# Version History

|  |  |  |
| --- | --- | --- |
| Version | Date | Release Note |
| V0.1 | 10-Mar-2020 | Initial Draft |
| V1.0 | 2-Apr-2020 | 1st Release |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

index

1 Version History 2

2 Exercise 1 – Form Submission 4

2.1 Form Submission Exercise 4

3 Exercise 1 - bonus (to inject a bean) 14

4 Exercise 2 – to access database 16

5 Exercise 2 - bonus (to connect external database) 22

# Exercise 1 – Form Submission

The exercise will demostrate how to create your first Spring Application

Prerequisite

Java installed (e.g. JDK 11)

Maven installed (e.g. version 3.6.3)

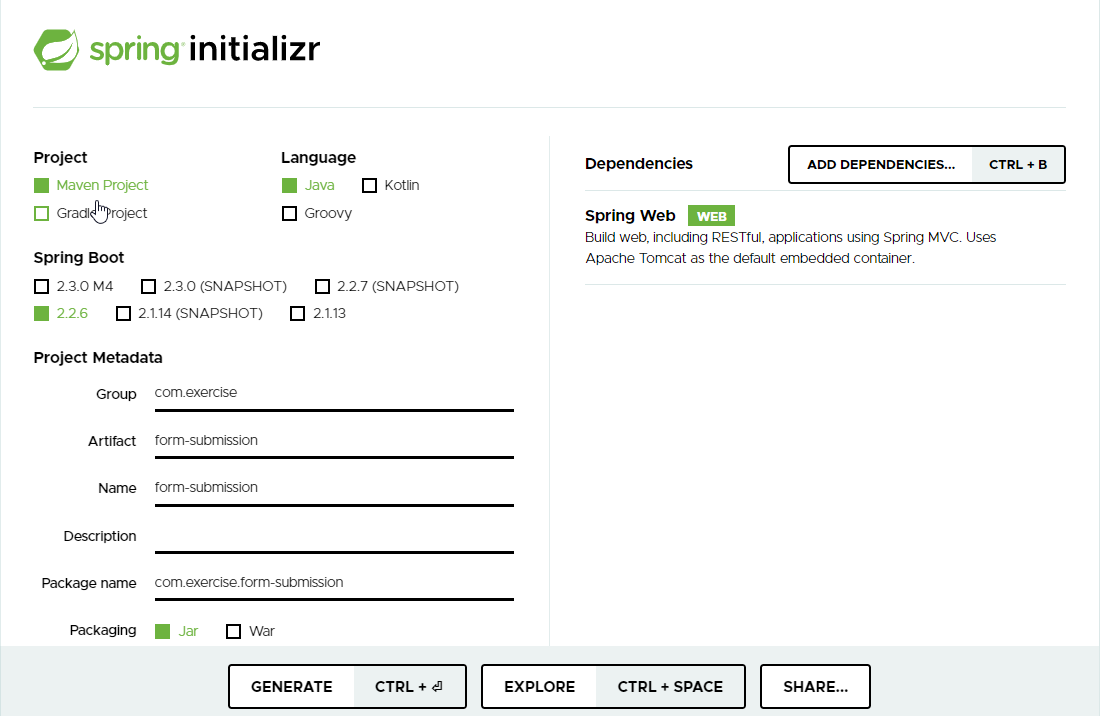
Spring Tool Suite (STS) (e.g. 4.5.1)

## Form Submission Exercise

### Generate sample pom.xml

Group: com.exercise

Artifact: form-submission

Dependencies: Spring Web

Or you may use this one:



2. Click “Generate” and extract the zipped file to your local directory, e.g. C:\java-ex:

C:.

└───form-submission

├───.mvn

│ └───wrapper

└───src

├───main

│ ├───java

│ │ └───com

│ │ └───exercise

│ │ └───formsubmission

│ └───resources

│ ├───static

│ └───templates

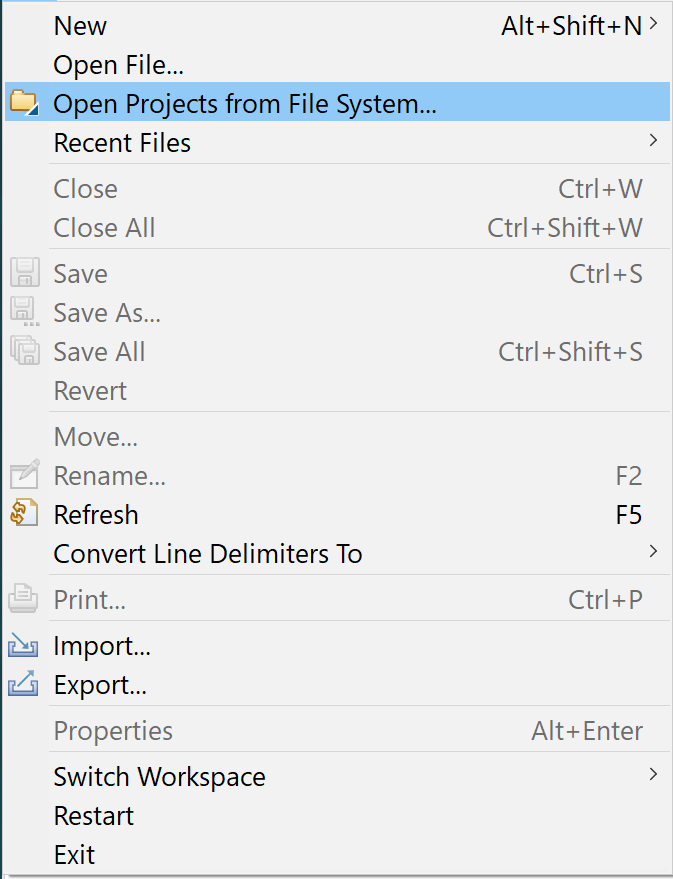
└───test

└───java

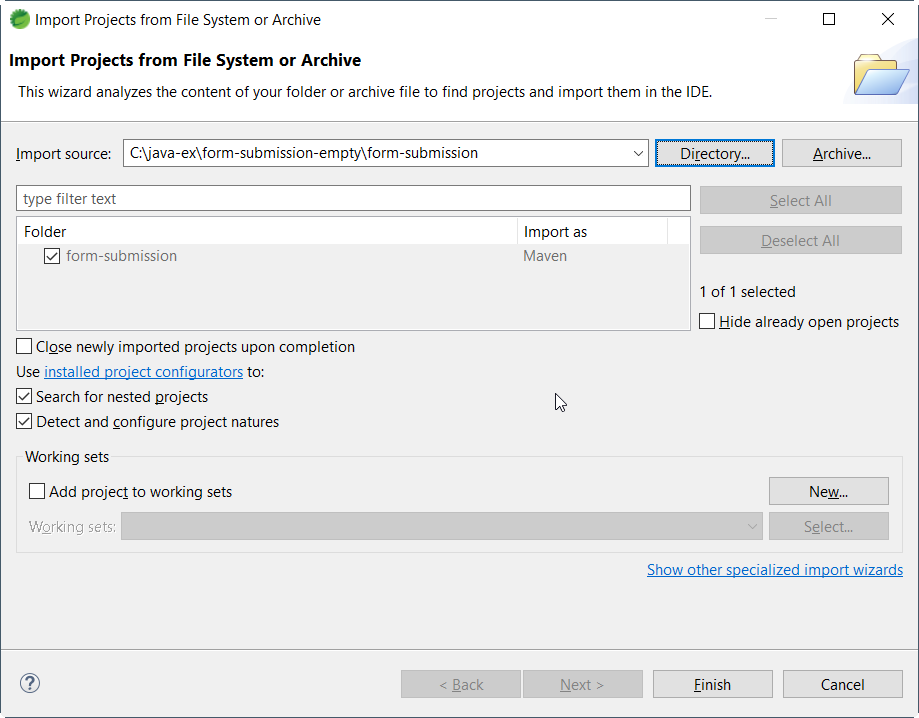
└───com

└───exercise

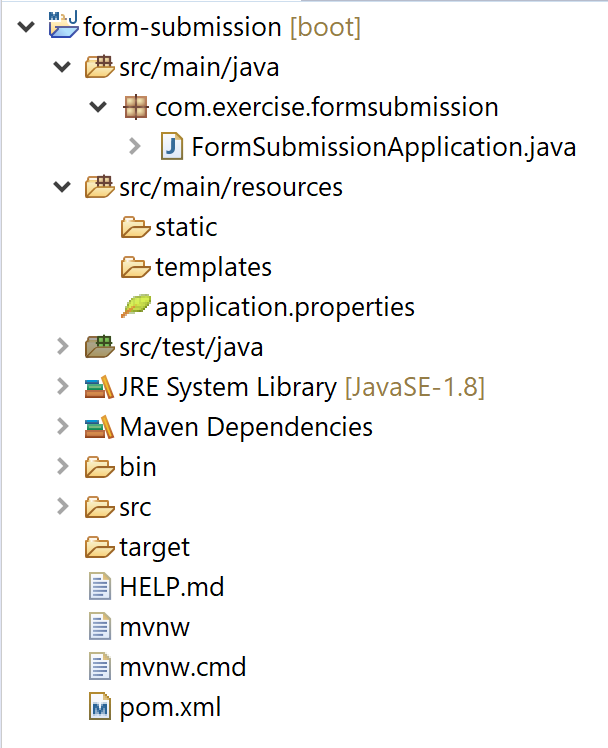
└───formsubmission

3. Start up STS, click “File” and select “Open Projects from File System…”:

4. Type or search your project folder “form-submission”:



5. The project structure should be shown:



6. In order to support JSP, we need to add dependency to the project. Now open the file “pom.xml”, and insert the following code to the <dependencies> tag.

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

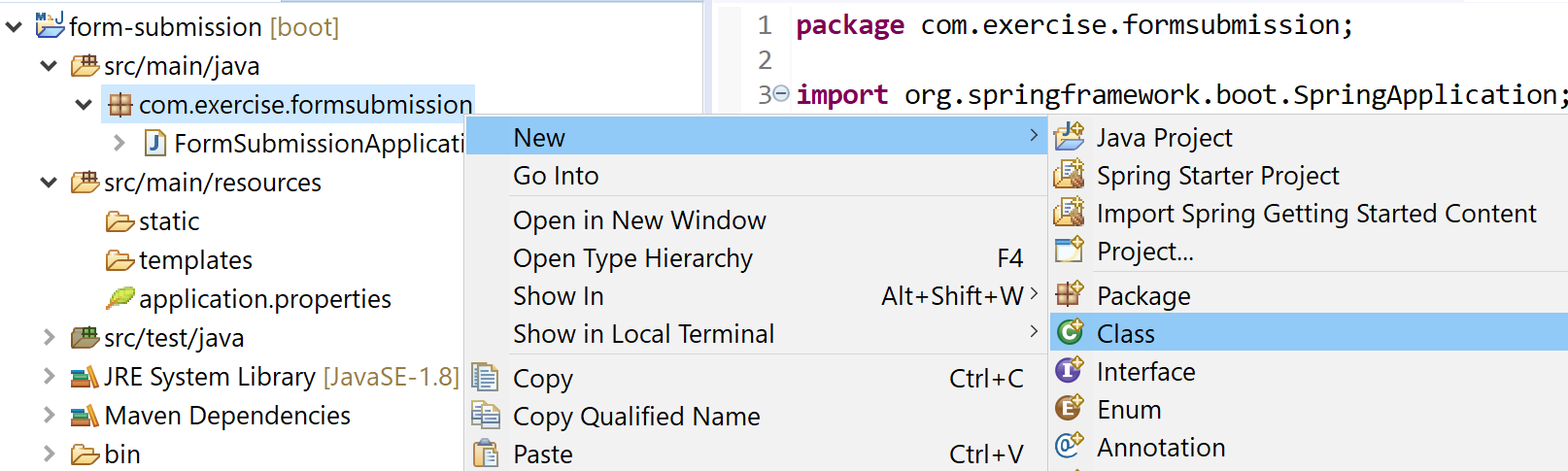


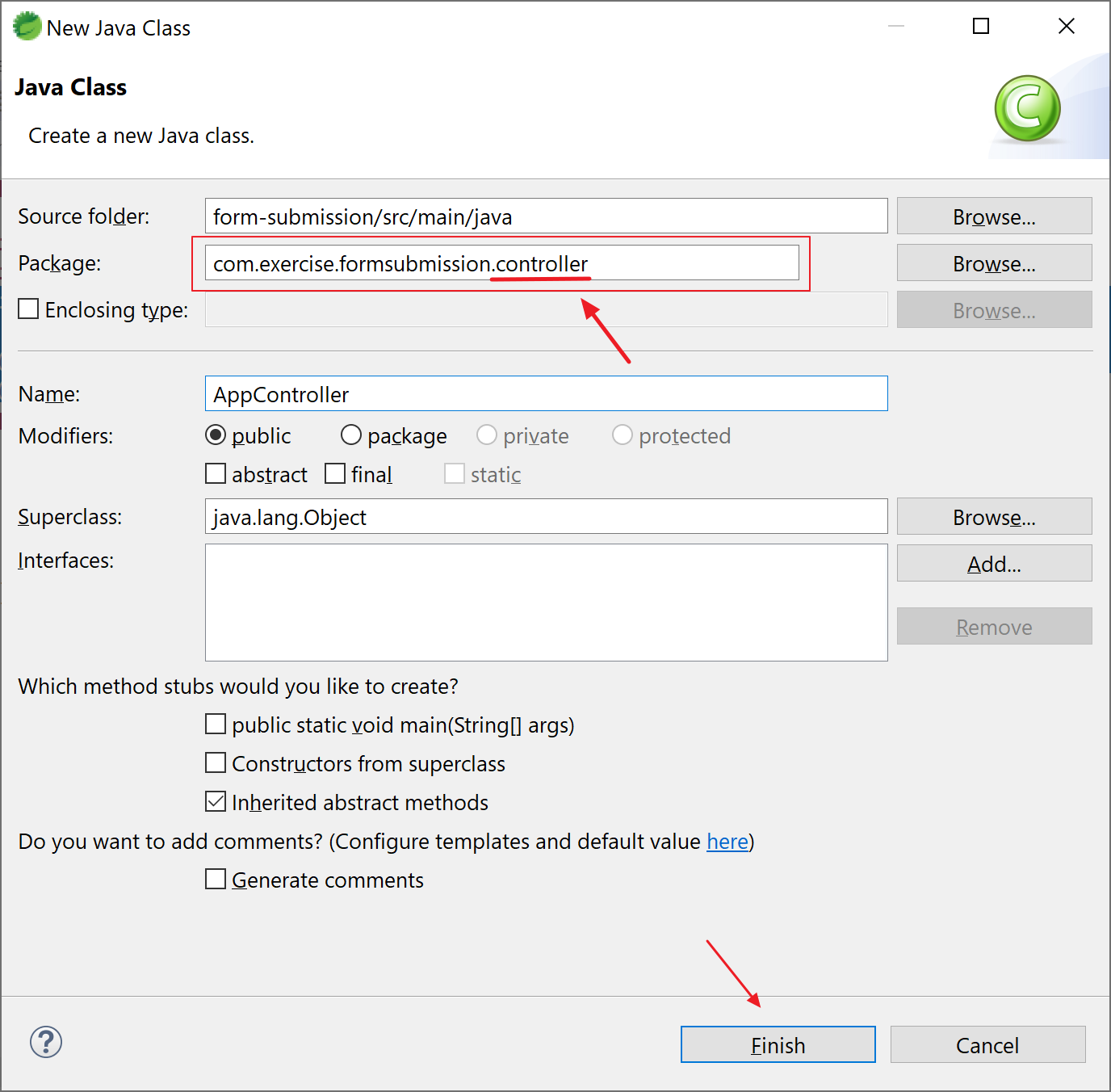
7. Open application.properties (.\src\main\resources\application.properties) to configure the MVC’s view. Add the following code to the file:

spring.mvc.view.prefix=/WEB-INF/jsp/

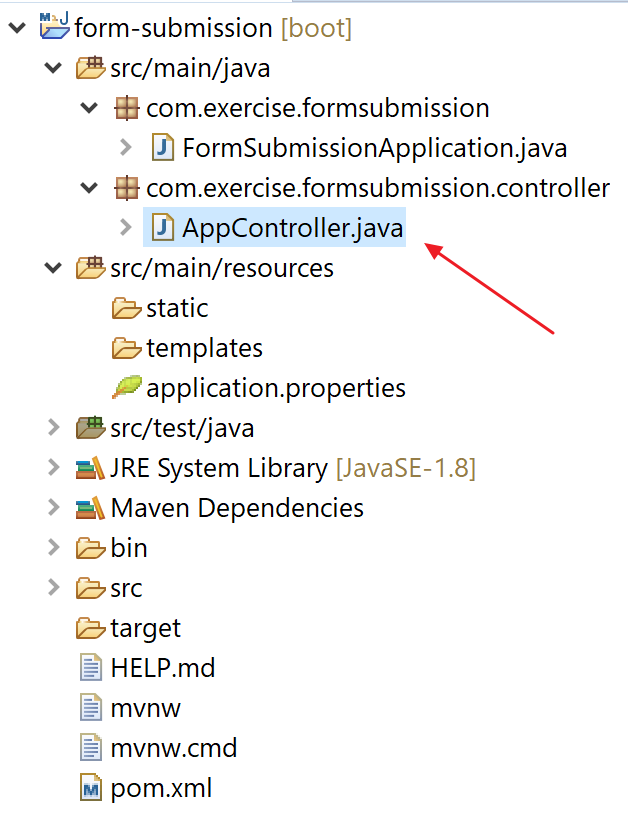
spring.mvc.view.suffix=.jsp

Note: if the view name is “showDetails”, then the view’s path will be “/WEB-INF/jsp/showDetails.jsp”.

8. Create the “Controller” class, right the package “com.exercise.formsubmission”, select “New” and then click “Class”:

9. Add a class “AppController” under the package “com.exercise.formsubmission.controller”:

Note: adding “.controller” to the package “com.exercise.formsubmission”.

10. Open the file “AppController.java” and insert the following code:

Here’s the list of the code:

package com.exercise.formsubmission.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

@Controller

public class AppController {

@GetMapping("/getAppForm")

public String getAppForm() {

return "appForm";

}

@PostMapping("/viewAppDetails")

public String viewAppDetails (@RequestParam("name") String name,

@RequestParam("email") String email, ModelMap modelMap) {

modelMap.put("name", name);

modelMap.put("email", email);

return "appDetails";

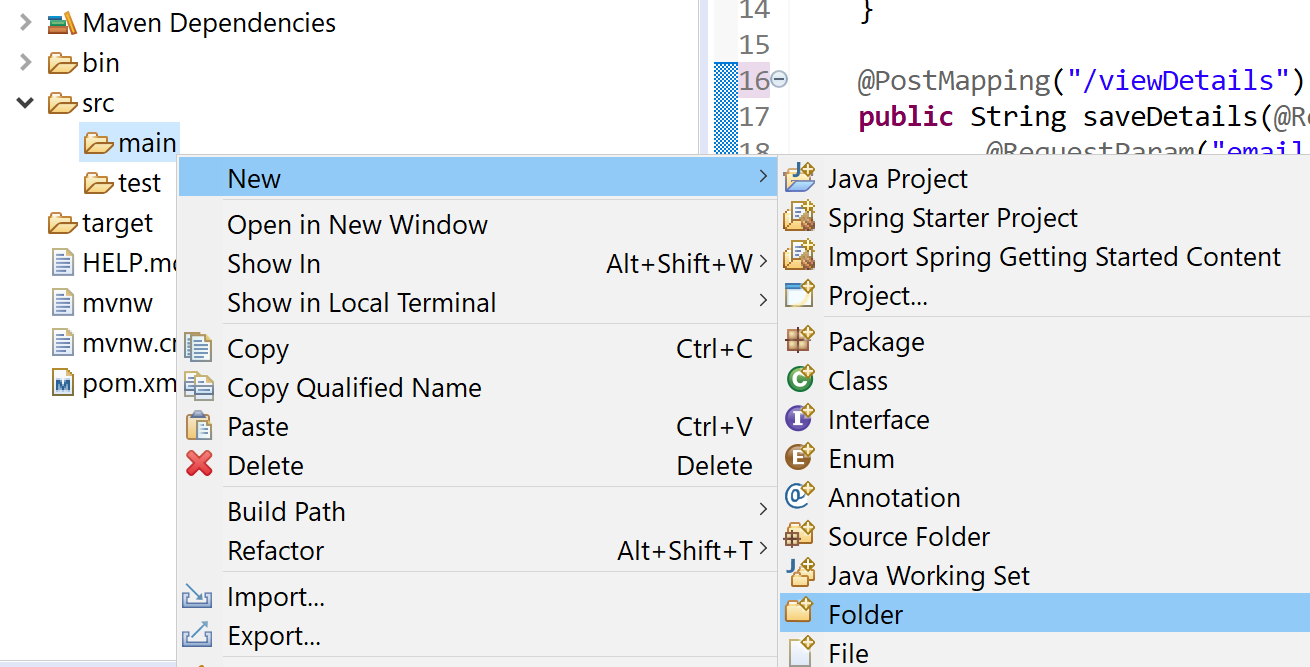
}

}

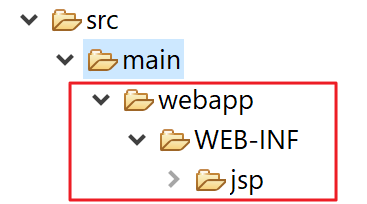
Note:

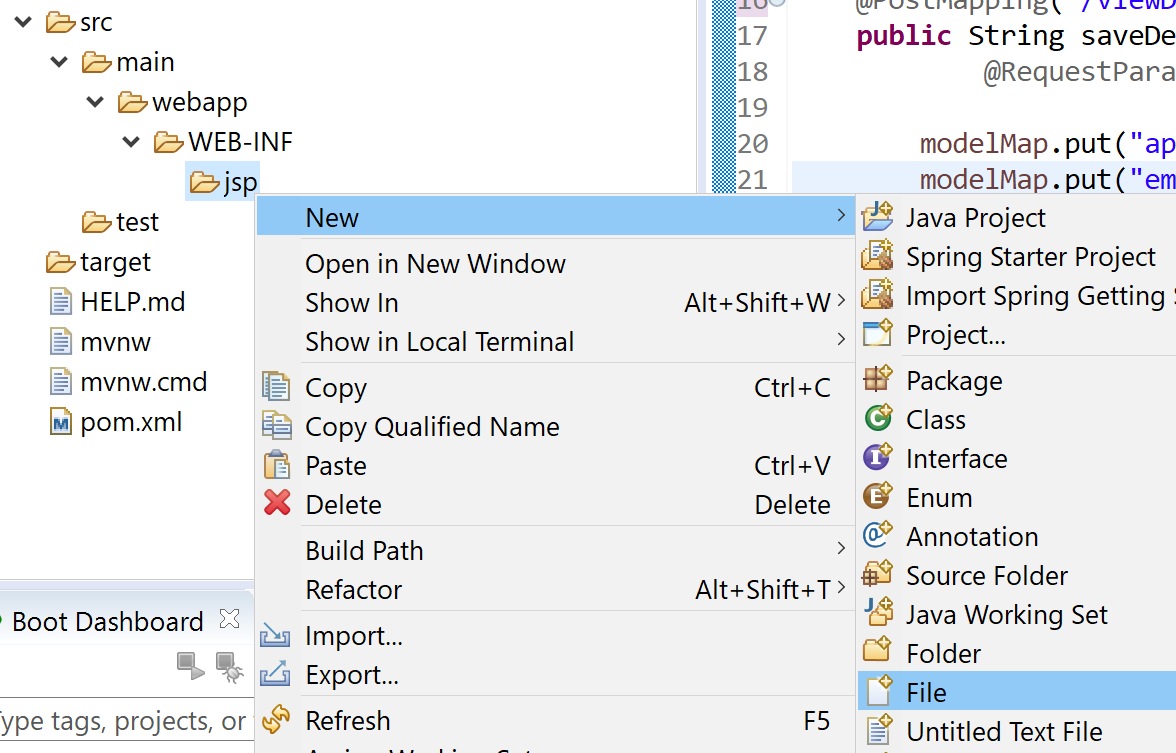
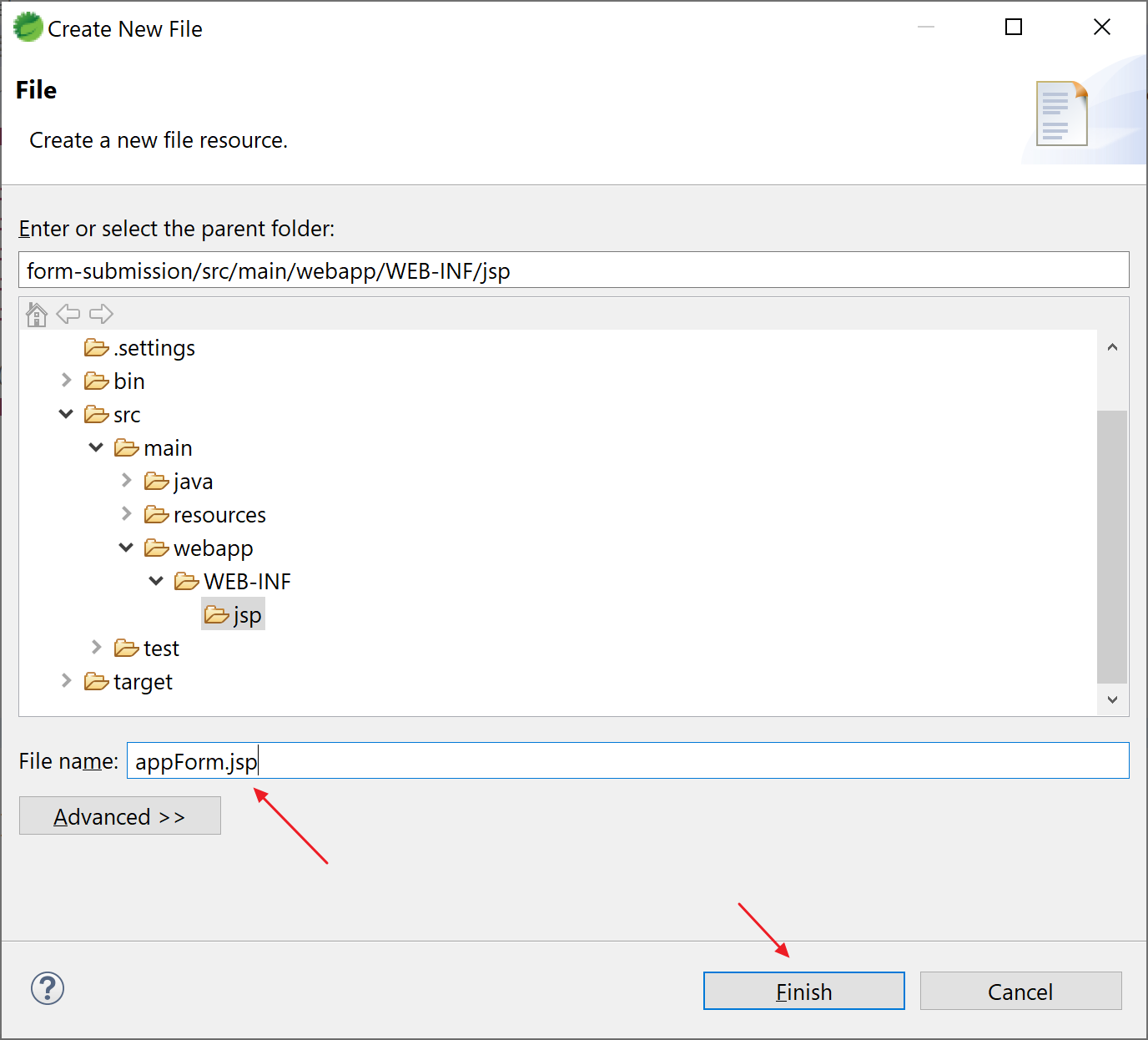
/getAppForm is the URL to the application form

/viewAppDetails is the URL to show the input details of the form

11. Implementing the views. Create a new folder “webapp” under “./src/main/”:

Repeat the steps to create the following structure (src/main/webapp/WEB-INF/jsp/):



11a. Create the file appForm.jsp under “src/main/webapp/WEB-INF/jsp/”:

Open the JSP and insert the following code:

<html>

<head>

<title>form submit exercise</title>

</head>

<body>

<h2>Application Details</h2>

<form method="post" action="viewAppDetails">

Enter Name : <input type="text" name="name" /> <br>

Enter Email Address: <input type="email" name="email"> <br>

<input type="submit" value="Submit">

</form>

</body>

</html>

11b. Repeat the step to create the file appDetails.jsp under “src/main/webapp/WEB-INF/jsp/”:

<%@ page contentType="text/html;charset=UTF-8" language="java"%>

<html>

<head>

<title>form submit exercise</title>

</head>

<body>

<h2>Details of the form</h2>

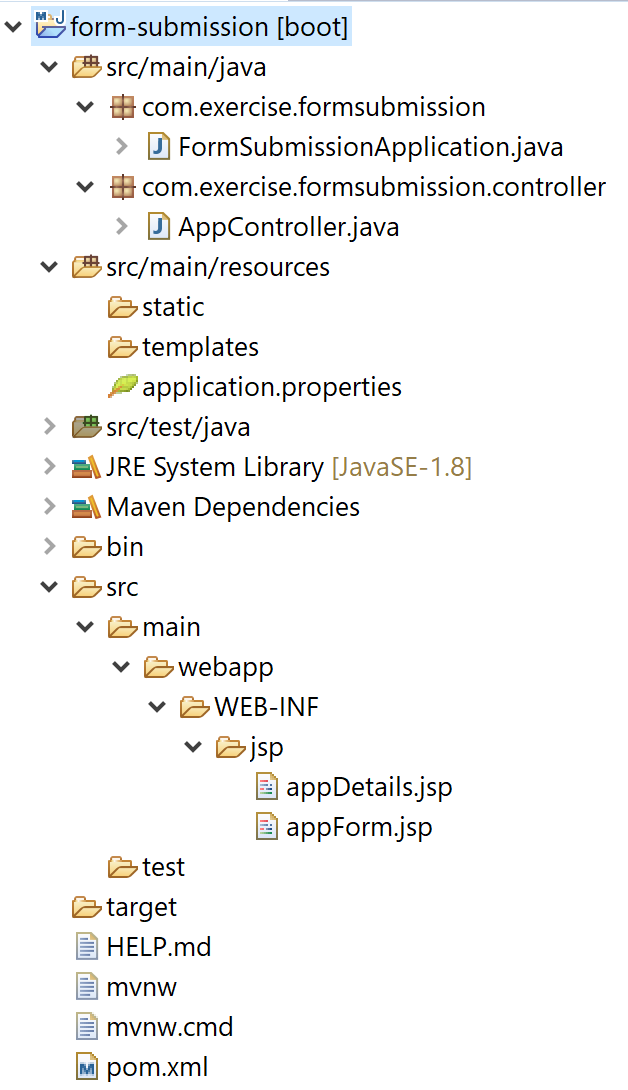
<h4>Name : ${name}</h4>

<h4>Email : ${email}</h4>

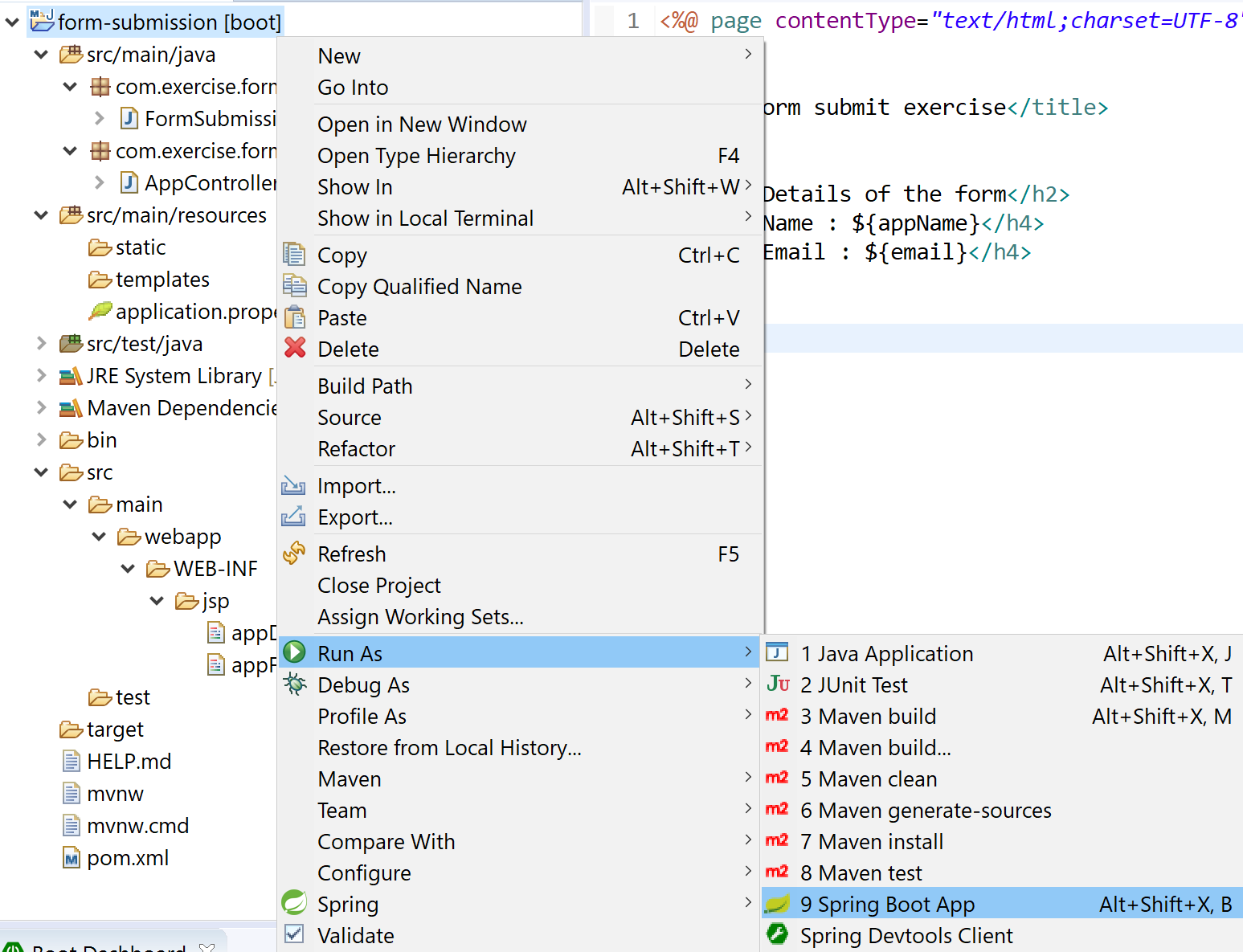
</body>

</html>

12. The final project look like:



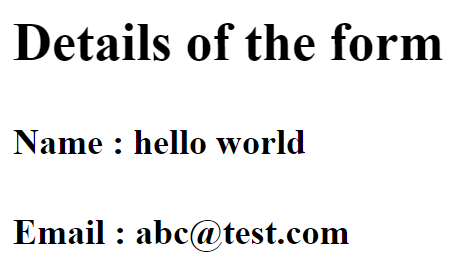
13. Run the application - Right click “form-submission”, click “Run As” and select “9 Spring Boot App”:



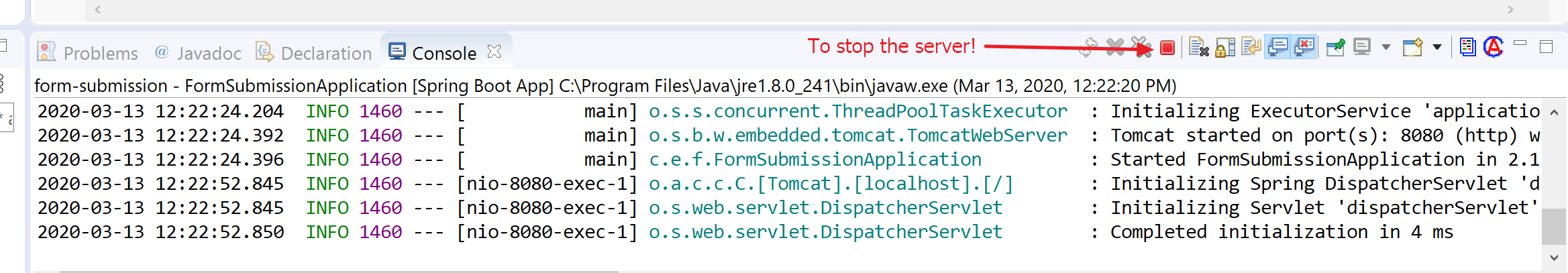
14. Verify the page – Open a browser and visit the following page:

<http://localhost:8080/getAppForm>

After inputting the name and email, click “Submit”:



15. Stop the server:



16. In case you want to use command line to run the application, start up the “Command Prompt” and change directory to your project, e.g. C:\java-ex\form-submission. Type the following command:

mvn spring-boot:run

Note: press Ctrl-C to stop the server.

17. Congratulations, if you can see the page! Try this one if you have problem with this exercise:



# Exercise 2 - bonus (to inject a bean)

Prerequisites:

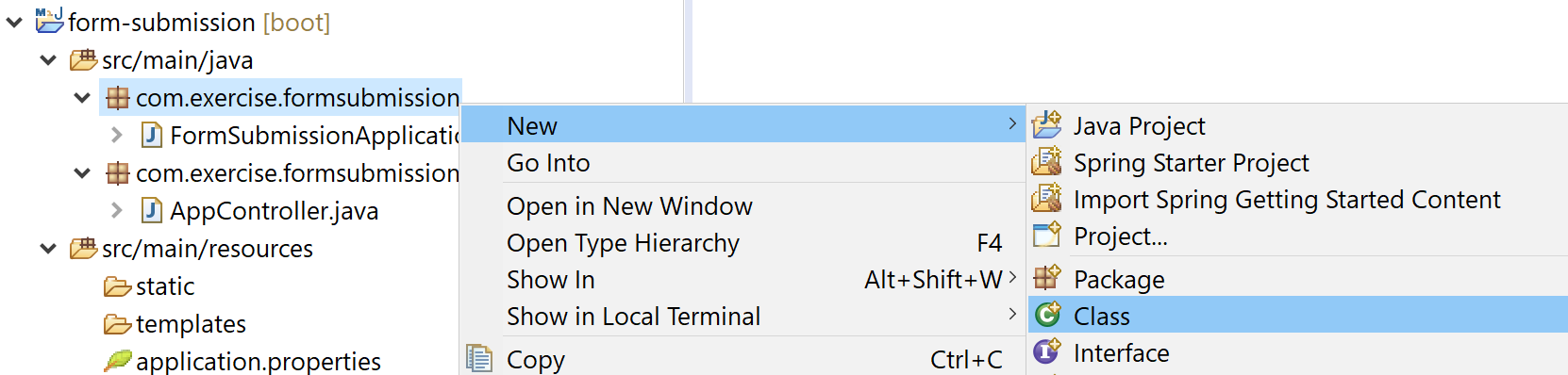
Java installed (e.g. JDK 1.8)

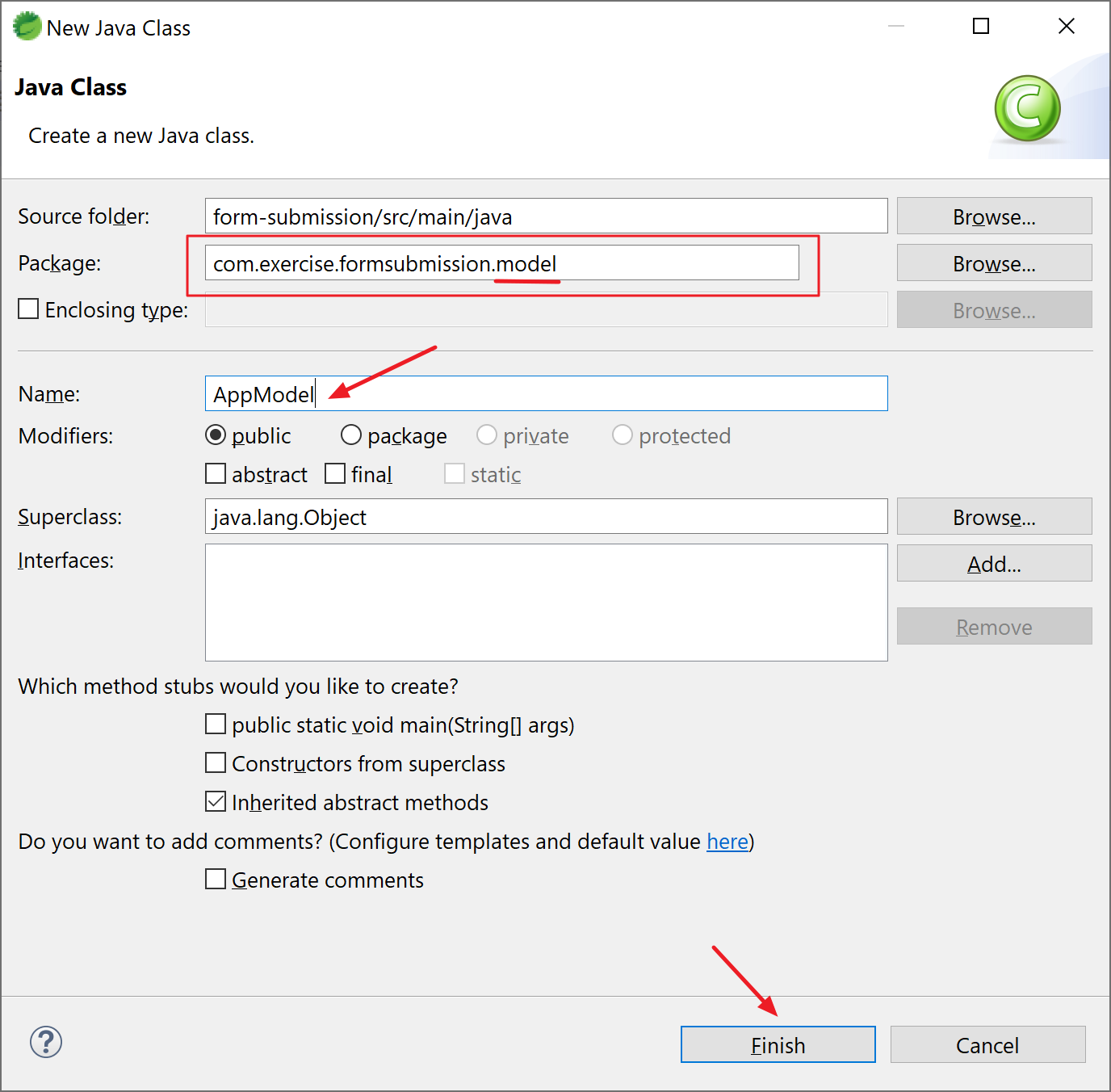
Maven installed (e.g. version 3.6.3)

Spring Tool Suite (STS) (e.g. 4.5.1)

Exercise 1 is finished

1. Create a model class “AppModel” under “./src/main/java/formsubmission/model/”:





2. Open “AppModel.java” and insert the following code:

package com.exercise.formsubmission.model;

public class AppModel {

String appName;

public String getAppName() {

return appName;

}

public void setAppName(String appName) {

this.appName = appName;

}

}

3. Modified “AppController.java”:

3a. Create and autowire the bean – insert the following code to “AppController.java”:

@Autowired

private AppModel appModel;

@Bean

public AppModel getAppName() {

AppModel appModel = new AppModel();

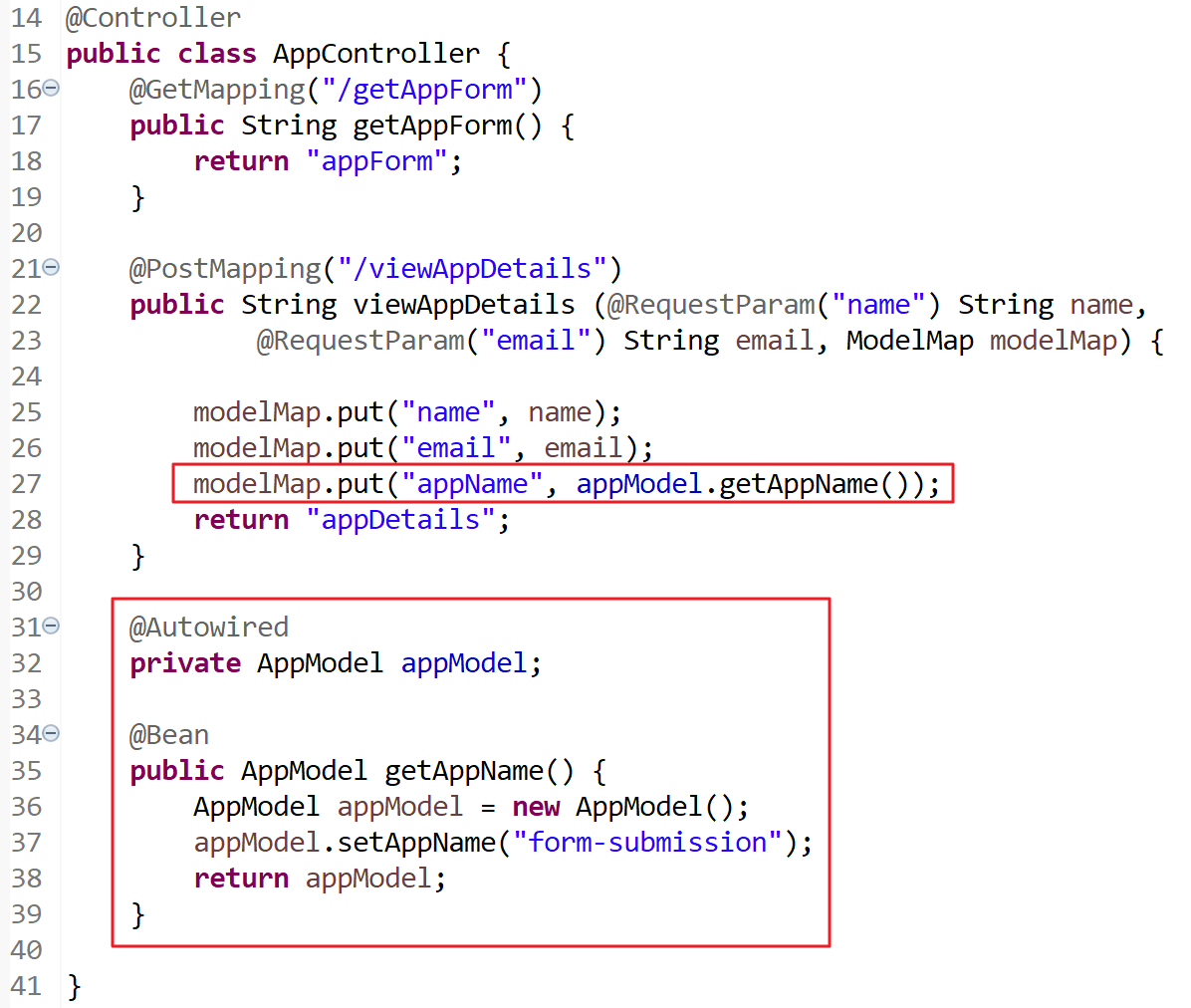
appModel.setAppName("form-submission");

return appModel;

}

3b. Pass the value from the bean to the JSP – insert the following code to function viewAppDetails():

modelMap.put("appName", appModel.getAppName());



Note:

i. we set the appName to “form-submission” (hard coded).

ii. for your reference:

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Bean;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import com.exercise.formsubmission.model.AppModel;

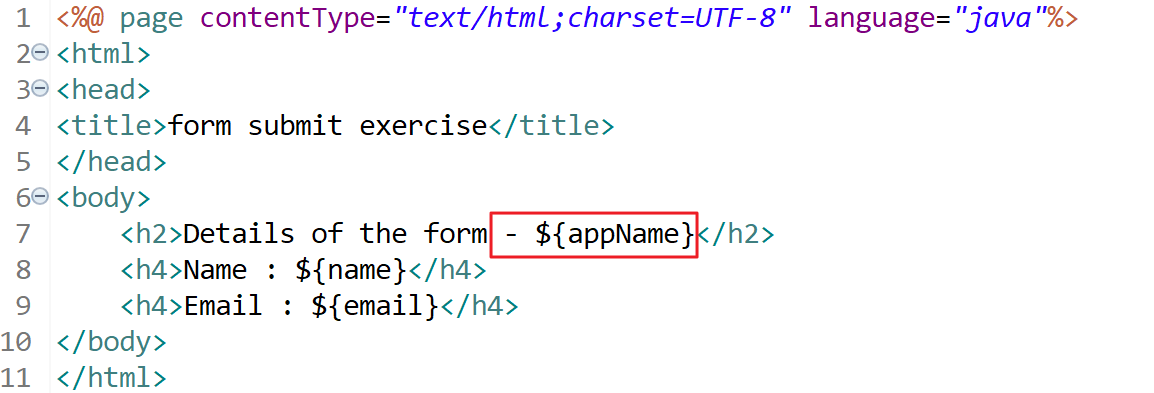
4. Display the value in JSP – edit “appDetails.jsp”

change from

<h2>Details of the form</h2>

to

<h2>Details of the form - ${appName}</h2>

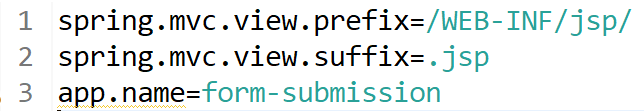


5. Re-start the application, and you should see the appName (form-submission):



6. Get value from property file – edit “application.properties”, insert the following code:

app.name=form-submission



7. Modify the bean to read value from property file – edit “AppController.java”

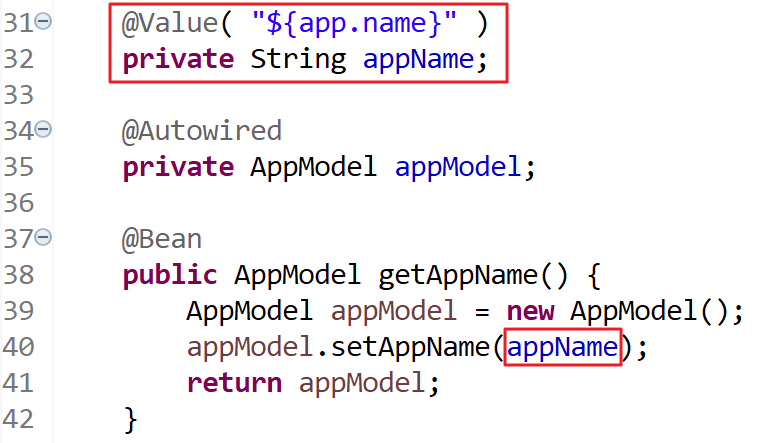
7a. Define the appName:

@Value( "${app.name}" )

private String appName;

7b. Set the bean:

appModel.setAppName(appName);



8. Re-start the application, and you should see the appName (read from property file):



Note: you may modify the value in “application.properties” to see it really works.

9. Congratulations, if you can see the page! Try this one if you have problem with this exercise:



# Exercise 3 – to access database

Prerequisites:

Java installed (e.g. JDK 1.8)

Maven installed (e.g. version 3.6.3)

Spring Tool Suite (STS) (e.g. 4.5.1)

exercise 2 finished

1. Modify “pom.xml” to add dependencies for database persistence. Open the file and insert the following code:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.hsqldb</groupId>

<artifactId>hsqldb</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

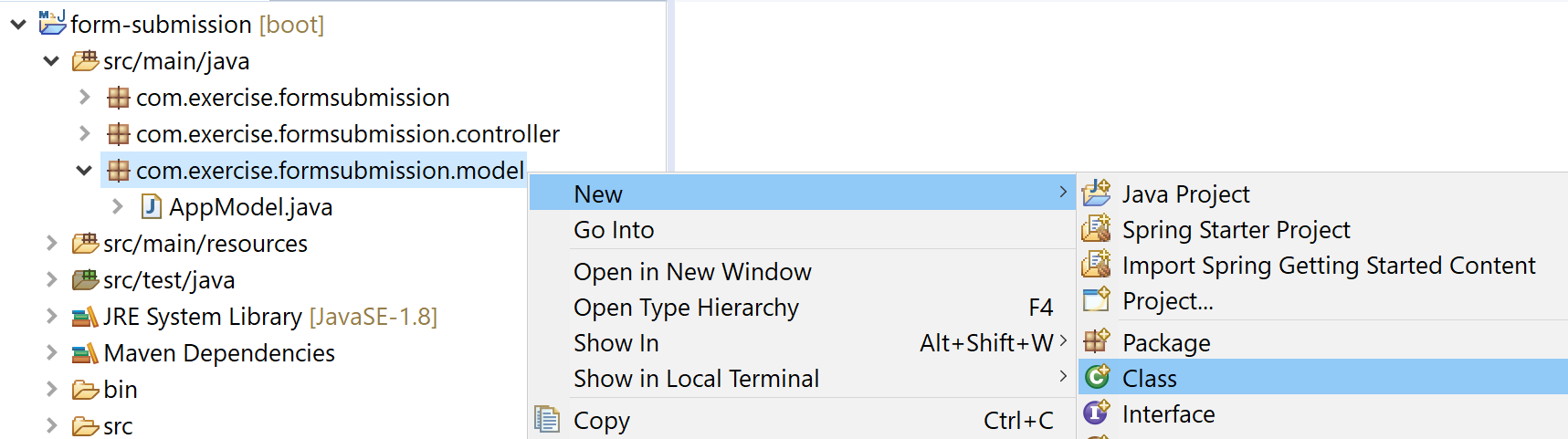


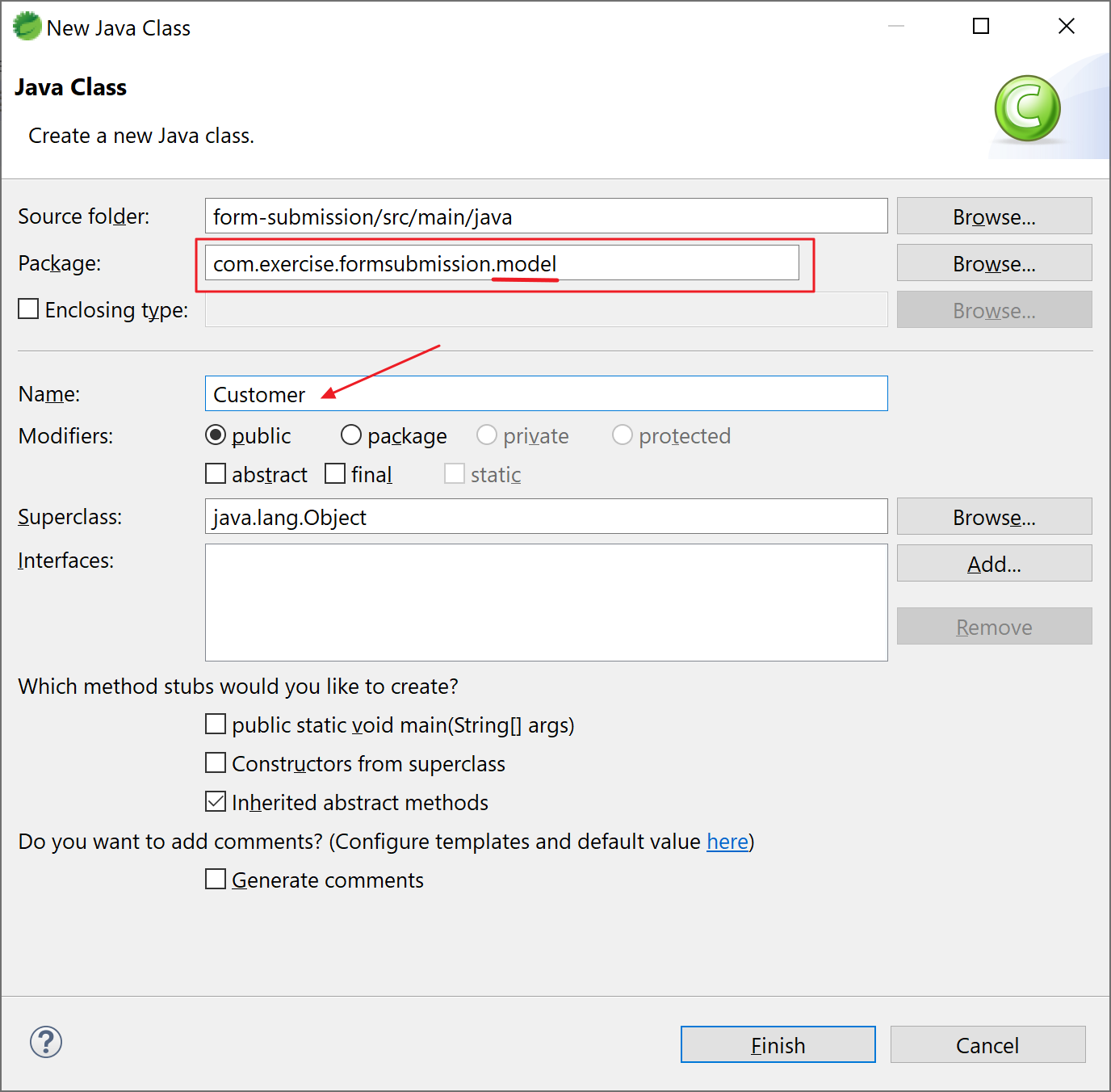
Note: “spring-boot-starter-data-jpa” is for Spring Data JPA

“hsqldb” is for Hyper SQL Database

“jstl” is for JSTL in JSP

2. Create the model class “Customer” under package “com.exercise.formsubmission.model”:





3. Open the newly created file and insert the following code:

package com.exercise.formsubmission.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "customers")

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private long id;

private String name;

private String email;

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

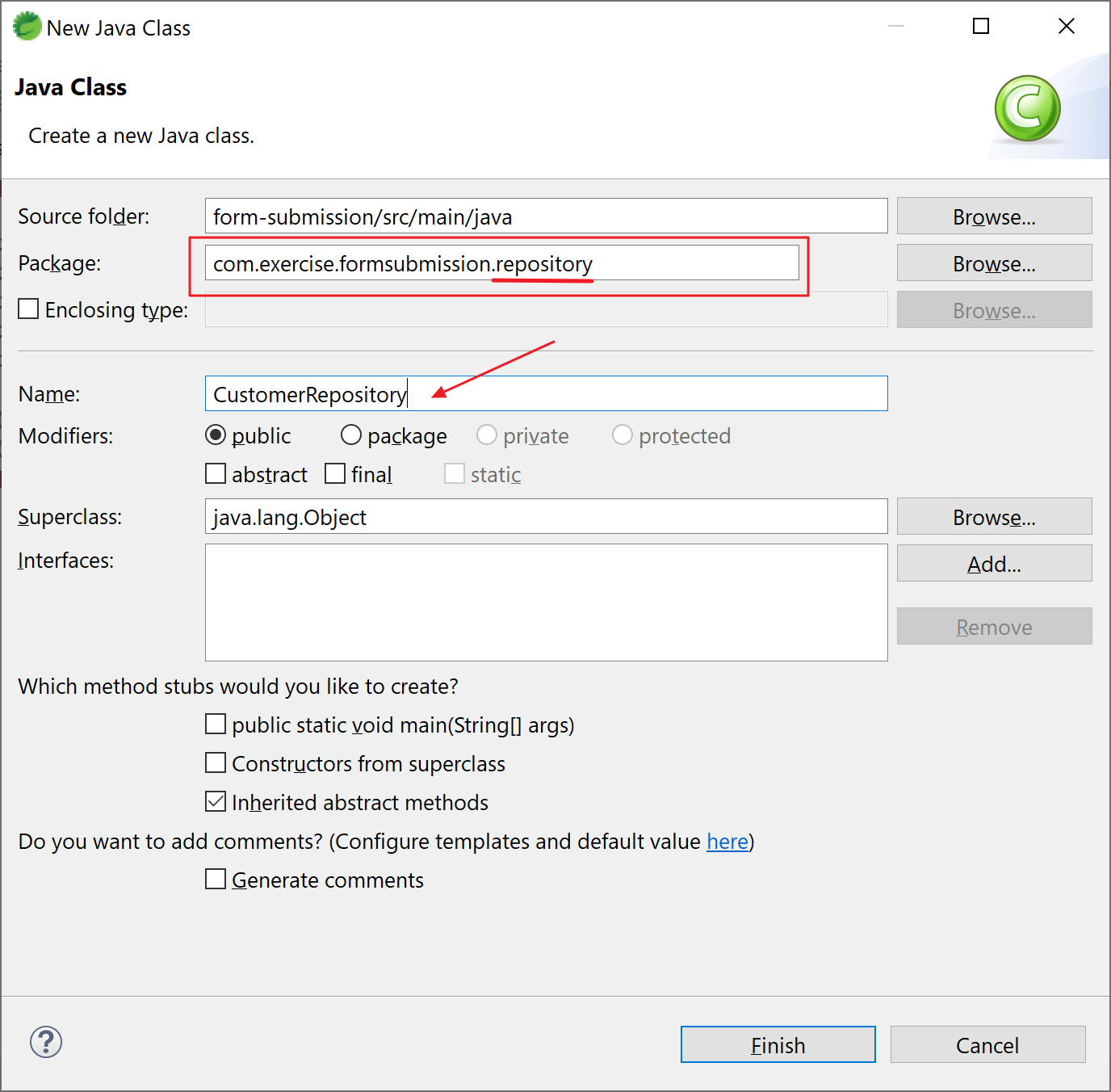
}

Note: @Entity is to annotate this is a model class for persistence

@Table is to annotate the table name in DB

@Id is to annotate the primary key of the table

4. Create the repository class “CustomerRepository” under package “com.exercise.formsubmission.repository” for accessing the database:



5. Open the newly created file and insert the following code:

package com.exercise.formsubmission.repository;

import org.springframework.data.jpa.repository.JpaRepository;

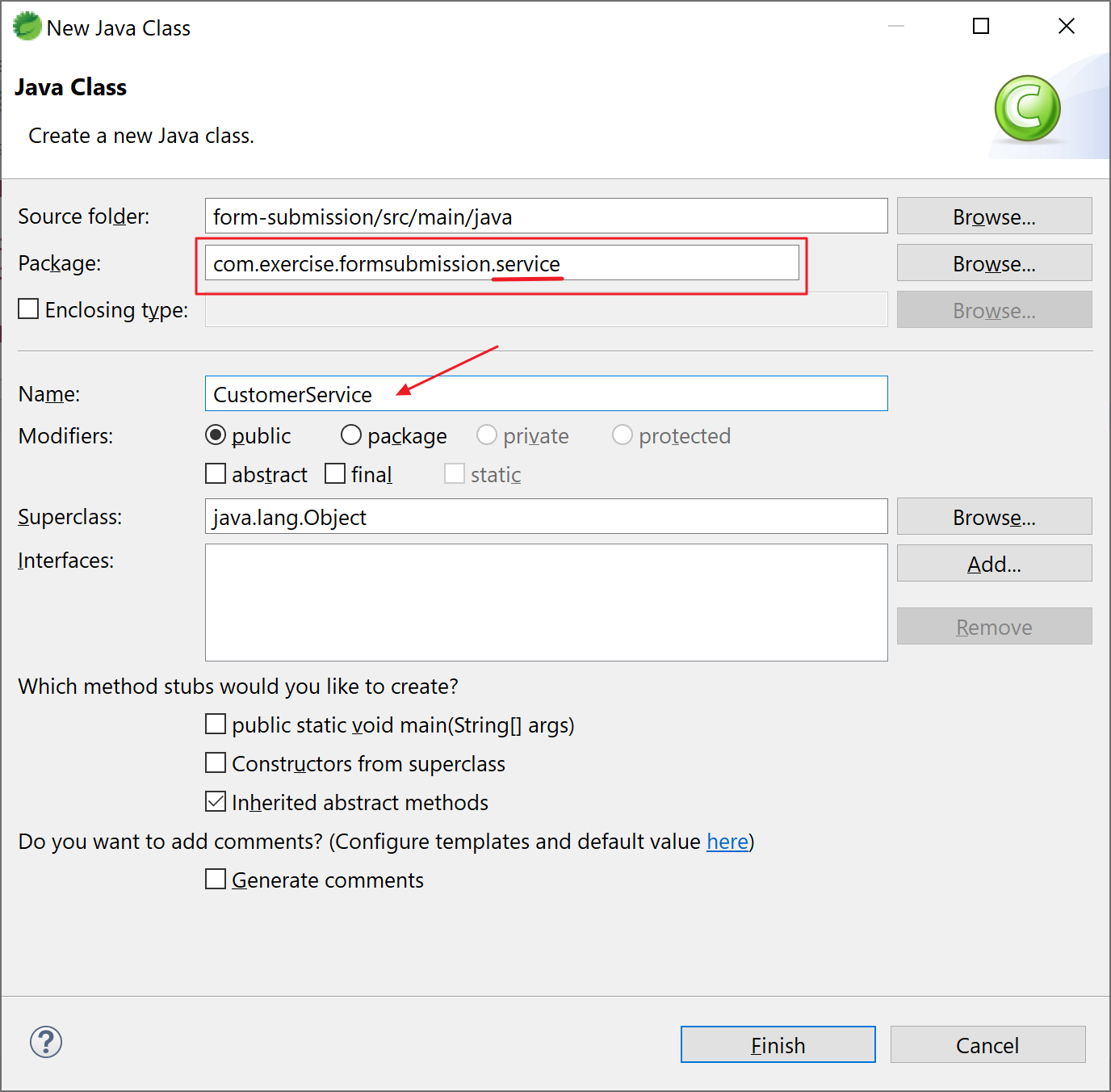
import com.exercise.formsubmission.model.Customer;

public interface CustomerRepository extends JpaRepository<Customer, String> {

}

Note: we don’t need to implement the functions for basic operations, e.g. save(), findAll(), etc.

6. Create a service for controller to call. Create the service class “CustomerService” under package “com.exercise.formsubmission.service”:



7. Open the newly created file and insert the following code:

package com.exercise.formsubmission.service;

import java.util.ArrayList;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.exercise.formsubmission.model.Customer;

import com.exercise.formsubmission.repository.CustomerRepository;

@Service

public class CustomerService {

@Autowired

CustomerRepository customerRepository;

public List<Customer> getAllCustomers() {

List<Customer> customers = customerRepository.findAll();

return customers;

}

public void addCustomer(Customer customer) {

Customer c = customerRepository.save(customer);

}

public void updateCustomer(String id, Customer customer) {

customerRepository.save(customer);

}

public void deleteCustomer(String id) {

customerRepository.deleteById(id);

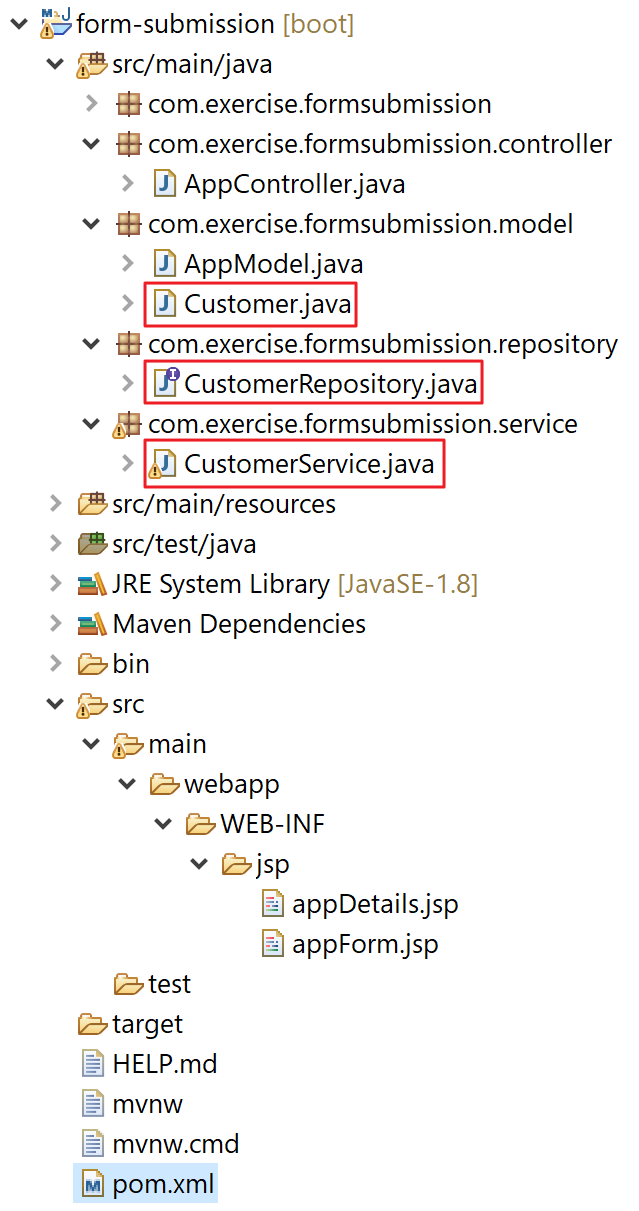
}

}

Note:

1. use @Autowired to inject CustomerRepository
2. we don’t need to implement the funcitons findAll, save and deleteById yourself, Spring Data will implement the functions for us.

8. the structure of the project now should look like this:



9. Use the service in controller – open “AppController.java” and edit the functions accordingly.

9a. getAppForm():

@GetMapping("/getAppForm")

public String getAppForm(ModelMap modelMap) {

logger.info("get customers");

List<Customer> customers = customerService.getAllCustomers();

for(Customer c : customers) {

logger.info("###### {}-{}-{}", c.getId(), c.getName(), c.getEmail());

}

modelMap.addAttribute("customers", customers);

return "appForm";

}

Note: calling customerService.getAllCustomers() to get all customers from database.

9b. viewAppDetails():

@PostMapping("/viewAppDetails")

public String viewAppDetails (@RequestParam("name") String name,

@RequestParam("email") String email, ModelMap modelMap) {

modelMap.put("name", name);

modelMap.put("email", email);

modelMap.put("appName", appModel.getAppName());

Customer customer = new Customer();

customer.setName(name);

customer.setEmail(email);

customerService.addCustomer(customer);

logger.info("add customer");

return "appDetails";

}

Note: saving the form to database using customerService.save().

Here’s the code of AppController.java:

package com.exercise.formsubmission.controller;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.context.annotation.Bean;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import com.exercise.formsubmission.model.AppModel;

import com.exercise.formsubmission.model.Customer;

import com.exercise.formsubmission.service.CustomerService;

@Controller

public class AppController {

final Logger logger = LoggerFactory.getLogger(AppController.class);

@Autowired

CustomerService customerService;

@GetMapping("/getAppForm")

public String getAppForm(ModelMap modelMap) {

logger.info("get customers");

List<Customer> customers = customerService.getAllCustomers();

for(Customer c : customers) {

logger.info("###### {}-{}-{}", c.getId(), c.getName(), c.getEmail());

}

modelMap.addAttribute("customers", customers);

return "appForm";

}

@PostMapping("/viewAppDetails")

public String viewAppDetails (@RequestParam("name") String name,

@RequestParam("email") String email, ModelMap modelMap) {

modelMap.put("name", name);

modelMap.put("email", email);

modelMap.put("appName", appModel.getAppName());

Customer customer = new Customer();

customer.setName(name);

customer.setEmail(email);

customerService.addCustomer(customer);

logger.info("add customer");

return "appDetails";

}

@Value( "${app.name}" )

private String appName;

@Autowired

private AppModel appModel;

@Bean

public AppModel getAppName() {

AppModel appModel = new AppModel();

appModel.setAppName(appName);

return appModel;

}

}

10. Let’s show the retrieved data in the JSP. Open “appForm.jsp” and add the following code below the form.

10a. use JSTL in JSP, add the following at the first line:

<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

10b. add the following code in body:

<br />

Customer:

<table>

<c:forEach items="${customers}" var="customer">

<tr>

<td><c:out value="${customer.id}" /></td>

<td><c:out value="${customer.name}" /></td>

<td><c:out value="${customer.email}" /></td>

</tr>

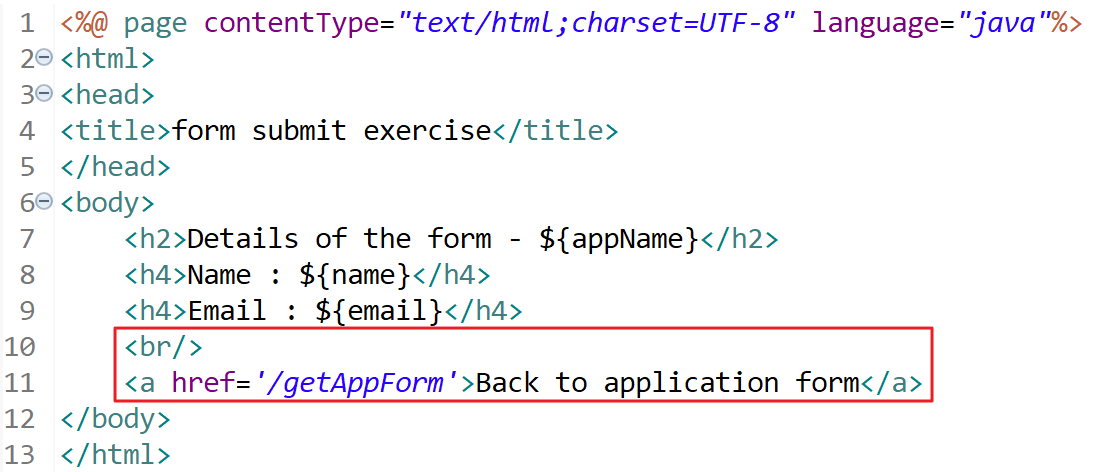
</c:forEach>

</table>



11. Add a link in “appDetails.jsp” to redirect the page back to “appForm.jsp”:

<br/>

 <a href='/getAppForm'>Back to application form</a>

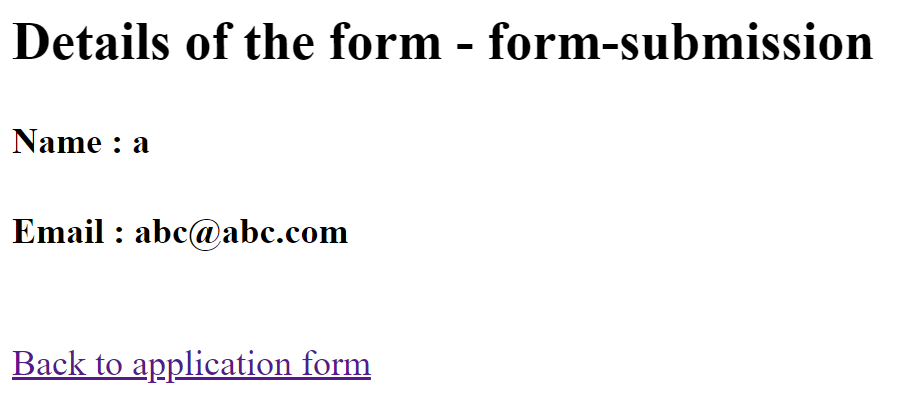
12. Run the application and examine the results. If you do not remember how to start the application, please revisit Exercise 1 step 13.

Open a browser and visit:

<http://localhost:8080/getAppForm>

Note: the list of “Customer” is empty.

After submitting the form, this details and a link “Back to application form” show:



After clicking “Back to application form”, the customer list shows (retrieve from database):



13. Congratulations, if you can see the page! Try this one if you have problem with this exercise:



# Exercise 4 - bonus (to connect external database)

Prerequisites:

Java installed (e.g. JDK 1.8)

Maven installed (e.g. version 3.6.3)

Spring Tool Suite (STS) (e.g. 4.5.1)

Exercise 3 is finished

1. Question: In above exercise, we don’t have any configurations about connecting the database, why does it work?

Answer: You need only include a build dependency to the **embedded** database that you want to use. All the work will be done by Spring Data.

Question: How can I configure to connect to an **external** database?

Answer: Follow this exercise to find the answer.

2. Before any changes to the code, we need an external database. In this exercise, we are going to use HSQLDB again.

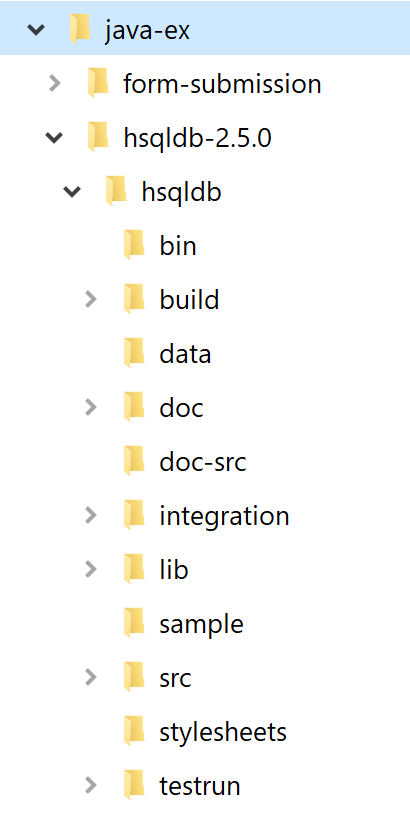
2a. Download the HSQLDB (e.g. hsqldb-2.5.0.zip) from

<http://hsqldb.org/>

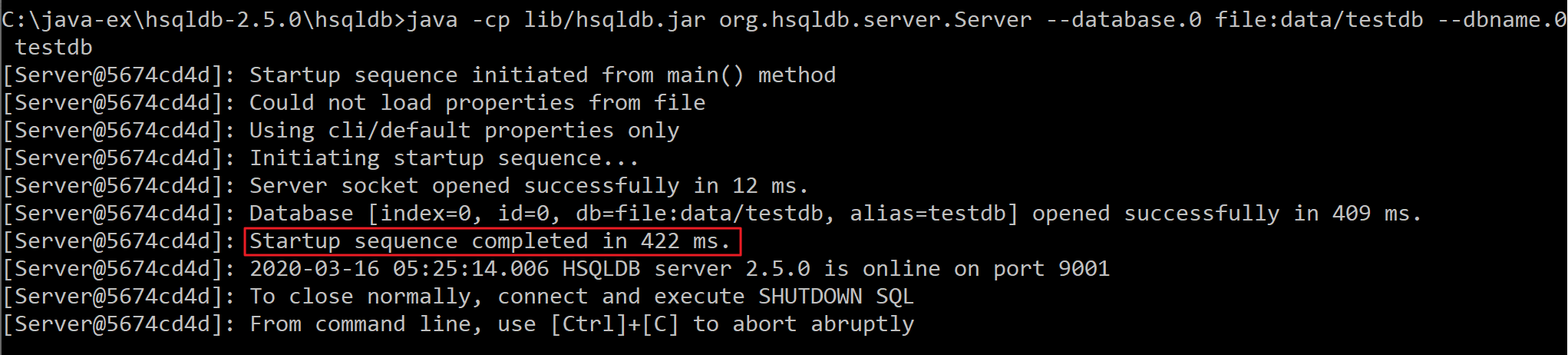
or directly from

<https://sourceforge.net/projects/hsqldb/files/latest/download>.

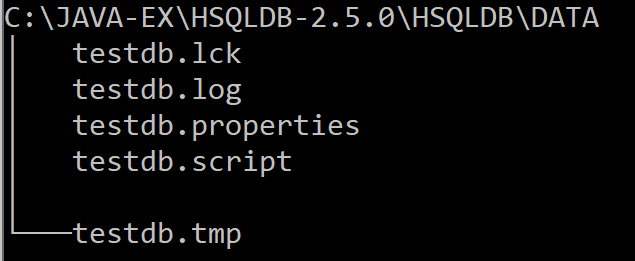
2b. Extract your downloaded file to a folder, e.g. C:\java-ex\hsqldb-2.5.0\.



2c. Start the server. Start a “Command Prompt” and go to the home directory of HSQLDB server, e.g. C:\java-ex\hsqldb-2.5.0\hsqldb and run the following command:

java -cp lib/hsqldb.jar org.hsqldb.server.Server --database.0 file:data/testdb --dbname.0 testdb

Note: data files will be stored in “data” directory with prefix “testdb”:



3. Add configuration for the database connection in “application.properties”:

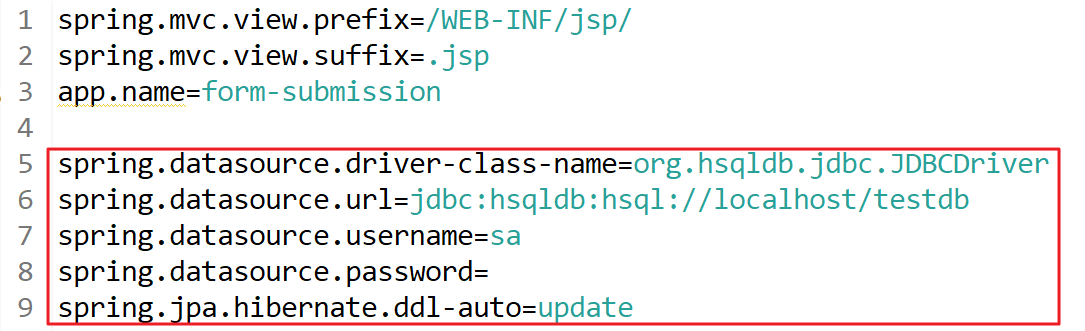
spring.datasource.driver-class-name=org.hsqldb.jdbc.JDBCDriver

spring.datasource.url=jdbc:hsqldb:hsql://localhost/testdb

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

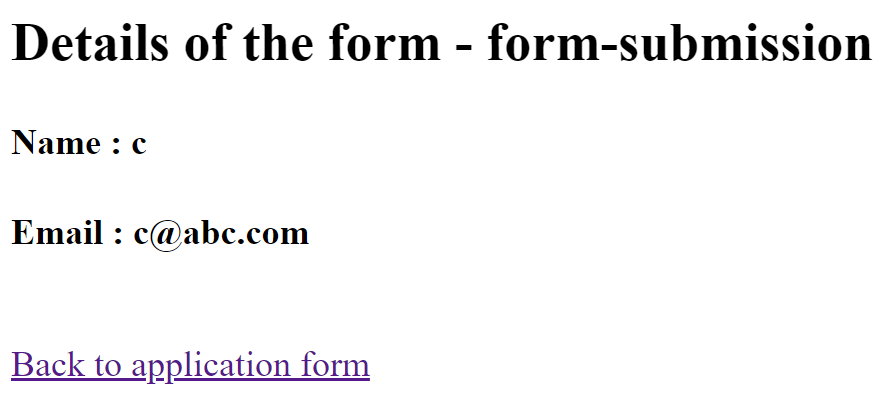


Note: Do not contain any white space in the above configurations.

4. Restart the application and visit the following page:

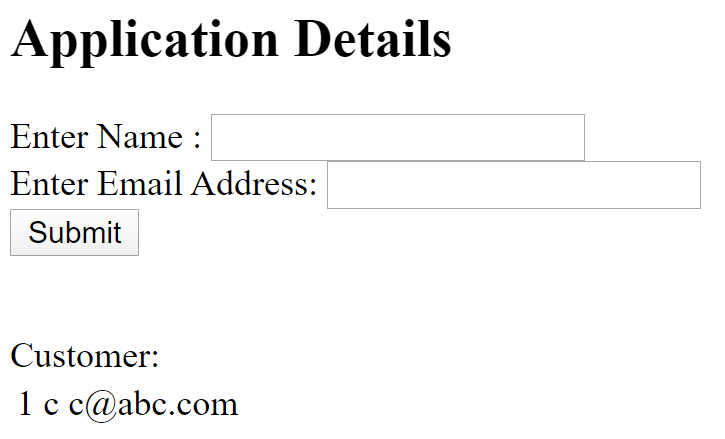
<http://localhost:8080/getAppForm>

4a. Fill in the form and submit:



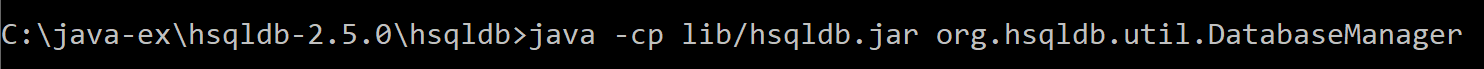
4b. Click “Back to application from”:

4c. The Customer list should show:



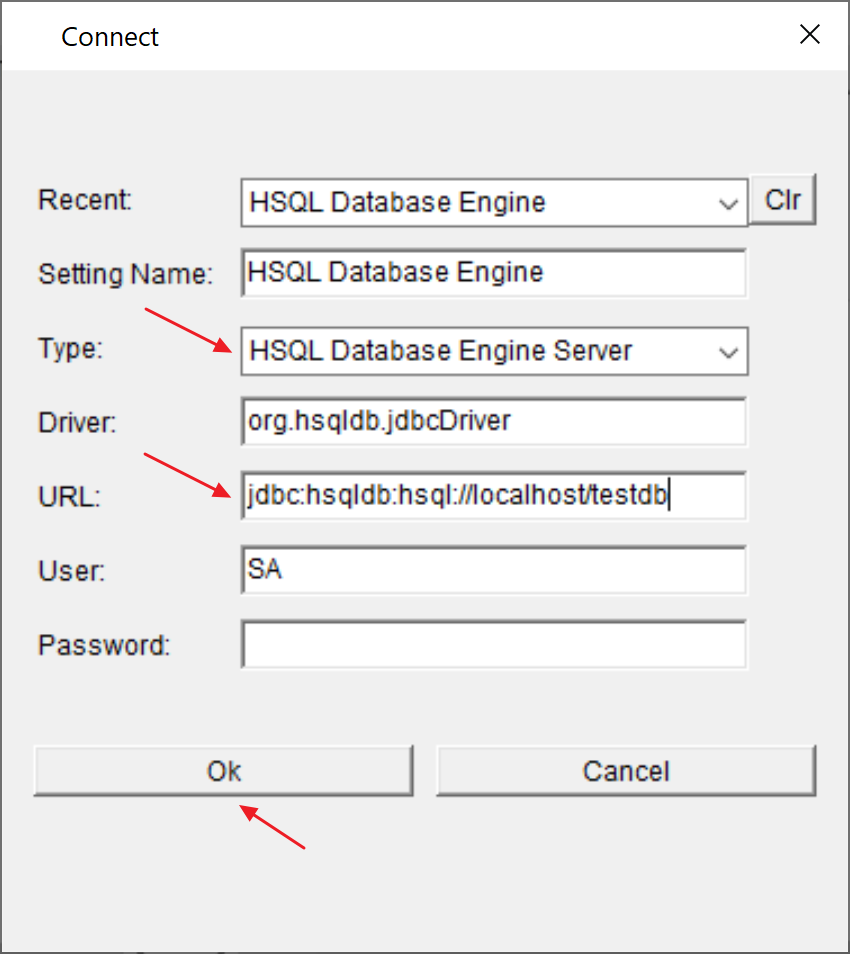
5. Verify if the data is stored in the external database. Start a “command prompt” and go to the home directory of HSQLDB server, e.g. C:\java-ex\hsqldb-2.5.0\hsqldb and run the following command:

java -cp lib/hsqldb.jar org.hsqldb.util.DatabaseManager

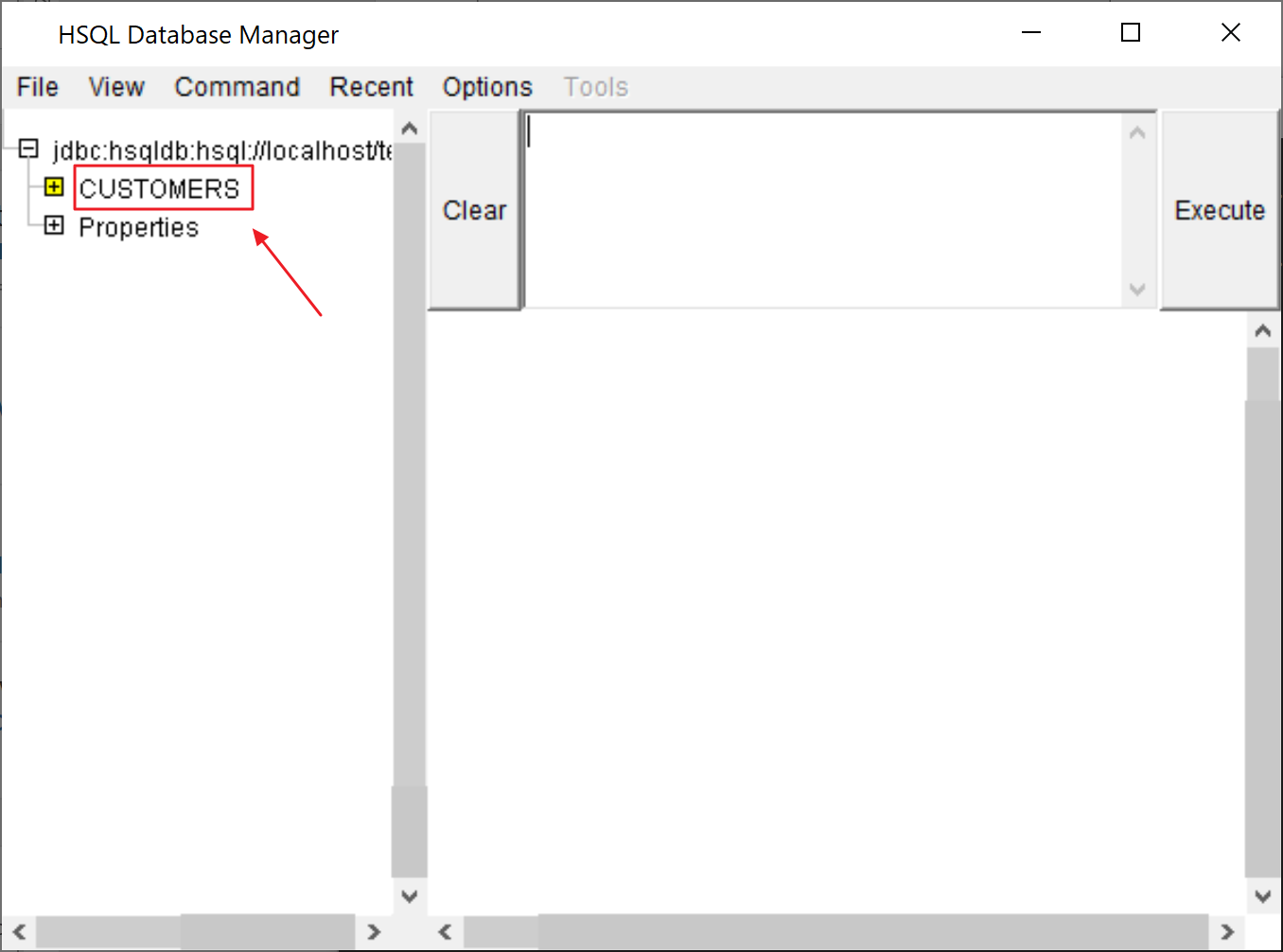


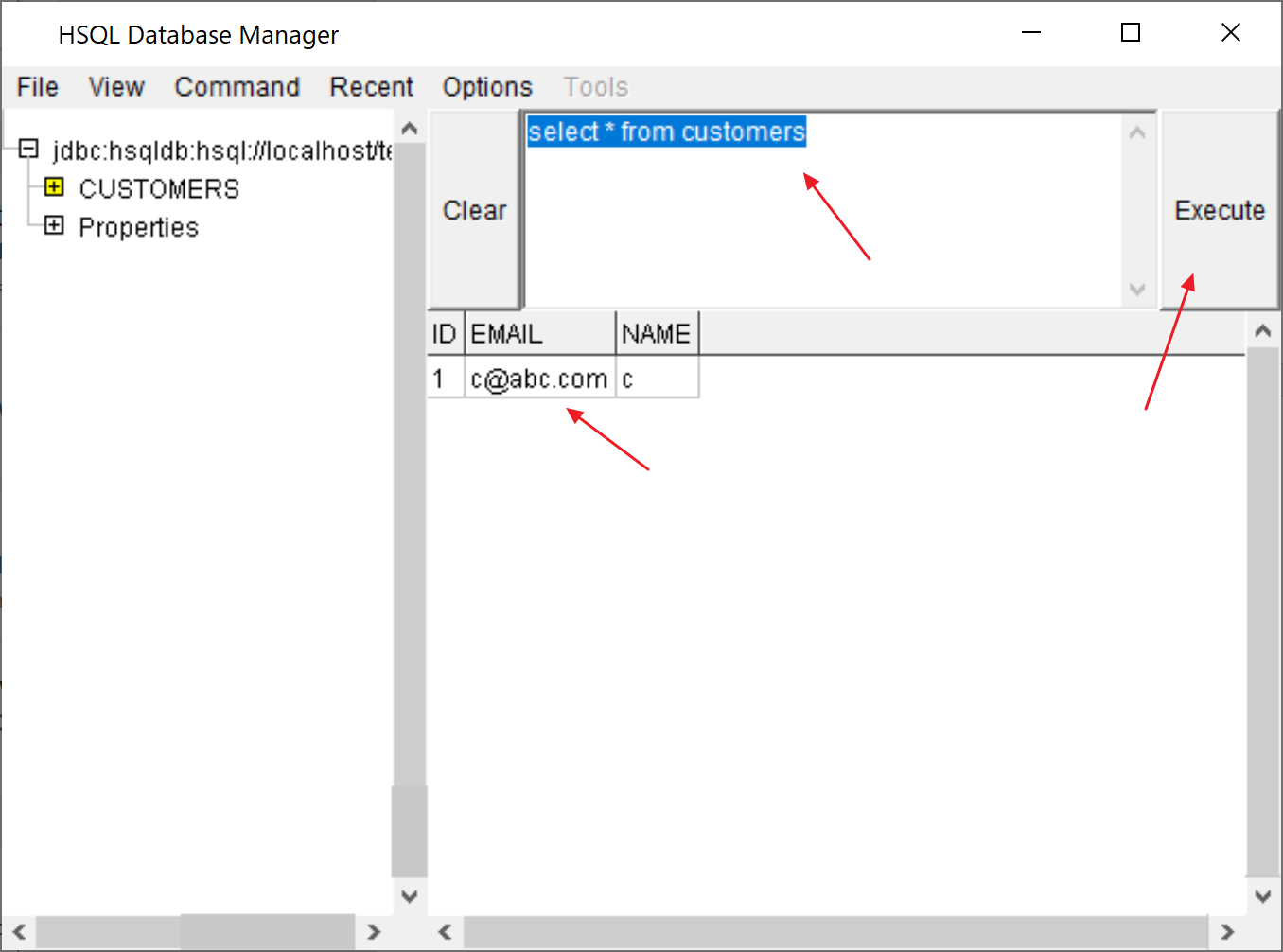
5a. database manager should show with a connection input form:

Type: HSQL Database Engine Server

 URL: jdbc:hsqldb:hsql://localhost/testdb

Note: leave Password empty (this is the default password)

5b. the following panel should show:

5c. verify the table, execute the following SQL:

select \* from customers

You should see the data that you inputted in the form previously.

6. Congratulations! That’s it, we only need to configure the settings in “application.properties”. In case you have problem in this exercise. Try this one if you have problem with this exercise:



~ End of the exercises ~