Lab 12: Spring and the Web

In this lab, you will put a Web front end on a Spring application.

Objectives:

- To use SpringMVC

Part 1: Spring MVC with Java Configuration

In this part, you will write a SpringMVC Web application that will act as the user interface for your Spring beans. You will use the Java configuration approach.

Steps:

_1. Your first job is to generate an Eclipse project for this lab. We will use the Spring Boot and Gradle to satisfy dependencies and generate the project. Follow this procedure:

   a. In the Windows Start menu, at the bottom, enter **cmd** and press Enter to open a command prompt window.

   b. Change to the lab's directory. If your computer has the standard lab setup:

```
cd \Users\username\webclass\workspace\lab12
```

   c. Use the Spring Boot Initializr to generate a starter project:

```
spring init -b 1.3.1.RELEASE -d=web --build gradle -x
```

   d. In the same fashion as in earlier labs, use the *gradlew* script to generate an Eclipse project, then import it into Eclipse.

   e. Expand the new project's *src/main/java* folder and right-click on the *com.example.demo* package and choose *Refactor* - *Rename* to change the package name to **com.oaktreeair**.

   f. In the *com.oaktreeair* package, right-click on DemoApplication.java and rename it to **FrequentFlierProgram**.

_2. Next, copy files from the *lab06JavaConfig* project to the new Web project:

   a. In the Project Explorer pane, expand the *lab06JavaConfig* project's *src/main/java* folder, then right-click on the *com.oaktreeair.ffprogram* package and choose *Copy*.

   b. In the Project Explorer, expand the *lab12* folder, then right-click on the *src/main/java* folder and choose *Paste* to copy the Spring bean classes to the new project.

_3. Next, let's create a simple Spring Java configuration file that defines a couple of Flier components (note that this is artificial just for this lab: in the real world, the Flier info would be stored in a database):

   a. Right-click on the *com.oaktreeair* package and create a class named **FrequentFlierConfig**.

   b. Annotate the class with *@Configuration*.

   c. Write two methods named **flier01()** and **flier02** that create, initialize and return Flier objects. Annotate each of the methods with *@Bean*. Initialize the Flier objects to values of your choice. Note that the method

Copyright © Descriptor Systems 2016. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

names assign names to the beans, so when the program runs, the container will have beans named *flier01* and *flier02*.

Note: Be sure to assign each of the Flier objects an initialized ContactInfo object.

_4. Now, create an HTML input form so the user can enter a flier bean ID:

a. Right-click on the project and create a new folder named **src/main/webapp/WEB-INF/jsp**.

b. Right-click on the newly created folder and choose *New - Other - Web - JSP File* and create a JSP named **findFlier.jsp**. On the third page of the wizard, uncheck the box to "Use JSP Template".

Eclipse opens the new JSP into the editor.

c. Enter the following:

```
<html>
  <head>
    <title>Find Flier by Bean ID</title>
  </head>
  <body>
    <form action="findFlier.html" method="post">
      <p>
        Enter flier bean ID:
        <input type="text" name="flierId">
      </p>
      <p>
        <input type="submit" value="Find">
      </p>
    </form>
  </body>
</html>
```

This defines a standard HTML input form into which the user can enter a bean ID.

_5. Next, define a JSP that will display the results of finding a bean:

a. Right-click on the *jsp* folder and create a new JSP named **showFlier.jsp**, again using no JSP Template.

b. Enter the following text:

Copyright © Descriptor Systems 2016. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core"
  prefix="c"%>
<html>
  <head>
    <title>Flier Information</title>
  </head>
  <body>
    <h3>Flier Information</h3>
    <p>${msg}</p>
    <p>
      <c:if test="${not empty flierName}">
        Name: ${flierName}
      </c:if>
    </p>
    <p>
      <c:if test="${not empty flierEmail}">
        Email: ${flierEmail}
      </c:if>
    </p>
  </body>
</html>
```

This page looks for the flier information stored at request scope and displays it.

Note: You can ignore errors and warnings in this file on the JSTL *taglib* directive and c:if lines.

_6. Next, create an *application controller* class that responds to the input form and stores the flier info in a *model and view* object:

a. Create a new class in a new *com.oaktreeair.ffprogram.controllers* package named **FindFlierController**.

b. Configure the controller class with the following annotations:

```
@Controller
@RequestMapping("/findFlier")
```

These annotations configure the class as a SpringMVC controller and indicate that it should respond to URLs that contain "/findFlier".

c. Within the class, setup an injection into a field so that Spring can inject an application-context reference:

```
 @Autowired private ApplicationContext applicationContext;
```

d. Write a method that Spring MVC will invoke when the user issues an HTTP "get" to the controller:

Copyright © Descriptor Systems 2016. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
@RequestMapping(method = RequestMethod.GET)
public String setupForm(Model model)
{
    return "findFlier";
}
```

Note that the method does nothing yet; you will add to it later.

e. Start writing a method to handle the posted form submit request:

```
@RequestMapping(method = RequestMethod.POST)
public String submitForm(@RequestParam("flierId") String flierId,
        Model model)
{


    return "showFlier";
}
```

Note how we annotate one of the method's parameters with @RequestParam so that Spring MVC injects the value that the user enters into the form. Check back in findFlier.jsp and ensure you understand how this will work.

f. In the *submitForm* method, write a try-catch block:

```
try
{

}
catch(NoSuchBeanDefinitionException e)
{

}
```

g. In the *try* block, use the Spring application context in the normal way to attempt to get the bean whose bean ID is stored in the flierId parameter:

```
Flier flier = . . .;
```

This will throw a NoSuchBeanDefinitionException if there's no bean with the specified ID in the application context.

Then store the flier info into the model object:

Copyright © Descriptor Systems 2016. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
model.addAttribute("msg", "Flier found successfully");
model.addAttribute("flierName", flier.getFlierName());
ContactInfo inf = flier.getContactInfo();
model.addAttribute("flierEmail", inf.getEmailAddress());
```

h. In the *catch* block, store a not-found message into the model-and-view object:

```
model.addAttribute("msg", "Flier " + flierId +
        " not found");
```

_7. Edit the *src/main/resources/application.properties* file and add:

```
spring.mvc.view.prefix: /WEB-INF/jsp/
spring.mvc.view.suffix: .jsp
```

This configures Spring MVC so it can find your JSPs.

Note: Ensure that there are no trailing spaces in those two lines.

_8. Edit the build.gradle file and modify it:

a. In the *apply-plugin* list, add:

```
apply plugin: 'war'
```

b. In the bottom-most *dependencies* block, add:

```
providedRuntime("org.apache.tomcat.embed:tomcat-embed-jasper")
providedRuntime("javax.servlet:jstl")
```

_9. To run and test, in your command prompt, use *gradlew bootRun* to start the application within the embedded Tomcat server. Then open a web browser and enter:

```
http://localhost:8080/findFlier
```

You should see your input form. Enter **flier01** into the form and press Submit - you should see the flier info.

Press the back button in the browser and enter **flier04** and press Submit - you should see the not-found message.

_10. The application worked, but let's fix up a minor inconvenience:

a. Open findFlier.jsp into the editor and find the *input* tag for the flier ID. Add the following attribute to the tag:

```
value="${flierId}"
```

b. Open the FindFlierController.java class into the editor, and modify the setupForm method so that it initializes a flier ID for the form:

Copyright © Descriptor Systems 2016. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.

```
        model.addAttribute("flierId", "flier01");
```

_11. To test with the updates, in your command prompt, kill, then restart the application.

Back your web browser, do a hard refresh (hold the Ctrl key and press Refresh) of the *http://localhost/findFlier* URL and note that the form is "pre-filled" with a good flier bean ID -- that's because your controller returned a model object from the *setupForm* method.

Copyright © Descriptor Systems 2016. All Rights Reserved. Materials may not be reproduced in any form without prior written consent of Joel Barnum.