

Assignment 2 - Concurrent programming

Romeo and Juliet

Felipe Orihuela-Espina

Contents

1	About the assignment	1
2	Problem statement	1
3	Templates	2
3.1	Methods description	3
4	What to submit?	6
5	Rubric	6

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 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 playwright, William Shakespeare, gets acquaintance with the two lovers and an exchange of love letters commence with the playwright 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 analogous 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 attributes, 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.

3.1 Methods description

Juliet.java

Remember that Juliet is a server. So its duty is to accept service requests, process those requests and then send back the request outcome to the client. The service is calculating the next value of the ODE sinusoidal part.

- **public Juliet(double initialLove):** Class constructor. **Create the server socket.** The IP address corresponds to the localhost (i.e. 127.0.0.1). The port can be any number, but you are suggested to use 7779 for Juliet. The `initialLove` value helps to initialize the attribute `currentLove`.
- **public Pair<InetAddress,Integer> getAcquaintance():** Receives lover's socket information and share's own socket. This method is called by the playwrighter. In this method, the server (Juliet) **returns the server IP address and its port number.**
- **public double receiveLoveLetter():** Retrieves the lover's love. This method **accepts the server's service requests.** Upon receiving the service request, it parses the message to remove the termination character from the incoming message. The remaining part of the incoming message contains the current ODE value from the counterpart server (Romeo) via the client (the playwrighter). Note that the message will arrive in the form of a string so a casting to a double will be required before processing the request.
- **public double renovateLove(double partnerLove):** **Provided. No need to do anything.** The ODE system. Given the lover's love at time t , estimate the next love value for Romeo at time $t + 1$. This method **provides the actual service.** Note how the value of the ODE is stored in attribute `currentLove`.
- **public void declareLove():** Communicate love back to playwrighter. This method prints Juliet's good night's message (*Good night, good night! Parting is such sweet sorrow,\n That I shall say good night till it be morrow.*) and **return the outcome of the request to the client;** that is, it sends back to the playwrighter a message with the ODE value (Juliet's `currentLove`). The message uses a termination character. Although the termination character can be any character that it is not part of the message, we suggest you use "J" for traceability.
- **public void run ():** **Provided. No need to do anything.** Execution.

Romeo.java

Remember that Romeo is a server. So its duty is to accept service requests, process those requests and then send back the request outcome to the client. The service is calculating the next value of the ODE cosinusoidal

part.

- **public Romeo(double initialLove):** Class constructor. **Create the server socket.** The IP address corresponds to the localhost (i.e. 127.0.0.1). The port can be any number, but you are suggested to use 7778 for Romero. The `initialLove` value helps to initialize the attribute `currentLove`.
- **public Pair<InetAddress,Integer> getAcquaintance():** Receives lover's socket information and share's own socket. This method is called by the playwrighter. In this method, the server (Romeo) **returns the server IP address and its port number.**
- **public double receiveLoveLetter():** Retrieves the lover's love. This method **accepts the server's service requests.** Upon receiving the service request, it parses the message to remove the termination character from the incoming message. The remaining part of the incoming message contains the current ODE value from the counterpart server (Romeo) via the client (the playwrighter). Note that the message will arrive in the form of a string so a casting to a double will be required before processing the request.
- **public double renovateLove(double partnerLove):** **Provided. No need to do anything.** The ODE system. Given the lover's love at time t , estimate the next love value for Romeo at time $t + 1$. This method **provides the actual service.** Note how the value of the ODE is stored in attribute `currentLove`.
- **public void declareLove():** Communicate love back to playwrighter. This method prints Romeo's good night's message (*I would I were thy bird*) and **return the outcome of the request to the client**; that is, it sends back to the playwrighter a message with the ODE value (Romeo's `currentLove`). The message uses a termination character. Although the termination character can be any character that it is not part of the message, we suggest you use "R" for traceability.
- **public void run ():** **Provided. No need to do anything.** Execution.

PlayWriter.java

Remember that PlayWriter is a client to both servers. So its duty is to request both services, and wait to be responded with the request outcome. Both services are requested a number of times (the `novelLength`) before the final output (the novel) is dumped to a .csv file.

- **public PlayWriter():** **Provided. No need to do anything.** Class constructor. Initializes all the attributes of the class.
- **public void createCharacters():** Create the lovers. **Create one object of type Romeo and one object of type Juliet** and saves them to attributes

myRomeo and myJuliet respectively. Then, it launches the execution of the corresponding threads.

- **public void charactersMakeAcquaintances():** This method **interrogate each server for their IP address and port number**.
- **public void requestVerseFromRomeo(int verse):** **Connects to the server Romeo and request service** from the Romeo server. In order to do so, it has to create a service request message containing the counterpart server service value i.e. Juliet's love, and add a terminating character. Although the termination character can be any character that it is not part of the message, we suggest you use "J" for traceability. Once the message is ready, it does send the message to the Romeo server to indicate the request of the service. The method's parameter **verse** helps to keep track of the iteration and hence where from the novel array should Juliet's current value be read.
- **public void requestVerseFromJuliet(int verse):** Analogous to the service request method for the Romeo server but for server Juliet. Although the termination character can be any character that it is not part of the message, we suggest you use "R" for traceability.
- **public void receiveLetterFromRomeo(int verse):** **Received the outcome of the service** from the server Romeo. Upon receiving this outcome, it parses the message to remove the termination character from the incoming message. The remaining part of the incoming message contains the new ODE value from the server (Romeo). Note that the message will arrive in the form of a string and with a termination character, so a removal of the termination character and a casting to a double will be required before saving the ODE's value into the novel array. The method's parameter **verse** helps to keep track of the iteration and hence where to in the novel array should Romeo's new value be written. Finally, it **closes the connection** with the server's socket.
- **public void receiveLetterFromJuliet(int verse):** Analogous to the service outcome reception method for the Romeo server but for server Juliet.
- **public void storyClimax():** This is the **main loop** where the services from both servers are requested and received.
- **public void charactersDeath():** **Terminates the execution of both servers**.
- **public void writeNovel():** **Provided. No need to do anything**. This is the playwrighter main method.
- **public void dumpNovel():** **Provided. No need to do anything**. This is the playwrighter ability to write the novel array to a file.

- **public static void main (String[] args):** **Provided. No need to do anything.** This is the overall main method. It creates an object of type `Playwriter` and call its main method `writeNovel`. At the end, it `dumpNovel`

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.

5 Rubric

- `Romeo.java`: 20%
- `Juliet.java`: 20%
- `Playwriter.java`: 40%
- `RomeoAndJuliet.csv`: 10%
- Output `RomeoAndJuliet_Execution.txt`: 10%