Randy Oram (roram@ufl.edu)
CNT4007C
Project 1: Quiz Server & Quiz Client
February 18th, 2019

Note: This project (and this report) are adaptations from my project 1 submission.

Application Description

Quiz Server is an application that allows users to create and maintain a database of questions on a remote machine. The questions can be accessed on their own (as raw data) or they can be added to contests by the admin (contestmeister). Contests allow for multiple users (contestants) to connect and answer questions, obtaining real-time feedback as the quiz progresses. The contestmeister can also review details for particular contests.

This application utilizes a client-server architecture to form a logical link between various machines – one server (acting as the middleman), one contestmeister (acting as the administrator), and one or more contestants (users who can participate in contests. As such, the code powering the application is separated into those three major functionalities (server, contestmeister, and contestant).

The server uses a socket to listen for incoming TCP connections from a contestmeister. When it receives one, it passes that connection off to a second TCP connection within its own thread. Through this socket, the server can receive messages containing commands and, if needed, data.

The contestmeister can choose to begin a contest. In this case, the server creates a contest object thread. This thread opens a new TCP socket to listen for incoming connections from contestants. When it receives one, it passes that connection off to a separate TCP socket. This socket, along with unique in and out streams, are stored within the contest object.

The contestants can then respond to the questions presented by the server and receive real-time feedback. After the contest is completed, the contestant objects disconnect from the server, print out results to the contestant, and then self-terminate.

The results of the contestant are stored persistently on the server-side. The contestmeister can access these details through its interface with the server.

Persistent Storage

Question Bank

Persistent storage of questions was achieved by using a JSON data structure. The structure of each question is minimal, containing only these five fields:

Field	Description
Tags	Descriptor tags, used to define the question's type. Stored as comma-separated strings.
Question	The actual question to be answered. Stored as a string.
Answers	The answers to the question. Stored as a JSON array object containing strings. Format for a single entry: "(a) <answer text>".
Correct	The correct answer (as a letter) to the question. Stored as a string.
qNum	The question number, as created by the server. Stored as an integer.

These question objects are stored in a JSON array, which is tied to an overall "questions" JSON object.

The JSON file (with title "QuestionBank.json") must be present when running the server. This file, at a minimum, must contain the following:

{"questions":[]}

An example of a suitable JSON file containing three questions follows:

{"questions":[{"qNum":1,"question":"Who are you?","correct":"b","answers":["(a) I am nothing","(b) I am","(c) I am not you","(d) I am you"],"tags":"philosophy"},{"qNum":2,"question":"FITB: Make Trance _____ Again","correct":"c","answers":["(a) Stellar","(b) Great","(c) Uplifting","(d) Popular"],"tags":"Trance, EDM"},{"qNum":3,"question":"Where is the best place to study?","correct":"c","answers":["(a) Volta","(b) Grog","(c) Mai Kai"],"tags":"Stores, Gainesville"}]}

Contest Bank

Persistent storage of contests was achieved by using a JSON data structure. The structure contains the following fields:

Field	Description
qNums	List of question numbers associated with this contest.
qAvgs	Average correct for each question in particular.
hasRun	Boolean indicating whether the contest has been run.
maxScore	The maximum score a contestant has achieved on this quiz.
cNum	The contest number associated with this contest.
timesRun	The number of times this contest has been run.
numParticipants	The number of contestants who have taken this quiz.

These contest objects are stored in a JSON array, which is tied to an overall “contests” JSON object.

The JSON file (with title “QuestionBank.json”) must be present when running the server. This file, at a minimum, must contain the following:

{“contests”:[]}

An example of a suitable JSON file containing three questions follows:


```
{"contests”:[ {
                "qNums": [1],
                "avgScore": null,
                "qAvgs": [100, 100, 100],
                "hasRun": false,
                "maxScore": null,
                "cNum": 2,
                "timesRun": 0,
                "numParticipants": 0
        }]}
```

Object Specifications

Client (i.e. contestmeister)

Members

Name	Type	Function
clientSocket	Socket	The TCP socket object; used as a logical link between the client and server.
in	BufferedReader	Uses an InputStreamReader (tied to the clientSocket object) to obtain server input from the socket.
out	PrintWriter	Uses an OutputStreamReader (tied to the clientSocket object) to write output to the server using the socket.
stdIn	BufferedReader	Obtains command line input from the end-user.

Methods

Function	Description
Constructor	Instantiates the client object by creating the client socket and then calling open interface.
openInterface	Instantiates the input and output channels (in, out, and stdIn), then calls commandRouter. This function was used to add another layer of abstraction.
commandRouter	Accepts commands from the standard input (command line). A switch statement is used to handle each command (either by sending the command to the server, calling a client-based function, or both).
putQuestion	Handles the client-side of adding a question.
getQuestion	Handles the client-side of acquiring a question. A special Boolean argument determines how to handle the difference between a get question and a random question command.
reviewContest	Handles the client-side of receiving details about a particular contest.
listContests	Handles the client-side of receiving details about all particular contests.

help	Prints help text to the console.
closeClient	Closes all members and then shuts the program down.

Server (i.e. cserver)

Members

Name	Type	Function
server	ServerSocket	The TCP server socket that listens for incoming connections.
serverSocket	Socket	Creates another socket (tied to the server member) that is used for communication with the current client.
in	BufferedReader	Uses an InputStreamReader (tied to the serverSocket object) to obtain client input from the socket.
out	PrintWriter	Uses an OutputStreamReader (tied to the serverSocket object) to write output to the client using the socket.
questionBank	JSONArray	The current question bank (initially obtained from the file; updated dynamically as the client interacts with the server)
contestBank	JSONArray	The current contest bank (initially obtained from the file; updated dynamically as the client interacts with the server)
on	Boolean	Maintains state of server (whether the server is currently on or off). This is used to allow serial client connections.
(c/q)BankFilepath	String	Contains the filepath of question bank and contest bank JSON files.

Methods

Function	Description
Constructor	Instantiates the server object. Populates the JSONArray with the file contents. Sets *on* to true.

acceptCommands	The server's command router. This function utilizes a switch statement to route incoming commands to the proper helper functions.
acceptNewQuestion	Creates a new JSONObject and populates it with incoming data from the client.
deleteQuestion	Deletes a question from the questionBank (using qNum).
dumpQuestion	Obtains a specific question from the question bank (by qNum) and returns it to the client.
getRandomQuestion	Obtains a random question from the question bank and returns it to the client.
checkAnswer	Checks a given question (determined by qNum) to see if it's correct answer is the same as what the client provided.
reviewContest	Provides contest details for a specific contest to the contestmeister.
listContests	Lists contests and minor details about the contest to the contestmeister.
beginContest	Begins a contest specified by a contest number.
addToContest	Adds a specific question to the contest.
createContest	Creates a contest with a specific contest number.
questionExists	Checks the JSONArray to see if a question exists (this is accomplished via checking qNums).
closeServer	Closes all members and shuts the server down by setting the *on* Boolean to false.
updateQuestionBank	Will write the contents of JSONArray to the local JSON file.
updateQNums	Deprecated function (it is unused in the code). If used, it will run through the question bank, setting the question numbers to 1, 2, 3, etc.

Contestant

Members

Name	Type	Function
clientSocket	Socket	The TCP socket object; used as a logical link between the client and server.
in	BufferedReader	Uses an InputStreamReader (tied to the clientSocket object) to obtain server input from the socket.
out	PrintWriter	Uses an OutputStreamReader (tied to the clientSocket object) to write output to the server using the socket.
stdIn	BufferedReader	Obtains command line input from the end-user.
nickname	String	The name associated with this contestant.

Methods

Function	Description
Constructor	Instantiates the contestant object.
OpenInterface	Populates readers and writers, calls sendIntro, and opens the command interface.
sendIntro	Communicates with the server to send a nickname and ensure it is acceptable.
ContestInterface	Handles the receiving of questions, sending of answers, and retrieval of statistics after sending an answer.

Protocol Specifications

Command & Message Definitions

Type	Description
p	Command: p Notifies the server that the client will be sending a new question. The client will then send the question in a specific format. The server will add this data to the question bank, append a question number, and return this question number to the client to indicate success. The client must format the question input as follows: Question tag 1, question tag 2, question tag n Question text . (a) First choice . (b) Second choice . (…) Nth choice . . correctAnswerAsLetter The server then returns the question number it assigned to this question.
d	Command: d qNum Notifies the server that the client wants to delete a question (by sending the server a question number). The server will then handle this task, deleting the question (and updating the question bank). The server then explicitly indicates success to the client as follows: Deleted question qNum
g	Command: g qNum Requests a question from the server by qNum. The server will obtain this question and provide it to the client in the following format:

	Question tag 1, question tag 2, question tag n Question text . (a) First choice . (b) Second choice . (…) Nth choice . . correctAnswerAsLetter qNum
r	Command: r cNum Requests details about contest with contest number cNum from the server. The results are returned to the contestmeister in the following format: cNum 1 question, run, average correct: 0.5; maximum correct: 1 qNum 50% correct
s	Command: s cNum Asks the server to create a new contest with contest number cNum.
a	Command: a cNum qNum Asks the server to add question with question number qNum to contest with contest number cNum.
b	Command: b cNum Asks the server to begin contest with contest number cNum. The server will then spawn a new thread for this contest and allow contestants to connect to the contest. The thread will complete the contest and update the contestants with feedback. After the contest completes, the contest thread will save the results to the persistent contest bank before terminating.
l	Command: l Asks the server to list details about all contests. The server will provide a response in the following format: cNum numQuestions, run, average correct: 0.5; maximum correct: 1

	cNum 0, not run
k	Command: k Asks the server to terminate. The server will safely close its data members, update the JSON file, and then terminate.
q	Command: q This command does not interact with the server. Quits the client. The server will remain open, allowing other clients to connect to it.
h	Command: h This command does not interact with the server. Prints help text for each command.

Error Handling

Socket and IO Error Checking

Try-catch statements were used with all IO sections of the code (i.e. when the client and server are communicating, or when the client and server are initializing or closing their sockets).

Null Checks

Both the client and server do simple error checking to ensure incoming messages (either from the other party, or, in the client's case, from standard input) are not null.

Regular Expressions

The client program uses regular expressions to verify user commands. This prevents the user from typing commands incorrectly. If they do, it notifies them that they tried to use an invalid command, and to type "h" for help.

Command Parsing

Both the client and server properly parse incoming commands using substrings to obtain the command itself (i.e. p, g, d, etc.), and (if applicable), a qNum or answer choice.