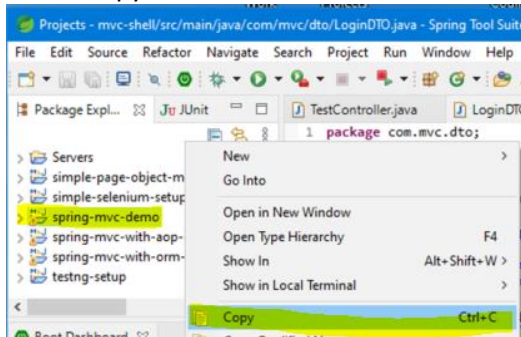
Create new Maven project to use ORM

Create the new project based on a previous project

Copy the spring-mvc-demo project

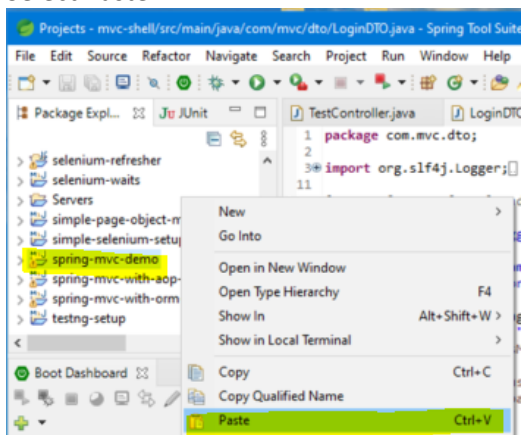
Right click on the project

Select Copy



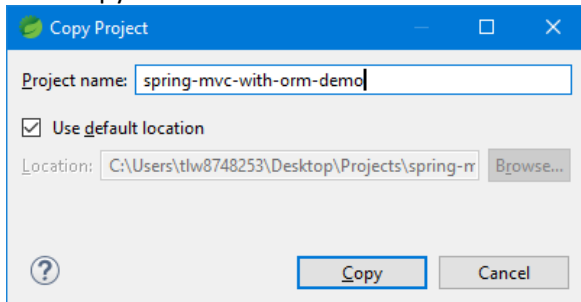
Right click again

Select Paste



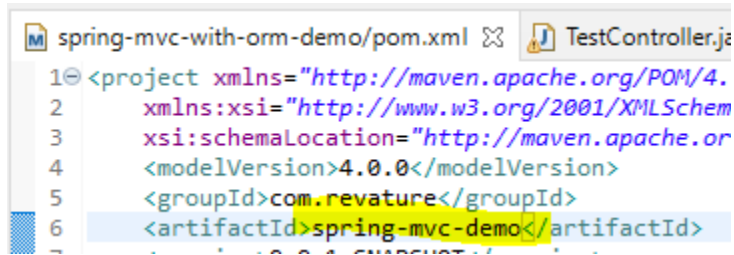
Name the project: "spring-mvc-with-orm-demo"

Click Copy



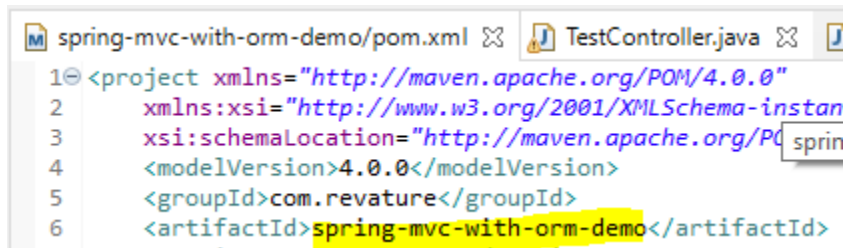
Update the POM.xml file

Update the project name



```
1<project xmlns="http://maven.apache.org/POM/4.0.0"
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4  <modelVersion>4.0.0</modelVersion>
5  <groupId>com.revature</groupId>
6  <artifactId>spring-mvc-demo</artifactId>
```

```
<artifactId>spring-mvc-with-orm-demo</artifactId>
```



```
1<project xmlns="http://maven.apache.org/POM/4.0.0"
2  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
4  <modelVersion>4.0.0</modelVersion>
5  <groupId>com.revature</groupId>
6  <artifactId>spring-mvc-with-orm-demo</artifactId>
```

Since we copied the spring-mvc-demo we should have the following in the POM.xml already:

```
10<properties>
11  <maven.compiler.source>1.8</maven.compiler.source>
12  <maven.compiler.target>1.8</maven.compiler.target>
13</properties>
14
15<dependencies>
16  <!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
17  <dependency>
18    <groupId>javax.servlet</groupId>
19    <artifactId>javax.servlet-api</artifactId>
20    <version>4.0.1</version>
21    <scope>provided</scope>
22  </dependency>
23  <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
24  <dependency>
25    <groupId>com.fasterxml.jackson.core</groupId>
26    <artifactId>jackson-databind</artifactId>
27    <version>2.12.5</version>
28  </dependency>
29  <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
30  <dependency>
31    <groupId>org.springframework</groupId>
32    <artifactId>spring-webmvc</artifactId>
33    <version>5.3.9</version>
34  </dependency>
35  <!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
36  <dependency>
37    <groupId>org.projectlombok</groupId>
38    <artifactId>lombok</artifactId>
39    <version>1.18.20</version>
40    <scope>provided</scope>
41  </dependency>
```

No updates to the above are necessary.

Add database related dependencies to POM.xml

The following dependencies “Artifact Ids” are added for this project

- spring-orm
- hibernate-core
- tomcat-dbcp
- mariadb-java-client

We can search, copy and paste from the Maven Repository to find the versions we want to use
<https://mvnrepository.com/>

For this demo purpose, copy the text below and add to end of POM.xml just above the
</dependencies> line.

```
<!-- Database Related dependencies -->
<!-- https://mvnrepository.com/artifact/org.springframework/spring-orm -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>5.3.9</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.5.7.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-dbcp -->
<dependency>
  <groupId>org.apache.tomcat</groupId>
  <artifactId>tomcat-dbcp</artifactId>
  <version>10.0.10</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client -->
<dependency>
  <groupId>org.mariadb.jdbc</groupId>
  <artifactId>mariadb-java-client</artifactId>
  <version>2.7.4</version>
</dependency>
```

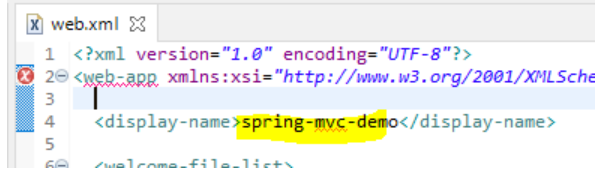
```
43 <!-- Database Related dependencies -->
44 <!-- https://mvnrepository.com/artifact/org.springframework/spring-orm -->
45 <dependency>
46   <groupId>org.springframework</groupId>
47   <artifactId>spring-orm</artifactId>
48   <version>5.3.9</version>
49 </dependency>
50 <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
51 <dependency>
52   <groupId>org.hibernate</groupId>
53   <artifactId>hibernate-core</artifactId>
54   <version>5.5.7.Final</version>
55 </dependency>
56 <!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-dbcp -->
57 <dependency>
58   <groupId>org.apache.tomcat</groupId>
59   <artifactId>tomcat-dbcp</artifactId>
60   <version>10.0.10</version>
61 </dependency>
62 <!-- https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client -->
63 <dependency>
64   <groupId>org.mariadb.jdbc</groupId>
65   <artifactId>mariadb-java-client</artifactId>
66   <version>2.7.4</version>
67 </dependency>
68 </dependencies>
69 </project>
70
```

Update the WEB.xml file

Update the display name

Any time you open the web.xml file you might see an error. This is a bug with the IDE. Once the file is change in some way, add or delete a space, or make a needed change and the file is saved, the error should resolve.

Update the display name from:



```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
3
4 <display-name>spring-mvc-demo</display-name>
5
6 <welcome-file-list>
```

To:

spring-mvc-with-orm-demo



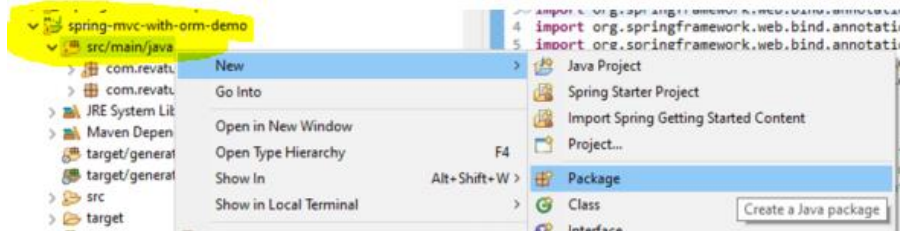
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
3
4 <display-name>spring-mvc-with-orm-demo</display-name>
5
```


Create configuration package & class

Create configuration package

Right click on source folder

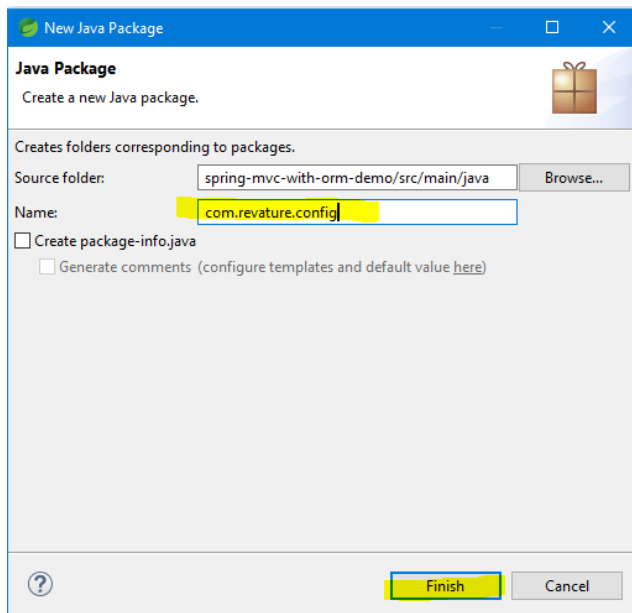
Select "New" → "Package"



Enter the package "Name:"

com.revature.config

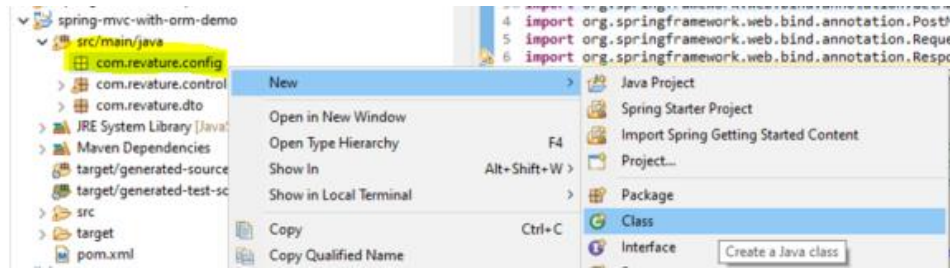
Click "Finish"



Create configuration class

Right click on configuration package

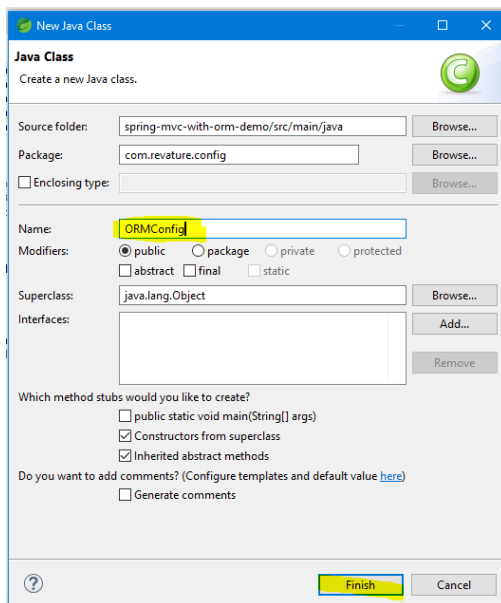
Select "New" → "Class"



Enter class "Name"

ORMConfig

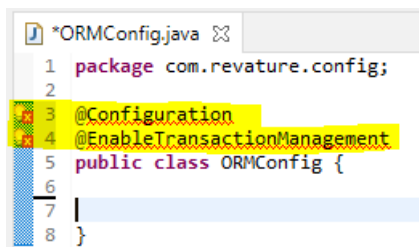
Click "Finish"



Add class annotations to configuration class

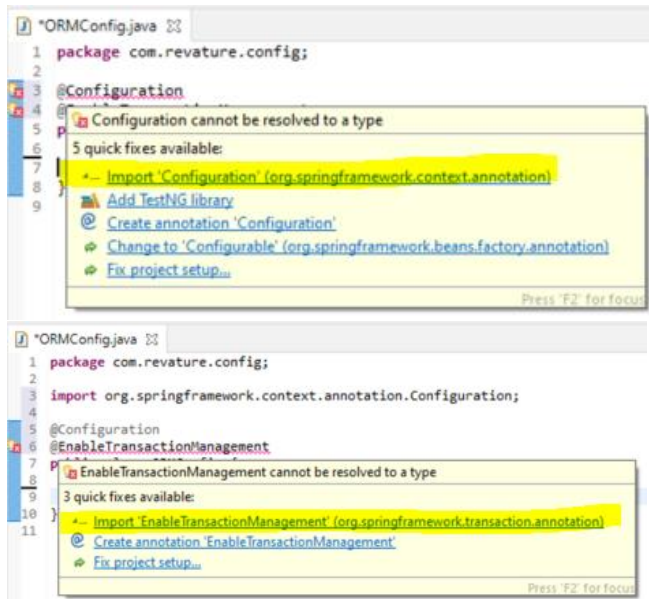
Add configuration and transaction management annotation to the class.

```
@Configuration
@EnableTransactionManagement
```



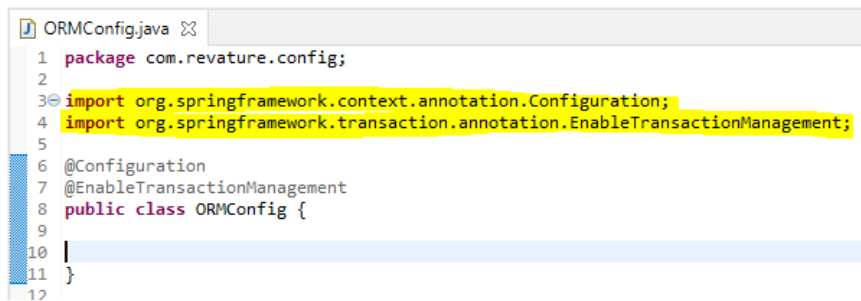
If you type the annotations in the class the imports for each should be added automatically.

If you copy and paste then you can hover the mouse over each error to add the imports



You can also copy and paste these imports:

```
import org.springframework.context.annotation.Configuration;
import org.springframework.transaction.annotation.EnableTransactionManagement;
```



Add three beans from the lecture

Add a data source bean

```
@Bean
public DataSource dataSource() {
    BasicDataSource dataSource = new BasicDataSource();

    dataSource.setDriverClassName("org.mariadb.jdbc.Driver");

    dataSource.setUrl(System.getenv("db_url"));
    dataSource.setUsername(System.getenv("db_username"));
    dataSource.setPassword(System.getenv("db_password"));

    return dataSource;
}
```

```
*ORMConfig.java
1 package com.revature.config;
2
3 import org.springframework.context.annotation.Configuration;
4 import org.springframework.transaction.annotation.EnableTransactionManagement;
5
6 @Configuration
7 @EnableTransactionManagement
8 public class ORMConfig {
9
10     @Bean
11     public DataSource dataSource() {
12         BasicDataSource dataSource = new BasicDataSource();
13
14         dataSource.setDriverClassName("org.mariadb.jdbc.Driver");
15
16         dataSource.setUrl(System.getenv("db_url"));
17         dataSource.setUsername(System.getenv("db_username"));
18         dataSource.setPassword(System.getenv("db_password"));
19
20         return dataSource;
21     }
22
23
24 }
25 }
```

Correct the import issues as shown with red underline in the above screenshot. Either hover the mouse over each and add the import or copy and paste the following imports:

```
import javax.sql.DataSource;
import org.apache.tomcat.dbcp.dbcp2.BasicDataSource;
import org.springframework.context.annotation.Bean;
```

```
ORMConfig.java
1 package com.revature.config;
2
3 import javax.sql.DataSource;
4
5 import org.apache.tomcat.dbcp.dbcp2.BasicDataSource;
6 import org.springframework.context.annotation.Bean;
7 import org.springframework.context.annotation.Configuration;
8 import org.springframework.transaction.annotation.EnableTransactionManagement;
9
```

Verify data source environment variables

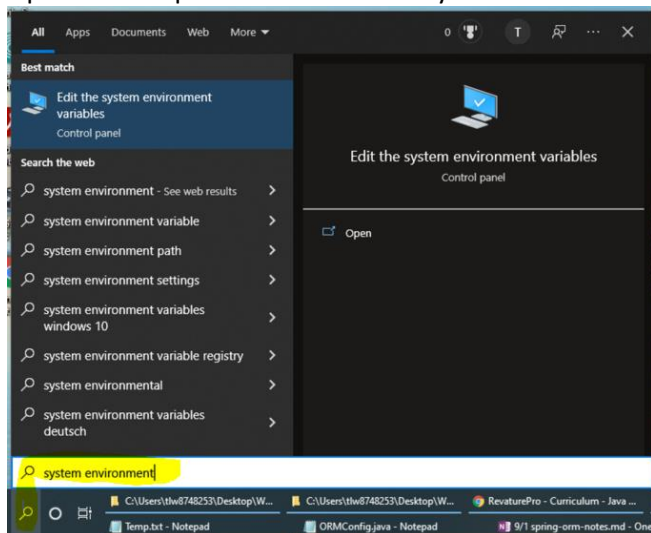
In a previous lecture and project you should have created system environment variables for the database URL, username, and password. The variable names might not be the same as used in the lecture, so you should verify what is on your machine.

```
dataSource.setUrl(System.getenv("db_url"));  
dataSource.setUsername(System.getenv("db_username"));  
dataSource.setPassword(System.getenv("db_password"));
```

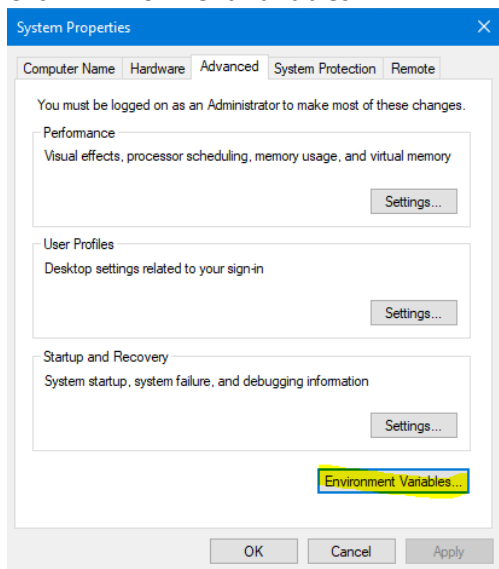
Click on the Window search icon in the task bar

Enter system environment

Open Control panel to the Edit the system environment variables area



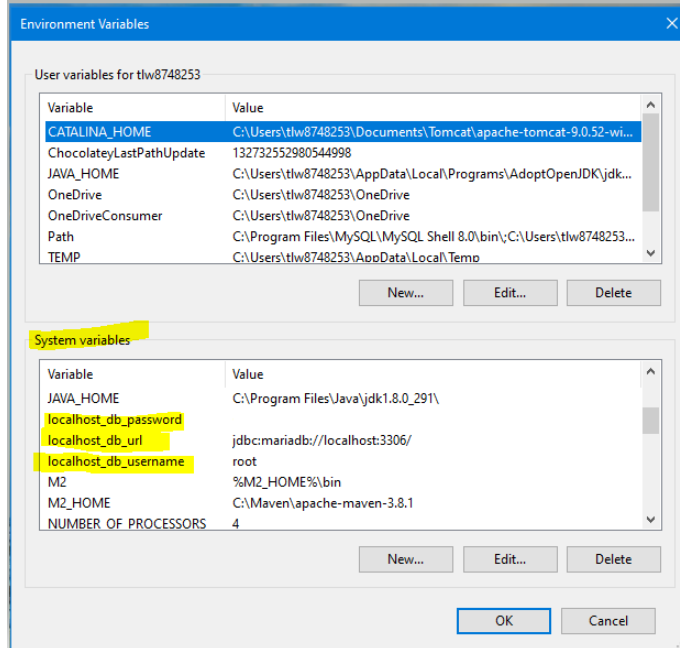
Click “Environment Variables...”



Scroll in the “System variables” area to see what environment variable are defined on your machine.

As you can see mine are different than what is in the lecture.

Make sure your spring-mvc-with-orm-demo are using the database variables defined on your machine.



Remember if you change or add any new environment variables you will need to restart the SpringTool IDE if it is in use.

Add a session factory bean

```
@Bean
public LocalSessionFactoryBean sessionFactory() {
    LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();

    sessionFactory.setDataSource(dataSource()); // setter injection
    sessionFactory.setPackagesToScan("com.revature.model");

    // Create the properties object
    // import from java.util.Properties;
    Properties hibernateProperties = new Properties();

    //Use the create parameter to create the database then switch to validate.
    hibernateProperties.setProperty("hibernate.hbm2ddl.auto", "create");
    //hibernateProperties.setProperty("hibernate.hbm2ddl.auto", "validate");
    hibernateProperties.setProperty("hibernate.dialect", "org.hibernate.dialect.MariaDB103Dialect");

    // Configure hibernate properties
    sessionFactory.setHibernateProperties(hibernateProperties);

    return sessionFactory;
}
```

```

27 @Bean
28 public LocalSessionFactoryBean sessionFactory() {
29     LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
30     sessionFactory.setDataSource(dataSource()); // setter injection
31     sessionFactory.setPackagesToScan("com.revature.model");
32
33     // Create the properties object
34     // import from java.util.Properties;
35     Properties hibernateProperties = new Properties();
36
37     //Use the create parameter to create the database then switch to validate.
38     hibernateProperties.setProperty("hibernate.hbm2ddl.auto", "create");
39     //hibernateProperties.setProperty("hibernate.hbm2ddl.auto", "validate");
40     hibernateProperties.setProperty("hibernate.dialect", "org.hibernate.dialect.MariaDB103Dialect");
41
42     // Configure hibernate properties
43     sessionFactory.setHibernateProperties(hibernateProperties);
44
45     return sessionFactory;
46 }

```

Correct the import issues as shown with red underline in the above screenshot. Either hover the mouse over each and add the import or copy and paste the following imports:

```

import java.util.Properties;
import org.springframework.orm.hibernate5.LocalSessionFactoryBean;

```

```

3 import java.util.Properties;
4
5 import javax.sql.DataSource;
6
7 import org.apache.tomcat.dbcp.dbcp2.BasicDataSource;
8 import org.springframework.context.annotation.Bean;
9 import org.springframework.context.annotation.Configuration;
10 import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
11 import org.springframework.transaction.annotation.EnableTransactionManagement;

```

Add a transaction manager bean

```

@Bean
public PlatformTransactionManager hibernateTransactionManager() {
    HibernateTransactionManager transactionManager = new HibernateTransactionManager();

    transactionManager.setSessionFactory(sessionFactory().getObject());

    return transactionManager;
}

```

```

52 @Bean
53 public PlatformTransactionManager hibernateTransactionManager() {
54     HibernateTransactionManager transactionManager = new HibernateTransactionManager();
55
56     transactionManager.setSessionFactory(sessionFactory().getObject());
57
58     return transactionManager;
59 }

```

Correct the import issues as shown with red underline in the above screenshot. Either hover the mouse over each and add the import or copy and paste the following imports:

```

import org.springframework.orm.hibernate5.HibernateTransactionManager;
import org.springframework.transaction.PlatformTransactionManager;

```

```
3- import java.util.Properties;
4
5 import javax.sql.DataSource;
6
7 import org.apache.tomcat.dbcp.dbcp2.BasicDataSource;
8 import org.springframework.context.annotation.Bean;
9 import org.springframework.context.annotation.Configuration;
10 import org.springframework.orm.hibernate5.HibernateTransactionManager;
11 import org.springframework.orm.hibernate5.LocalSessionFactoryBean;
12 import org.springframework.transaction.PlatformTransactionManager;
13 import org.springframework.transaction.annotation.EnableTransactionManagement;
```

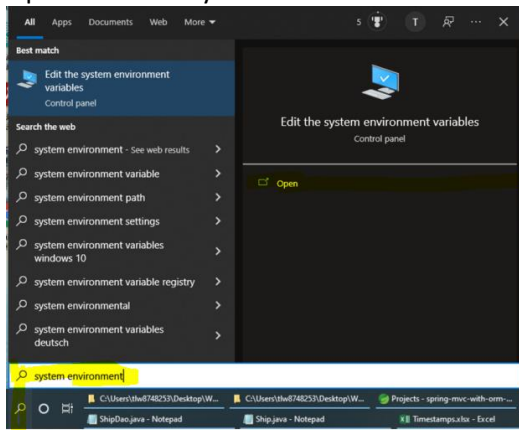

Setup a database for the demo

Change the system environment variable

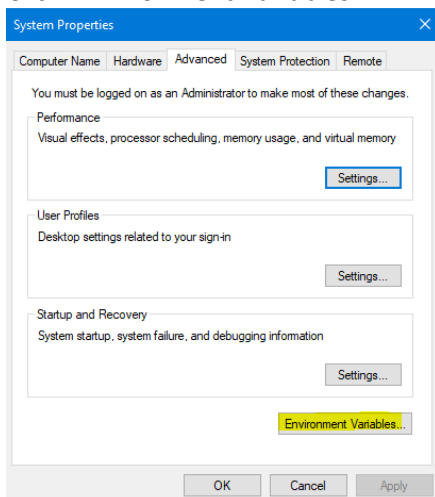
Open System Properties

Search for system environment in the Window search window

Open “Edit the system environment variables”

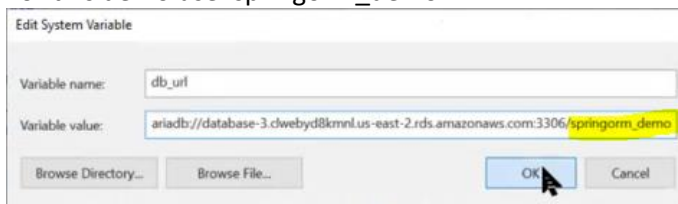


Click “Environment Variables...”



Change the database name in the db_url environment variable to a different database.

For this demo use: springorm_demo.



See “[Appendix: Alternative to db_url environment variable](#)” for a different way to setup your environment variables.

Create the demo database

Open DBeaver

Right click "Databases" → "Create New Database"

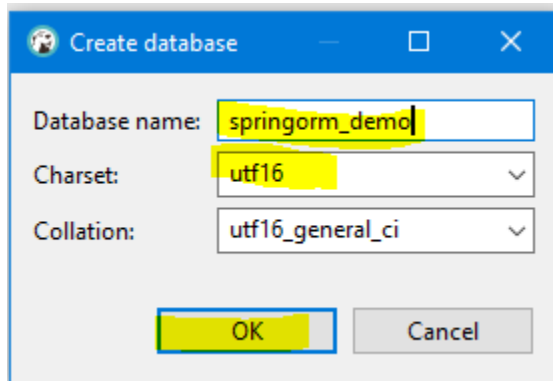


Enter "Database name:" springorm_demo

Select "Charset:" utf16

"Collation:" should automatically change to utf16_general_ci

Click "OK"



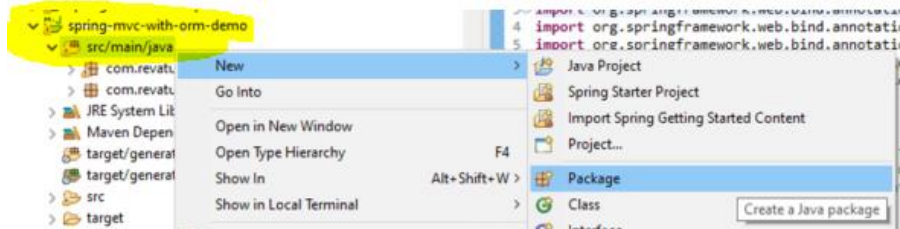
A database shell is created for this demo. The tables will be created in a later step after creating a class for the model package.

Create DAO and Model packages

Create DAO package

Right click on source folder

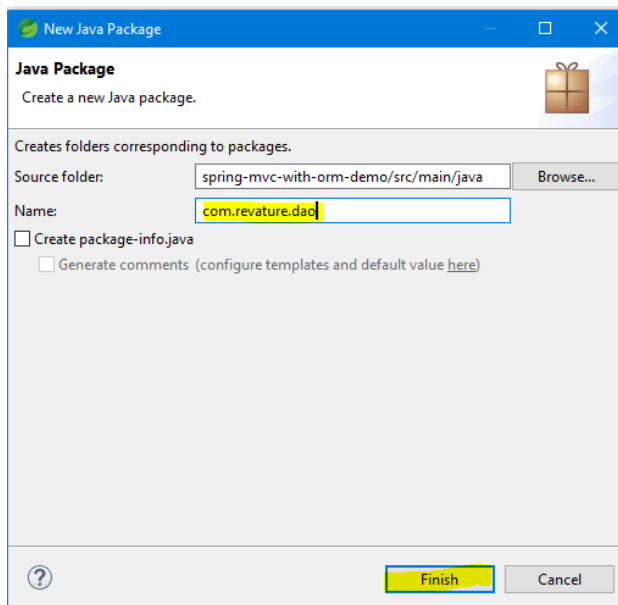
Select "New" → "Package"



Enter the package "Name:"

com.revature.dao

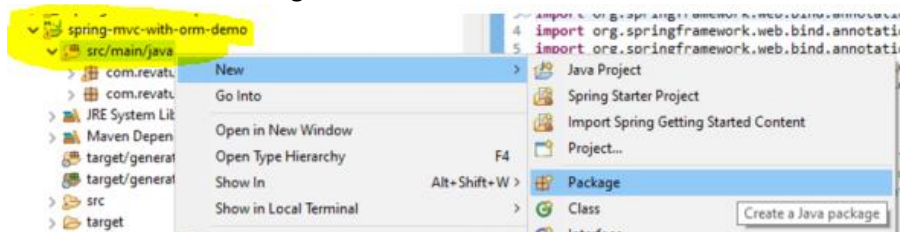
Click "Finish"



Create model package

Right click on source folder

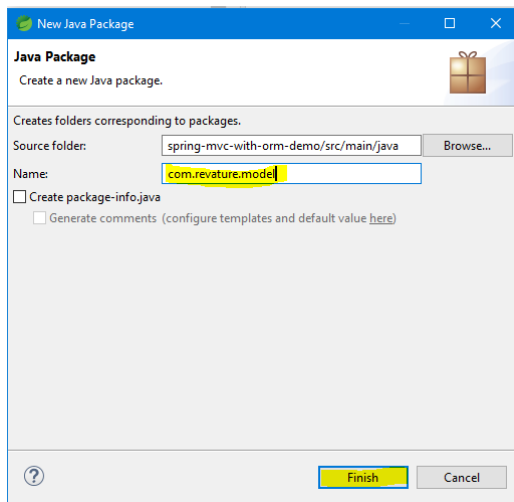
Select "New" → "Package"



Enter the package "Name:"

com.revature.model

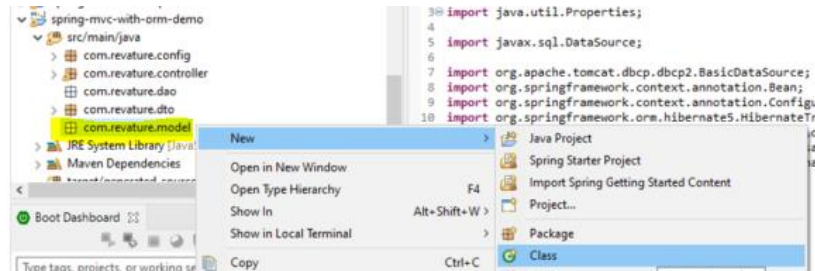
Click “Finish”



Create model class

Right click on model package

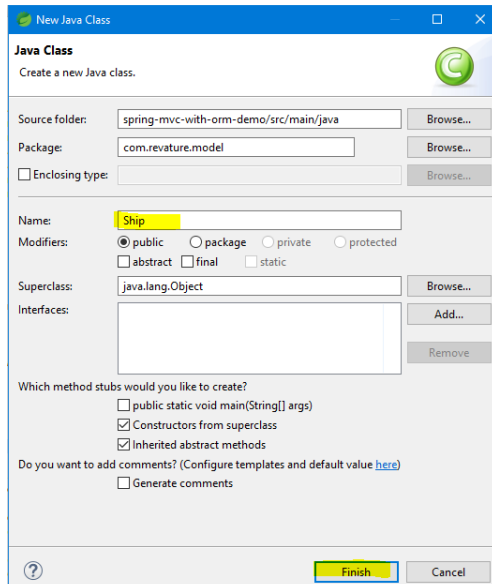
Select “New” → “Class”



Enter class “Name”

Ship

Click “Finish”



Complete the model class Ship

The Ship model class will use the Hibernate notation learned and used in previous lecture and for Project 1. It will also use the Lombok notation learned in the Aug 31st lecture.

Hibernate notation – will create / update the object in the database.

```
@Entity
@Table(name = "table_name")
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "column_name")
```

Lombok notation – will indicate the use of Getters, Setters, others without generating the actual code.

```
@Getter @Setter @NoArgsConstructor @EqualsAndHashCode @ToString
```

Update the base Ship class code generated when created

If you choose to type each line of class code beginning with @Entity, the imports should automatically be added. You can also copy and paste all the code below and replace the class shell code generate when the class was created.

```
package com.revature.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Entity
@Table(name = "ship")
@Getter @Setter @NoArgsConstructor @EqualsAndHashCode @ToString
public class Ship {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "ship_name")
    private String shipName;

    @Column(name = "age")
    private int age;

    public Ship(String name, int age) {
        this.shipName = name;
        this.age = age;
    }
}
```

```

1 package com.revature.model;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.Id;
8 import javax.persistence.Table;
9
10 import lombok.EqualsAndHashCode;
11 import lombok.Getter;
12 import lombok.NoArgsConstructor;
13 import lombok.Setter;
14 import lombok.ToString;
15
16 @Entity
17 @Table(name = "ship")
18 @Getter @Setter @NoArgsConstructor @EqualsAndHashCode @ToString
19 public class Ship {
20
21     @Id
22     @GeneratedValue(strategy = GenerationType.IDENTITY)
23     private int id;
24
25     @Column(name = "ship_name")
26     private String shipName;
27
28     @Column(name = "age")
29     private int age;
30
31     public Ship(String name, int age) {
32         this.shipName = name;
33         this.age = age;
34     }
35 }

```

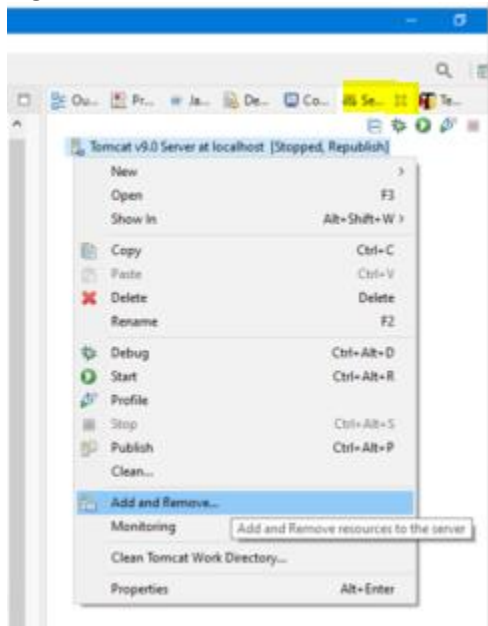
Test the database connection

Add this project to the IDE Tomcat server

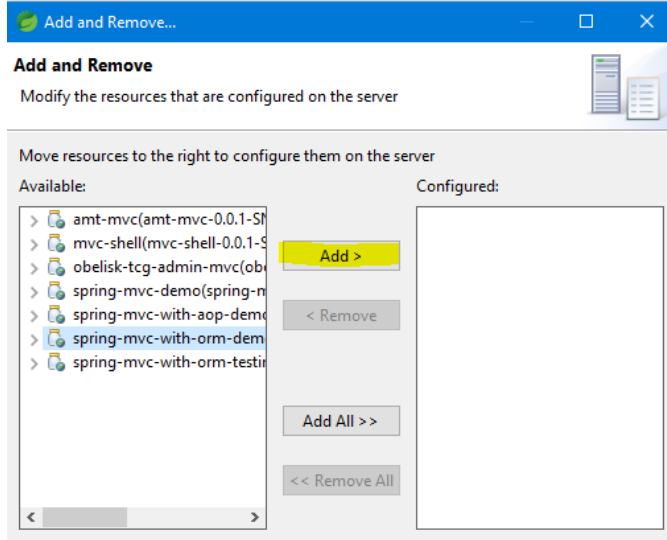
The location of the Server tab and window can vary depending on your IDE layout.

Select the Server tab

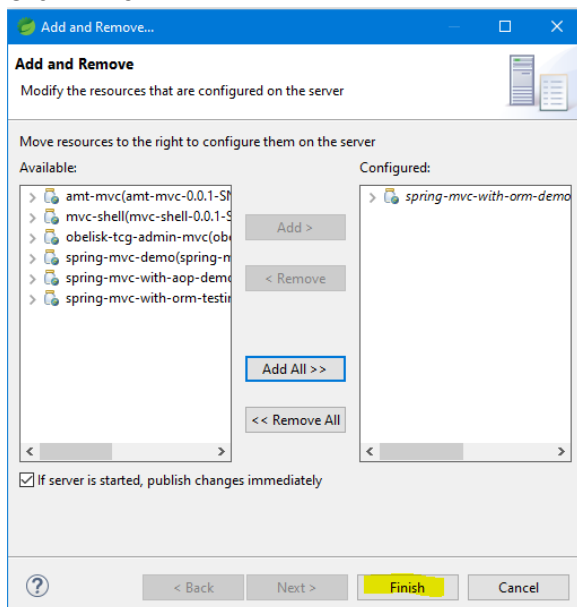
Right click "Tomcat ... Server ..." → "Add and Remove..."



Select this project “spring-mvc-with-orm-demo” from “Available:” window.
Click “Add >”




Click “Finish”

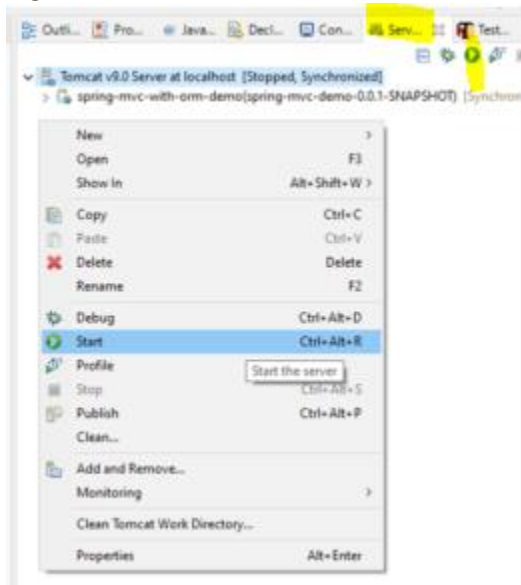


Verify the sessionFactory() method in ORMConfig class is set to “create”

```
36 @Bean
37 public LocalSessionFactoryBean sessionFactory() {
38     LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
39
40     sessionFactory.setDataSource(dataSource()); // setter injection
41     sessionFactory.setPackagesToScan("com.revature.model");
42
43     // Create the properties object
44     // import from java.util.Properties;
45     Properties hibernateProperties = new Properties();
46
47     //Use the create parameter to create the database then switch to validate.
48     hibernateProperties.setProperty("hibernate.hbm2ddl.auto", "create");
```

Start the Tomcat server

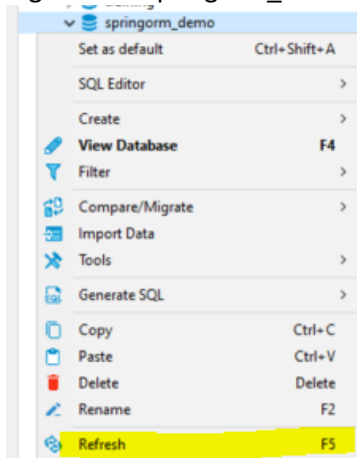
In the server tab press the green start button  or
Right click "Tomcat ... Server ..." → "Start"



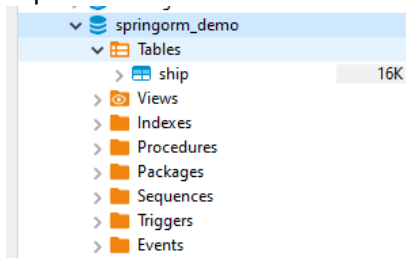
Once the server has fully started, a table should be created in the database.

Verify the database in DBeaver

Right click "springorm_demo" → "Refresh"



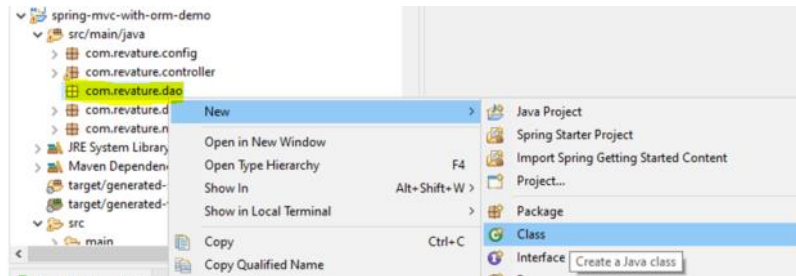
Expand the database and "Tables"



Create DAO class

Right click on DAO package

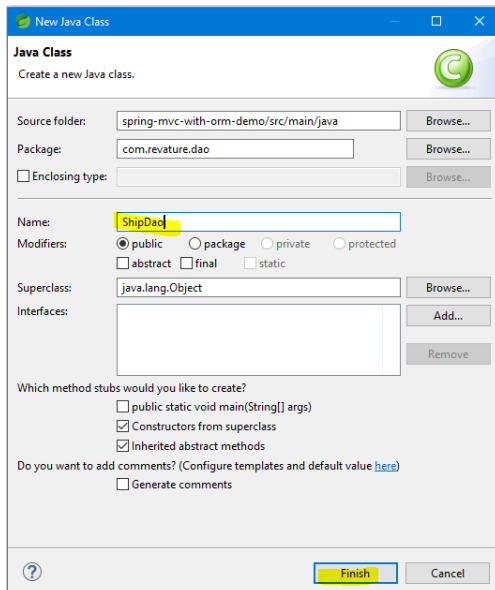
Select "New" → "Class"



Enter class "Name"

ShipDao

Click "Finish"



Complete the DAO class ShipDao

While completing this class a DTO class is needed. This section will show an alternative way of creating a new package and class as the dependency occurs.

The DAO class will use SpringFramework related notation:

```
@Autowired - injection of the sessionFactory bean
@Transactional - identifies and handles database related functions
```

Update the base ShipDao class code generated when created

If you choose to type each line of class code beginning with @Repository, the imports should automatically be added. You can also copy and paste all the code below and replace the class shell code generate when the class was created. Either way you will still have an error (red line in the IDE) the ship DTO class. This will be resolved using an alternate / shortcut way.

```
package com.revature.dao;
```

```

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.revature.dto.AddShipDTO;
import com.revature.model.Ship;

@Repository
public class ShipDao {

    @Autowired
    private SessionFactory sessionFactory;

    @Transactional // by default, this method will use the outer transaction or create a
new transaction if non is available
    public Ship addShip(AddShipDTO addShipDto) {
        Session session = sessionFactory.getCurrentSession();

        Ship newShip = new Ship(addShipDto.getName(), addShipDto.getAge());

        session.persist(newShip);

        return newShip;
    }

    @Transactional
    public List<Ship> getAllShips() {
        Session session = sessionFactory.getCurrentSession();

        List<Ship> ships = session.createQuery("FROM Ship s").getResultList();

        return ships;
    }
}

```

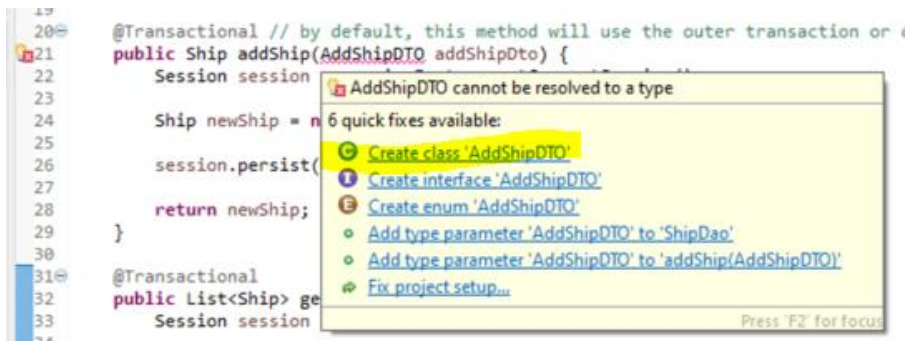
After completing the code, there are errors related to the ship DTO.

```
ShipDao.java
1 package com.revature.dao;
2
3 import { spring-mvc-with-orm-demo/src/main/java/com/revature/dao/ShipDao.java
4
5 import org.hibernate.Session;
6 import org.hibernate.SessionFactory;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.stereotype.Repository;
9 import org.springframework.transaction.annotation.Transactional;
10
11 import com.revature.dto.AddShipDTO;
12 import com.revature.model.Ship;
13
14 @Repository
15 public class ShipDao {
16
17     @Autowired
18     private SessionFactory sessionFactory;
19
20     @Transactional // by default, this method will use the outer transaction or create a new transaction if non is available
21     public Ship addShip(AddShipDTO addShipDto) {
22         Session session = sessionFactory.getCurrentSession();
23
24         Ship newShip = new Ship(addShipDto.getName(), addShipDto.getAge());
25
26         session.persist(newShip);
27
28         return newShip;
29     }
30
31     @Transactional
32     public List<Ship> getAllShips() {
33         Session session = sessionFactory.getCurrentSession();
34
35         List<Ship> ships = session.createQuery("FROM Ship s").getResultList();
36
37         return ships;
38     }
39
40 }
```

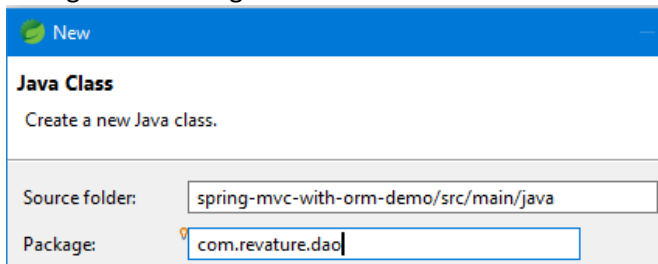
Create DTO class

Either after copying and paste of the DAO class or as you typed the code, the error on line 21 above is encountered.

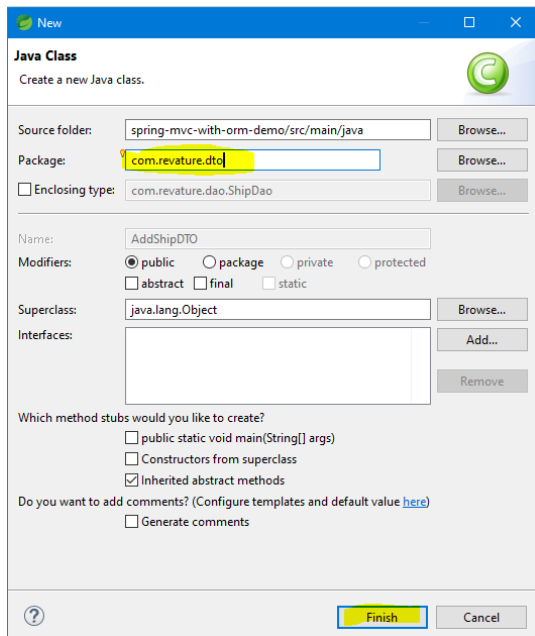
Hover your mouse over the error / red line and select “Create class ‘AddShipDTO’”.



Change the “Package:” to .dto



Click “Finish”



Complete the DTO class ShipDTO

The DTO class will use the Lombok notation –indicate the use of Getters, Setters, others without generating the actual code.

```
@Getter @Setter @NoArgsConstructor @EqualsAndHashCode @ToString
```

Update the base ShipDTO class code generated when created

If you choose to type each line of class code beginning with @Getter, the imports should automatically be added. You can also copy and paste all the code below and replace the class shell code generate when the class was created.

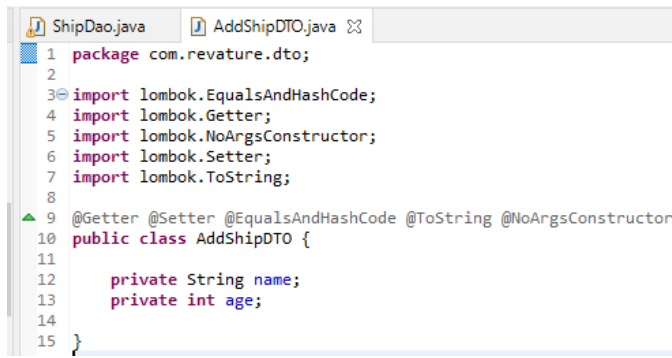
```
package com.revature.dto;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Getter @Setter @EqualsAndHashCode @ToString @NoArgsConstructor
public class AddShipDTO {

    private String name;
    private int age;

}
```



```
1 package com.revature.dto;
2
3 import lombok.EqualsAndHashCode;
4 import lombok.Getter;
5 import lombok.NoArgsConstructor;
6 import lombok.Setter;
7 import lombok.ToString;
8
9 @Getter @Setter @EqualsAndHashCode @ToString @NoArgsConstructor
10 public class AddShipDTO {
11
12     private String name;
13     private int age;
14
15 }
```

Once the AddShipDTO class is completed the errors in the ShipDao should be resolved.

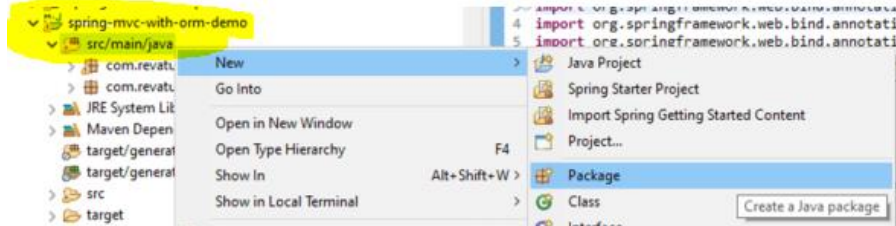
Create Service package and class

In the lecture, the service package and a test service class exists and the class was deleted. I believe the service class was created to show some problematic functionality that will not be used. This document does not show the creation of the TestService class. It is not needed in the final demo. If your project happens to have the class, it can be deleted.

Create Service package

Right click on source folder

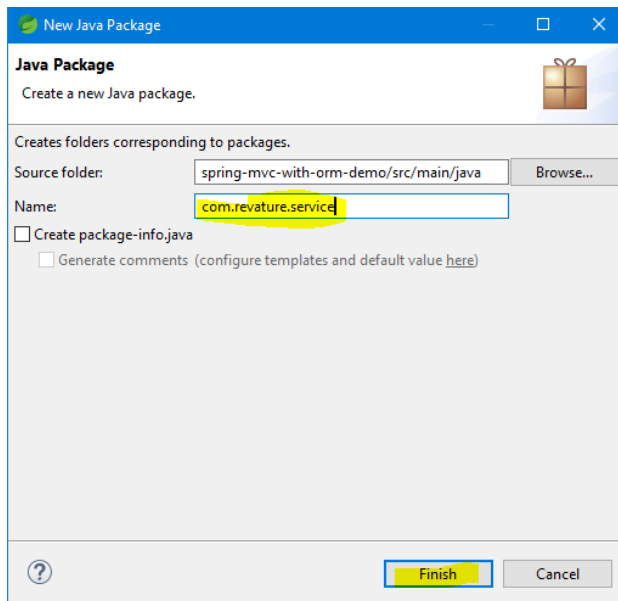
Select “New” → “Package”



Enter the package “Name:”

com.revature.service

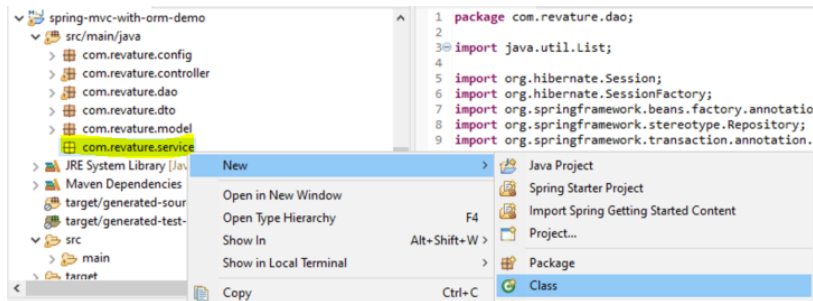
Click “Finish”



Create service class

Right click on service package

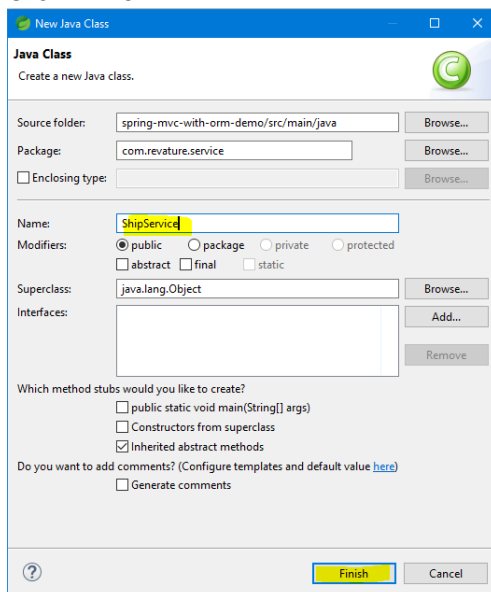
Select “New” → “Class”



Enter class “Name”

ShipService

Click “Finish”



Complete the service class ShipService

The Service class will use SpringFramework related notation:

@Service - annotates classes that perform service tasks
@Autowired - injection of the DAO class (constructor injection)

Update the base ShipService class code generated when created

If you choose to type each line of class code beginning with @Service, the imports should automatically be added. You can also copy and paste all the code below and replace the class shell code generate when the class was created.

```
package com.revature.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

```
import com.revature.dao.ShipDao;
import com.revature.dto.AddShipDTO;
import com.revature.exception.BadParameterException;
import com.revature.exception.ShipNotFoundException;
import com.revature.model.Ship;

@Service
public class ShipService {

    private ShipDao shipDao;

    @Autowired
    public ShipService(ShipDao shipDao) {
        this.shipDao = shipDao;
    }

    public Ship addShip(AddShipDTO addShipDto) throws BadParameterException {
        if (addShipDto.getName().trim().equals("")) {
            throw new BadParameterException("You cannot have a blank name for a
ship");
        }

        if (addShipDto.getAge() < 0) {
            throw new BadParameterException("You cannot have a negative age for a
ship");
        }

        Ship ship = shipDao.addShip(addShipDto);

        return ship;
    }

    public List<Ship> getAllShips() throws ShipNotFoundException {
        List<Ship> ships = shipDao.getAllShips();

        if (ships.size() == 0) {
            throw new ShipNotFoundException("No ships were found in the system");
        }

        return ships;
    }
}
```


Similar to the DAO class there are dependencies with classes not created yet. You can use the alternate method as you type each line to create the package and class dependencies. You can also create the package and classes after copy and paste of the entire service class. This document will show creating the dependent package and classes after copy and paste of service class.

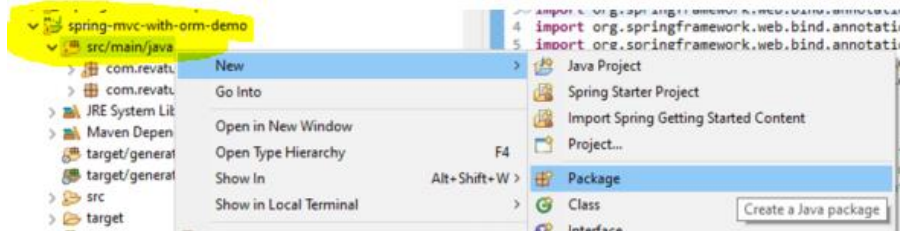
```
ShipService.java
1 package com.revature.service;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import com.revature.dao.ShipDao;
9 import com.revature.dto.AddShipDTO;
10 import com.revature.exception.BadParameterException;
11 import com.revature.exception.ShipNotFoundException;
12 import com.revature.model.Ship;
13
14 @Service
15 public class ShipService {
16     private ShipDao shipDao;
17
18     @Autowired
19     public ShipService(ShipDao shipDao) {
20         this.shipDao = shipDao;
21     }
22
23     public Ship addShip(AddShipDTO addShipDto) throws BadParameterException {
24         if (addShipDto.getName().trim().equals("")) {
25             throw new BadParameterException("You cannot have a blank name for a ship");
26         }
27
28         if (addShipDto.getAge() < 0) {
29             throw new BadParameterException("You cannot have a negative age for a ship");
30         }
31
32         Ship ship = shipDao.addShip(addShipDto);
33
34         return ship;
35     }
36
37     public List<Ship> getAllShips() throws ShipNotFoundException {
38         List<Ship> ships = shipDao.getAllShips();
39
40         if (ships.size() == 0) {
41             throw new ShipNotFoundException("No ships were found in the system");
42         }
43
44         return ships;
45     }
46 }
47
48 }
```

Create Exception package and classes

Create Service package

Right click on source folder

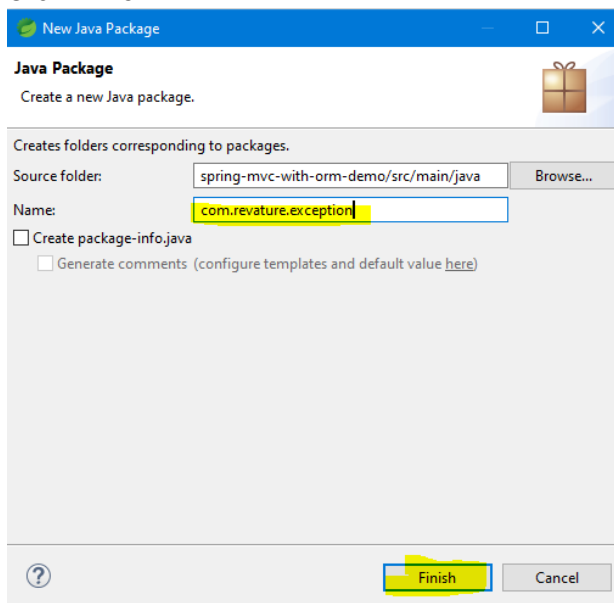
Select "New" → "Package"



Enter the package "Name:"

com.revature.exception

Click "Finish"



Create exception classes

Create bad parameter exception

This class is created from the ShipService class error. This will show the alternate method of creating the class as you type and encounter the error. It will also show creating the exception class with the wizard instead of copy and paste.

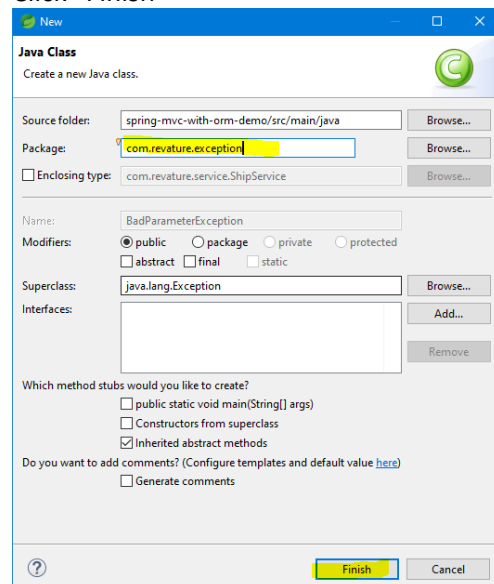
Hover mouse over BadParameterException redline → click “Create class ‘BadParameterException’”



Change “Package:” to

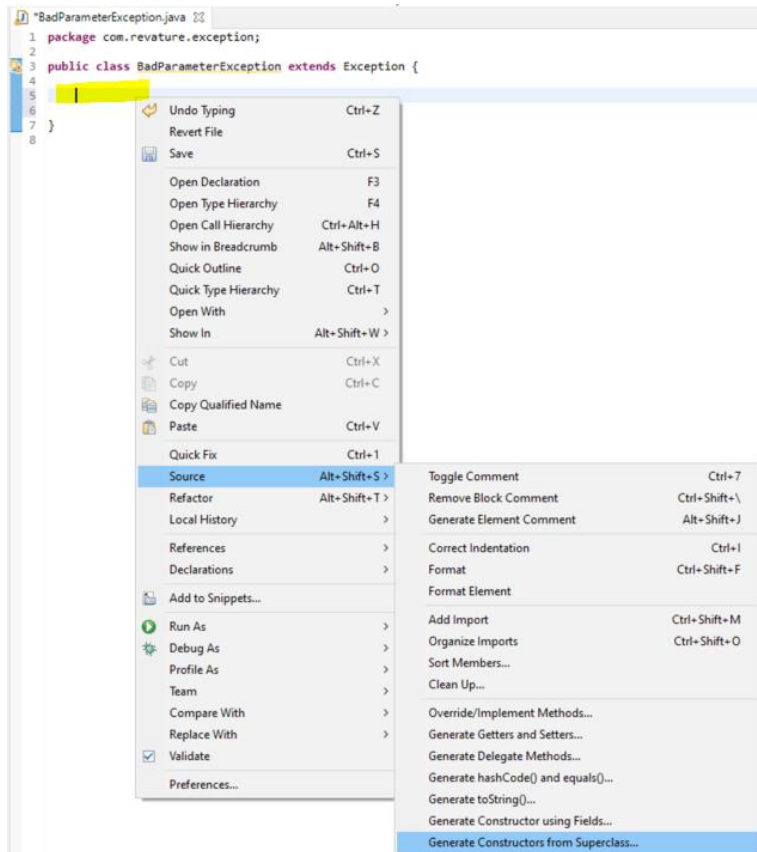
com.revature.exception

Click “Finish”

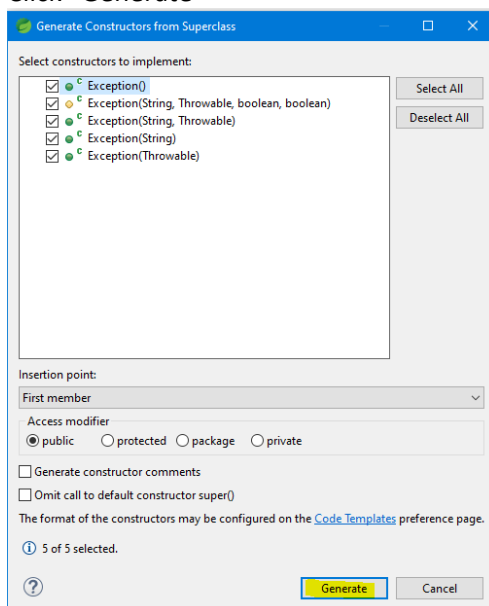


Use the source wizard to generate methods.

Right click inside the class ➔ "Source" ➔ "Generate Constructors from Superclass..."



Click "Generate"



Clean up auto generated code removing the TODO lines:

```
1 package com.revature.exception;
2
3 public class BadParameterException extends Exception {
4
5     public BadParameterException() {
6         super();
7         // TODO Auto-generated constructor stub
8     }
9
10    public BadParameterException(String message, Throwable cause, boolean enableSuppression,
11        boolean writableStackTrace) {
12        super(message, cause, enableSuppression, writableStackTrace);
13        // TODO Auto-generated constructor stub
14    }
15
16    public BadParameterException(String message, Throwable cause) {
17        super(message, cause);
18        // TODO Auto-generated constructor stub
19    }
20
21    public BadParameterException(String message) {
22        super(message);
23        // TODO Auto-generated constructor stub
24    }
25
26    public BadParameterException(Throwable cause) {
27        super(cause);
28        // TODO Auto-generated constructor stub
29    }
30 }
```

```
1 package com.revature.exception;
2
3 public class BadParameterException extends Exception {
4
5     public BadParameterException() {
6         super();
7     }
8
9     public BadParameterException(String message, Throwable cause, boolean enableSuppression,
10        boolean writableStackTrace) {
11        super(message, cause, enableSuppression, writableStackTrace);
12    }
13
14    public BadParameterException(String message, Throwable cause) {
15        super(message, cause);
16    }
17
18    public BadParameterException(String message) {
19        super(message);
20    }
21
22    public BadParameterException(Throwable cause) {
23        super(cause);
24    }
25
26 }
27 }
```

If you are typing in the code in ShipService and correcting the BadParameterException error as you go, you will need to add the throws declaration.

In the ShipService class:

Hover your mouse over the redline → “Add throws declaration”

```
19 public Ship addShip(AddShipDTO addShipDto) {
20     if (addShipDto.getName().trim().equals("")) {
21         throw new BadParameterException("You cannot have a blank name for a ship");
22     }
23 }
24
25 }
26 }
```

Unhandled exception type BadParameterException
1 quick fix available:
Add throws declaration

The throws declaration is added. You will need to finish the method with the two other highlighted line to get rid of the error / redline.

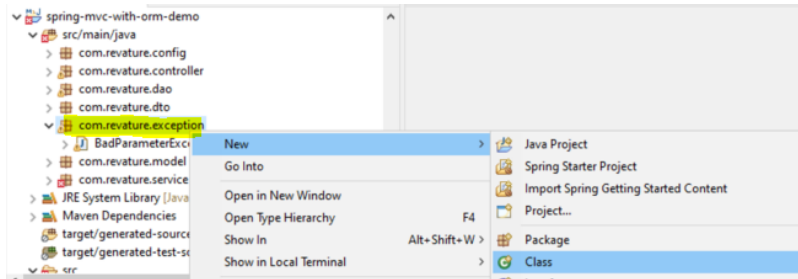
```
19 public Ship addShip(AddShipDTO addShipDto) throws BadParameterException {
20     if (addShipDto.getName().trim().equals("")) {
21         throw new BadParameterException("You cannot have a blank name for a ship");
22     }
23
24     Ship ship = shipDao.addShip(addShipDto);
25
26     return ship;
27 }
```

Create not found exception

This class is created using copy and paste.

Right click on exception package

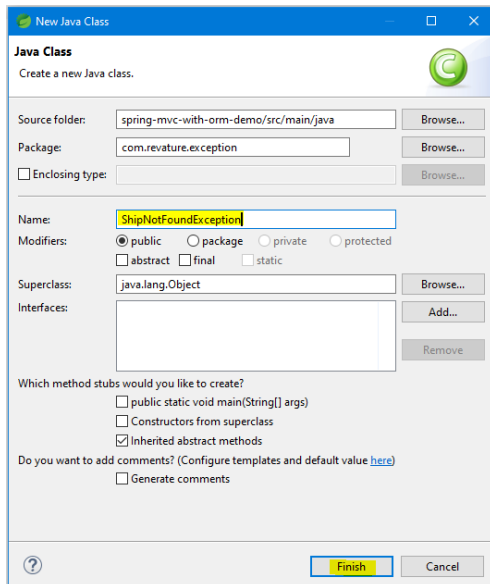
Select "New" → "Class"



Enter class "Name"

ShipNotFoundException

Click "Finish"



Complete the service class ShipNotFoundException

You can copy and paste all the code below replacing the shell generated when creating the class.

```
package com.revature.exception;

public class ShipNotFoundException extends Exception {

    public ShipNotFoundException() {
    }

    public ShipNotFoundException(String message) {
        super(message);
    }

    public ShipNotFoundException(Throwable cause) {
        super(cause);
    }

    public ShipNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

```

    }

    public ShipNotFoundException(String message, Throwable cause, boolean
enableSuppression,
                                boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
}

```

```

ShipNotFoundException.java
1 package com.revature.exception;
2
3 public class ShipNotFoundException extends Exception {
4
5     public ShipNotFoundException() {
6     }
7
8     public ShipNotFoundException(String message) {
9         super(message);
10    }
11
12    public ShipNotFoundException(Throwable cause) {
13        super(cause);
14    }
15
16    public ShipNotFoundException(String message, Throwable cause) {
17        super(message, cause);
18    }
19
20    public ShipNotFoundException(String message, Throwable cause, boolean enableSuppression,
21                                boolean writableStackTrace) {
22        super(message, cause, enableSuppression, writableStackTrace);
23    }
24
25 }

```

The errors in the ShipService class should now be corrected.

Update Controller package and classes

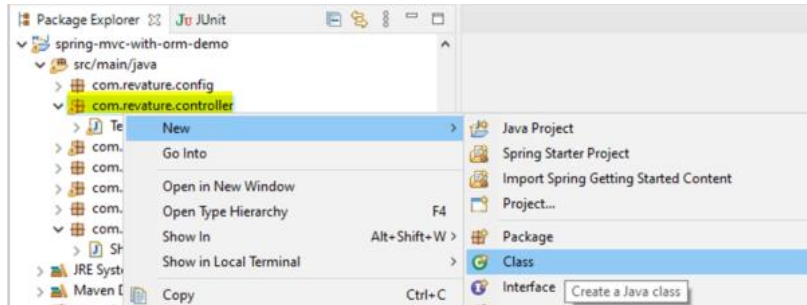
Update TestController class

If this class contains a reference to a TestService class, it should be removed since the TestService class in the service package was removed or never created.

Create ShipController class

Right click on controller package

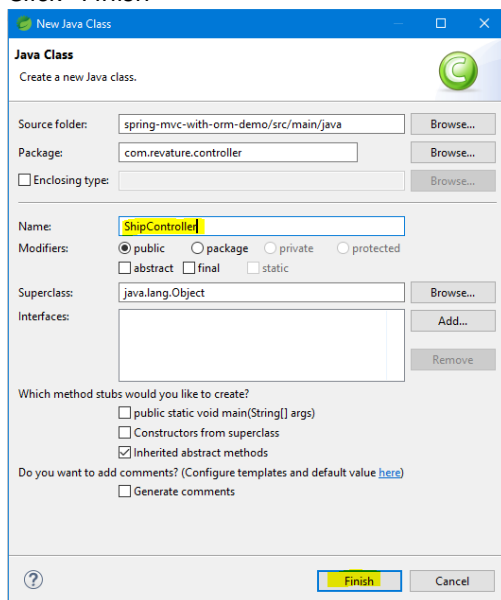
Select “New” → “Class”



Enter class “Name”

ShipController

Click “Finish”



Complete the controller class ShipController

The Controller class will use SpringFramework related notation:

```
@RestController - allows to handle all REST APIs such as GET, POST, Delete, PUT requests
@CrossOrigin - method or type as permitting cross origin requests
@Autowired - injection of the DAO class (constructor injection)
@PostMapping - annotation for mapping HTTP POST requests onto specific handler methods
@GetMapping - annotation for mapping HTTP GET requests onto specific handler methods
```


Update the base ShipController class code generated when created

If you choose to type each line of class code beginning with `@RestController`, the imports should automatically be added. You can also copy and paste the code below as presented in a sequence below. This class will be built in steps to demonstrate alternate ways to use coding wizards.

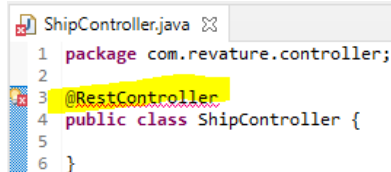
This is the auto generated base code after creating the class in the previous step.

```
package com.revature.controller;

public class ShipController {

}
```

Add `@RestController` annotation to the class.

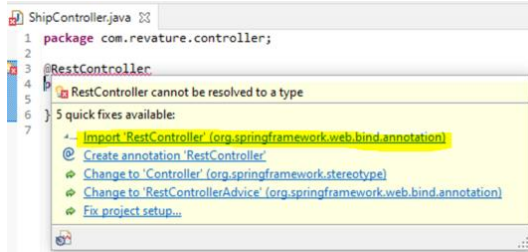


```
ShipController.java
1 package com.revature.controller;
2
3 @RestController
4 public class ShipController {
5
6 }
```

Use the wizard to add the `RestController` import.

Hover mouse over error / redline

Click on the import hyperlink highlighted below



```
ShipController.java
1 package com.revature.controller;
2
3 @RestController
4 public class ShipController {
5
6 }
```

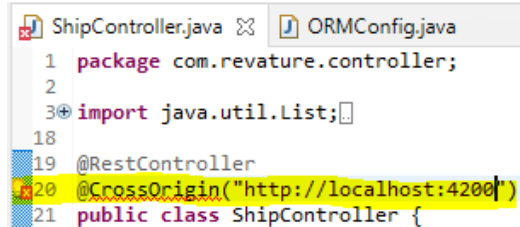
RestController cannot be resolved to a type
5 quick fixes available:

- Import 'RestController' (org.springframework.web.bind.annotation)
- Create annotation 'RestController'
- Change to 'Controller' (org.springframework.stereotype)
- Change to 'RestControllerAdvice' (org.springframework.web.bind.annotation)
- Fix project setup...

Add `@CrossOrigin` annotation to the class

```
@CrossOrigin("http://localhost:4200")
```

By default an Angular application will look for port 4200. During the start up of the Angular application port 4200 might be in use. If so the code should be change to another port like 4201.

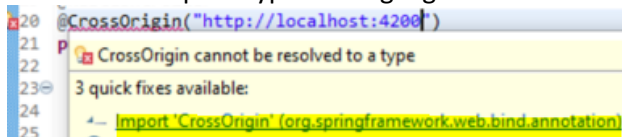


```
ShipController.java
1 package com.revature.controller;
2
3 import java.util.List;
4
5 @RestController
6 @CrossOrigin("http://localhost:4200")
7 public class ShipController {
8
9 }
```

Use the wizard to add the `CrossOrigin` import.

Hover mouse over error / redline

Click on the import hyperlink highlighted below



```
ShipController.java
1 package com.revature.controller;
2
3 import java.util.List;
4
5 @RestController
6 @CrossOrigin("http://localhost:4200")
7 public class ShipController {
8
9 }
```

CrossOrigin cannot be resolved to a type
3 quick fixes available:

- Import 'CrossOrigin' (org.springframework.web.bind.annotation)

You can also add the following code documentation related to @CrossOrigin:

```
//Whenever our browser makes a request
//to the backend, this annotation is what the backend will use to tell the browser
//that the source of the JavaScript code that is sending the request to the backend
//is a "trusted" source
//CORS is a security feature on most modern browsers
//We don't have CORS issues with Postman, because Postman doesn't care
//However, browsers are used by consumers who we should definitely be caring about in terms
//of providing a secure browsing experience

//The reason CORS exists is because users might visit malicious websites that have malicious
//JavaScript that will attempt to send HTTP requests to a backend that is not in their control

//So the backend will inform the browser about what JavaScript sources are actually trusted
```

```
20 @RestController
21 @CrossOrigin("http://localhost:4200")
22 //Whenever our browser makes a request
23 //to the backend, this annotation is what the backend will use to tell the browser
24 //that the source of the JavaScript code that is sending the request to the backend
25 //is a "trusted" source
26 //CORS is a security feature on most modern browsers
27 //We don't have CORS issues with Postman, because Postman doesn't care
28 //However, browsers are used by consumers who we should definitely be caring about in terms
29 //of providing a secure browsing experience
30
31 //The reason CORS exists is because users might visit malicious websites that have malicious
32 //JavaScript that will attempt to send HTTP requests to a backend that is not in their control
33
34 //So the backend will inform the browser about what JavaScript sources are actually trusted
35 public class ShipController {
```

Add an Autowired class variable

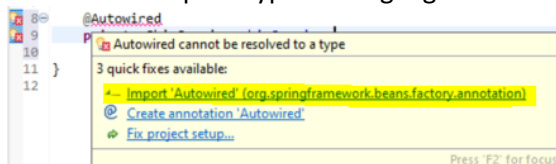
```
@Autowired
private ShipService shipService;
```

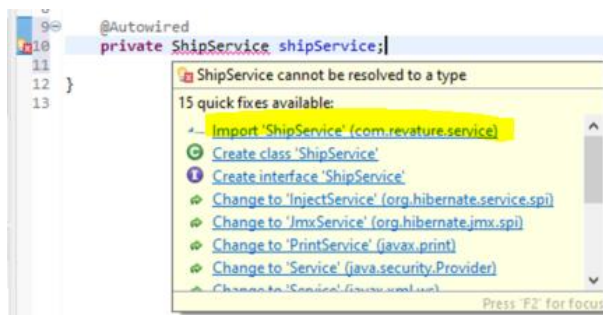
```
*ShipController.java
1 package com.revature.controller;
2
3 import org.springframework.web.bind.annotation.RestController;
4
5 @RestController
6 public class ShipController {
7
8     @Autowired
9     private ShipService shipService;
10
11 }
12
```

Correct the import errors.

Hover mouse over an error / redline

Click on the import hyperlink highlighted below





Add a post API using PostMapping annotation.

```

@PostMapping(path = "/ship")
public ResponseEntity<Object> addShip(@RequestBody AddShipDTO addShipDTO) {

    Ship ship = shipService.addShip(addShipDTO);

}

```

```

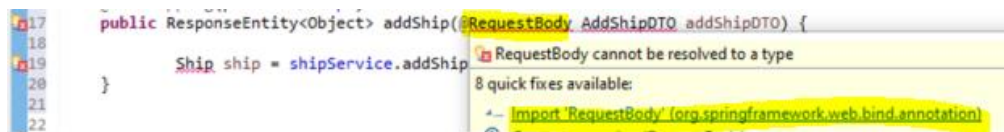
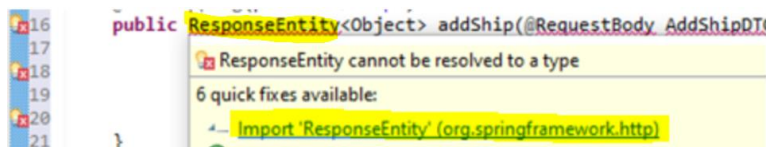
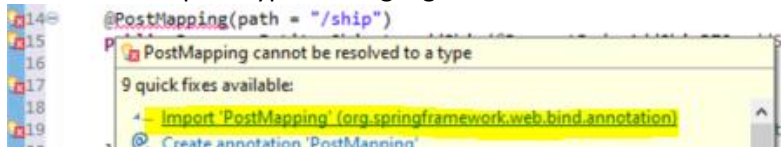
14 @PostMapping(path = "/ship")
15 public ResponseEntity<Object> addShip(@RequestBody AddShipDTO addShipDTO) {
16
17     Ship ship = shipService.addShip(addShipDTO);
18 }

```

Correct the import errors.

Hover mouse over an error / redline

Click on the import hyperlink highlighted below



Once all the import errors are corrected you will noticed new errors. One error is related to the ship service method throwing an exception. The other error is related to the method needing to return a value. We will take care of both by handing the exception error. We can use the wizard to add a try / catch block.

```
19 @PostMapping(path = "/ship")
20 public ResponseEntity<Object> addShip(@RequestBody AddShipDTO addShipDTO) {
21
22     Ship ship = shipService.addShip(addShipDTO);
23 }
24
```

This method must return a result of type ResponseEntity< Object>

Unhandled exception type BadParameterException

Hover mouse over an error / redline
Select "Surround with try/catch"

```
22     Ship ship = shipService.addShip(addShipDTO);
23 }
24
25
26
27
```

Unhandled exception type BadParameterException

2 quick fixes available:

- Add throws declaration
- Surround with try/catch

Update the try / catch block that was created

```
23     try {
24         Ship ship = shipService.addShip(addShipDTO);
25     } catch (BadParameterException e) {
26         // TODO Auto-generated catch block
27         e.printStackTrace();
28     }
29
```

Add a successful response code.

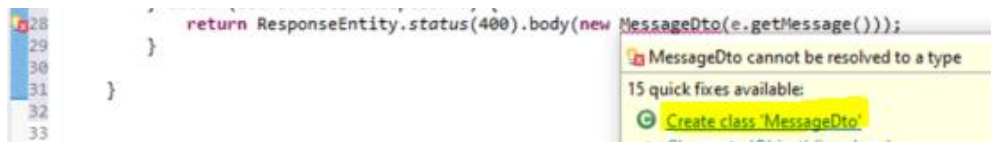
Replace the printStackTrace with an error response code.

```
    try {
        Ship ship = shipService.addShip(addShipDTO);
        return ResponseEntity.status(201).body(ship); // 201 created
    } catch (BadParameterException e) {
        return ResponseEntity.status(400).body(new MessageDto(e.getMessage()));
    }
}
```

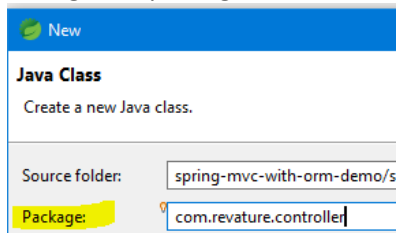
With the error response we want to use a new message DTO.

```
23     try {
24         Ship ship = shipService.addShip(addShipDTO);
25
26         return ResponseEntity.status(201).body(ship); // 201 created
27     } catch (BadParameterException e) {
28         return ResponseEntity.status(400).body(new MessageDto(e.getMessage()));
29     }
30
```

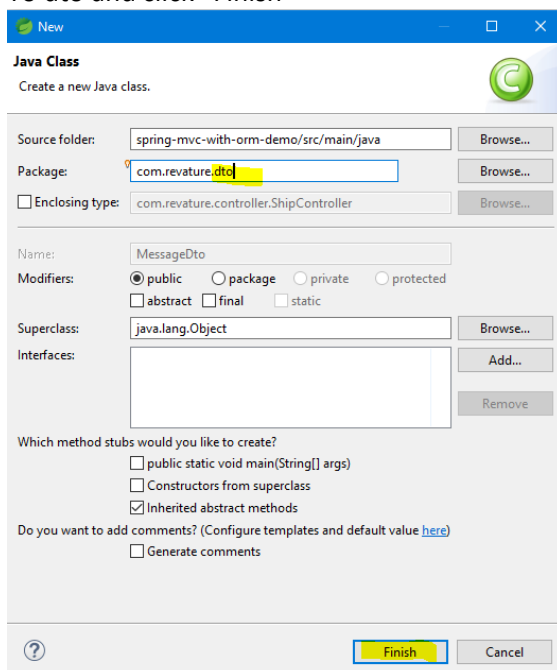
Hover mouse over the error / redline and use the wizard to create the new DTO class



Change the package from “controller”



To dto and click “Finish”



Update the Message.Dto class to the following:

```
package com.revature.dto;

import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@Getter @Setter @EqualsAndHashCode @NoArgsConstructor @ToString
public class MessageDto {

    private String message;

    public MessageDto(String message) {
        this.message = message;
    }
}
```

```
}
```

Continue Updating the base ShipController class

Once the ShipController class is saved, the Message.Dto error should be resolved.

Now add a get API using GetMapping annotation.

```
@GetMapping(path = "/ship")
public ResponseEntity<Object> getAllShips() {
    try {
        List<Ship> ships = shipService.getAllShips();

        return ResponseEntity.status(200).body(ships);
    } catch (ShipNotFoundException e) {
        return ResponseEntity.status(404).body(new
MessageDto(e.getMessage()));
    }
}
```

```
33 @GetMapping(path = "/ship")
34 public ResponseEntity<Object> getAllShips() {
35     try {
36
37         List<Ship> ships = shipService.getAllShips();
38
39         return ResponseEntity.status(200).body(ships);
40     } catch (ShipNotFoundException e) {
41         return ResponseEntity.status(404).body(new MessageDto(e.getMessage()));
42     }
43 }
```

Correct the errors introduced with the new code.

Hover mouse over an error / redline

Click on the import hyperlink highlighted below

```
33 @GetMapping(path = "/ship")
34
35
36
37
38
```

GetMapping cannot be resolved to a type

9 quick fixes available:

- Import 'GetMapping' (org.springframework.web.bind.annotation)

```
38 List<Ship> ships = shipService.get
39
40
41 } ca
42
43
44
45
46
47
48
```

List cannot be resolved to a type

33 quick fixes available:

- Import 'List' (antlr.collections)
- Import 'List' (com.sun.xml.bind.v2.sche)
- Import 'List' (java.awt)
- Import 'List' (org.hibernate.mapping)
- Import 'List' (java.util)

```
43 } catch (ShipNotFoundException e) {
44     return
45
46
47
48
```

ShipNotFoundException cannot be resolved to a type

12 quick fixes available:

- Import 'ShipNotFoundException' (com.revature.exception)

This change completes this class and should clear all the errors in this class.

The copy and paste version of the ShipController class is as follows:

```
package com.revature.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.revature.dto.AddShipDTO;
import com.revature.dto.MessageDto;
import com.revature.exception.BadParameterException;
import com.revature.exception.ShipNotFoundException;
import com.revature.model.Ship;
import com.revature.service.ShipService;

@RestController
public class ShipController {

    @Autowired
    private ShipService shipService;

    @PostMapping(path = "/ship")
    public ResponseEntity<Object> addShip(@RequestBody AddShipDTO addShipDTO) {

        try {
            Ship ship = shipService.addShip(addShipDTO);

            return ResponseEntity.status(201).body(ship); // 201 created
        } catch (BadParameterException e) {
            return ResponseEntity.status(400).body(new MessageDto(e.getMessage()));
        }
    }

    @GetMapping(path = "/ship")
    public ResponseEntity<Object> getAllShips() {
        try {

            List<Ship> ships = shipService.getAllShips();

            return ResponseEntity.status(200).body(ships);
        } catch (ShipNotFoundException e) {
            return ResponseEntity.status(404).body(new MessageDto(e.getMessage()));
        }
    }
}
```

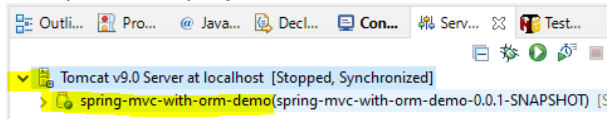
```

1 package com.revature.controller;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.http.ResponseEntity;
7 import org.springframework.web.bind.annotation.GetMapping;
8 import org.springframework.web.bind.annotation.PostMapping;
9 import org.springframework.web.bind.annotation.RequestBody;
10 import org.springframework.web.bind.annotation.RestController;
11
12 import com.revature.dto.AddShipDTO;
13 import com.revature.dto.MessageDto;
14 import com.revature.exception.BadParameterException;
15 import com.revature.exception.ShipNotFoundException;
16 import com.revature.model.Ship;
17 import com.revature.service.ShipService;
18
19 @RestController
20 public class ShipController {
21
22     @Autowired
23     private ShipService shipService;
24
25     @PostMapping(path = "/ship")
26     public ResponseEntity<Object> addShip(@RequestBody AddShipDTO addShipDTO) {
27
28         try {
29             Ship ship = shipService.addShip(addShipDTO);
30
31             return ResponseEntity.status(201).body(ship); // 201 created
32         } catch (BadParameterException e) {
33             return ResponseEntity.status(400).body(new MessageDto(e.getMessage()));
34         }
35     }
36
37     @GetMapping(path = "/ship")
38     public ResponseEntity<Object> getAllShips() {
39         try {
40
41             List<Ship> ships = shipService.getAllShips();
42
43             return ResponseEntity.status(200).body(ships);
44         } catch (ShipNotFoundException e) {
45             return ResponseEntity.status(404).body(new MessageDto(e.getMessage()));
46         }
47     }
48
49 }

```

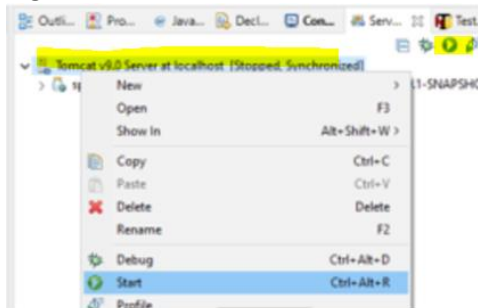

Test the POST API

This project was added to the Tomcat server in a previous step.
Verify that the project is still in the server.



Start the Tomcat.

Right click the server → select “Start”



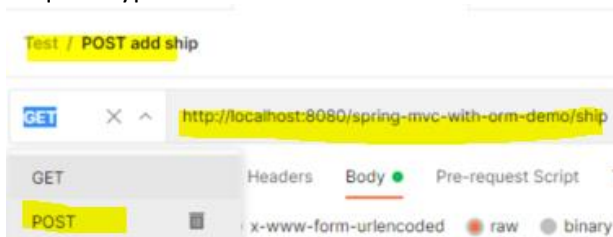
Testing the POST API

Create a test case in Postman website.

Name: POST add ship

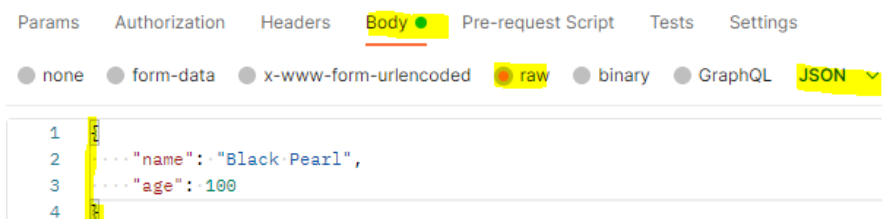
URL: <http://localhost:8080/spring-mvc-with-orm-demo/ship>

Request type: POST

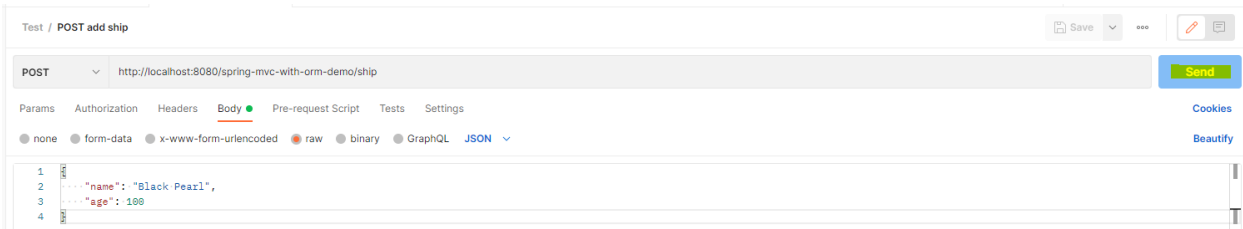


Body (raw, JSON):

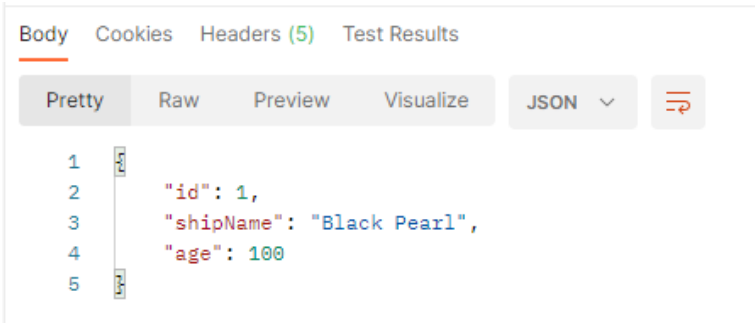
```
{
  "name": "Black Pearl",
  "age": 100
}
```



Press “Send”



Expected results:



If you got the expected results above or similar skip to section title [Update ORMConfig sessionFactory\(\) method](#). If you did not receive good results proceed to the next section.

Possible Testing Issues

Postman web (cloud) based issue

At some point the web based version of Postman no longer allows sending a request to localhost. You will need to use the Desktop Agent (install it if not already done).

Response



Could not send request

Cloud Agent Error: Can not send requests to localhost. Select a different agent.

Use Postman's Desktop Agent

After installing and logging into the desktop version of Postman, you should have access to any Collections in your Workspace created using the web based version. Including the POST test case created above.

Project URL issue

During the lecture when attempting a test of the POST there was an issue using the correct project name: spring-mvc-with-orm-demo in the URL. Even after updating the names in pom.xml and web.xml which is outline in the correct order in this document. The following URL received a 404 response:

http://localhost:8080/spring-mvc-with-orm-demo/ship

The project URL did not work. However if we use a URL with **spring-mvc-demo** in the name, the request worked. This name is from the project we copied as the base code for this project.

http://localhost:8080/spring-mvc-demo/ship

[Appendix: Correct Issue with Project URL](#) gives detailed instructions on how to resolve the issue with the project name in the URL string. If the correct project URL above does not work and you receive the “<title>HTTP Status 404 – Not Found</title>” error in the Postman respond, then following the instructions in the appendix.

Update ORMConfig sessionFactory() method

After correcting any issues with test the POST API, update database configuration.

At this time change the sessionFactory() property in the ORMConfig class from create to validate. Recall that in create mode the database tables are dropped and created each run of the application. In this mode any data in the tables will be lost. The validate mode preserves any data create in the database tables.

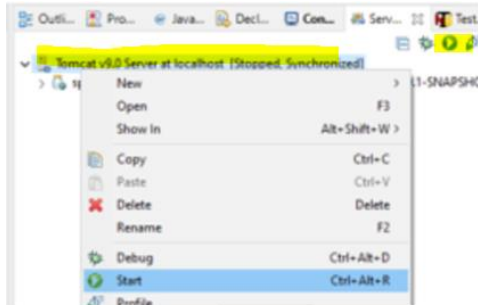
```
--
36 @Bean
37 public LocalSessionFactoryBean sessionFactory() {
38     LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();
39
40     sessionFactory.setDataSource(dataSource()); // setter injection
41     sessionFactory.setPackagesToScan("com.revature.model");
42
43     // Create the properties object
44     // import from java.util.Properties;
45     Properties hibernateProperties = new Properties();
46
47     //Use the create parameter to create the database then switch to validate.
48     //hibernateProperties.setProperty("hibernate.hbm2ddl.auto", "create");
49     hibernateProperties.setProperty("hibernate.hbm2ddl.auto", "validate");
50     hibernateProperties.setProperty("hibernate.dialect", "org.hibernate.dialect.MariaDB103Dialect");
51
52     // Configure hibernate properties
53     sessionFactory.setHibernateProperties(hibernateProperties);
54
55     return sessionFactory;
56 }
```

Test the GET API

After correcting any issues with test the POST API, proceed to testing the GET API.

Start the Tomcat.

Right click the server → select “Start”



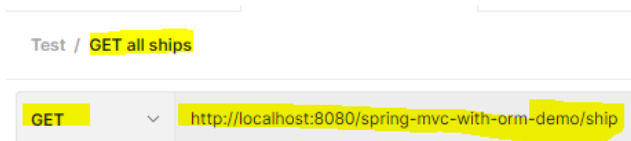
Testing the GET API

Create a test case in Postman Desktop Agent.

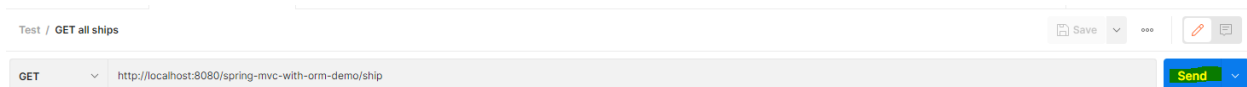
Name: GET all ships

URL: <http://localhost:8080/spring-mvc-with-orm-demo/ship>

Request type: GET



Press “Send”



Expected results (might vary depending on how many ships were added):



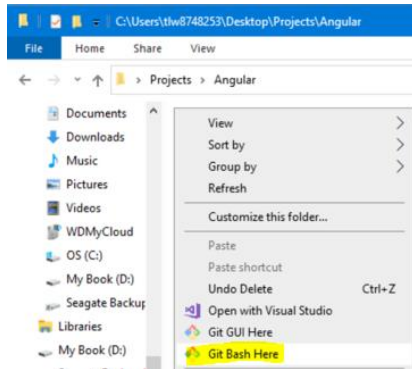
Create an Angular Web Application for the Demo

Generate an Angular folder “display-ships-app”

In your Angular workspace folder open GitBash.

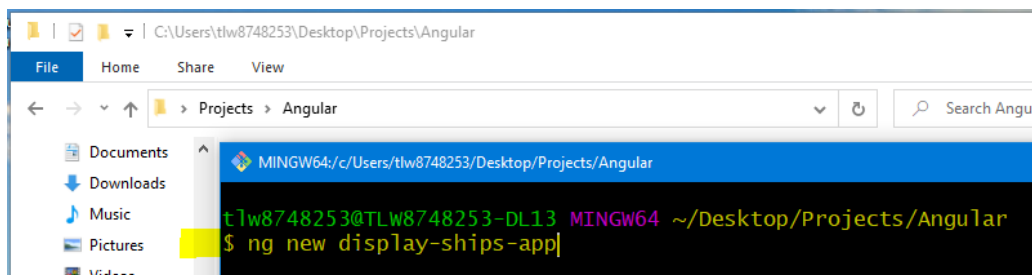
Right click in the folder

Select “Git Bash Here”

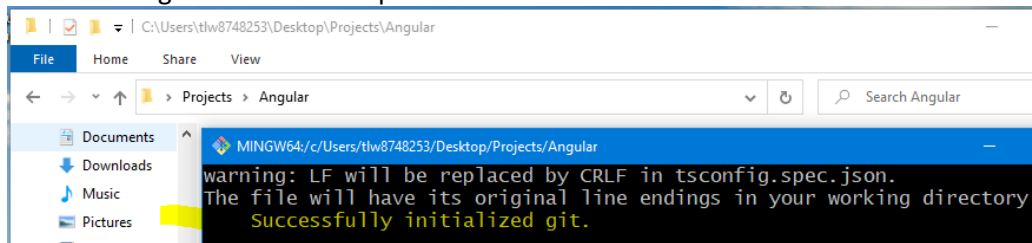


Use the following command to generate the Angular default application

```
ng new display-ships-app
```

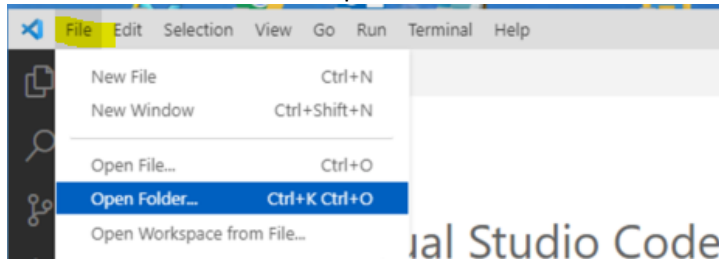


Wait for the generation to complete.



Open the default project with VS Code

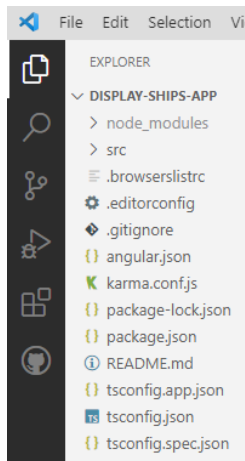
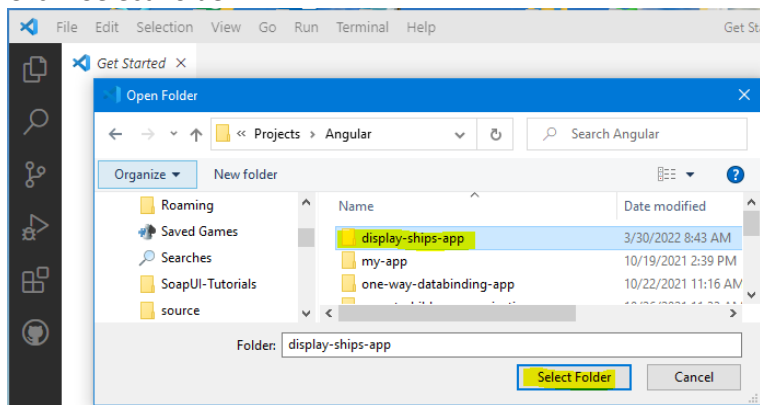
On the Toolbar: “File” → “Open Folder...”



Navigate to your Angular workspace folder

Select the “display-ships-app” folder

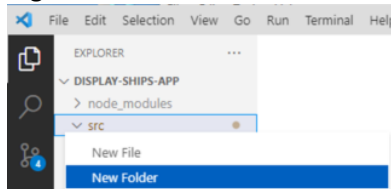
Click “Select Folder”



Create a ship Model

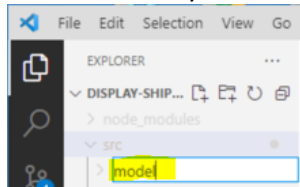
Create a folder: model

Right click on “src” folder → select “New Folder”



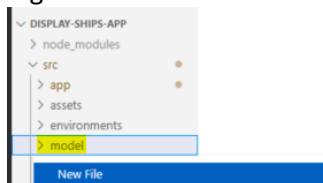
Enter folder name: “model”

Press return key



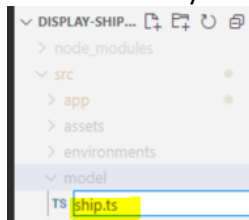
Create a model: ship.ts

Right click on folder created “model” → select “New File”



Enter file name: ship.ts

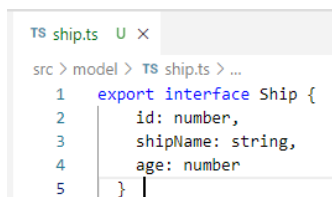
Press return key



Update the model file

Add the following code:

```
export interface Ship {  
  id: number,  
  shipName: string,  
  age: number  
}
```



Generate a ship service

In the Git Bash window

Change directory to the project folder

```
cd display-ships-app
```

```
MINGW64/c/Users/tlw8748253/Desktop/Projects/Angular/display-ships-app
tlw8748253@TLW8748253-DL13 MINGW64 ~/Desktop/Projects/Angular
$ cd display-ships-app
tlw8748253@TLW8748253-DL13 MINGW64 ~/Desktop/Projects/Angular/display-ships-app (master)
$ |
```

Generate the server, enter the following command:

```
ng generate service ship
```

```
MINGW64/c/Users/tlw8748253/Desktop/Projects/Angular/display-ships-app
tlw8748253@TLW8748253-DL13 MINGW64 ~/Desktop/Projects/Angular/display-ships-app (master)
$ ng generate service ship
```

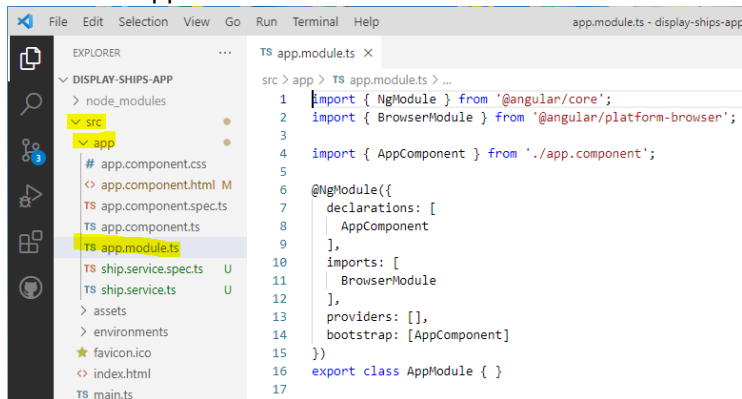
Wait till it completes

```
MINGW64/c/Users/tlw8748253/Desktop/Projects/Angular/display-ships-app
tlw8748253@TLW8748253-DL13 MINGW64 ~/Desktop/Projects/Angular/display-ships-app (master)
$ ng generate service ship
CREATE src/app/ship.service.spec.ts (347 bytes)
CREATE src/app/ship.service.ts (133 bytes)
```

Update ship service

First add import to the application

Select file: app.module.ts



```
File Edit Selection View Go Run Terminal Help
app.module.ts - display-ships-app

EXPLORER
DISPLAY-SHIPS-APP
  node_modules
  src
    app
      app.component.css
      app.component.html M
      app.component.spec.ts
      app.component.ts
      app.module.ts
      ship.service.spec.ts U
      ship.service.ts U
  assets
  environments
  favicon.ico
  index.html
  main.ts

src > app > TS app.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3
4 import { AppComponent } from './app.component';
5
6 @NgModule({
7   declarations: [
8     AppComponent
9   ],
10  imports: [
11    BrowserModule
12  ],
13  providers: [],
14  bootstrap: [AppComponent]
15 })
16 export class AppModule { }
17
```

Add import: HttpClientModule

```
import { HttpClientModule } from '@angular/common/http';

imports: [
  BrowserModule,
  HttpClientModule
```

```
TS app.module.ts M X
src > app > TS app.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { HttpClientModule } from '@angular/common/http';
4
5 import { AppComponent } from './app.component';
6
7 @NgModule({
8   declarations: [
9     AppComponent
10  ],
11   imports: [
12     BrowserModule,
13     HttpClientModule
14  ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19
```

Next update ship service

Select file: ship.service.ts

```
EXPLORER
node_modules
src
  app
    # app.component.css
    <> app.component.html M
    TS app.component.spec.ts
    TS app.component.ts
    TS app.module.ts
    TS ship.service.spec.ts U
    TS ship.service.ts U

TS ship.service.ts U X
src > app > TS ship.service.ts > ...
1 import { Injectable } from '@angular/core';
2
3 @Injectable({
4   providedIn: 'root'
5 })
6 export class ShipService {
7
8   constructor() { }
9
10 }
```

Update the service with the following:

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Ship } from '../model/ship';

constructor(private http: HttpClient) { }

getShips(): Observable<Ship[]> {
  return this.http.get<Ship[]>('http://localhost:8080/spring-mvc-with-orm-demo/ship');
}
```

Alternative copy and paste replacing the default code with:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Ship } from '../model/ship';
```

```

@Injectable({
  providedIn: 'root'
})
export class ShipService {

  constructor(private http: HttpClient) { }

  getShips(): Observable<Ship[]> {
    return this.http.get<Ship[]>('http://localhost:8080/spring-mvc-with-orm-demo/ship');
  }
}

```

End result:

TS ship.service.ts U X

```

src > app > TS ship.service.ts > ...
1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3  import { Observable } from 'rxjs';
4  import { Ship } from '../model/ship';
5
6
7  @Injectable({
8    providedIn: 'root'
9  })
10 export class ShipService {
11
12   constructor(private http: HttpClient) { }
13
14   getShips(): Observable<Ship[]> {
15     return this.http.get<Ship[]>('http://localhost:8080/spring-mvc-with-orm-demo/ship');
16   }
17
18 }

```

Add service to application

Select file: app.component.ts

Add service items to the application:

```
import { ShipService } from './ship.service';
import { Ship } from '../model/ship';

ships: Ship[] = [];
constructor(private shipService: ShipService) {}

ngOnInit() {
  this.shipService.getShips().subscribe((data) => {
    this.ships = data;
  });
}
```

Alternative copy and paste replacing the default code with:

```
import { Component } from '@angular/core';
import { ShipService } from './ship.service';
import { Ship } from '../model/ship';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  ships: Ship[] = [];
  constructor(private shipService: ShipService) {}

  ngOnInit() {
    this.shipService.getShips().subscribe((data) => {
      this.ships = data;
    });
  }
}
```

```
}  
}
```

End result:

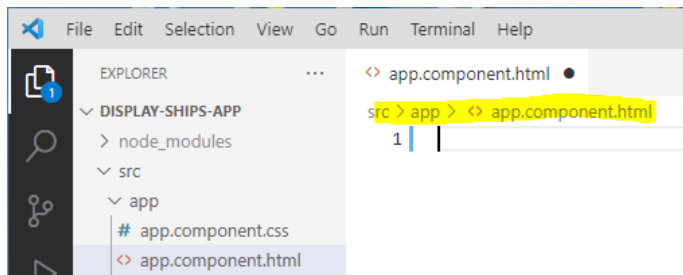
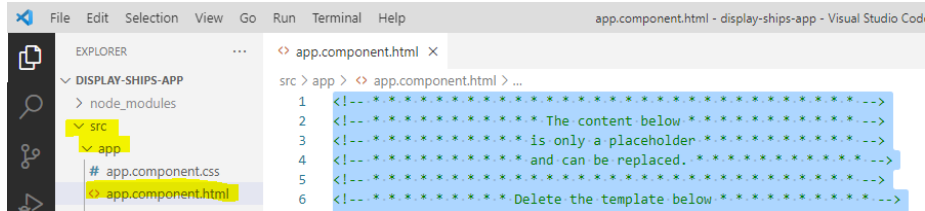
```
TS app.component.ts M X  
src > app > TS app.component.ts > AppComponent  
1  import { Component } from '@angular/core';  
2  import { ShipService } from '../ship.service';  
3  import { Ship } from '../model/ship';  
4  
5  @Component({  
6    selector: 'app-root',  
7    templateUrl: './app.component.html',  
8    styleUrls: ['./app.component.css']  
9  })  
10 export class AppComponent {  
11  
12    ships: Ship[] = [];  
13    constructor(private shipService: ShipService) {}  
14  
15    ngOnInit() {  
16      this.shipService.getShips().subscribe((data) => {  
17        this.ships = data;  
18      });  
19    }  
20  }
```

Modify file "app.component.html"

Open file: "app.component.html"

Highlight all the code: ctrl-a

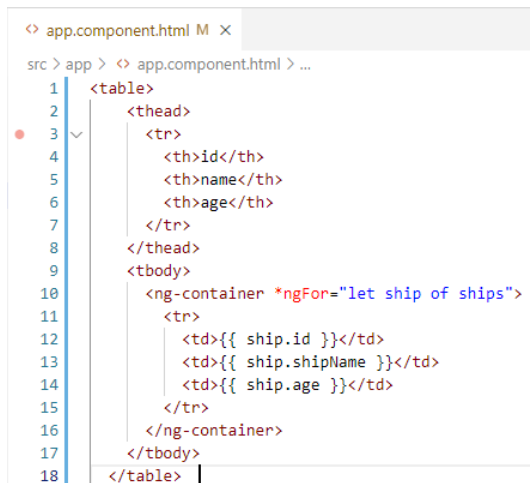
Delete all the default code



Copy and paste the following code into the html file

```
<table>
  <thead>
    <tr>
      <th>id</th>
      <th>name</th>
      <th>age</th>
    </tr>
  </thead>
  <tbody>
    <ng-container *ngFor="let ship of ships">
      <tr>
        <td>{{ ship.id }}</td>
        <td>{{ ship.shipName }}</td>
        <td>{{ ship.age }}</td>
      </tr>
    </ng-container>
  </tbody>
</table>
```

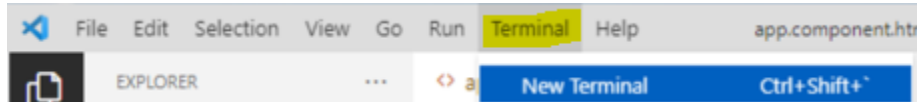
End result:



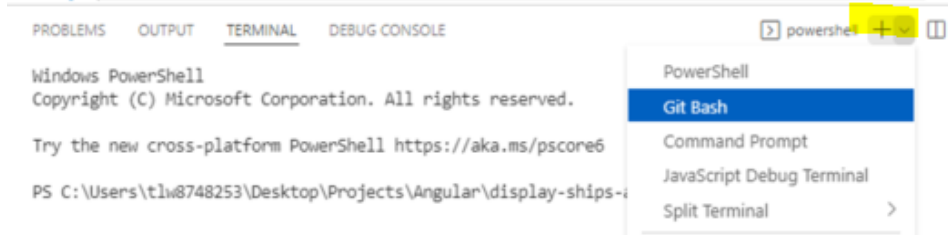
Run the Angular Application

Open a Git Bash terminal window

On the toolbar select “Terminal” → “New Terminal”



If Git Bash is not the default, open from the + dropdown



Command ng serve

Enter the following command in the Git Bash window:

```
ng serve
```

Recall that “ng serve” is mapped to the command “npm run start” in package.json file by default. Either command can be used to start the application.



On my machine there were no additional prompts and no conflict with port 4200. If you see additional prompts including port 4200 conflict see the section: [Additional Angular prompts that might be encountered](#). Also see [Command ng serve – with port number](#).

```
tlw8748253@TLW8748253-DL13 MINGW64 ~/Desktop/Projects/Angular/display-ships-app (master)
$ ng serve
^ Generating browser application bundles (phase: setup)...Compiling @angular/core : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/common/http : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
✓ Browser application bundle generation complete.

Initial Chunk Files | Names | Size
vendor.js           | vendor | 2.15 MB
polyfills.js        | polyfills | 466.55 kB
styles.css, styles.js | styles | 344.45 kB
main.js             | main | 12.19 kB
runtime.js          | runtime | 6.63 kB

| Initial Total | 2.96 MB

Build at: 2022-03-30T20:07:54.975Z - Hash: c70b153b2e4396615b5f - Time: 30664ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.
✓ Browser application bundle generation complete.

5 unchanged chunks

Build at: 2022-03-30T20:07:56.754Z - Hash: a2e2434f65c6e87fc052 - Time: 900ms

✓ Compiled successfully.
```

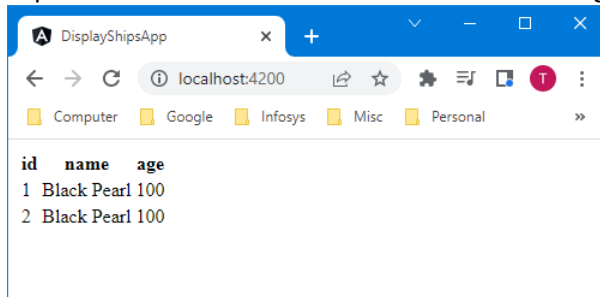
Look for the line that starts with “** Angular Live Development Server ...”

Copy the http string where the application is listening:

```
http://localhost:4200/
```

Paste into a browser. **NOTE** make sure the server is running in the SpringTool IDE.

Expected results should be similar to the following:

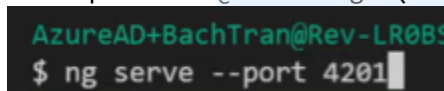


Command ng serve – with port number

If you know or believe that port 4200 is in use on your machine. You can force to use a specific port with the following command in the Git Bash window:

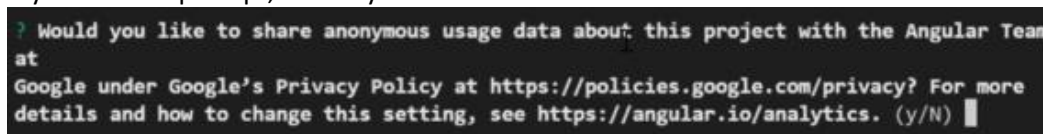
```
ng serve --port 4201
```

Then update the `@CrossOrigin("http://localhost:4200")` line to 4201 or the port you specified.

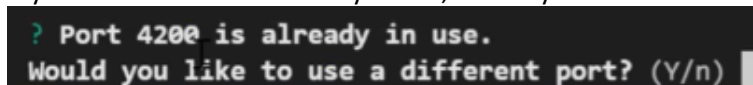


Additional Angular prompts that might be encountered

If you see this prompt, enter “y”

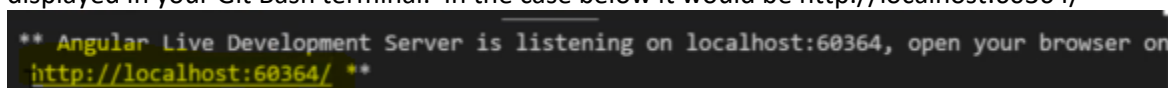


If you see Port 4200 is already in use, enter “y”



You will also need to update the `@CrossOrigin("http://localhost:4200")` port in the ShipController file to the port identified in the Git Bash terminal upon completion. As shown below this would be port 60364.

In addition your listening port will note be `http://localhost:4200/`. You will need to use the http string displayed in your Git Bash terminal. In the case below it would be `http://localhost:60364/`



Appendix: Alternative to db_url environment variable

Instead of having to change the db_url environment variable to point to different database names or to point to localhost versus hosted environments, you can setup different environment variables.

You can setup variables for a hosted environment (AWS), for local host, and for database names. If you use separate variables for the database name, you will not have to edit the db_url string every time you switch to a new or previous project.

For new projects you would just need to add a new database name variable.

Remember if you change or add any new environment variables you will need to restart the SpringTool IDE if it is in use.

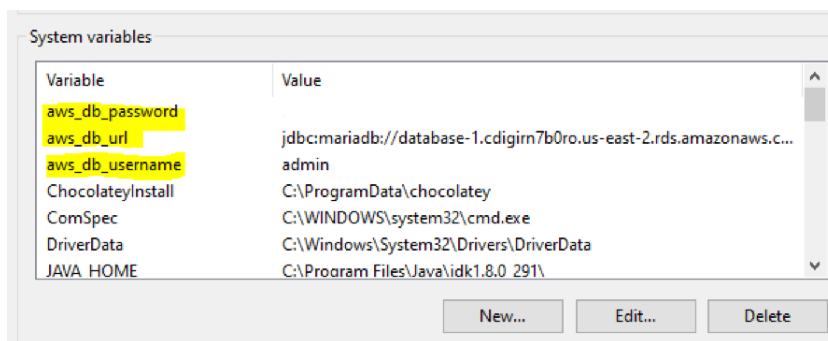
Example for hosted is to use:

aws_db_password

aws_db_url (without the database name and ends in forward slash):

jdbc:mariadb://your-aws-db-string.amazonaws.com:3306/

aws_db_username



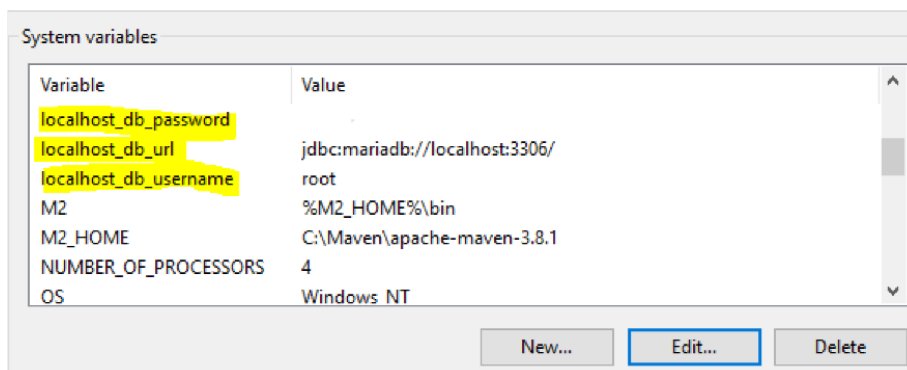
Example for localhost is to use:

localhost_db_password

localhost_db_url (without the database name and ends in forward slash):

jdbc:mariadb://localhost:3306/

localhost_db_username



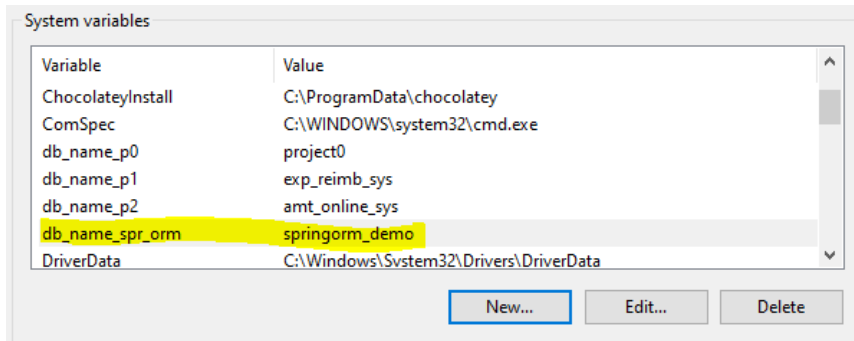
Example for database name is to use:

db_name_p0 – for project 0

db_name_p1 – for project 1

...

db_name_spr_orm for the demo in this document



Update DataSource bean in ORMConfig file

To use separate environment variable as described above, change the dataSource() method in the ORMConfig file as indicated by the highlighted code below.

```
@Bean
public DataSource dataSource() {
    BasicDataSource dataSource = new BasicDataSource();

    dataSource.setDriverClassName("org.mariadb.jdbc.Driver");

    String dbName = System.getenv("db_name_spr_orm");
    String dbURL = System.getenv("localhost_db_url");
    dbURL += dbName;

    dataSource.setUrl(dbURL);
    dataSource.setUsername(System.getenv("localhost_db_username"));
    dataSource.setPassword(System.getenv("localhost_db_password"));

    return dataSource;
}
```

The code above could be further modified to use a hosted (AWS) environment by using the aws defined system variable. It could be further modify to switch between environments based on a global constant making the project code more verbose.

Appendix: Correct Issue with Project URL

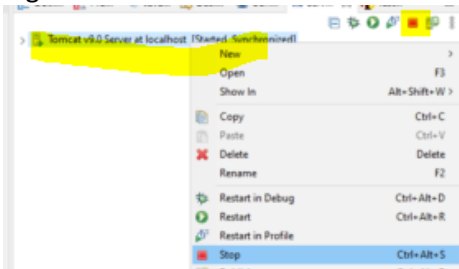
During the lecture when attempting a test of the POST API there was an issue using the correct project name: spring-mvc-with-orm-demo in the URL. Even after updating the names in pom.xml and web.xml which is outline in the correct order in this document.

`http://localhost:8080/spring-mvc-with-orm-demo/ship`

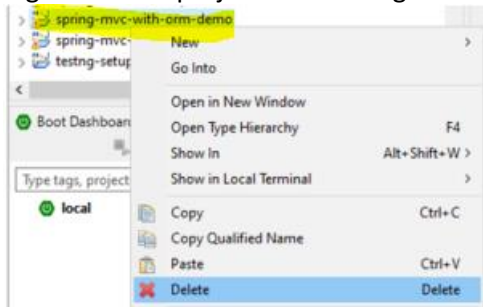
The URL did not work unless the project name was spring-mvc-demo. This name is from the project we copied as the base code for this project.

To resolve this issue:

1. Stop the IDE Tomcat server if started
Right click on the server and select "Stop:

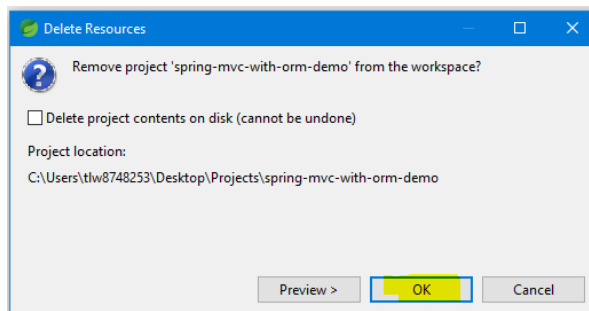


2. Delete the spring-mvc-with-orm-demo IDE
Right click the project in the navigation window and select "Delete"



DO NOT delete the content on disk

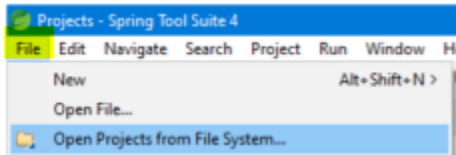
Leave the check box unchecked and click "OK"



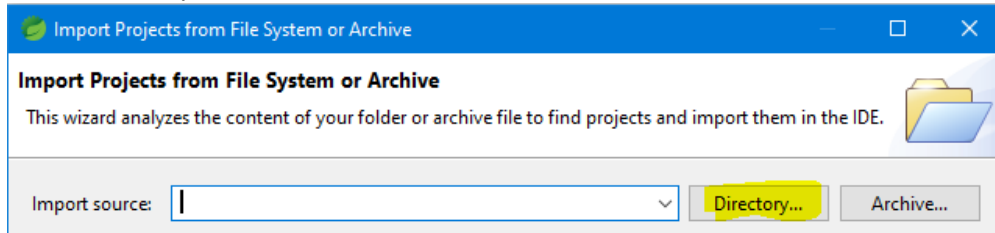
3. Import the spring-mvc-with-orm-demo

In the toolbar menu

Select “File” → “Open Projects from File System...”

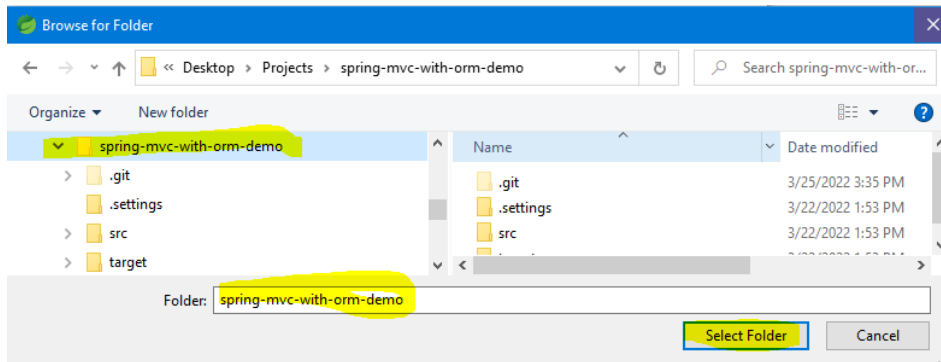


Click “Directory...”

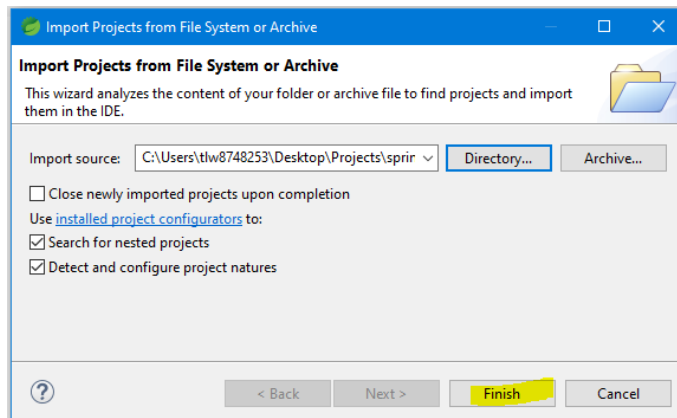


Select “spring-mvc-with-orm-demo” from your workspace

Click “Select Folder”

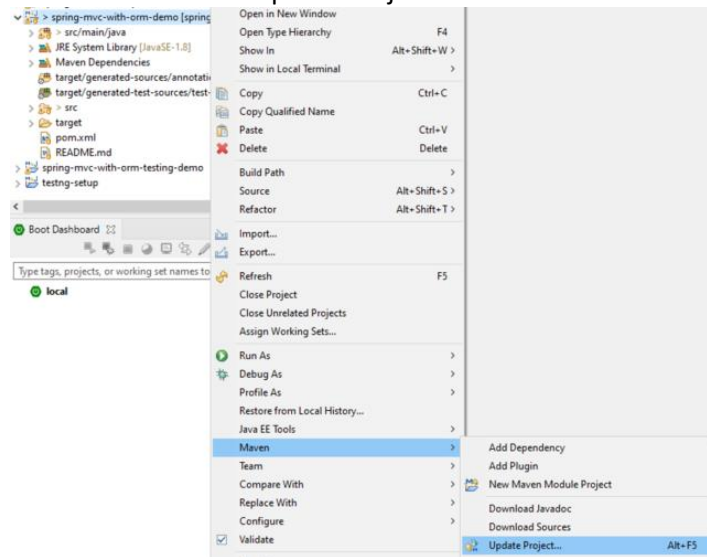


Click “Finish”

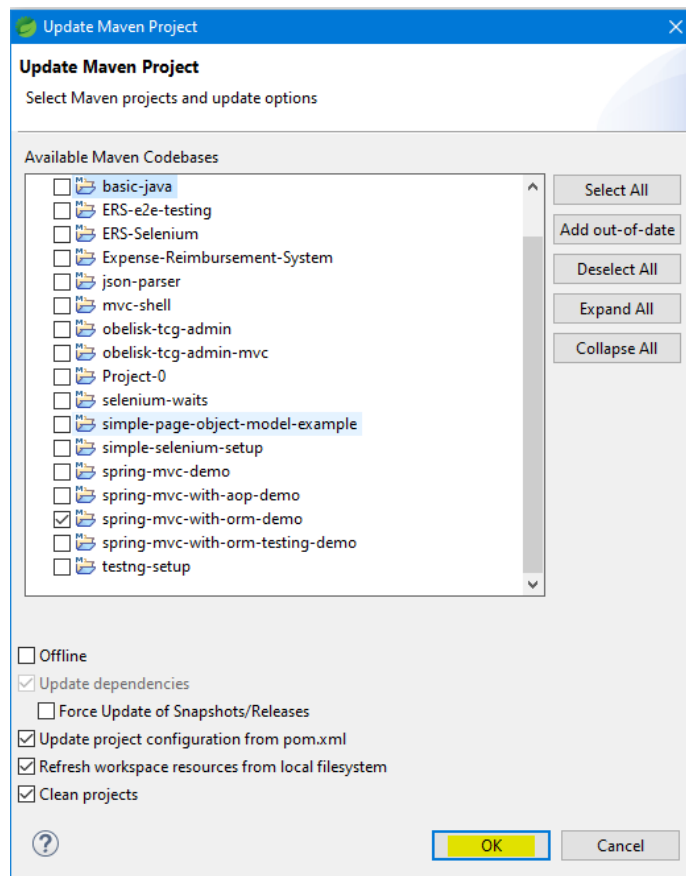


4. Rebuild the Maven project

Right click the “spring-mvc-with-orm-demo” project
Select “Maven” → “Update Project...”

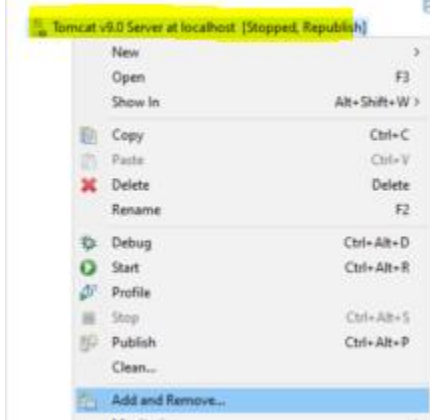


The “spring-mvc-with-orm-demo” should be checked by default
Click “OK”

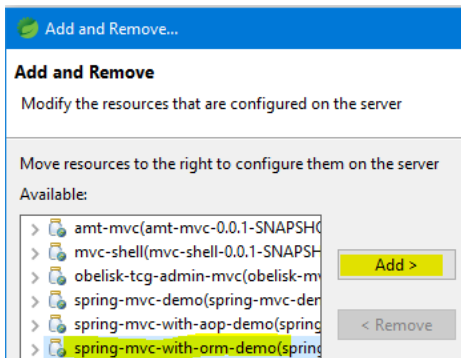


5. Add the spring-mvc-with-orm-demo the Tomcat server

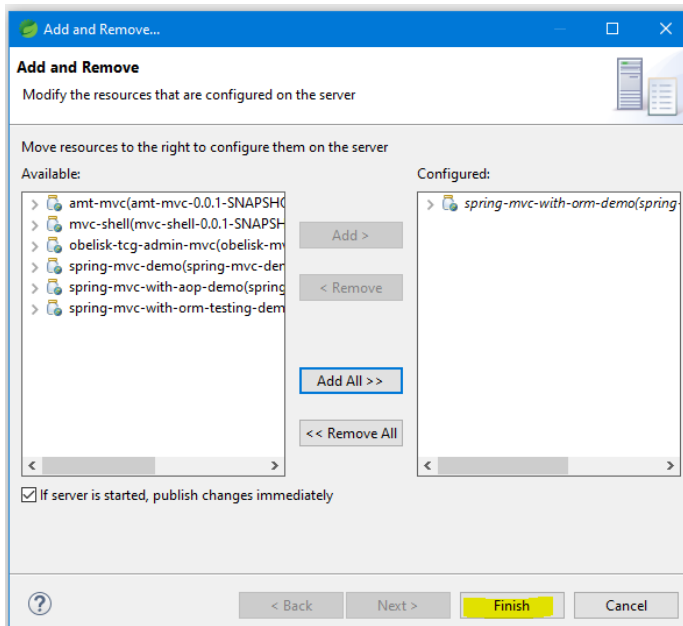
Right click the server and select “Add and Remove...”



Select the spring-mvc-with-orm-demo project
Click “Add >”

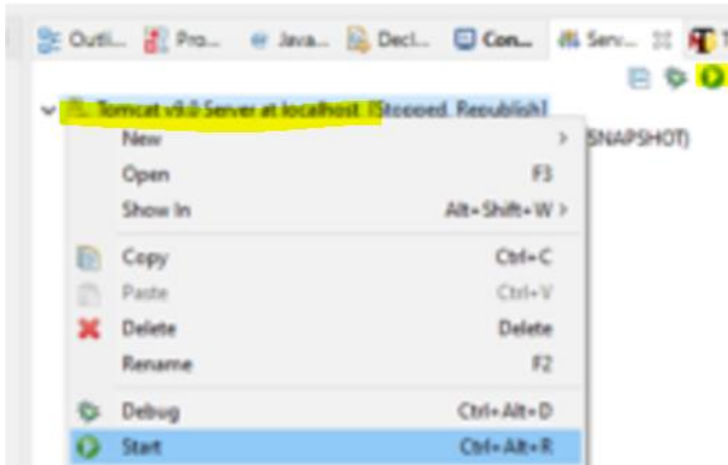


Click “Finish”



6. The server was restarted

Right click the server and click "Start"



7. The correct URL with spring-mvc-with-orm-demo should worked.

NOTE: I added rebuilding of the project and then still needed to do the steps above more than once. So you might need to do the same plus clean the server and clean the project. Right click the server and select clean, right click the project and select clean.