The University of Hong Kong
Department of Computer Science
COMP2396 Object-oriented Programming and Java

Assignment 5

Deadline: 11:55pm, 1ˢᵗ Dec, 2017.

Overview

This assignment tests your understanding of networking and multi-threading, and their implementations in Java. You are going to modify the Big Two card game you developed in assignment 4, and make it support 4 players playing over the internet. A number of classes and interfaces will be provided to aid your implementation. These include all the classes and interfaces provided in assignment 4, as well as a BigTwoServer class (and its superclass CardGameServer) which models a game server for the Big Two card game, a CardGameMessage class (and its superclass GameMessage) which models the messages used for communication between a game server and its clients, and a NetworkGame interface. You may refer to their Javadoc for details of these classes and interfaces. You should **NOT** modify any of these classes and interfaces in completing your assignment.

You are required to implement a BigTwoClient class which models a client for the Big Two card game. This class, like the BigTwo class in the previous assignment, implements the CardGame interface and models the Big Two card game logics. Besides, it also implements the NetworkGame interface and handles the communication with other clients by sending and receiving messages to and from a game server. In order to allow players to exchange chat messages during the game, you will also need to modify your BigTwoTable class, which implements the CardGameTable interface, accordingly. You are free to introduce new instance variables and methods to these classes. Besides, you are also free to design and introduce new classes in the inheritance trees as appropriate. With a proper OO design, it is most likely not necessary for you to touch the rest of the classes you implemented in assignment 4. You should write Javadoc for all public classes and their public class members.

Specifications

Behavior of the game server

Below is a description of the general behavior of the provided Big Two card game server:

- Upon establishing a successful connection with a new client, the server will send a message of the type PLAYER_LIST to this client. The playerID in this message specifies the playerID (i.e., index) of the local player of this client, and data is a reference to a regular array of strings specifying the names of the existing players. A null value in the array means that particular player does not exist.
- If the server is already full when a new client establishes a connection with it, the server will send a message of the type FULL to this client and then close the connection immediately. The playerID and data in this message are -1 and null, respectively, which can simply be ignored.
- When a connection to a client is lost, the server will broadcast a message of the type QUIT to its clients. The playerID in this message specifies the playerID (i.e., index) of the player who loses the connection, and data is a reference to a string representing the IP address and TCP port of this player (e.g., "/127.0.0.1:2396").

- Upon receiving a message from a client, the server will look up its playerID based on its socket connection, and replace the `playerID` in the message with this value. This implies that a client can simply assign `-1` to the `playerID` in a message it sends to the server.
- Upon receiving a message of type `JOIN` from a client, the server will broadcast a message of the type `JOIN` to its clients. The `playerID` in this message specifies the playerID (i.e., index) of the player who joins the game, and `data` is a reference to a string representing the name of this player.
- Upon receiving a message of the type `READY` from a client, the server will broadcast a message of the type `READY` to its clients. The `playerID` in this message specifies the playerID (i.e., index) of the player who is ready, and `data` is `null` which can simply be ignored.
- When all the clients are ready, the server will broadcast a message of the type `START` to its clients. The `playerID` in this message is `-1`, which can simply be ignored, and `data` is a reference to a newly created and shuffled BigTwoDeck object (to be used for the new game).
- Upon receiving a message of the type `MSG` from a client, the server will broadcast a message of the type `MSG` to its clients. The `playerID` in this message specifies the playerID (i.e., index) of the player who sends out this chat message, and `data` is a reference to a formatted string including the player's name, his/her IP address and TCP port, and the original chat message (e.g., `"Kenneth (/127.0.0.1:2396): Hello!"`).
- Upon receiving a message of the type `MOVE` from a client, the server will simply broadcast this message to its clients.

Behavior of the client

Below is a description of the general behavior of the Big Two card game client:

- When the client starts, it should prompt the user to enter his/her name. (You may use `JOptionPane.showInputDialog()` to show an input dialog box for the user to enter his/her name.)
- After the user has entered his/her name, the client should make a connection to the game server. (You might either prompt the user to enter the IP address and TCP port for the server, or simply use a hardcoded IP address, e.g., `"127.0.0.1"`, and TCP port, e.g., `2396`, for the server.)
- Upon establishing a successful connection to the server, the client should send a message of type `JOIN`, with `playerID` and `data` being `-1` and a reference to a string representing the name of the local player, respectively, to the server.
- After sending a message of the type `JOIN` to the server, the client should send a message of type `READY`, with `playerID` and `data` being `-1` and `null`, respectively, to the server.
- Upon receiving a message of the type `PLAYER_LIST` from the server, the client should set the playerID of the local player and update the names in the player list. The `playerID` in this message specifies the playerID (i.e., index) of the local player, and `data` is a reference to a regular array of strings specifying the names of the players. A `null` value in the array means that particular player does not exist.
- Upon receiving a message of the type `JOIN` from the server, the client should add a new player to the player list by updating his/her name. The `playerID` in this message specifies the playerID (i.e., index) of the new player, and `data` is a reference to a string specifying the name of this new player.

- Upon receiving a message of the type FULL from the server, the client should display a message in the text area of the BigTwoTable that the server is full and cannot join the game. The playerID and data in this message are -1 and null, respectively, which can simply be ignored.

- Upon receiving a message of the type QUIT from the server, the client should remove a player from the game by setting his/her name to an empty string. The playerID in this message specifies the playerID (i.e., index) of the player who leaves the game, and data is a reference to a string representing the IP address and TCP port of this player (e.g., "/127.0.0.1:9394"). If a game is in progress, the client should stop the game and then send a message of type READY, with playerID and data being -1 and null, respectively, to the server.

- Upon receiving a message of the type READY from the server, the client should display a message in the text area of the BigTwoTable that the specified player is ready. The playerID in this message specifies the playerID (i.e., index) of the player who is ready, and data is null which can simply be ignored.

- Upon receiving a message of the type START from the server, the client should start a new game with the given deck of cards (already shuffled). The playerID in this message is -1, which can simply be ignored, and data is a reference to a (shuffled) BigTwoDeck object. (You should call the start() method from the CardGame interface to start a new game. Note that you should **NOT** shuffle the deck again inside the start() method.)

- When the local player makes a move during a game, the client should send a message of the type MOVE, with playerID and data being -1 and a reference to a regular array of integers specifying the indices of the cards selected by the local player, respectively, to the server. (When the user presses either the "Play" or "Pass" button to make a move, you should call the makeMove() method from the CardGame interface to send a message of the type MOVE to the server.)

- Upon receiving a message of the type MOVE from the server, the client should check the move played by the specified player. The playerID in this message specifies the playerID (i.e., index) of the player who makes the move, and data is a reference to a regular array of integers specifying the indices of the cards selected by the player. (You should call the checkMove() method from the CardGame interface to check the move.)

- When the local player press "Enter" in the text input field, the client should send a message of the type MSG, with playerID and data being -1 and a reference to a string representing the text in the text field, respectively, to the server. The client should then reset the text input field to empty.

- Upon receiving a message of the type MSG from the server, the client should display the chat message in the chat window. The playerID in this message specifies the playerID (i.e., index) of the player who sends out the chat message, and data is a reference to a formatted string including the player's name, his/her IP address and TCP port, and the original chat message (e.g., "Kenneth (/127.0.0.1:9394): Hello!").

- When the game ends, the client should display the game results in a dialog box. When the user clicks the "OK" button on the dialog box, the dialog box should close and the client should send a message of the type READY, with playerID and data being -1 and null, respectively, to the server. (You may use JOptionPane.*showMessageDialog*() to show information in a message dialog box to the user.)

- If the client has not yet established a connection to the server, and the user selects "Connect" from the menu, the client should make a connection to the server. (You should call the `makeConnection()` method from the NetworkGame interface to connect to the server.)
- If the user selects "Quit" from menu, the client should close the window and terminate itself. (You may use `System.exit()` to terminate the client.)

The BigTwoClient class

The BigTwoClient class implements the CardGame interface and NetworkGame interface. It is used to model a Big Two card game that supports 4 players playing over the internet. Below is a detailed description for the BigTwoClient class.

Specification of the BigTwoClient class:

public constructor:

    `BigTwoClient()` – a constructor for creating a Big Two client. You should (i) create 4 players and add them to the list of players; (ii) create a Big Two table which builds the GUI for the game and handles user actions; and (iii) make a connection to the game server by calling the `makeConnection()` method from the NetworkGame interface.

private instance variables:

    `int numOfPlayers` – an integer specifying the number of players.

    `Deck deck` – a deck of cards.

    `ArrayList<CardGamePLayer> playerList` – a list of players.

    `ArrayList<Hand> handsOnTable` – a list of hands played on the table.

    `int playerID` – an integer specifying the playerID (i.e., index) of the local player.

    `String playerName` – a string specifying the name of the local player.

    `String serverIP` – a string specifying the IP address of the game server.

    `int serverPort` – an integer specifying the TCP port of the game server.

    `Socket sock` – a socket connection to the game server.

    `ObjectOutputStream oos` – an ObjectOutputStream for sending messages to the server.

    `int currentIdx` – an integer specifying the index of the player for the current turn.

    `BigTwoTable table` – a Big Two table which builds the GUI for the game and handles all user actions.

CardGame interface methods:

    `int getNumOfPlayers()` – a method for getting the number of players.

    `Deck getDeck()` – a method for getting the deck of cards being used.

    `ArrayList<CardGamePlayer> getPlayerList()` – a method for getting the list of players.

    `ArrayList<Hand> getHandsOnTable()` – a method for getting the list of hands played on the table.

    `int getCurrentIdx()` – a method for getting the index of the player for the current turn.

    `void start(Deck deck)` – a method for starting/restarting the game with a given shuffled deck of cards. You should (i) remove all the cards from the players as well as

from the table; (ii) distribute the cards to the players; (iii) identify the player who holds the 3 of Diamonds; (iv) set the `currentIdx` of the BigTwoClient instance to the playerID (i.e., index) of the player who holds the 3 of Diamonds; and (v) set the `activePlayer` of the BigTwoTable instance to the playerID (i.e., index) of the local player (i.e., only shows the cards of the local player and the local player can only select cards from his/her own hand).

`void makeMove(int playerID, int[] cardIdx)` – a method for making a move by a player with the specified playerID using the cards specified by the list of indices. This method should be called from the BigTwoTable when the local player presses either the "Play" or "Pass" button. You should create a CardGameMessage object of the type `MOVE`, with the `playerID` and `data` in this message being -1 and `cardIdx`, respectively, and send it to the game server using the `sendMessage()` method from the NetworkGame interface.

`void checkMove(int playerID, int[] cardIdx)` – a method for checking a move made by a player. This method should be called from the `parseMessage()` method from the NetworkGame interface when a message of the type `MOVE` is received from the game server. The `playerID` and `data` in this message give the playerID of the player who makes the move and a reference to a regular array of integers specifying the indices of the selected cards, respectively. These are used as the arguments in calling the `checkMove()` method.

`boolean endOfGame()` – a method for checking if the game ends.

NetworkGame interface methods:

`int getPlayerID()` – a method for getting the playerID (i.e., index) of the local player.

`void setPlayerID(int playerID)` – a method for setting the playerID (i.e., index) of the local player. This method should be called from the `parseMessage()` method when a message of the type `PLAYER_LIST` is received from the game server.

`String getPlayerName()` – a method for getting the name of the local player.

`void setPlayerName(String playerName)` – a method for setting the name of the local player.

`String getServerIP()` – a method for getting the IP address of the game server.

`void setServerIP(String serverIP)` – a method for setting the IP address of the game server.

`int getServerPort()` – a method for getting the TCP port of the game server.

`void setServerIP(int serverPort)` – a method for setting the TCP port of the game server.

`void makeConnection()` – a method for making a socket connection with the game server. Upon successful connection, you should (i) create an ObjectOutputStream for sending messages to the game server; (ii) create a thread for receiving messages from the game server; (iii) send a message of the type `JOIN` to the game server, with `playerID` being -1 and `data` being a reference to a string representing the name of the local player; (iv) send a message of the type `READY` to the game server, with `playerID` and `data` being -1 and `null`, respectively.

`void parseMessage(GameMessage message)` – a method for parsing the messages received from the game server. This method should be called from the thread

responsible for receiving messages from the game server. Based on the message type, different actions will be carried out (please refer to the general behavior of the client described in the previous section).

`void sendMessage(GameMessage message)` – a method for sending the specified message to the game server. This method should be called whenever the client wants to communicate with the game server or other clients.

inner class:

`class ServerHandler` – an inner class that implements the Runnable interface. You should implement the `run()` method from the Runnable interface and create a thread with an instance of this class as its job in the `makeConnection()` method from the NetworkGame interface for receiving messages from the game server. Upon receiving a message, the `parseMessage()` method from the NetworkGame interface should be called to parse the messages accordingly.

public static methods:

`void main(String[] args)` – a method for creating an instance of BigTwoClient.

`Hand composeHand(CardGamePlayer player, CardList cards)` – a method for returning a valid hand from the specified list of cards of the player. Returns `null` is no valid hand can be composed from the specified list of cards.

Graphical user interface

Besides the minimum requirement specified in assignment 4, your GUI should

- Have a text area showing the chat messages sent by the players.
- Have a text input field for a player to send out chat messages.
- Replace the "Restart" menu item with a "Connect" menu item for establishing a connection to the game server.

Sample output

Figure 1 shows an example of GUI for the Big Two client that satisfies the minimum requirement. You are not required to reproduce the same GUI exactly.
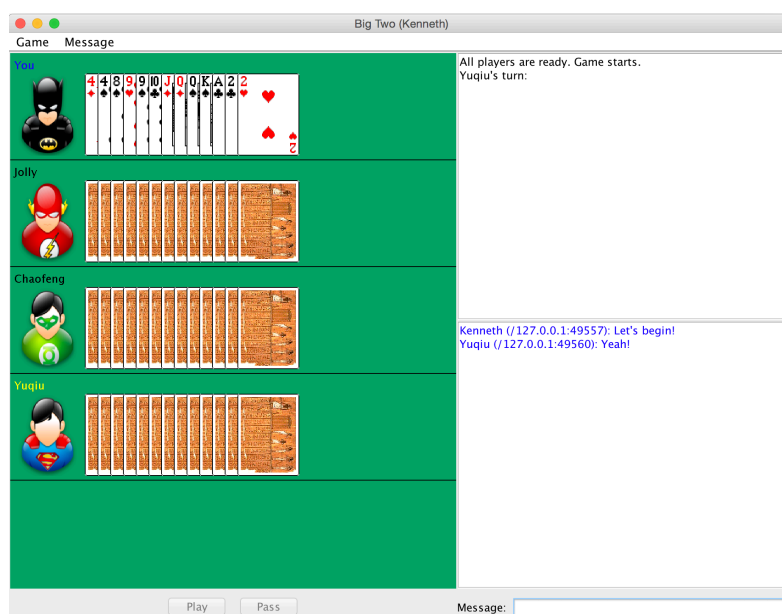

Figure 1. An example of GUI for the Big Two client.

Marking Scheme

Marks are distributed as follows:

- Establishing a socket connection to the game server (5%)
- Sending a message of the type JOIN to the server upon connection (5%)
- Sending a message of the type READY to the server upon connection (5%)
- Creating a thread for receiving incoming messages from the server (10%)
- Handling incoming messages of the type PLAYER_LIST (5%)
- Handling incoming messages of the type JOIN (5%)
- Handling incoming messages of the type FULL (5%)
- Handling incoming messages of the type QUIT (10%)
- Handling incoming messages of the type READY (5%)
- Handling incoming messages of the type START (5%)
- Handling incoming messages of the type MOVE (5%)
- Handling incoming messages of the type MSG (5%)
- Handling events from the "Play" and "Pass" buttons (5%)
- Handling events from the chat input field (5%)
- Implementation of the chat message area (5%)
- Implementation of the chat input field (5%)
- Implementation of the "Connect" menu item (5%)
- Handling possible concurrency problem (5%)

Submission

Please pack the source code (*.java) and images of your application into a single zip file, and submit it to the course Moodle page.

A few points to note:

- Always remember to write Javadoc for all public classes and their public class members.
- Always remember to submit the source code files (***.java**) but **NOT** the bytecode files (*.class).
- Always double check after your submission to ensure that you have submitted the most up-to-date source code files.
- Your assignment will not be marked if you have only submitted the bytecode files (*.class). You will get zero mark for the assignment.
- Please submit your assignment on time. Late submission will not be accepted.

~ End ~