Spring Boot Microservices - Guide to Running the Practicals

The project is called Fleetman - it manages a fleet of vehicles.

Start code can be found in the course download or the GitHub repository
https://github.com/DickChesterwood/Starting-Code-for-Fleetman

Chapters 1-3 don't need this code - you can work in a clean IDE for this.

Chapter 4 runs through the entire process of running the system. *Do ensure you watch chapter 4 in full - although it is a bit tedious, there's some important information about the functions of the services. You'll be adding extra code to the project also.*

In chapters 5 through to 11, if you return to the course after a break, you'll need to restore your application by running all of the steps again - this is a guide to what you need to do.

> *These manual steps are definitely tedious. In real life, you wouldn't want to be doing all of these steps before you can start coding - you definitely want to have scripts, configuration control and local virtual machines. This will be the focus of the next course in this series!*

Chapter 5:

At the start of this chapter you will need to have your microservices up and running as described in chapter 4. Here's a recap of the instructions:

1. Start up your ActiveMQ server as demonstrated in Chapter 2.
   Ensure any code you have running from chapters 2 and 3 has been terminated!
   (If you forget to run the broker, the microservices will "back off" and wait until the broker is running, so this doesn't have to be done first).

2. Start up the Position Simulator. You can do this as on the video by opening the workspace and running it as usual in Eclipse. However, we rarely alter the code in this microservice, so you can run it on the command line using **mvn spring-boot:run** and forget about it. (If you don't have mvn on your path, you can use the **mvnw** contained in the project directory)

3. Start up the Position Tracker[1]. We will be modifying code in this project, so I recommend you start this in an instance of your IDE - you should have set up the workspace for this in Chapter 4. Run the main class **PostionreceiverApplication.java.**

[1] Warning: if you've just taken this project from the starting code, you need to have completed the work in chapter 4 of writing an MDP to read from the Queue. If you really can't be bothered, you can copy the class MessageProcessor.java from the solution code.

4. Start up the web front end. We called this project "fleetman", although "fleetman-webapp" might have been a better name (rename it if you want). You definitely want to do this in your IDE. Run the main class **FleetmanApplication**.

5. Visit http**://localhost:8080/website/vehicles/list.html** to get a list of vehicles (should be two - don't worry about the nulls) and follow a link to see the map. You will need to have included a google maps API key, as described in chapter 4.

You should now be good to study chapter 5.

Chapters 6-11:

You will need to follow all the steps as described above, AND you will need to run your Eureka server. Although Richard uses the IDE to run Eureka, it rarely changes so you can safely run it in a terminal using mvn spring-boot:run (or mvnw spring-boot:run)

After the Course

Once you've finished, to run your full system you will need to do all of the above AND run the Config Server - again this is best run on the command line rather than within an IDE.