SOFTENG 325 Software Architecture
Lab 1

Ian Warren

July 22, 2017

The purpose of this lab is to reinforce week 1's lecture material and to give you some practical experience with developing simple distributed applications. In addition, it requires you to refamiliarise yourself with Maven, which we'll be using extensively throughout the first part of the course.

Tasks

Task 1: Build the supplied Concert application

A simple client/server application has been developed using Java's support for sockets and the TCP protocol. The server maintains a collection of concerts, and offers an interface with CRUD (create/retrieve/update/delete) functionality to clients. The application implements a simple request/reply protocol, allowing clients to send requests to the server, which processes the requests and generates replies.

The project is named `softeng325-labl1-sockets`, and is a *multi-module* Maven project. It comprises 3 modules:

1. `softeng325-labl1-sockets-common`. This contains classes that are shared by the client and server modules. Specifically it contains request and response message classes, class `Concert` to represent concerts, and a `Config` class that defines network properties.

2. `softeng325-labl1-sockets-client` The client module contains a single JUnit class that implements several test cases to test the server's functionality.

3. `softeng325-labl1-sockets-server` This module contains class `Server` that implements the server program.

Multi-module projects are structured using a *parent* project (`softeng325-labl1-sockets`) that contains modules. The parent project has a POM file that declares the modules. Each module also has its own POM file that inherits declarations, e.g. plugin declarations, from the parent POM.

(a) Import the project

Import the project into your Eclipse workspace. When importing the project using Eclipse, select the parent project – and this will automatically import the associated modules. In Eclipse's Package Explorer view, you will see four projects: the parent plus the 3 modules. The parent project doesn't contain any code, but, as noted above, it has an important POM file.

Aside: importing a project does not make a copy of the project in your Eclipse workspace directory. It is good practice to use a separate development directory, e.g. `softeng325-dev`, to store all of your source code. After you've imported a project from your development directory into the workspace, any edits to sources files that you make using Eclipse will update the files in the development directory. It is typical that a workspace directory contains only Eclipse metadata.

(b) Build the project

Using Maven, build the complete project. The easiest way to do this is to run a Maven goal, e.g. `install`, on the parent project. This will build the `common`, `client` and `server` modules and place the resulting JAR files in your local Maven repository.

(c) Run the application

Run the server using the generated JAR file outside of Eclipse. Using a command window/terminal, navigate to the directory, within your development directory, containing the JAR file. Assuming that your development directory is named `softeng325-dev` and that the parent project is `softeng325-labl1-sockets`:

```
softeng325-dev/softeng325-labl1-sockets/softeng325-lab1-sockets-server/target
```

From this directory, run the JAR (as a single command on one line):

```
java -cp
softeng325-lab1-sockets-server-0.0.1-SNAPSHOT-jar-with-dependencies.jar
nz.ac.auckland.concert.server.Server
```

To run the client (unit tests), simply run the `Client` from within Eclipse as a JUnit test.

(d) Study the application

Study the application source code to understand how the application works. In your journal, critique the application and address the following questions.

- What would be involved in developing a similar application whose server maintained a collection of some other type of information, e.g. movies instead of concerts?
- How efficient is the server? *Hint*: how does the server cope with multiple clients.
- What enhancements do you think should be made to the server?

Task 2:Build the supplied Whiteboard application

The shared whiteboard application, discussed in lectures, is available to download from Canvas. Project `softeng325-labl1-rmi-whiteboard` follows the same structure as the sockets project you've built in task 1.

Download the project, unzip it and store it in your development directory. Import the project into your Eclipse workspace. You can run the RMI server similarly to running the socket-based server. Likewise, the RMI client is a JUnit test class and can be run in same way as in task 1.

The shared whiteboard application is useful to study when reviewing the lecture material and when working through the next task.

Task 3:Redevelop the Concert application using Java RMI

(a) Develop the Java RMI application

Download the skeleton project structure for `softeng325-labl1-rmi-concert` from Canvas. Unzip the project and store it in your development directory. Import the project into your Eclipse workspace. The project structure mirrors that of the structure of the projects for tasks 1 and 2.

The `common` module defines Concert service's remote interface. Study the Javadoc comments for `ConcertService` and flesh out the remainder of the project as necessary to implement a JUnit client and a server.

(b) Reflect on the Java RMI application

Consider the impact of using Java RMI (instead of working directly with sockets) on your development. Record your thoughts in your journal.

You might have noticed that in the shared whiteboard application, `ShapeServant` and `ShapeFactoryServant` instances are threadsafe – since all methods are synchronized. Recall that where all methods of a class are synchronized, only a single thread can be executing any method on the object at any one time. In your journal, answer the following questions:

- Why have `Shape` and `ShapeFactory` been made threadsafe?
- Does your `Concert` service need to be threadsafe?
- Should a RMI server use multiple threads when handling remote invocations?

Assessment and submission

Run Maven's `clean` goal on your task 3 parent project to clear all generated code. Zip up the project, including the 3 modules, and upload it to the Assignment Drop Box (https://adb.auckland.ac.nz).

The submission deadline is 18:00 on Friday 4 August. Participating in this lab is worth 1% of your SOFTENG 325 mark.

Resources

The following resource on Maven might be of interest to you:

`http://books.sonatype.com/mvnex-book/reference/index.html`.