# ENSF 607 – Advanced Software Design and Architecture

Fall 2021

SCHULICH
School of Engineering
UNIVERSITY OF CALGARY

## Lab Assignment # 4:

| Due Dates | |
|---|---|
| **Post-Lab** | Submit electronically on Git **before 11:59 PM on <u>Monday November 1st</u>** |

Students have an option of working in groups for this assignment. Maximum of 2 students per group.

## The objectives of this lab are to understand the concepts of:

1. Multithreading and Java Threads
2. Client-server communication using Java Sockets
3. Experience with client-server architecture

**<u>The following rules apply to this lab and all other lab assignments in future:</u>**

1. Before submitting your lab reports, take a moment to make sure that you are handing in all the material that is required. If you forget to hand something in, that is your fault; you can't use `I forgot' as an excuse to hand in parts of the assignment late.
2. **<u>20% marks</u>** will be deducted from the assignments handed in up to **<u>24 hours</u>** after each due date. It means if your mark is X out of Y, you will only gain 0.8 times X. There will be no credit for assignments turned in later than 24 hours after the due dates; they will be returned unmarked.

## Exercise - 1: Communicating between threads (10 Marks)

Communication between threads can be accomplished by passing values in a shared variable or a shared data area. In C or C++, several threads could share a global variable. However, Java does not have global variables. In Java, each thread's local variables and methods are private to it because each thread has its own private activation stack. However, threads are not given private copies of static class variables or object instance variables. Therefore, you might ask what's a good way to share a variable in Java?

**Task 1** – Download the `SimpleThread.java` and the `Resource.java` classes from D2L. This program has two threads that are supposed to increment the counter of the resource and print the new value. Carefully read the code, and Run the `SimpleThread` program, and consider the output. why doesn't the value of the `counter` increase in some steps?

Whenever two or more threads (or processes) share a variable (or data area), any computation on the shared variable by one thread must be protected from interference by the other threads. Only by careful analysis can we identify these critical regions and then use a synchronization primitive to protect them. In Java, this synchronization primitive is a Hoare-style monitor associated with the object. The monitor guarantees that at most one thread can execute within a critical region for that object at any time.

**Task 2** – Fix this program to avoid the above issue. If you run the program after fixing it, you should see each number only once in the console.

**Task 3** – Change the program to use the runnable interface in the implementation of your multi-thread program

**What to Hand in:** Please commit **all the java files** including the files you have modified

## Exercise - 2: Practice with Threads (20 Marks)

### Task 1 - Randomized Number (10 Marks)

**What to Do**: Write a program that runs 5 threads. Each thread is responsible to generate a random number in the range of (1, 100). The program waits for all the threads to finish, and then calculates the sum of the numbers (which were randomized), and prints their sum. Implement a Runnable class that randomizes a number and store it in a member field. Your main program can go over all the objects and check the stored values in each object.

### Task 2 - Modified Randomized Number I (5 Marks)

**What to Do**: Modify the program from Task1 of this exercise, so that instead of each object keeping its own score, you will use one collection to store all the results in.

### Task 3 - Modified Randomized Number II (5 Marks)

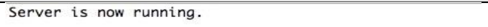
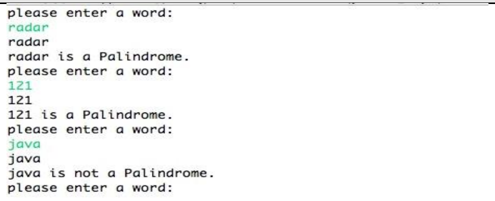
**What to Do**: Modify the program from Task 2 of this exercise, so that the program uses a fixed thread pool instead of individual threads. The thread pool should have 5 threads and be instantiated using an `ExecutorService` object.

**What to Hand in:** Commit your programs for Tasks 1, 2, and 3.

## Exercise 3: A Very Simple Client-Server Program (8 marks)

In this exercise, you will complete the implementation of a very simple client-server program. Where client prompts the user to enter a word (in fact a string of characters), then passes the string to the server. The server is supposed to examine the string and send a message to the server indicating whether the user input is a palindrome or not. Your job in this exercise is to write only the definition of the class `Server`, the definition of class `Client` is given.

**What to Do:** Download the definition of the class `Client` from D2L. The client connects to the server running on port 8099 of the "localhost". The main job of the server that you need to implement is to read a message from socket, examine it for palindrome and send an appropriate message to back to the client. For your information: A string is called palindrome if it spells the same word from both ends. As an example, the word "radar" is palindrome. You can assume that user inputs are always lower case and contain only alphanumeric characters, and no punctuation. You don't need to do any error checking for these requirements.
To test your program, you should first start running the server. When server started running, it is always a good idea to show a message on the terminal or console screen that:
Server is running ...
Do not make any change to the Client class. Here are the screenshots of the outputs of the client and server programs.

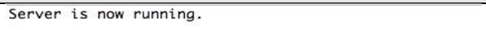
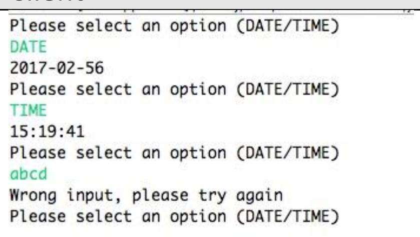
| Server | Client |
|---|---|
|  `Server is now running.` |  `please enter a word:`<br>`radar`<br>`radar`<br>`radar is a Palindrome.`<br>`please enter a word:`<br>`121`<br>`121`<br>`121 is a Palindrome.`<br>`please enter a word:`<br>`java`<br>`java`<br>`java is not a Palindrome.`<br>`please enter a word:` |

**What to Submit:** Please commit all the java files including the files you have created/modified.

## Exercise 4 Date-time server (8 Marks)

In this exercise, you need to implement a simple client that connects to a date server. The date server class is given to you. Please download `DateServer.java` from D2L. The server runs on port 9090 of your local machine and waits for a client and when a client connects to the server, client can select to receive `DATE` or `TIME`. Depending on the client's selection, server sends the response to the client. A sample output of the program is given.

Your task is to create a client class that connects to the date server. Please do not make change to the date server class.

| Server | Client |
|---|---|
|  `Server is now running.` |  `Please select an option (DATE/TIME)`<br>`DATE`<br>`2017-02-56`<br>`Please select an option (DATE/TIME)`<br>`TIME`<br>`15:19:41`<br>`Please select an option (DATE/TIME)`<br>`abcd`<br>`Wrong input, please try again`<br>`Please select an option (DATE/TIME)` |

**What to Hand in:** Please commit all the java files including the files you have created/modified.

## Exercise 5: Tic-Tac-Toe with a Thread pool (22 Marks)

In this exercise, you should use the first version of your Tic-Tac-Toe Game and you should modify it in a way that it can support multiple games simultaneously. Each game will have its own thread and the communication between players is managed via sockets.

One obvious step that you need to do in this exercise is to define a class that represents the client-side of the game (a client user interface). This class is not only responsible to take the player's inputs and pass them to the server-side of the game, but also is responsible to receive the opponent's inputs from server-side and display them on the board.

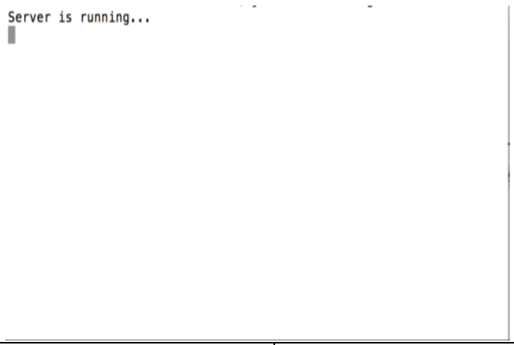
The client side must run in two different terminal-windows and a game should not start before two players are available (must be monitored by the server). The server-side must use a threadpool to start a new game for every two clients that connect.

For the server-side one possible option is to create a new class for the server and make your existing class `Game` implement `Runnable`. This means the server class should include data fields such as `ServerSocket and ExecutorService`. It must create and run a new game when two clients (players) ask for a connection and those connections are accepted.

**What to Submit:**
Please commit all the java files including the files you have created/modified.

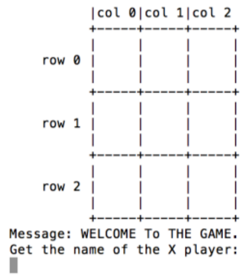
The following page shows a few screenshots of the program.

Screen Shot from Server Treminal



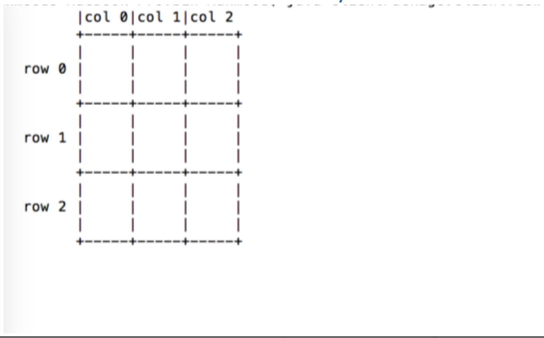
```
Server is running...
```

Screen Shots from X-Player Terminal

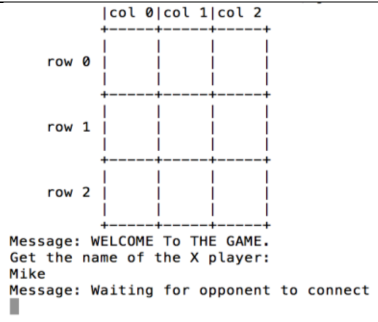


```
          |col 0|col 1|col 2
          +-----+-----+-----+
          |     |     |     |
   row 0  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 1  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 2  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
Message: WELCOME To THE GAME.
Get the name of the X player:
```

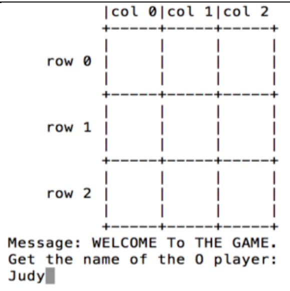
Screen Shot from O-Player Terminal



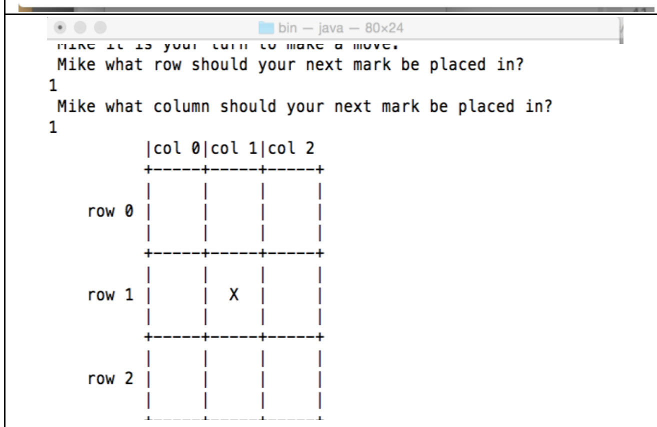
```
          |col 0|col 1|col 2
          +-----+-----+-----+
          |     |     |     |
   row 0  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 1  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 2  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
```



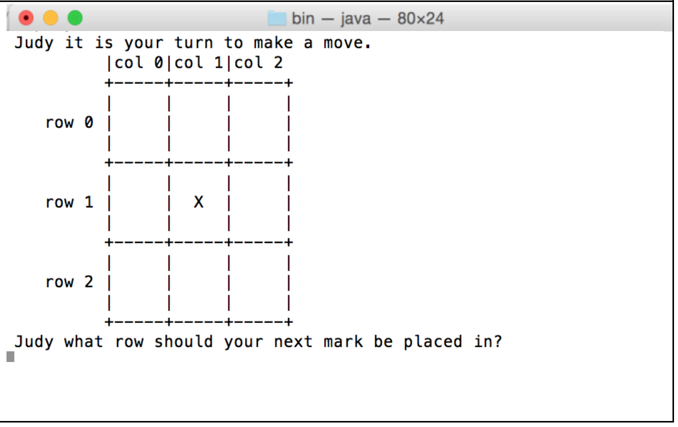
```
          |col 0|col 1|col 2
          +-----+-----+-----+
          |     |     |     |
   row 0  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 1  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 2  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
Message: WELCOME To THE GAME.
Get the name of the X player:
Mike
Message: Waiting for opponent to connect
```



```
          |col 0|col 1|col 2
          +-----+-----+-----+
          |     |     |     |
   row 0  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 1  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 2  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
Message: WELCOME To THE GAME.
Get the name of the O player:
Judy
```



```
 Mike it is your turn to make a move.
 Mike what row should your next mark be placed in?
1
 Mike what column should your next mark be placed in?
1
          |col 0|col 1|col 2
          +-----+-----+-----+
          |     |     |     |
   row 0  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 1  |     |  X  |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 2  |     |     |     |
          |     |     |     |
```



```
Judy it is your turn to make a move.
          |col 0|col 1|col 2
          +-----+-----+-----+
          |     |     |     |
   row 0  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 1  |     |  X  |     |
          |     |     |     |
          +-----+-----+-----+
          |     |     |     |
   row 2  |     |     |     |
          |     |     |     |
          +-----+-----+-----+
Judy what row should your next mark be placed in?
```