SKJ ASSIGNMENT 1 REPORT

Thi Thu Trang Do 17774

A. General Solution Description

I. Example of execution

1. **To enter the game:** Run "bin/Main.class" or compile all files *.java in src and then run Main.class
   i) Player 1:
      Run Main.class
      Enter "localhost 1234 localhost 1234"
      No one to play
   ii) Player 2:
      Run Main.class
      Enter "localhost 1298 localhost 1234".
      Player 2 plays with player 1
   iii) Player 3:
      Run Main.class
      Enter "localhost 1245 localhost 1234"
      Player 3 plays with player 1 and 2
   iv) Player 4:
      Run Main.class
      Enter "localhost 1237 localhost 1245"
      Player 4 plays with player 1, 2 and 3
2. **To play all duels:**
   Let the program do itself
3. **To quit:**
   Enter "QUIT" (insensitive case)
4. **To check the duels in the tournament:**
   Open browser, type "localhost:8000"
   Refesh to update the status of the tournament

II. Classes and User Library

1. **User library: http-2.2.1.jar**
   For creating HTTP Server
2. **Player**
   Each instance is a pair of IP address and Port number of a player.
3. **Peer**
   Each instance is a node (agent) in the network. Class Peer contains methods to:
   1. join the tournament
   2. play all duels with all others already in the tournament
   3. play duels when a new one invites
   4. print result of all duels and quit

4. **Main**
   Containing method static void main(String[] args) to run the program.
5. **ReceiveMessageThread**
   A thread that listen on the server of a specific player. This class is for receiving messages from other peers (agents).
6. **Coordinator**
   The purpose of creating this class is to keep track of all players join and quit, inform all others about this and give the new player information about others already in the tournament and the turns (who starts first) for all duels it will make.
   It also reponds to HTTP Server requests from "localhost 8000".
7. **NewCoordinatorThread**
   Create a thread containing a server that listen on a special Port (65000 – its port) for each peer. When the current Coordinator quit, if it tells this peer to be new Coordinator, this server accepts this connection and set this peer (agent) as a new Coordinator.

III. General solution

There is a tournament in the local network. Each player has to enter its IP, Port and Introducing IP, Port. The first player is its own introducing peer.
The most recently entered peer will be the Coordinator of the game to ensure the connections between peers.

There is a HTTP Server for people to follow the tournament "localhost:8000".

1. Phrase 1: Join and Get Data from Coordinator

i) In Main class, user enters data from keyboard. If the input is invalid, host is unknown, port is already used, or introducing player's IP and Port is wrong, the user is asked to enter data again. When data format is correct, create a new instance of **Peer** (peer A).
ii) Create a server and an instance of **ReceiveMessageThread** to receive connections from others.
iii) Call method **join** on peer A.
iv) Peer A connects to the Introducing and get information about the Coordinator. If it is its own Introducing, it sets itself as Coordinator.
v) In case peer A is not the Coordinator, it connects to the Coordinator, the Coordinator adds it to the list allPlayers, gets list of all other players and list of all turns for each duel (randomly created). Store them into allPlayers and allDuels (the list excludes itself, only others). When peer A has got all data, it tells the Coordinator to inform others about it. Others add player A to list allPlayers.

2. Phrase 2: Play all duels with whom joins before and new coming players

(Describe below in B.III. How all duels after joining the tournament by a new player were implemented)

3. Phrase 3: Quit

1.i.1. When user types "QUIT" (case insensitive), method **quit** is called on peer A. Set the quitting flag to true, so it will not add any new player.
1.i.2. If peer A is the last one in the tournament, simply close the program.

1.i.3. Otherwise, firstly, remove this player A from the list of allPlayers of Coordinator. So when a new player comes, the list of allPlayers given to this new player will not contain peer A. And peer A just needs to wait for all duels to finish.

1.i.4. When peer A has already played with will others. It checks again whether it is the last player in the game. If so, simply close the program.

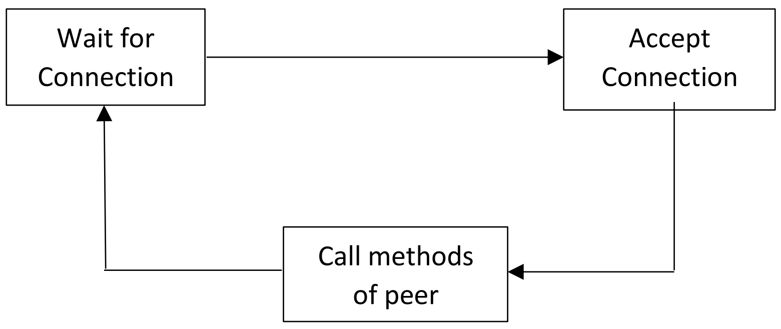
1.i.5. In case there are other players in the tournament, if peer A is the Coordinator, peer A tells its coordinator to select a new coordinator, tells it that it is the new coordinator and gives it data for response for HTTP Server "localhost:8000". The Coordinator closes the HTTP Server. New Coordinator opens HTTP Server again.
If peer A is not Coordinator, tells the Coordinator that it quits, the Coordinator removes it from the list and informs all others. All others remove peer A from the list.

1.i.6. Peer A close the program.

B. Precise Solution Description

I. How is connection established?

1. **To accept connections from other peers (agent)**
   New server dedicated for a peer is created right after getting valid information from keyboard.
   Then new instance of ReceiveMessageThread is created. This thread listens on this server all the time. When it receives a message, it decides what to do next and calls methods on peer, and breaks the connection. Then it waits for next connections.



2. **To create a connection to another peer (agent)**
   Whenever there is a need to connect to a peer, create a socket of type Socket. After finishing all communication, the socket is closed.
3. **New player and all others know each other**
   There is a Coordinator who tells new player about all others and tells all others about new player.

II. How data is kept

1. There is a Coordinator kept by the most recently entered peer.
2. When a new player entered, if it is the first player, create a new Coordinator.
   If it is not, it tells the Coordinator to add it to the list, give it information about other players, and then inform all others about this new one. Other players will add new player to the list.
3. When a player quits, if it is the last player, it simply closes the program.
   If it is not, it tells the Coordinator to remove it from the list so new players will not know

about it. On the other hand, it sets quitting flag to true, it will not add new player to list allPlayers. If it is the current Coordinator or it is selected to be new Coordinator, it will select the most recently entered player to be new Coordinator, gives it all data. Then the Coordinator informs all others about quitted player. The player finish quitting. All others remove this player from list.

4. If the current Coordinator and selected new Coordinator quitted one after another, the selected new Coordinator will have to select another player as new Coordinator before quitting.

III. How all duels after joining the tournament by a new player are implemented

1. Peer A after joining, starts to play all duels by calling method **playAllDuels**.
2. For each player and turn, peer A connects to the opponent, sends it a message “Want to play?”, gives the opponent information about itself and the turn for the opponent. The opponent receives the message. Both call method **startTheDuel** to play with each other. If they agree about the winner, they store the winner into list of all winners **allWinners**.
3. If the duel is OK, peer A tell the Coordinator about the result. The Coordinator stores result into a list allDuels. Then Coordinator changes response for HTTP Server “localhost:8000”
4. Each duel is played one after another. Each player can receive only one invitation at a moment. Only after finishing duel can it receive next invitation.
5. When there is a new player comes, new player gets information about all other peers from Coordinator (including peer A if peer A is not quitting). Peer A also receives message about new player from Coordinator. If it is not quitting, it adds new player to the list. The new coming player sends peer A message and initiates the duel.

III. How fairness is granted

1. The turn (who starts first) is randomly generated by the Coordinator.
2. The number nA and nB are randomly generated
3. Peer A sends an invitation to peer B. Peer A needs to choose a number first and then sends number A plus a random number (key) (nA + key A) to peer B. Only after already choosing number A, peer A can receive message from B. Peer B also sends in the same way (chooses a number first and sends nB + key B) to peer A. Both will not know whether the opponent’s number is odd or even.
4. Only after receiving (the opponent number + opponent key) can one send a message “Sending random number” to the other. When one receives message “Sending random number”, it waits a second. Then they send their random key to each other. So each will send its key only when it receives a message to tell it to send the key.
5. Peer can read the opponent’s number only when it already sends its key.
6. nA and nB are displayed only when two players have received the opponent’s true number.
7. Two peers calculate who the winner is. If both agree about the winner, the duel stops. If they do not, start the duel again.
8. If there is a problem with any duel of a new player, the duel is repeatedly played 3 times, the new player is forced to quit. Probably it is because the new player has cheated.
9. If the first player is the one that changed the program, all new players will be forced to quit, this is an exception. The program will not handle it. The second player knows the first player since in order to enter the game, the second player needs to know an Introducing (which is the first player). The second player will have to deal with the first player somehow.

C. Conclusion

The program successfully deals with 3 main problems: how data is kept, how all duels are implemented and how fairness is granted.

It also implements simple HTTP server “localhost:8000” as a monitor **(User library: http-2.2.1.jar).**

Contents