CS 342 **Project#3** **Spring 2022**

The Ancient Game of Morra

Due Dates:

Part #1: UML class diagrams, Sunday, April 3 2022, @ 11:59pm
Part #2: server and client programs, Sunday, April 10 2022, @ 11:59pm

Description:

Morra is a hand game that dates back thousands of years. The rules are simple. If a two player game, each player (at the same time) must reveal their hand holding out zero to five fingers; at the same time, they must call out their guess about how many fingers total will be revealed by both players. If a player guesses correctly, they win a point. If both players guess correctly, no points are awarded. If no one guesses correctly, no points are awarded. The total number of points to win is determined before the game starts.

Your implementation will be a two player game where each player is a separate client and the game is run by a server. Your server and clients will use the same machine; with the server choosing a port on the local host and clients knowing the local host and port number (just as was demonstrated in class). Games will be played until one of the players has two points. At the end of each game, each user will be able to play again or quit.

All networking must be done utilizing Java Sockets (as shown in class). **The server must run on its own thread and handle each client on a separate thread. Each client must connect and communicate with the server on a separate thread**.
You may work in teams of two but do not have to.

Part #1: UML Class Diagrams:

You are required to create two separate UML Class Diagrams; one for the server program and one for the client program. Each UML diagram should including all classes, data members and methods of those classes, interfaces and interactions between them. Format these as PDFs. There is NOT a late day for part one.

CS 342 Project#3 Spring 2022

Implementation Details:

You will create *two separate programs*, each with a GUI created in JavaFX, one for the server and one for the client. Use the Maven templates provided. This project will be developed as a Maven project using the JavaFX_MavenTemplate posted under "Sample Code" on our course BB site.

You will need to change the <artifactId> in the pom file for each program: for instance, the server program would have:
<artifactId>serverProgramProjectThree</artifactId>

And the client program would have:
<artifactId>clientProgramProjectThree</artifactId>

You may use FXML, .CSS files with controller classes if you would like.

For the server GUI:

- A way to chose the port number to listen to
- Have a button to turn on the server.
- Display the state of the game(you can display more info, this is the minimum):
    - how many clients are connected to the server.
    - what each player played.
    - how many points each player has.
    - if someone won the game.
    - are they playing again.
- Any other GUI elements you feel are necessary for your implementation.

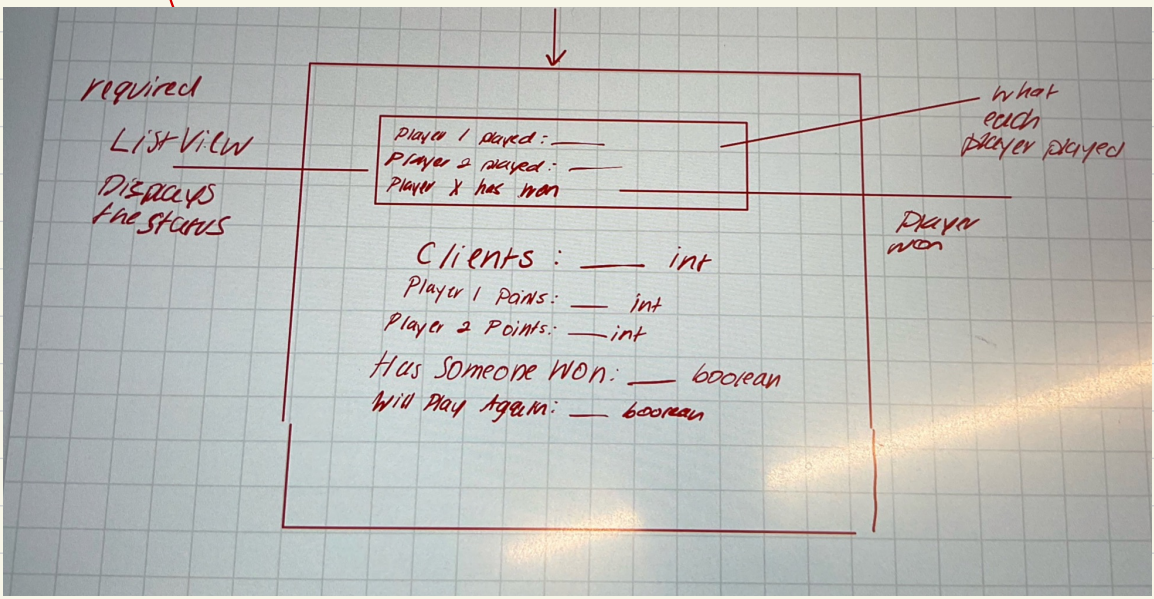
Notes: Your server GUI must have a minimum of two scenes: an intro screen that allows the user to input the port number and start the server and another that will display the state of the game information. To display the game information, you must incorporate a listView (as seen in class) with any other widgets used. Keep in mind, you can dynamically add items to the listView without using an ArrayList.

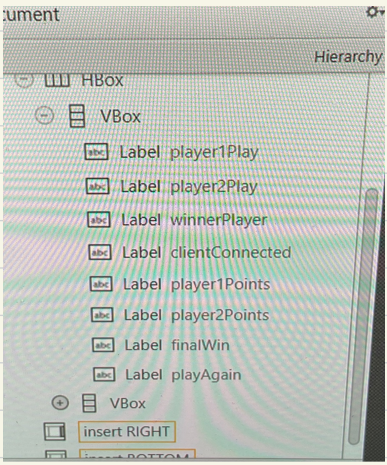
For the server logic:

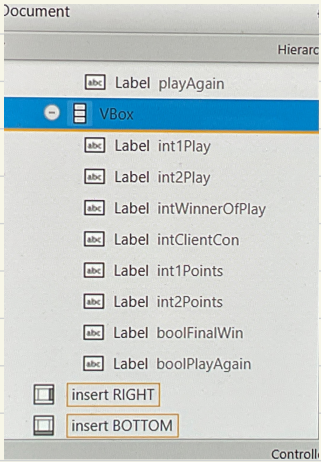
- It will only allow a game to start if there are two clients connected.
- It will notify a client if they are the only one connected.
- It will keep track of what each player played.
- It will evaluate who won each each round.
- It will evaluate if a client has won the game.
- It will update each client with the above items in time.
- It will do all things necessary to run the game.

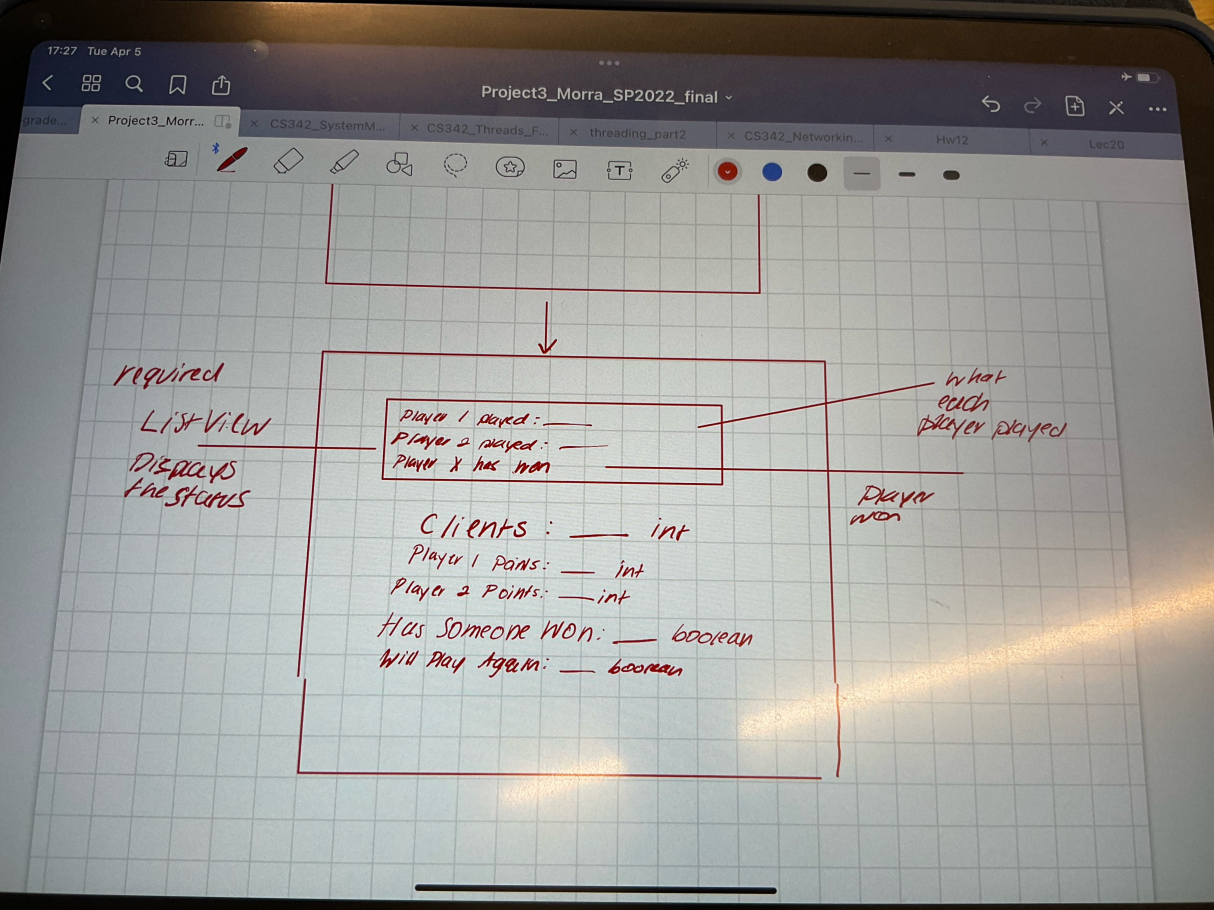
For the server GUI:

- A way to chose the port number to listen to
- Have a button to turn on the server.
- Display the state of the game(you can display more info, this is the minimum):
  - how many clients are connected to the server.
  - what each player played.
  - how many points each player has.
  - if someone won the game.
  - are they playing again.
- Any other GUI elements you feel are necessary for your implementation.








17:27 Tue Apr 5
Project3_Morra_SP2022_final
Project3_Morr...
CS342_SystemM...
CS342_Threads_F...
threading_part2
CS342_Networkin...
Hw12
Lec20
required
ListView
Displays the status
Player 1 played: ___
Player 2 played: ___
Player X has won
what each player played
Player won
Clients: ___ int
Player 1 Pairs: ___ int
Player 2 Points: ___ int
Has Someone Won: ___ boolean
Will Play Again: ___ boolean

① run program
↳ server

Client button · (HBox(400, server choose

① scenemap store diff scene

* list item is
center of border point

① client : connect to server
① create client no server crash
② client connect server & loopprint

— server program
— client program

- client : swap scene and title
↳ new client
↳ class :
↳ call start() = means extend threads
extend thread

Threads : class extend threads

- Client extend thread (UI) → run on it's own thread

↳ run on it's thread (blocking call) → on the same thread user interface won't work

① talk to server

↳ input / output

↳ 2 client.start

↳ connect to server

while true

- program sent to server too

- String go to server also


application
thread

It is expected that your server code will open, manage and close all resources needed and handle all exceptions in a graceful way. For game play, each client will send the server the number they are playing (0-5) and their total guess. The server will determine who if anybody receives points and then update each client with what the other played, guessed and the resulting state of the game.
If a client has won the hand and has reached two points, the server will send what the other player played and guessed, the resulting state of the game and require each client to make a choice as to play again or quit. If a player quits, the server will end that connection. If one player quits and the other wants to play again, the server will notify the client that they must wait for another person to connect. If both want to play again, the server will start a new game.

For the client GUI:

- A way for the user to enter the port number and ip address to connect to
- A button to connect to the server.
- A way to display the points each player has.
- A way to display what the opponent played each round using images.
- A way to display what the opponent guessed each round.
- Clickable images to choose what to play.
- A way to make a guess on total played.
- A way to display messages from the server.
- Buttons to choose to play again or quit.
- Any other GUI elements you feel are necessary for your implementation.

Notes: Your client GUI must have a minimum of three scenes: an intro screen that allows the user to input the port number and ip address to connect to the server, another that will display the state of the game information and game play and a third of your choice. To display the game information, you must incorporate a listView (as seen in class) with any other widgets used. Keep in mind, you can dynamically add items to the listView without using an ArrayList. Buttons should be disabled when not in use or when waiting for a response from the server.

For the client logic:

After entering the port number and ip address, the user will click to connect to the server. When the server notifies that there is another client to play, the game will start. The user will select what number to play and their total number guess and send to the server. The server will respond with what the opponent played and guessed, who won and what points were distributed. The client GUI will update with this information and allow the user to either keep playing or, if someone has won, chose to either play again or quit. Quit will end the client program. It is expected that your client code will open, manage and close all resources needed and handle all exceptions in a graceful way.

CS 342 Project#3 Spring 2022

Passing info between clients and server:

You must implement the MorraInfo class. class MorraInfo implements Serializable{}

You will add serializable data members to this class that keep track of the state of the game(i.e. int p1Points, int p2Points, String p1Plays, String p2Plays , Boolean have2players…..). This class will be used to send information back and forth between the server and two clients. This is the only way you are allowed to send and receive information.

Testing Code:

You are required to include JUnit 5 test cases for your program. Add these to the src/test/java directory of your Maven Project.

Use of templates found on the web:

You may use ideas from templates(fxml, css or other) for styling your programs found on the web. You may not import those templates and pass them off as your own work. If you use an idea or part of such template, include a reference to that work in the header of the file where used. Failure to do this will result in academic misconduct charges.

Electronic Submission:

Part #1:

submit your UML class diagrams as a single PDF to the link provided. Name it netid + Project3UML. If you are working with a partner, only one of you needs to submit this. There is NOT a late day for this; it must be submitted on time.

Part #2:

If you worked with a partner, only one of you needs to submit a project. You must submit a PDF file called Collaboration.pdf to the assignment link when you submit part two. In that document, put both of your names and netIds as well as a description of who worked on what in the project. Zip the Maven projects MorraClient_Project3_Sp2022, MorraServer_Project3_Sp2022, and name it with your netid + Project3: for example, I would have a submission called mhalle5Project3.zip, and submit it to the link on Blackboard course website.

CS 342 Project#3 Spring 2022

Assignment Details:

You may submit your code, part two, up to 24 hours late for a 10% penalty. Anything later than 24 hours will not be graded and result in a zero.

We will test all projects on the command line using Maven 3.6.3. You may develop in any IDE you chose but make sure your project can be run on the command line using Maven commands. Any project that does not run will result in a zero. If you are unsure about using Maven, come see your TA or Professor.

Unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml.

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing

your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or class participation. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml.