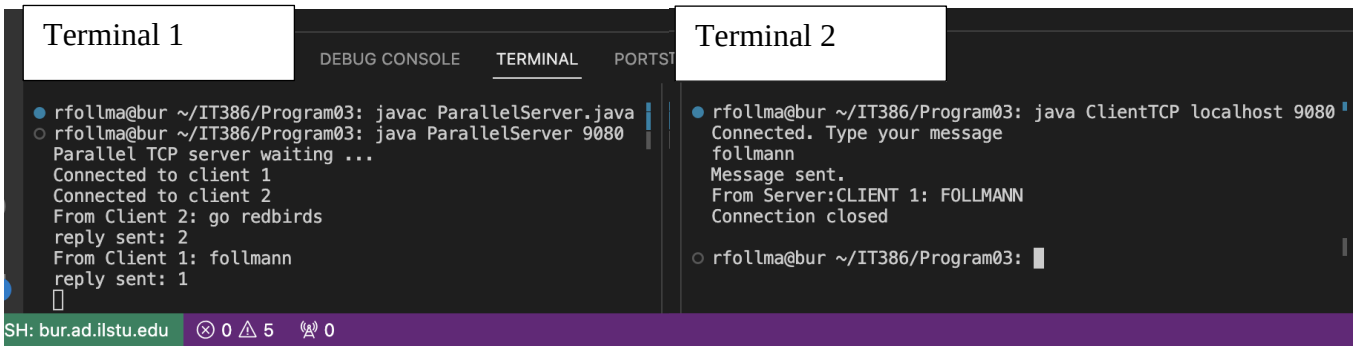
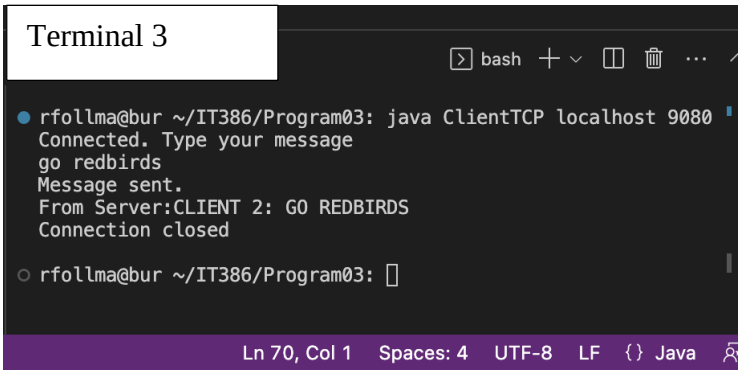
IT 386 - Program 03 - Parallel Programming with Java threads

Problem 1 (30 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 sockets 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 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. 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**
- Terminal 1: run the server with port number of your choice
- Terminal 2: run the client with localhost and port number that server is listening to, **but do not send a message.**
- Terminal 3: 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:

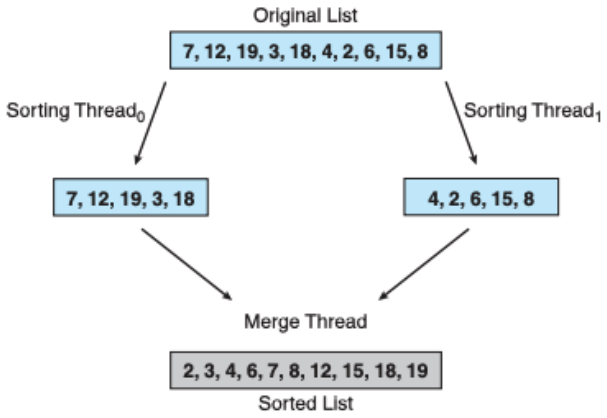




- Take a screenshot of your three terminals showing the successful interaction between the two clients and the parallel server, and place screenshot in a word document.

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 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. To ensure that the two sorting threads have completed execution, the main thread will need to use the join() method on the two sorting threads before passing the two sorted lists to the merge thread. Similarly, the main thread will need to use join() on the merge thread before it outputs the complete sorted list (if it is a small list).

- The length of the array should be entered as a command argument input
- Write helper method to generate the random array of integers of any length to be sorted
- Write helper method to check if your final array is sorted correctly, and any other method you need.
- Test your code with a small, unsorted list (smaller than 20), and have the unsorted and sorted lists printed to screen, otherwise if it is a large list you are sorting, do not print to screen.
    a. Take a screenshot of you successfully testing your code, and place it in the word document
- Measure the elapsed time of your multithreaded Sort application and compare with the sequential equivalent. Run three trials using an unsorted array of length 100,000 and report your numbers and take screenshot, place it in the word document
- 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 along with the screenshots and answers to last item to this submission on Reggienet.