CS 5123: Cloud and Distributed Systems
Fall 2017

Homework 1 (15% of your final grade)
Due: 11:59pm Wednesday September 27

Group: You may work with up to two other students. Maximum group size is three (3). Minimum group size is one (1).

Goal: Write a complete client/server program to play a game of cards. You will have to use network/socket programming and thread management.

Guidelines:

1) Start **early**!
2) Ask for help: e-mail instructor, e-mail class list, post a question on discussion forum on D2L, come to office hours. Don't wait till the night before to ask for help, as you might not get a response.
3) Submit your source code, a compiled code (for example jar file if using Java or executable if using C/C++), a README.txt file, and a short manual/tutorial in a zipped file through D2L DropBox.
4) If you are using an IDE, make sure your code compiles and runs on CSx. Highly recommended to use an IDE such as Eclipse, IntelliJ, NetBeans and a version control system such as git or svn.
5) See syllabus for penalty on late submission.
6) If you need help finding a team, contact the instructor as soon as possible.
7) This is a programming-heavy assignment. Use good programming and debugging skills. Sketch out how the solution should look like before coding.
8) You may use any other programming language of your choice.
9) Feel free to use Google but all code should be **your own**.

Deliverables (in a zipped file):

1) Your source code.
2) A README.txt file describing how to compile and run your code.
3) A brief manual/tutorial showing a user how to use your program.
4) A jar file if using java or executable if using C/C++.

Description:

You will write a program for four players to play a card game over the Internet. It is a very simple card game with a deck of 52 cards. There are four suits (spade, heart, diamond, club), each with 13 cards of 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A (in ascending order). The four players will form two teams of two players. The goal is to reach 250 points first. At the beginning of every round of the game, each player is dealt 13 cards. Each player will then bid. The bid indicates how many tricks the player thinks he/she can take. If you are used to Bridge or Spades or other trick-taking game, this is a simple version without trumps. Each player has to follow suit. For example, if *Player1* plays 2 Spade, *Player2* has to play a Spade. If *Player2* has no Spade, *Player2* can play any card but he/she won't get the trick. The

highest card takes the trick. For example, if 2 of Spade, 5 of Spade, J of Spade, and K of Heart are played, since Spade started, the player who played the J of Spade won the trick. Note that this is a team game, so if *Player2* bid 3 tricks and *Player4* bid 2 tricks, both *Player2* and *Player4* have to win 5 tricks in total (it doesn't matter who wins which trick). The player who won the trick starts next.

At the end of the round, that is, all 13 cards are played, the score for each team is updated. If a team bid X tricks, and it won Y >= X tricks, its score is (X*10 + Y-X). If the team won fewer than X tricks, then its score is (-X*10). Note the negative score. For example, if *Player2* and *Player4* bid 5 tricks total and they won 7 tricks, then their score for this round is 52 points. However, if they only won 3 tricks, then their score for this round is -50 points. Their total score (from previous rounds) is added up. Whenever a team reaches at least 250 points, the game is over. If both teams' scores exceed 250 points, the team with the higher score wins the game. At the end of each round, the next round starts: 13 cards are dealt, the players bid, and play their cards.

Play is sequential. For the first round, *Player1* starts the bid and plays the first card. For the next round, *Player2* starts the bid and plays the first card. And so on. Within a round, the player who wins the trick, plays the next card.

Your program will have two components: a server side and a client side.

The server code will manage all the clients, ensure that all clients are following the protocol, no client is cheating, keep track of the score, and deal the cards. Pick your own port number for the server.

You will have four running instances of the client code. The client will connect to the server and interact with the user and the server. Assume the first client to connect is called *Player1*, the second client is called *Player2*, the third client to connect is *Player3*, and the last client is *Player4*. You may use either a “settings” file to store the server name (or IP address) or have the client code ask the user for the server name or IP address. You can also assume that *Player1* and *Player3* are on the same team and *Player2* and *Player4* are on the same team. You can also assume that *Player1* starts playing first on the first round. For the second round, *Player2* bids and plays first. For the fifth round (if any), *Player1* bids and plays first.

NOTE: You can use either synchronous communications or asynchronous communications.

At client startup, ask the user for server name or IP address. Then ask the user for his/her name. Send the name to the server. Wait for reply from server. Once all users enter their name, display all the names and teams. Connect to server to obtain 13 cards. The server will create a deck of 52 cards and “deal” 13 cards to each client. Each card is unique; two clients cannot have the same card and one client cannot have two of the same card. Once all cards are dealt, the client will ask the user for a bid. The bid is sent to the server and shown to all the clients. Once all the clients have bid, then the total bid for each team is shown. Then play starts. You have to show each client's played card and who won the trick. Make sure to tally the number of tricks won correctly. When a client plays a card, that card is sent to the server. The server then notifies the next client that it is its turn to play. The server needs to keep track of who won the trick to prevent possible cheating. At the end of each round, update the score and show the total number of tricks and bids for each team. Start the next round (if any).

To represent the cards in text format, use S for Spade, H for Heart, D for Diamond, and C for Club.

When displaying all 13 cards, group by suit so that all cards in one suit are grouped together. Sort the cards in ascending order. It doesn't matter which suit is displayed first, but be consistent. For example, a client's cards can be displayed as 2S 5S JS; 3H 4H 9H 10H QH AH; ; 5C QC KC AC

Your program will require user interactions and network interactions. Make sure to manage all the threads you might have. Also, make sure to check for errors. Your program should not crash if the user enters an invalid card.

Sketch out your program, use good programming principles, use an IDE, use a code-sharing tool.

If your clients are unable to connect to the server, check that the ports are the same and that the IP address (or host name) is correct. Also check your firewall settings. Your code might not work on CSx due to firewall.

Grading:

- Code: 90%
  - Does it compile?
  - Does it run?
  - Does it run correctly?
  - Is it easy to play?
  - Did you include error checking?
  - Does your program crash at unexpected input/output?
  - Did you use good programming paradigms?
  - Is the program easy to use and understand?
  - Does your card game follow the rules?
- Manual: 10%
  - Is it easy to read and understand?
  - Can a user use and deploy the program easily?
- Misc: negative points if missing
  - Jar file or executable is easy to use.
  - No Manual and/or no README.txt file.
- Extra Credit:
  - Do you have a nice GUI?

Need help? E-mail instructor, e-mail class list, post a question on discussion forum on D2L, come to office hours, and/or use Google.