Assignment 2 - Concurrent programming Romeo and Juliet

Felipe Orihuela-Espina

Contents

1 About the assignment

- **Deadline**: Monday 6th-Mar-2023 at 16:00 (UK time).
- **Late submissions policy**: No late submissions allowed.
- **What to submit**: A single `.zip` file as explained in Section 4
- **Learning outcome**: Design and implement server-side application software.
- **Where has the skill been learned in the module**:
  - Week 3 slides for concepts and code on Threads' creation, execution and termination.
  - Week 4 slides for concepts and code on Client-Server Architecture.
  - Week 4 Lab exercise for code on TCP/IP architecture and sockets.

2 Problem statement

In this exercise you are going to program a a well-known ordinary differential equation (ODE) known as *Romeo and Juliet* over a TCP/IP client-server architecture. Don't worry, you do NOT need to know calculus at all or understand ODEs to solve the exercise; the method implementing the ODE will be provided to you, so you focus only on the TCP/IP client-server architecture.

The figurative scenario is a variant of the famous Romeo and Juliet play by William Shakespeare. Again, you do not have to have read the play in order to understand or solve the assignment. In the original tragedy, the two lovers face an impossible relation because of the ongoing rivalry between their families. Here, our scenario is just slightly different; whereby the playwriter, William Shakespeare, gets acquaintance with the two lovers and an exchange of love letters commence with the playwriter playing a kind of matchmaker between the two while writing the play about the love story. In this figurative scenario, the verses of the play correspond to the different values of the ODE over time.

In a more real scenario; you will be building a 1 client (the `Playwriter`) and 2 single-threaded servers (`Romeo` and `Juliet`) concurrent system. You are provided with templates for these three files with only a few gaps to be filled; the gaps control the client-server communication processes. The servers (`Romeo` and `Juliet`) are analgous in the service they provide although each one solve one of the equations of the *Romeo and Juliet* ODE. They are single-threaded. The client, i.e. the `Playwriter`, will be responsible to communicate with the servers to get the ODE values over time (i.e. iterations). For each verse of the play (i.e. iteration of the ODE), the `Playwriter` will request the service from *both* servers and annotate their answers in the novel. At the end of the iterations, the novel will be dumped into a `.csv` file. The method to do so is also provided to you, so you do not have to worry about it.

In *all* classes, you are requested to treat exceptions in the method that first can raise them i.e. do not rely in `throws` clauses to deal with exceptions at a later stage. If some exceptions requires you to stop the execution of either the servers, or the client without a correct resolution of the request, make sure that in those cases you exit the program with code 1. Exit with code 0 if program execution is correct.

3 Templates

You are provided with templates for all three classes in canvas. The exercise requires you to fill the gaps clearly identified with;

```
//TO BE COMPLETED
```

The templates already contain all the atributes, methods, and correct signatures. There is no need to create more attributes for the classes, but you can declare local variables within the methods as you may need. Do not alter the signature of the methods as the automarker will rely on these.

4 What to submit?

This is a sumative assignment.

1. Copy the output of the execution into a text file called:

   `RomeoAndJuliet_Execution.txt`

   This file should contain the output of all threads together (See provided exemplary execution for reference).
2. Compress into a single `.zip` file the following;
   - the three .java files corresponding to `Romeo`, `Juliet` and the `Playwriter` classes,
   - the novel (`RomeoAndJuliet.csv`) and
   - the execution (`RomeoAndJuliet_Execution.txt`),
3. Submit the `.zip` file into canvas.

☞ **IMPORTANT**: Do NOT use winrar for compressing but standard .zip. Do ONLY zipped the afore mentioned files. Do NOT create or organize the files into folders. If you develop your code in some IDE, e.g. IntelliJ, extract only the source files and do not submit the whole project. A penalty will be applied to your marks if any repackaging is necessary at our end.