Exercises

Summary

Build a semi-standard system using the Thymeleaf template engine! MASSIVE firepower here, and you NEED to make this run!

If you've fallen behind in the exercises, there is literally no other time now to catch up. Do it today (literally today!) or risk repeating the course! Seriously! If you're not completely caught up with the class by the midnight tonight or so, you're risking flat out failure. I suspect you can see this material is cumulative, and also that this course moves fast!

Enough lecture though. 😊 My guess is that at least 90% of you are exactly caught up and target. Give yourselves a pat on the back if you're ready to begin this exercise right now!

Step 1

Create a new project called ex32_thymeleaf which uses Spring Web, Thymeleaf, and Lombok repos.

Inside, create a new POJO in a ca.sheridancollege.<yourUserName>.beans package called Course. In Course, create a private String prefix, a private String code, and a private String name. Always a good idea to create a private Long id too – they come in handy sometimes!

Lombok your Course POJO with @Data, @NoArgsConstructor, @AllArgsConstructor, and @Builder.

Step 2

Now create a CourseController class in a ca.sheridancollege.<yourUserName>.controllers package. Annotate your class with @Controller! A simple, but easily forgettable step!

Inside CourseController, use the following line of code at the top to store a List of Course instances. It uses a bit of Polymorphism to stuff a CopyOnWriteArrayList into a List Interface shaped hole, as we've discussed previously.

No need to do any injection here today like last time though, unless you really want to and are confident you can make it work with some modifications to the code below! For today, simple and easy so we can focus on the Thymeleaf instead!

```java
List<Course> courseList = new CopyOnWriteArrayList<Course>();
```

Write a GET mapped method for requests to "/" called index. Make sure your method grabs a Model model instance as a parameter too! We'll need it in a sec..   your method should return to "index".

Just BEFORE your return to "index" though, add your courseList instance to your model as well.

```
model.addAttribute("courseList", courseList);
```

Step 3

Now create an HTML file called index.html in your resources / templates directory. This will have two parts: a table to display our courses so far, and a form where we can easily add a new one!

Don't forget, we must add our Thymeleaf XML namespace to this page to have it function with Thymeleaf templating technology! Change your <html> tag to the following:

```
<html xmlns:th="https://www.thymeleaf.org">
```

To display the courses, we need a <table> with table headings (<th>Heading!</th>) for each field, ideally. This part is easy.

Also inside the table, but as a second row (or third or whatever – depends on what we want to display!), add the following code:

```
<tr th:each="aCourse:${courseList}">
      <td th:text="${aCourse.id}">55</td>
      <td th:text="${aCourse.prefix}">PROG</td>
```

Repeat for code and name, then close the row tag and table with closing tags.

Step 4

Now, underneath your table, create a form.

Our form should use the POST method. It needs an action, but for styling purposes, our regular action will just go to "#", otherwise known as nowhere because our server isn't there to process it yet. Meanwhile, if we're using Thymeleaf with Spring Boot, on the other hand, a th:action attribute going to @{/addCourse} should do the trick!

```
<form method="post" action="#" th:action="@{/addCourse}">

</form>
```

Be mindful of the symbols in front of Thymeleaf variables! Use $ for most model attributes, and use @ for locations!

Inside the form tags, we need regular fields for our Course POJO params. Here are a few you can copy if you wish:

```
ID: <input type="number" name="id" /><br>
Prefix: <input type="text" name="prefix" /><br>
Code: <input type="number" name="code" /><br>
Name: <input type="text" name="name" /><br>
<input type="submit" value="Add Course!" />
```

Step 5

This last part is easy! Back at the Controller, create a method to handle POST requests to "/addCourse", and retrieve your @RequestParams per the form. This part is on you!

Create a new Course instance out of your request params, and add it to your list as follows:

```
Course course = new Course(id, prefix, code, name);
courseList.add(course);
```

Finally, add your updated list to your model for display by Thymeleaf again, and then return to "index" as before!

```
model.addAttribute("courseList", courseList);
```

Show me this working when you get it done! If you have time, make it look pretty too! It's code we'll use again and again, and having an easy pattern to follow is sometimes helpful! 😊

For the exercise submission, add at least three unique courses using our system, then take a screenshot of them displayed using Spring Boot and Thymeleaf with a bit of style! Awesome!