Assignment #4
CSCI 201 Fall 2017
4.5% of course grade

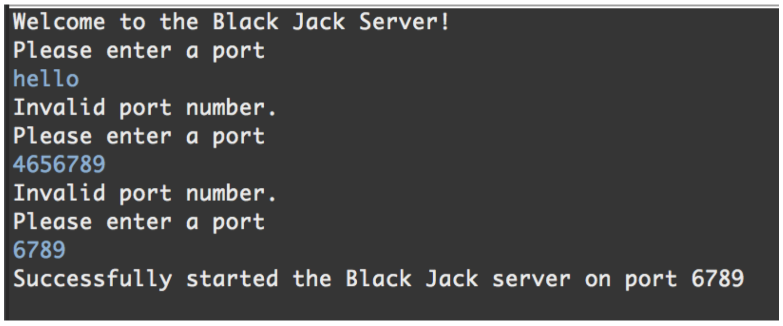
Title
Creating a Simple Networked Game

Lecture Topics Needed
Basic Java Programming
Multi-Threading
Networking

Introduction
For this assignment, we will take a break from building the 201 course website, and implement a standalone networked game. You will be implementing a simplified version of the card game black jack. You will be using pure Java to create the game, with a simple console based UI. You will need to use your knowledge of networking and multi-threading in order to complete the program.

Server/Client Set Up
Your application will need two programs: the server and the client. The server should keep track of all the current games in progress and the players associated with each game. The server program should be started as follows:

1) A welcome message should be displayed.
2) The user should be prompted for the server's port number.
3) If the port number is invalid (i.e. an exception was thrown upon trying to create a ServerSocket), go back to step 2.
4) Display a message to the user that the server has been successfully started, and the server should continue running.

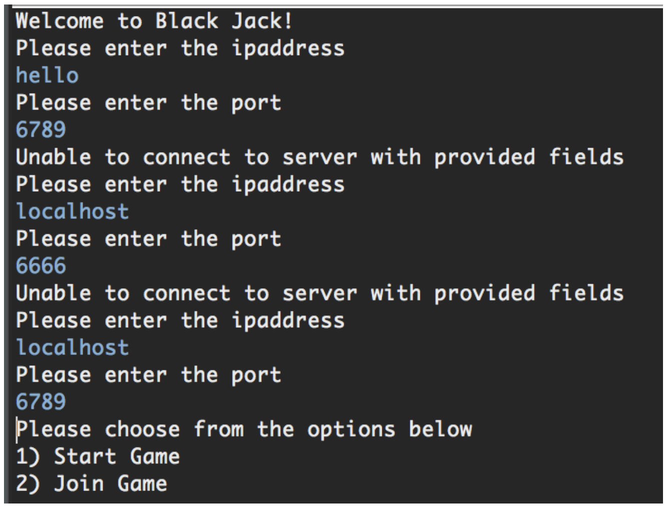


```
Welcome to the Black Jack Server!
Please enter a port
hello
Invalid port number.
Please enter a port
4656789
Invalid port number.
Please enter a port
6789
Successfully started the Black Jack server on port 6789
```

Image 1: Starting the server

The client program should be started as follows:

1) A welcome message should be displayed.
2) The user should be first prompted for the IP address of the server.
3) The user should then be prompted for the port number of the application on the server.
4) If the combination of the IP address and port is invalid (i.e. an exception was thrown upon creating a Socket), go back to step 2).



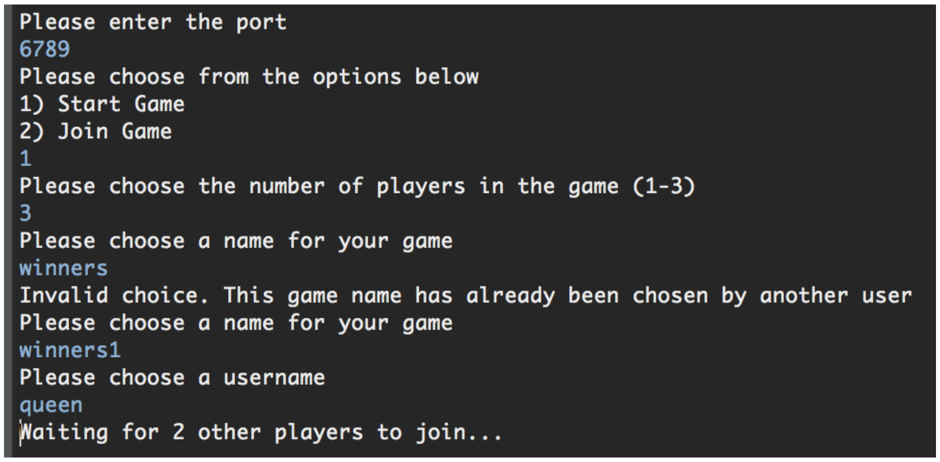
```
Welcome to Black Jack!
Please enter the ipaddress
hello
Please enter the port
6789
Unable to connect to server with provided fields
Please enter the ipaddress
localhost
Please enter the port
6666
Unable to connect to server with provided fields
Please enter the ipaddress
localhost
Please enter the port
6789
Please choose from the options below
1) Start Game
2) Join Game
```

Image 2: Connecting to the server from the client program

Game Set Up

On the client side, when the user successfully connects to the Black Jack server, a menu should print with the option to either start a game or join a game. If the user inputs a ‘1’ indicating they wish to start a game, the program should behave as follows:

1) Ask the user to choose the number of players in the game. This will be a number between 1 and 3, inclusively. Make sure you check for appropriate values here.
2) The user should then be prompted for a unique name for their game.
    a) If the user enters a name that is already in use by another **ongoing** game, they should receive a message that their input was invalid, and the program should go back to step 2).
3) The user should then be prompted for their username, which can be anything except the empty string.
4) The user should then see a message with the number of players that must join before the game can start.

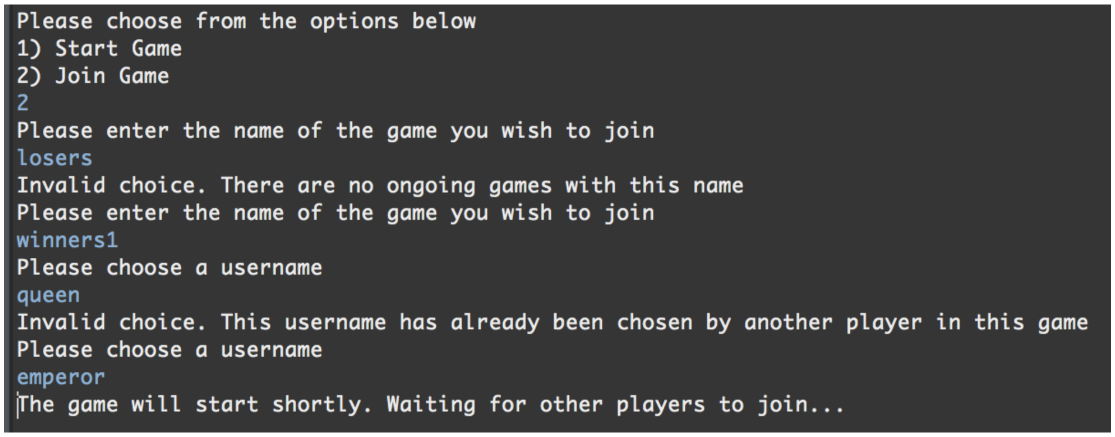


```
Please enter the port
6789
Please choose from the options below
1) Start Game
2) Join Game
1
Please choose the number of players in the game (1-3)
3
Please choose a name for your game
winners
Invalid choice. This game name has already been chosen by another user
Please choose a name for your game
winners1
Please choose a username
queen
Waiting for 2 other players to join...
```

Image 3: Starting a game

If the user inputs a '2', indicating they wish to join a game, the program should behave as follows:

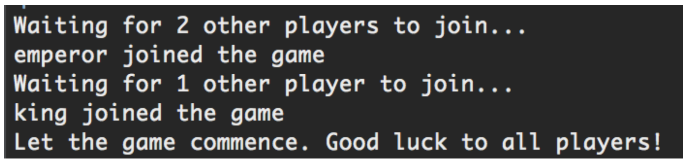
1) The user should be prompted for the name of the game they wish to join.
    a) If a game with this name **that is waiting for players** does not exist, then print a message that the choice was invalid. Have the program go back to step 1).
2) Have the user enter their choice of username.
    a) If this username has already been chosen by another player **in this game**, then print a message that the choice was invalid. Have the program go to 3)
3) Print a message that the game will begin shortly after other players join.



```
Please choose from the options below
1) Start Game
2) Join Game
2
Please enter the name of the game you wish to join
losers
Invalid choice. There are no ongoing games with this name
Please enter the name of the game you wish to join
winners1
Please choose a username
queen
Invalid choice. This username has already been chosen by another player in this game
Please choose a username
emperor
The game will start shortly. Waiting for other players to join...
```

Image 4: Joining a game

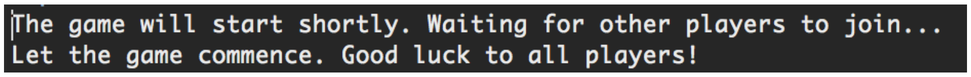
For the user that created the game, they should see a message on their console every time another player joins the game.



```
Waiting for 2 other players to join...
emperor joined the game
Waiting for 1 other player to join...
king joined the game
Let the game commence. Good luck to all players!
```

Image 5: Player that started the game gets updated on when players join

For the users that joined the game, they should not get any updates, and are only notified when the game starts.



```
The game will start shortly. Waiting for other players to join...
Let the game commence. Good luck to all players!
```

Image 6: Player that joined the game is only notified when the the game has been started

Rules of the Game

Whether or not you are familiar with the game of Black Jack, please **do not skip** this section as the game has been simplified for your implementation.

Goal of the Game

The goal of the game is to accumulate cards to the point where the sum of their values is more than that of the dealer (assuming the dealer doesn't go over 21) but less than or equal to 21 (face cards have the following values: Jack = 10, Queen = 10, King = 10, Ace = 1 or 11).

Gaining and Losing Chips

Before a round of the game begins, the players bet an amount of their total chips.

- The players will lose their chips to the dealer if they 'bust' during game play. A player 'busts' if they add additional cards to their hand to the point where the sum exceeds 21.
- The player will gain double the chips they bet if they get 'blackjack' and the dealer does not. A 'blackjack' hand is when the sum of the first two cards dealt equals 21 exactly.
    - If both the dealer and the player have 'blackjack', then the player neither gains chips nor loses chips.
- The player will gain the amount that they bet if they have not busted and their card sum exceeds the card sum of the dealer.
- The player also will gain the amount that they bet if they have not busted and the dealer has busted.
- The player will lose their chips if their card sum is less than the card sum of the dealer (assuming the dealer didn't bust).
- The player will neither gain nor lose chips if their card sum is the same as the card sum of the dealer.

Round of Play
In each round, the dealer of the game shuffles the cards and deals two cards face up to each player. They then deal one card face down and one card face up to themselves. Once the cards have been dealt, each player has a turn to add more cards to their hand. The player has two choices: they can either 'hit' — they are dealt an additional card to them face up — or they can 'stay' — finalize their hand, at which point they can no longer add additional cards, and the game play moves to the next player.

If the player chooses to 'hit' and their new card sum exceeds 21, they automatically 'bust' and lose the round, giving their bet chips to the dealer. Until the player chooses to 'stay', the player can 'hit' as many times as they like (unless they 'bust' during the process). If the player has an Ace in their hand, they can treat it in their sum as either a 1 or an 11. If their hand would 'bust' if the Ace was valued at 11, then the Ace is automatically treated with a value of 1.

After each player stays or has busted, the dealer reveals their face down card. The dealer is then required to 'hit' until their card sum is greater than or equal to 17. If the dealer is dealt an Ace at any point, they should 'stay' only if the Ace valued at 11 brings their total to 17 or more. However, if the Ace valued at 11 causes the dealer to bust, the value of the Ace is 1.

At this point, chips are either distributed or collected from each player based on the hand of the dealer (as described in the second section above). If any of the players now has a chip total of 0, the game ends. The player with the most chips at the end will be declared the winner. If all players have 0 chips left, the dealer is declared the winner.

If none of the players have a chip total of 0, a new round begins.

<u>Game Play</u>
Each player will begin the game with 500 chips. The dealer (i.e. your program) should 'shuffle' the deck before each round— in other words the dealing of the cards should be pseudo-random. Each player, in order of when they joined the game, should be given the opportunity to bet chips for the round. Their bet must be greater than 0 and less than or equal to their chip total (you can assume the player will always enter a valid bet amount). Each time a player makes their bet, all other players should be notified with the player's name and bet amount. Then each player should see a message as to whose turn it is to bet next.

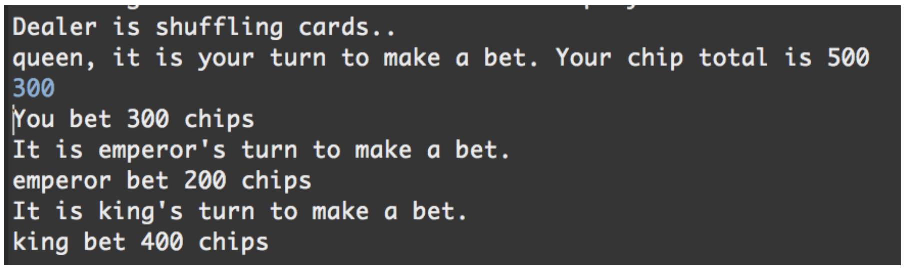
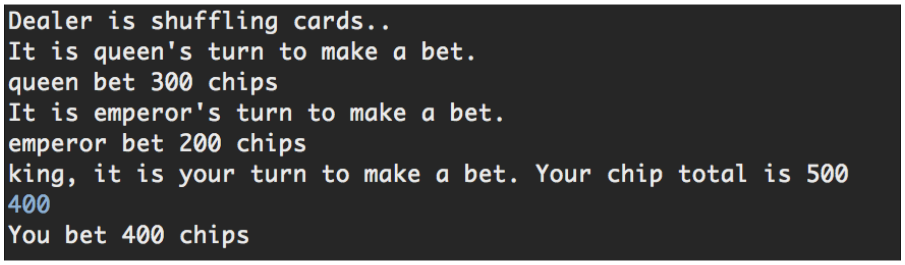

```
Dealer is shuffling cards..
queen, it is your turn to make a bet. Your chip total is 500
300
You bet 300 chips
It is emperor's turn to make a bet.
emperor bet 200 chips
It is king's turn to make a bet.
king bet 400 chips
```

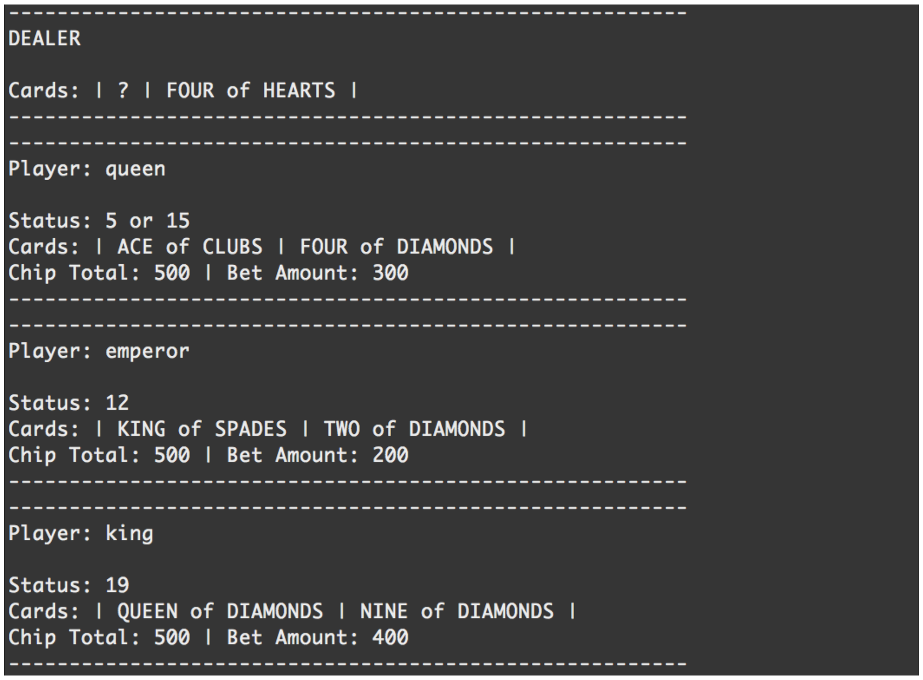
Image 7: Betting, from the perspective of Player 1 (i.e. 'queen')



```
Dealer is shuffling cards..
It is queen's turn to make a bet.
queen bet 300 chips
It is emperor's turn to make a bet.
emperor bet 200 chips
king, it is your turn to make a bet. Your chip total is 500
400
You bet 400 chips
```

Image 8: Betting, from the perspective of Player 3 (i.e. 'king')

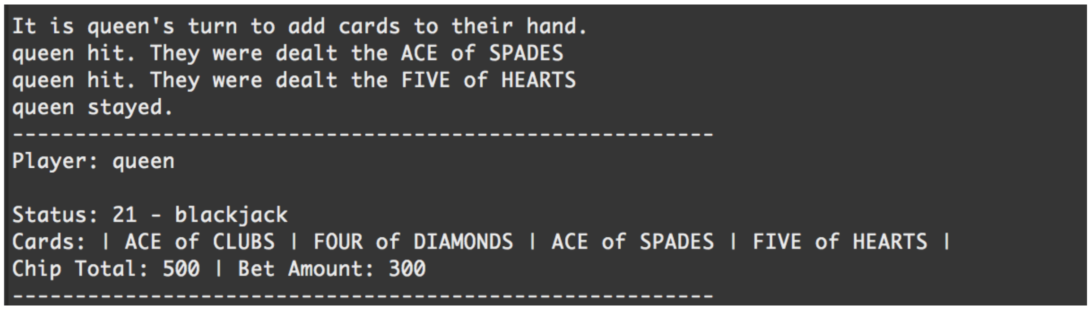
The dealer should assign two cards to each player and to themselves. The players' and dealer's cards should be displayed with one of the dealer's cards hidden. After the betting, the initial state of the game should be printed. It should show the values and suits of the face up cards, the chip total, the bet amount, and the status (card sum) of the cards for each player.



```
--------------------------------------------------------
DEALER

Cards: | ? | FOUR of HEARTS |
--------------------------------------------------------
--------------------------------------------------------
Player: queen

Status: 5 or 15
Cards: | ACE of CLUBS | FOUR of DIAMONDS |
Chip Total: 500 | Bet Amount: 300
--------------------------------------------------------
--------------------------------------------------------
Player: emperor

Status: 12
Cards: | KING of SPADES | TWO of DIAMONDS |
Chip Total: 500 | Bet Amount: 200
--------------------------------------------------------
--------------------------------------------------------
Player: king

Status: 19
Cards: | QUEEN of DIAMONDS | NINE of DIAMONDS |
Chip Total: 500 | Bet Amount: 400
--------------------------------------------------------
```

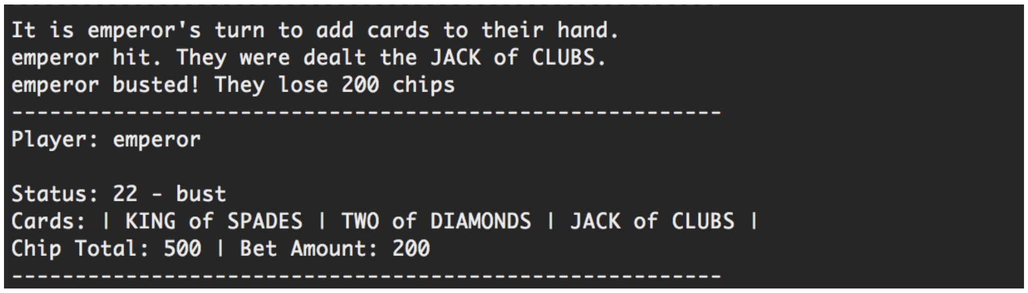
Image 9: The UI print out of each player's current state, including the dealer

Then each player should have a turn to add cards to their hand (in the order they joined the game, which in the screenshot example is ‘queen’, then ‘emperor’, then ‘king’). Each time a player chooses to either ‘hit’ or ‘stay’, the other players should see a message with that player’s choice and the dealt card (if the choice was to ‘hit’). When the current player either chooses to ‘stay’ or they ‘bust’, there should be a printout with the new state of that player’s hand.



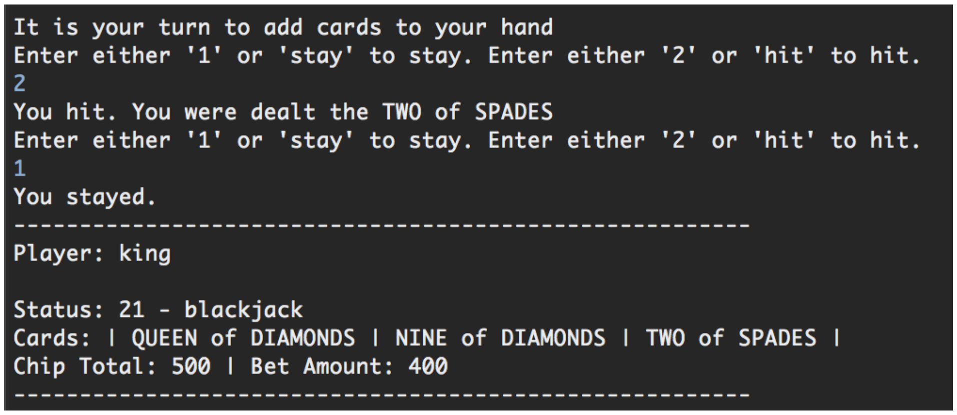
```
It is queen's turn to add cards to their hand.
queen hit. They were dealt the ACE of SPADES
queen hit. They were dealt the FIVE of HEARTS
queen stayed.
-------------------------------------------------------
Player: queen

Status: 21 - blackjack
Cards: | ACE of CLUBS | FOUR of DIAMONDS | ACE of SPADES | FIVE of HEARTS |
Chip Total: 500 | Bet Amount: 300
-------------------------------------------------------
```

Image 10: queen’s turn, from the perspective of Player 2 (‘emperor’) and Player 3 (‘king’)



```
It is emperor's turn to add cards to their hand.
emperor hit. They were dealt the JACK of CLUBS.
emperor busted! They lose 200 chips
-------------------------------------------------------
Player: emperor

Status: 22 - bust
Cards: | KING of SPADES | TWO of DIAMONDS | JACK of CLUBS |
Chip Total: 500 | Bet Amount: 200
-------------------------------------------------------
```

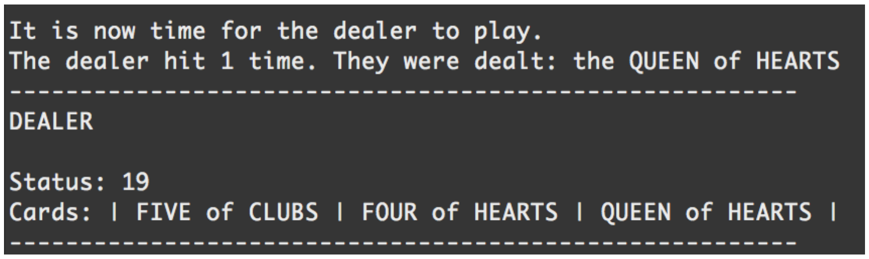
Image 11: emperor’s turn, from the perspective of Player 1 (‘queen’) and Player 3 (‘king’)



```
It is your turn to add cards to your hand
Enter either '1' or 'stay' to stay. Enter either '2' or 'hit' to hit.
2
You hit. You were dealt the TWO of SPADES
Enter either '1' or 'stay' to stay. Enter either '2' or 'hit' to hit.
1
You stayed.
-------------------------------------------------------
Player: king

Status: 21 - blackjack
Cards: | QUEEN of DIAMONDS | NINE of DIAMONDS | TWO of SPADES |
Chip Total: 500 | Bet Amount: 400
-------------------------------------------------------
```

Image 12: king’s turn, from the perspective of Player 3 (‘king’). Note the commands provided to the user for choosing to either ‘stay’ or ‘hit’

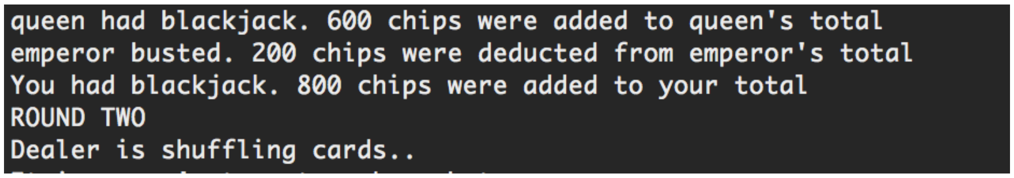
After each player has had a turn to add cards to their hand, the dealer then adds cards to their own hand if necessary (remember, the dealer must 'hit' until their sum is greater than or equal to 17).



```
It is now time for the dealer to play.
The dealer hit 1 time. They were dealt: the QUEEN of HEARTS
-------------------------------------------------------
DEALER

Status: 19
Cards: | FIVE of CLUBS | FOUR of HEARTS | QUEEN of HEARTS |
-------------------------------------------------------
```

Image 13: From the perspective of all players, UI print out of the dealer's hand

At this point, it should be determined for each player if they should be awarded chips, deducted chips or neither.



```
queen had blackjack. 600 chips were added to queen's total
emperor busted. 200 chips were deducted from emperor's total
You had blackjack. 800 chips were added to your total
ROUND TWO
Dealer is shuffling cards..
```

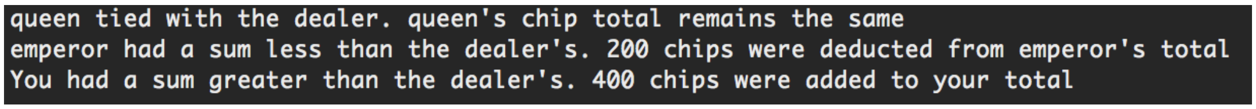
Image 14: From the perspective of Player 3 ('king'), determining new chip totals. Note that player's with 'blackjack' receive double their bet amounts.

The rounds should continue until one of the players runs out of chips. Each round, the 'Chip Total' field in the UI should update to show the new chip total for each player based on the results of the previous round.

When the game ends, print a message to each player stating the player that won the game and the player that lost the game (the player that ran out of chips). At this point, the client program can terminate.

Please make your console output as similar to the screenshots as possible. We will be looking for you to include the same detail in the outputted messages to players as is shown in these screenshots.

As an example of what to output for the other result states of the players, let us assume 'queen' tied with the dealer, 'emperor' had a lesser sum than the dealer, and 'king' had a greater sum than the dealer. The output would look like the following (assuming the players made the same bets as in the example game play in the previous screenshots):



```
queen tied with the dealer. queen's chip total remains the same
emperor had a sum less than the dealer's. 200 chips were deducted from emperor's total
You had a sum greater than the dealer's. 400 chips were added to your total
```

Image 15: From the perspective of Player 3 ('king'), example of alternate outcomes.

Program Execution

We will first run an instance of your server program, and then we will run multiple instances of your client program. Make sure that your server can handle multiple ongoing games at once, and multiple players in each game.

Note: You **will** be graded on code quality for this assignment. Please remember to include which classes contain the main methods for your server and client programs in your ReadMe.

Grading Criteria (4.5%)

Code quality (0.5%)

0.5% - your code conforms to the style rubric (see last page for details)

Starting Server/Client (0.15%)

0.05% - starting the server operates as described and displayed in **Image 1**
0.1% - connecting to the server from the client program operates as described and displayed in **Image 2**

Game Set Up (0.7%)

0.3% - starting a game operates as described and displayed in **Image 3** and **Image 5**
0.4% - joining a game operates as described and displayed in **Image 4** and **Image 6**

Game Play (2.65%)

0.3% - the betting operates as described and displayed in **Image 7** and **Image 8**
0.2% - the initial state of the round is printed similar to **Image 9**
0.6% - viewing the turns of other players operates as described and displayed in **Image 10** and **Image 11**
0.3% - the turn for the current user operates as described and displayed in **Image 12**
0.2% - the dealer's turn operates as described and displayed in **Image 13**
0.4% - the distribution of chips at the end of each round operates as described and displayed in **Image 14** and **Image 15**
0.2% - the program properly starts successive rounds with chip totals based on the results on the previous rounds.
0.45% - the program plays rounds until a player runs out of chips, at which point the loser and winner are announced to all players.

Other (0.5%)

0.5% - the server properly handles multiple ongoing games at once

Style Guide (with grading breakdown)

- Code commenting (Total = 0.1%)
    - Methods should have a 1-3 line comment above with a description of the inputs, outputs, and what the function does (0.08%)
    - For every 20 lines of code, there should be at least 1 comment (0.02%)
- Naming conventions (Total = 0.1%)
    - Use camel case for naming variables, classes and functions (0.05%)
        - For functions and variables: first letter is lower case
        - For classes: first letter is upper case
    - Unless indexing loops, use intuitive naming for variables to improve readability of your code (0.05%). Example:
      OK:
      ```
      for (int i = 0; i < array.length; i++) {
           System.out.println(array[i]);
      }
      ```
      NOT OK:
      ```
      List<Integer> i = new ArrayList<>();
      ```
- README: Your project must contain a README that includes the following (Total = 0.1%)
    - Name, student ID number, email, lecture section number(0.025%)
    - Anything important that should be noted to your grader - something hardcoded, functionality you know isn't working, special instructions (0.025%)
    - Description of your class structure, indicating where your main class and/or main method is (0.05%)
- You should be using multiple packages, multiple classes, and multiple functions within classes (Total = 0.15%)
    - You should have at least 2 packages (and no default package) (0.025%)
    - You should have at least 6 classes (0.025%)
    - You should have at least 18 functions (includes getters and setters), each with no more than 100 lines of code (0.1%)
- Use private variables as much as possible: if something is public explain in a comment why it must be public (0.025%)
- If you use any static objects or methods (besides the main method), explain in a comment why it must be static(0.025%)