CS 342 **Project#3** **Fall 2020**

Baccarat JavaFX GUI

Due Dates:

Part one: UML class diagrams for server and client programs and wireframe for the client program:
due: Sunday October 25th @11:59pm

Part two: code for server and client programs:
due: Sunday November 1st @11:59pm

Description:

In this project you will implement a networked version of the popular casino game Baccarat. This is a somewhat simple game to understand and play which should allow you to focus on event driven programing and networking with Java Sockets.

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).

Each client is playing a separate game with the server. So, each game has its own deck and own history.

All networking must be done utilizing Java Sockets (as shown in class). **The server must run on its own thread (not the JavaFX application thread) and handle each client on a separate thread. Each client should have their connection to the server on a separate thread other than the JavaFX application thread.**

This project will be developed as two Maven projects using the same Maven template you downloaded from BB for project #2. The artifact ID in the POM file for the server Maven project should be “projectThreeServer” and the artifact ID in the POM file for the client Maven project should be “projectThreeClient”

*****You may work in teams of two but do not have to***.**

How the game is played:

The user will first bid some amount of money on either The Banker to win or The Player to win or it will be a Draw. These are the only three outcomes of the game.

Next, the dealer will deal two cards each to The Banker and The Player. The Player always gets to go first.

For the cards: 10's and face cards are worth zero points. Ace's are worth one point and all other cards are worth their face value.

If either The Banker's hand or The Player's hand add up to 8 or 9 points, it is a "natural" win and the game is over and bets are paid. If both the players hand and bankers hand add up to 8 or 9 points, and the user bet on The Player or The Banker, the user loses and the game is over.

Otherwise, The Player will go first: if hand totals to 5 or less, The Player gets one more card. If the hand totals to 6 or 7 points, no more cards are given.

counting the points: if the total value of the two cards is greater than 9, remove the first number of the total. For example, if I had a 9 and a 6: 9 + 6 = 15 so I would have 5 after dropping the 1.

Next it's The Banker's turn: if the bankers first two cards total 7 or more, no more cards are dealt. If The Banker's cards total 2 or less, The Banker gets one additional card. If The Bankers first two cards total 3, 4, 5, or 6, it depends on if The Player drew another card and if so, the value of that card to determine if The Banker receives another card. ***See the included PDF for a chart.***

The winning hand is the one with a total of 9 or as close to 9 as possible.
See the included PDF for betting payouts.

Implementation Details:

You will create two separate programs, each with a GUI created in JavaFX, one for the server and one for the client. In this project, you are free to use scene builder, FXML and .CSS files. I would recommend that most of you stick with doing things programmatically (like project #2)

<u>For the server program GUI:</u>

- A way to chose the port number to listen to
- Have a button to turn on the server.
- Have a button to turn off 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.
    - The results of each game played by either client.
    - how much the client bet on each game
    - how much the client won or lost on each game
    - if a client drops off the server.
    - if a new client joins the server.
    - is the client playing another hand.
- 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.

<u>For the server program logic:</u>

It is expected that your server code will open, manage and close all resources needed and handle all exceptions in a graceful way.

You will implement the following in your server program:

public class BaccaratGame

You need to add the following members:

ArrayList<Card> playerHand
ArrayList<Card> bankerHand
BaccaratDealer theDealer
double currentBet
double totalWinnings

and the method:

public double evaluateWinnings()

This method will determine if the user won or lost their bet and return the amount won or lost based on the value in currentBet.

public class BaccaratGameLogic
You need to implement this class with the following methods:

public static String whoWon(ArrayList<Card> hand1, ArrayList<Card> hand2)
public static int handTotal(ArrayList<Card> hand)
public static boolean evaluateBankerDraw(ArrayList<Card> hand, Card playerCard)
public static boolean evaluatePlayerDraw(ArrayList<Card> hand)

The method whoWon will evaluate two hands at the end of the game and return a string depending on the winner: “Player”, “Banker”, “Draw”. The method handTotal will take a hand and return how many points that hand is worth. The methods evaluateBankerDraw and evaluatePlayerDraw will return true if either one should be dealt a third card, otherwise return false.

public class BaccaratDealer
You need to implement this class with the following members:
ArrayList<Card> deck;

and the methods:
public void generateDeck()
public ArrayList<Card> dealHand();
public Card drawOne()
public void shuffleDeck();
public int deckSize();

generateDeck will generate a new standard 52 card deck where each card is an instance of the Card class in the ArrayList<Card> deck. dealHand will deal two cards and return them in an ArrayList<Card>. drawOne will deal a single card and return it. shuffleDeck will create a new deck of 52 cards and “shuffle”; randomize the cards in that ArrayList<Card>. deckSize will just return how many cards are in this.deck at any given time.

public class Card
You need to implement this class with the following members:

String suite
int value

and a two argument constructor:

Card(String theSuite, int theValue);

Note: You must implement the above just as they are described in the project write up. We will use these data members and methods to test your projects. Failure to do so will result in significant loss of points.

When a client connects to the server, the server will wait to receive a message from the client to play. That message should include the amount they bet and what they bet on: Banker, Player or Draw. The client thread on the server should create a new BaccaratGame instance. It should play through one hand and then send the client all of the information for that game:

- Initial Banker and Player hand
- If either the Banker or Player got an extra card and what it was
- The result of the game based on the clients bet (did they win or lose and how much)

The server should then wait to see if the client wants to play another game. The server will know the client wants to play again because it will receive another play message that includes the amount bet and what they bet on.

For the client program GUI:

- A way for the user to enter the port number and ip address to connect to
- A button to connect to the server
- There should be an area to display both the Players and Bankers cards with each clearly labeled. You may use images or text to display the cards.
- There should be a button to start each round of play.
- There should be an area where the user can enter the amount to bid and decide if they are bidding on The Player, The Banker or a Draw.
- There should be an area displaying the current winnings for the user
- There should be an area that displays the results of each round of play
- There should be a button that allows the user to quit the program

Some examples of result displays after each round of play:

Player Total: 7 Banker Total 5
Player wins
Sorry, you bet Draw! You lost your bet!

Player Total: 3 Banker Total: 8
Banker wins
Congrats, you bet Banker! You win!

Notes: Your client GUI must have a minimum of two scenes: an intro screen that allows the user to input the ip address and port number and connect to the server and another that will display the game and game information and allow the user to play again or quit.

<u>Client program logic:</u>

- When your program starts, the user should be able to enter the ip address and port of the server and connect to it.
- Once connected, the user will need to bid a dollar amount and decide what they are betting on (Player, Banker or Draw). Once these choices are made, your client program will send that information to the server and wait for the results of the game.
- Once it receives the results of the game, your client program will display what was dealt for the Player and Banker's hands and pause for a few seconds.
- It will then end the game if there is a "natural" win and post a message and update the current winnings field or inform that user that there is no natural. Pause for a few seconds.
- It will then display whether the Player got another card and display it or not. Then pause again.
- It will then display whether the Banker got another card and display it or not. Then pause again,
- It will then report the results of the game and update the current winnings field.
- To play again, the user would press the play button. Pressing this button, will allow the user to select a different bet amount and choose either Banker, Player or Draw again. The user can also press a button to quit and exit the program.

Passing info between clients and server:

You must implement the BaccaratInfo class:

class BaccaratInfo implements Serializable{}.

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. The data fields are up to you and can vary depending on students individual implementations.

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. A minimum of two unit tests per required method and one per class constructor.

UI and UX design:

You are required to use the best practices we discussed in lecture for designing the client programs user interface. The client should know what is happening and what to do at all times. Your interface should be intuitive and not cluttered.

How to Start:

- **Deal a few hands of Baccarat, or play online, to understand the logic and game flow as well as the payouts for a winning bid.**
- **Create your wireframes and UML class diagrams. Use these to think through how the game and networking will be done**
- **Work on the classes in the server first without worrying about networking or GUI. If you get all those to work, you know you can play the game on the server.**
- **Work on the client GUI. How will you display cards and information? How will you add and delete cards from your interface?**
- **Try passing simple objects back and forth between the server and your clients before trying to pass all of the game information.**
- **Figure out all of the fields in your GameInfo class. How will game information be represented?**
- **Test everything as you go, do not wait till the end; it will save you a lot of debugging time!**

CS 342 Project#3 Fall 2020

Part 1: Wireframe and UML Class Diagrams:
You are required to create a wireframe for the client GUI and UML Class Diagrams for the server and client programs; including all classes, data members and methods of those classes, interfaces and interactions between them. Format these as PDFs.

Part 2: two Maven projects (a server program and a client program):
Your server program will be its own Maven project and your client program will be its own Maven project. You will zip both together for your submission.

Electronic Submission:
If you worked in a group, only one of you needs to submit parts one and two.
For part one:
just submit a pdf of the wireframe and two class diagrams. Make sure both of your names are in the pdf if you worked in a group.
For part two:
You must include a PDF file called Collaboration.pdf if you worked in a team of 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 both the client and server Maven projects (and PDF if you worked in a group) 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.

Assignment Details:
- You may NOT submit part #1 late.
- You CAN submit part #2 up to 24 hours late for a 15% 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.