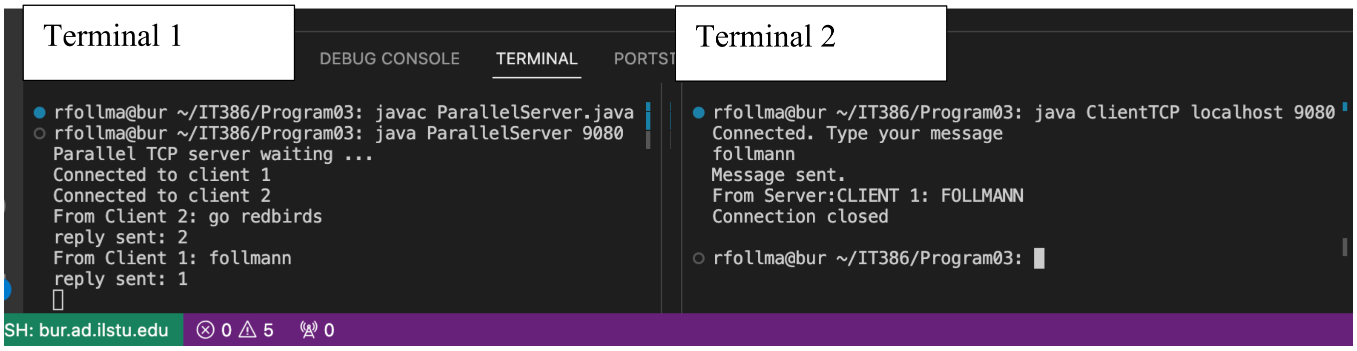
IT 386 - Program 03 - Parallel Programming with Java threads

Problem 1 (25 pts). Write a multithreaded TCP server in Java, and name it **ParallelServer.java**. Recap on socket programming by checking Week 5 materials, where you learned about socket programming. In our sequential version, only one client at the time could be processed, if another client would connect, it would need to wait for the first client to be fully processed.
You will write a parallel implementation of the server using Java Threads so the server can handle multiple clients at the same time. In your **ParallelServer.java** create a subclass **WorkerServer** that implements Runnable. The implementation of the run() method should receive the message from the client, process it by converting the characters to uppercase, and send the uppercase message along with the clientNumber back to the clientTCP. To keep track of clients, add a counter to count the number of clients. For example, if I am the 3rd client to connect to the server, then my clientNumber is 3. The port number should be inputted as argument input when running your code, make sure to comment and indent your code properly, and also add your name to your code.
Testing your ParallelServer implementation.

- Use the provided **ClientTCP.java** no need to change this code.
- Open three terminals
- Compile both the **ClientTCP.java** and **ParallelServer.java**
- <u>Terminal 1</u>: run the server with port number of your choice
- <u>Terminal 2</u>: run the client with localhost and port number that the server is listening to, **but do not send a message.**
- <u>Terminal 3</u>: run another client with localhost and same port number, and send the message "go redbirds"
- Go back to Terminal 2 and send a message with your last name. Your output should look like this:

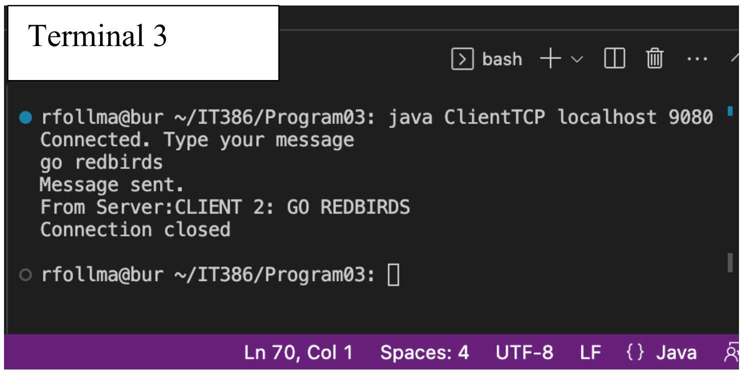

Terminal 1

```
rfollma@bur ~/IT386/Program03: javac ParallelServer.java
rfollma@bur ~/IT386/Program03: java ParallelServer 9080
Parallel TCP server waiting ...
Connected to client 1
Connected to client 2
From Client 2: go redbirds
reply sent: 2
From Client 1: follmann
reply sent: 1
```

Terminal 2

```
rfollma@bur ~/IT386/Program03: java ClientTCP localhost 9080
Connected. Type your message
follmann
Message sent.
From Server:CLIENT 1: FOLLMANN
Connection closed

rfollma@bur ~/IT386/Program03:
```


Terminal 3

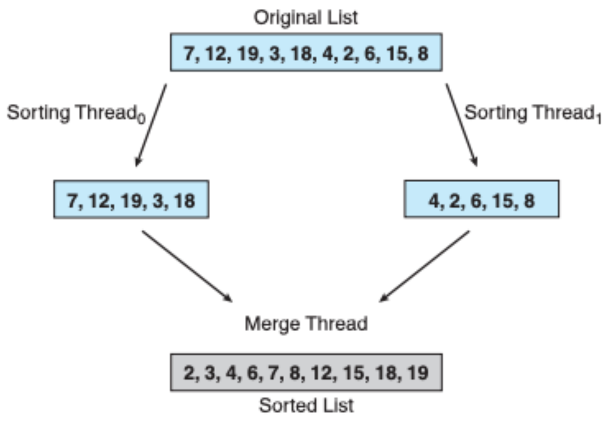
```
rfollma@bur ~/IT386/Program03: java ClientTCP localhost 9080
Connected. Type your message
go redbirds
Message sent.
From Server:CLIENT 2: GO REDBIRDS
Connection closed

rfollma@bur ~/IT386/Program03:
```

- Take a screenshot of your three terminals showing the successful interaction between the two clients and the parallel server, and place screenshot in a word document.

Problem 2 (40 pts). Write a multithreaded sorting program in Java, name it **ParallelSort.java.** Your code should work as follows:
A collection of items is divided into two lists/arrays of equal size. Each list is then passed to a separate thread (a sorting thread), which sorts the list using *insertion sort algorithm*. The two sorted lists are passed to a third thread (a merge thread), which merges the two separate lists into a single sorted list. Once the two lists have been merged, the complete sorted list is the output. If we were sorting integer values, this program should be structured as illustrated in the diagram below.

A. The length of the array should be entered as a command argument input.
B. Write a helper method to generate an array of random of any length to be sorted. The random numbers should be selected from random distribution of numbers smaller than 1,000,000.
C. Write a helper method to check if your final array is sorted correctly, and any other methods you need.
D. Test your code with a small, unsorted list (smaller than 20), and have the unsorted and sorted lists printed to screen, otherwise if the list you are sorting is larger than 20, do not print to screen.
   - Take a screenshot of you successfully testing your code, and place it in the word document
E. Measure the elapsed time of your multithreaded Sort application and compare it with the sequential Insertion Sort. Run three trials using an unsorted array of length 100,000, take a screenshot, report your elapsed time for the sequential and for the parallel.
F. Discuss the comparison of the timing results above.



Submission Guidelines:

Notes: Make sure to comment your codes, use proper indentation, add your name, course as a comment in your codes.
Attach your **ParallelSort.java** and **ParallelServer.java** codes.
In a word document place the screenshots for problem 1, and the screenshot for problem 2 along with the answer to items D, E and F, and attach to this submission.