CSCI 201, Fall 2024, Assignment 2
Implementation Recommendations

A server-client crossword game application is implemented in this assignment with certain requirements on game play logics and UI display. This document will primarily focus on the high-level interpretation to achieve the crossword game functionalities detailed described in the assignment instruction “Assignment2.pdf”. The key points will be emphasized, and the implementation recommendations are provided for potential inspirations.

Outline

Part 1: The general overview

1.1. Resource Management

1.1.1 All game files (.csv) are placed in *gamedata* folder under current project directory. The program should be able to

1) scan all .csv files in *gamedata*,
2) check if a randomly selected file is valid, and
3) store the word entries (across/down-index-question-answer) properly.

1.1.2 A game data object needs …

1) to contain word entries properly,
2) to be updated once a client answers an entry correctly (or to record client answers),
3) to be visually generated, and then displayed in console.

1.1.3 All server resources are shared with limited number of clients (Number of client lines need to exactly match the number provided by Player 1)
   1) The game will not start if the clients are less than this number.
   2) No more players can join the game if this number is reached.

1.2. Process Management

1.2.1 Server-client communication
   1) One server communicates with multiple clients independently. It sends the updated game status to all clients, receive the response from all clients, and sync other clients with the message received from certain clients.
   2) There is no direct communication between any two of the clients. They communicate with intermedia assistance of the server.
   3) The main functionality of client ends is sending players type-ins and displaying the server messages. It is optional for the client end to record the player's game info (e.g. number of correct answers the player guesses).

1.2.2 Multi-thread concurrency
   1) The clients are independent with each other.
      a. Players' reactions in the clients do not interrupt other client lines.
      b. After the game starts, when one player (active player) is communicating the server, all other clients need to wait for the active player to finish the communication.
   2) The process of game preparation is independent from the process of connecting other clients (if there is supposed to be any). The game preparation includes valid game file selection, game file info storage, and game board generation.

Part 2: The class design

Based on the above analysis on resource management and process management, one possible structure design for the Server class and the Client class can be as follows.

2.1 Server Class

2.1.1 Game board management

In this part, students …

1) Can consider to use *java.io.File* to read the game file list in a certain directory. Can also consider *java.io.FilenameFilter* for file name filtering.

2) Need to randomly select a game file with the usage of *java.util.Random.*

3) Can consider to parse the selected .csv file by *java.util.Scanner* or *java.io.FileReader*
(more info can be found: https://piazza.com/class/ly551p113od3l9/post/121)

   Organize the entry info in .csv file (question direction, index, description, answer) and use a proper data structure to store it (e.g. Map, List, etc.)

   Step -2) and 3) should be repeated until the game file satisfies the requirements (check “Assignment.pdf” for details). Once a valid game file is found, parsed, and stored properly,

4) Generate the visual game board by spatially placing all word entries into the board.

   To represent the game board, it is very flexible to choose any proper 2-D data structure. ArrayList<ArrayList<String>>, Vector<Vector<String>>, char[][], String[][], or any other structures that can satisfy the storage and afterwards printing and also updating/modifying requirements are okay to use. Choose a data structure based on your own dataflow design.

   Then, to organize the 2-D spatial positions of the words, here are many possible solutions to achieve this goal. One possible solution can be:
   - Sort all the words by decreasing length,
   - If there are words unplaced in game board,

- Pick the next unplaced word *w_unp* according to the sorting order
- Check if there is any placed word *w_pl_idx* in the other direction sharing the same index, if yes:
  - Place the word *w_unpl* starting from the head position of word *w_pl_idx,*
- else, start to check intersections in decreasing-length order with the previous placed words in the other direction (called reference words)
  - for a certain reference word *w_pl*, the intersection checking of word *w_pl* and word *w_unpl* can be conducted by a letter-wise examination from the end of the words to the beginning* (a double *for* loop involved).
- If the unplaced word *w_unp* cannot find its intersected word in other direction (either intersection common letter does not exist, or the intersection position is not available), it will be pushed to the end of the unplaced word array, waiting for another trial when there are more words in the board **.

* The example algorithm for intersection checking examines the possible intersection letter from the word end to word beginning, mainly to yield the priority of the FIRST letter of word *w_unpl* for possible intersection with other unplaced word in the future.

** Current strategy is aiming at making all words to overlap as much as possible. There are many other choices to deal with the case where word *w_unpl* fails to find a position ensuring intersection(s) with placed words. For example, word *w_unpl* can be temperately placed with certain cells away from the placed words, or try to look for other positions for the word placed in current intersection position which is also suitable for word *w_unpl* (intersection position for a certain word can be more than one).
Some randomness can even be introduced, e.g. in selecting word to place or in searching for placed words for intersection, to reduce the possibility of getting stuck in placing new words. Students are well welcome to look for other crossword entry-to-board algorithms.

5) Display the game board in console when needed. The display is slightly different from the game board storage. There are three possible statuses of a cell in a game board, empty (i.e. no letter involved), covered (i.e. not correctly guessed), and released (i.e. corrected guessed), and they should be displayed by space(s), underline(s), and the letter itself (possibly with spaces for ensure

the commonly shared cell length), respectively. The status of empty cells will not be modified at all, while only the covered cells will be updated to released cells in valid updating. Besides, some cells shared by two words as the FIRST letter, the word index remains after the first-time release. Please refer to "Assignment2.pdf" for more display details.

Additionally, the cell length in console display should be the same regardless of the cell status to provide the meaningful display. We don't want the letters in the Down words to shift in a zig-zag pattern. One possible solution is to store each row of the game board in an *ArrayList*, then call *Arrays.toString()* to generate the corresponding string, and don't forget to replace left square bracket ("["), comma (","), and right square bracket ("]") transformed from the ArrayList in the string with space (" ") for visual pleasure.

This whole part related to game board management handled by the server is relatively independent from the other functionalities. Moreover, 2.1.1-1) to 2.1.1-3) should be conducted simultaneously when the server waits or connects with other Players (if any) besides Player 1. Therefore, it is recommended to create a *Game* class extended from *Thread*, and put 2.1.1-1) to 2.1.1-3) in its *run()* method.

2.1.1-4) game board generation is more about organizing spatial positions of the word entries in a 2-D array. This step is one-time needed. Once the spatial structure is settled, it is the status of cells that need to be updated and displayed appropriately during game time (2.1.1-6).

2.1.2 General game status tracking and turns determination

Once the game starts, the server also needs to track the general game conditions, including

- The words that are not correctly guessed, i.e. available questions for next turn
- The number of correct guesses for each player, i.e. score counting
- Current turn and the evaluation of the current guess, and afterwards next turn determination

2.1.3 Communication with clients

Please carefully digest the related lectures and the corresponding example codes. Key lectures and codes: (reference the lecture slides: NetworkingCode.pdf, NetworkingCode-Multithreaded.pdf)

→ Networking with multiple clients:

- Networking with multiple clients: Use *ServerSocket()* to manage the networking. Use *Socket()* to manage the connection to a client.
- Create *ServerThread()* to manage the client-wise communication.

Server Networking Chat Room (multi-threading)

```
1 import java.io.IOException;
2 import java.net.ServerSocket;
3 import java.net.Socket;
4 import java.util.Vector;
5
6 public class ChatRoom {
7
8     private Vector<ServerThread> serverThreads;
9     public ChatRoom(int port) {
10          try {
11                System.out.println("Binding to port " + port);
12                ServerSocket ss = new ServerSocket(port);
13                System.out.println("Bound to port " + port);
14                serverThreads = new Vector<ServerThread>();
15                while(true) {
16                      Socket s = ss.accept(); // blocking
17                      System.out.println("Connection from: " + s.getInetAddress());
18                      ServerThread st = new ServerThread (s, this);
19                      serverThreads.add(st);
20                }
21          } catch (IOException ioe) {
22                System.out.println("ioe in ChatRoom constructor: " + ioe.getMessage());
23          }
24    }
25    public void broadcast(String message, ServerThread st) {
26          if (message != null) {
27                System.out.println(message);
28                for(ServerThread threads : serverThreads) {
29                      if (st != threads) {
30                            threads.sendMessage(message);
31                      }
32                }
33          }
34    }
35    public static void main(String [] args) {
36          ChatRoom cr = new ChatRoom(6789);
37    }
38 }
```

Server Networking Server Thread (multi-threading)

```
1 import java.io.BufferedReader;
2 import java.io.IOException;
3 import java.io.InputStreamReader;
4 import java.io.PrintWriter;
5 import java.net.Socket;
6
7 public class ServerThread extends Thread {
8
9     private PrintWriter pw;
10    private BufferedReader br;
11    private ChatRoom cr;
12    public ServerThread(Socket s, ChatRoom cr) {
13          try {
14                this.cr = cr;
15                pw = new PrintWriter(s.getOutputStream());
16                br = new BufferedReader(new InputStreamReader(s.getInputStream()));
17                this.start();
18          } catch (IOException ioe) {
19                System.out.println("ioe in ServerThread constructor: " + ioe.getMessage());
20          }
21    }
22
23    public void sendMessage(String message) {
24          pw.println(message);
25          pw.flush();
26    }
27
28    public void run() {
29          try {
30            while(true) {
31                String line = br.readLine();
32                      cr.broadcast(line, this);
33                }
34          } catch (IOException ioe) {
35                System.out.println("ioe in
ServerThread.run(): " + ioe.getMessage());
36          }
37 }
38 }
```

2.1.4 Synchronization control

1) *ServerThread.run()* method should be adaptive to the player activation status, as the waiting player(s) can only receive the sync messages after the active player finishes the interaction with the server. Besides, the waiting behavior can be managed by a lock.
2) The number of players that can simultaneously connected to the server can be controlled by *Semaphores()* in server for *ServerThread()* instances.

2.1.5 Termination

Once starts, the server will not stop unless be manually terminated. This feature can be achieved by an infinite *while* loop.

2.2 Client Class

2.2.1 Client thread communicate with the server

Client Networking Example (no multi-threading)

```
1   import java.io.*; // to save space
2   import java.net.Socket;
3   public class NetworkingClient {
4     public NetworkingClient() {
5       Socket s = null;
6       BufferedReader br = null;
7       PrintWriter pw = null;
8       try {
9         System.out.println("Starting Client");
10        s = new Socket("localhost", 6789);
11        br = new BufferedReader(new InputStreamReader(s.getInputStream()));
12        pw = new PrintWriter(s.getOutputStream());
13        String str = "Line being sent";
14        System.out.println("Sending: " + str);
15        pw.println(str);
16        pw.flush();
17        String line = br.readLine();
18        System.out.println("Line Received: " + line);
19      } catch (IOException ioe) {
20        System.out.println("IOE: " + ioe.getMessage());
21      } finally {
22        try {
23          if (pw != null)
24            pw.close();
25          if (br != null)
26            br.close();
27          if (s != null)
28            s.close();
29        } catch (IOException ioe) {
30          System.out.println("ioe: " + ioe.getMessage());
31        }
32      } // ends finally
33    } // ends NetworkingClient()
34    public static void main(String [] args) {
35      new NetworkingClient();
36    }
37  }
```

2.2.2 Termination

Once the game concludes and the clients received the results, the clients will automatically terminate. "*System.exit(0);*" can be used for program termination.

2.2.3 Synchronization (optional)

Synchronization in the clients is generally not needed. The 2 key points in the rubric on synchronization are on the common synchronization control for server and clients. Please have your own synchronization control design based on need. If your implementation works correctly without client synchronization, it will receive full points.