Java CP2561 – Project

Note that the project, as defined below, is meant to be continuation of development, and enhancement of functionality, of what was completed in our Collections assignment. It is assumed that your solution for the Collections assignment was fundamentally working before starting the project. If you have any concerns or questions after your feedback on the assignment, please let me know.

Specifications:

We have created a new game called "One Hundreds" but the next stage of development will be to enhance the playability of the game. We will do this primarily by adding the capability for players to connect from their own "client" application to the central game server. As part of this, the server will handle client requests in a multi-threaded fashion. Much of your development logic related to the game should still be very relevant, however it may require a restructuring to complete the necessary tasks. Feel free to create and/or modify classes as needed.

Networking Component:

Players should be able to connect to the game server from their own client.

- When a client connects to the server, the server should welcome them, and ask for a player name. The player name will be used to create the new player object.
- The server should contain a method to validate a name using a regular expression – i.e. a name should contain 4 – 10 characters (only letters). If a name does not meet the requirement, the server should ask the client to enter a new name.
- Essentially, the server should "deal" cards to the player (client) – and when the game starts, the players hand should be displayed/printed to the client.
- For each "hand", the player's card is sent to the server, and a result is returned to the client (it should show each of the player cards played for that round, and which player won that hand).
- When all hands have been played, the results of the game should be displayed to the client stating the overall winner.
- The player should be able to specify if they would like to play another game.

While it would be great to be totally dynamic from a player perspective, let's assume that the server will wait until 4 clients connect before the game starts. This can be a static requirement that's coded in.

If your solution requires the sending of objects from client to server, reading or writing objects to a socket is similar to how we wrote and read objects to a file (think serializable), however I have noticed in the past that it seems to require creating the ObjectOutputStream before the ObjectInputStream (at both ends – client and/or server). If I recall correctly, it was something about the stream header.

Multithreading Component:

The server should be written to be multithreaded. Essentially, each client connection should spawn a new thread when they connect, and each thread relates to the game play for that client. There is flexibility here in terms of implementation specifications (e.g. you can extend Thread or implement Runnable) as long as your program does implement true multithreading. Depending on your implementation, you may require the use of synchronization for certain aspects of game play.

NOTE:

To enable the running of multiple instances of a Java file in IntelliJ (e.g. when testing client connections), do the following:

- Select the "Run" option in the top menu.
- Choose "Edit Configurations"
- Select the class(es) that you'd like to run multiple instances of, then do one of the following (depends on the version of IntelliJ you have):
    - Click the checkbox in the upper right corner "Allow running in parallel"
    - Or, select the blue "Modify options" link and then select "Allow multiple instances"

Now, you will be able to launch/run a Java program multiple times, each execution will be a separate instance.

Other requirements:

- For all of your classes, comment your code and generate documentation using the Javadoc tool

Date Due:

Export your project and please submit to the dropbox (Project) on or before December 11th, 11:59 p.m.

On the next pages, I have included the Collections assignment for easy reference if you would like to check on previous specifications.

Previous Collections Assignment (for reference)

Collections

Let's create a new type of card game, called One Hundreds, with the following characteristics:

- Total cards in the deck are 100.
- Our cards have numeric values (integers) that identify them, and they are numbered 1 through 100.
- During each deck creation, four random cards become "wildcards", they will be identified as such (e.g. perhaps simply containing a string "w")
- All other cards (96) will be identified as normal (e.g. perhaps simply containing a string "n")
- The higher the numeric value, the higher the value of the card. E.g. 73 will beat a 32.
    - Exception: A wildcard will beat all – e.g. a wildcard 2 will beat a normal 100
- If 2 or more wildcards are played in the same hand:
    - The wildcard with the lowest numeric value wins
- Hands are continued to be played while there are cards remaining in the deck for all players to receive a card.
- Score is kept simply as a player winning a hand, e.g. if player two wins 6 hands, their score is 6.

Each of the required classes are defined below. As a general comment, make use of the collection capabilities and methods (e.g. Collections, ArrayList, LinkedList, etc.) where appropriate.

Create a class called Card:

- This class will create a card object that has two attributes, value and status
    - The value represents an integer, that is the value of the card
    - The status represents a String, that is an identified of either normal or wildcard (e.g. n or w)
- The class should have get and set methods for each attribute.
- You may choose to create other methods in your Card class to do any convenient action for your implementation of the game (i.e. there is flexibility for you to add other methods to the Card class).

Create a class called Player:

- This class will provide functionality related to players of the card game.
- The class will create a player object that has 2 attributes, name (String) and hand (LinkedList)
    - The name represents a player name – assume first name only – and should be a String.

    - The “hand” represents card objects assigned to the player, and should be a LinkedList
- The class should have get and set methods for each attribute (only name at this point, we may extend in the future).
- The class should have a method called PrintHand that outputs the values of card objects assigned to the LinkedList.

Create a class called CardDeck:

- Create a method called GenerateDeck that returns an ArrayList of card objects, and does the following:
    - Create 100 card objects, as described above, that represent a full deck.
    - You can create these card objects, programmatically, any way that you choose, as long as they are representative of the constraints of the game as described above.
    - As part of your “creation” implementation, assign 4 cards at random to receive “wildcard” status, the rest will be “normal” cards. Note, regarding random numbers, there are many ways to do this, but one standard way is to use java.util.Random.
    - Store the card objects in an ArrayList.

- Create a method called ShuffleDeck that takes 2 parameters and does the following:
    - First parameter is an ArrayList of card objects.
    - Second parameter is an integer, representing the number of times to shuffle
    - The ArrayList should be “shuffled” as many times as the integer that’s passed in (e.g. 1 would mean one shuffle).
    - An ArrayList is returned.

- Create a method called PrintDeck that returns a printed representation of the Deck ArrayList (note, make sure of a ListIterator as part of your implementation for this feature).

- Create a method called CardsRemaining that takes an ArrayList parameter representing a deck and returns that number of cards left in the deck.

Create a class called OneHundreds:

- In the main method:
    - Ask the user for the number of players (a range of 2-4)
    - Create a new player object for each of the players, using a name that is input from the terminal. The “hand” can be an empty LinkedList at this point.
    - A “score” HashMap should be created, with player name as the “key” and a mapped integer that represents wins. Wins can be zero initially.
    - Generate a new deck for the game

- Shuffle the deck
- "Deal" cards to each player. Dealing involves the following:
  - Take the top (first) card from the deck and assign to the 1st player (in their player object LinkedList), and so on with each additional player.
  - Note that when a card is "dealt" it should be removed from the deck.
  - Continue dealing cards from the deck while there are enough cards for all players to receive one (i.e. There may be cards left over in the deck)
- Once, finished "dealing", each player lays their first card (the "top" or first card from their associated "hand").
  - This should be displayed on the screen.
- The winner is determined from the values that are displayed, as outlined in the rules of the game.
  - A message should indicate who won the hand
- The score should be updated (HashMap) to reflect the win for that player.
- The next round begins, and each player lays their second card, etc. until there are no more cards left.
- Note that cards from each round should be discarded from the hand when played
- When there are no more cards in player hands, the game ends.
- At the end of the game:
  - Any remaining cards in the deck should be displayed to the screen (i.e. if there were cards left over in the deck when dealing).
  - The score should be displayed, with a specific message indicating which player won.