CNT4007C
Project 2
Due: 4/18/2019
Documentation

Michael Gordon

Key Classes

1. Request – Abstract class implements serializable. Holds **String** requestType and **boolean** res
   a. **ReviewRequest**- Object sent to server for > r command from contestmeister client. Holds **int** contestNum
   b. **ListRequest**- Object sent to server for > l command from contestmeister client. No additional data
   c. **AppendRequest**- Object sent to server for > a command from contestmeister. Holds **int** contestNum and **int** questionNum
   d. **BeginRequest**- Object sent to server for > b command from contestmeister client . Holds **int** contestNum
   e. **SetRequest**- Object sent to server for > s command from contestmeister client . Holds **int** contestNum
   f. **DeleteRequest** - Object sent to server for > d command from contestmeister client. Holds **int** questionNum
   g. **GetRequest** Object sent to server for > g and command from contestmeister client. Holds **int** questionNum
   h. **KillRequest** – Object sent to server to request termination from contestmeister client. Holds no additional data
   i. **PutRequest** - Object sent to server to upload a question to the bank from contestmeister client. Holds **Question** object
   j. **NicknameRequest** - Object sent to server when requesting a nickname from constent client. Holds **String** name
   k. **SendAnswer** – Object sent to server from Contestant client to respond to a question. Holds **char** answer.
2. Response - Abstract class implements serializable. Holds **String** responseType and **boolean** res
   l. **BadResponse** - Object sent to contestmeister client from server to indicate something wrong happened
   m. **ReviewResponse** - Object sent to contestmeister client for > r command response. Holds **String** review
   n. **ListResponse** - Object sent to contestmeister client for > l command response. Holds **String** list
   o. **AppendResponse** – Object sent to contestmeister client for > a command response. Holds **String** resText
   p. **SetResponse** – Object sent to contestmeister client for > a command response. Holds no additional data (uses **boolean** res to indicate success or failure)
   q. **BeginResponse** – Object sent to contestmeister client for > b command response. Holds **int** contestPort and **int** contestNum
   r. **DeleteResponse** – Object sent to contestmeister client for > d command response. Holds no additional data
   s. **GetResponse** - Object sent to contestmeister client for > g and >r command response. Holds **Question** q object

t. **KillResponse** – Object sent to contestmeister client to acknowledge kill request. Holds no additional data
u. **PutResponse** – Object sent to client for > p command response. Contains **int** questionNum
v. **NicknameResponse** – Object reply sent to contestant client. **boolean** res indicates success or failure when requesting a name. No additional data.
w. **SendQuestion** – Object sent to contestant client from server contains **Question** q
x. **AnswerResponse** – Object sent to contestant client from server. Contains **String** resText
y. **ContestOver** – Object sent to contestant client from server. Indicates the end of the contest holds no additional data

3. **Question** - Implements Serializable. Holds **String** questionTag, question; **int** questionNum,; **char** correctAnswer; **ArrayList< String >** choices; Used to represent questions.
4. **Contest** - Implements Serializable. Holds **int** contestNum, **char** correctAnswer; **ArrayList< Integer >** questionNums; HashMap<Integer, QuestionStat> questionStats**; boolean** run; **int** maxCorrect; **double** averageCorrect; **int** attempts; **int** correct; Used to represent Contests
5. **QuestionStat** - implements Serializable. Holds **int** questionNum, attempts, correct; Used to hold data for specific questions in a contest.
6. **RunningContest** – implements Runnable. Used to run a thread to run a contest initiated by the begin command
7. **ContestmeisterThread** – implements Runnable. Used to run a thread to manage interactions with a contestmeister client. (Note, it is not actually a Thread, it does not **extend** Thread it **implements** Runnable)
8. **ContestantThread** – implements Runnable. Used to run a thread to manage interactions with a contest client. (Note, it is not actually a Thread, it does not **extend** Thread it **implements** Runnable)
9. ContestantClient, ContestmeisterClient, and Server – These classes contain the source code for running the server and client.

Key Functions:

1. From ServerSocket
   a. **new ServerSocket(0)** – creates a **ServerSocket** open on the host, with an available port chosen by Java.
   b. **accept()** – Listens for an attempted connection. This blocks the thread until a connection is made.
   c. **close()** – terminates the ServerSocket.
2. From Socket
   a. **getOutputStream()** – obtains an output stream that the client/server can write to.
   b. **getInputStream()** – obtains an input stream that the client/server can read from.
   c. **close()** – terminates the Socket.
3. From ObjectOutPutStream and ObjectInputStream
   a. ObjectOutputStream os = new ObjectOutputStream(socket.getOutputStream()) - Obtains output stream from socket that the server/client can write to.
   b. ObjectInputStream is = new ObjectInputStream(socket.getInputStream()); - Obtains an input stream from the socket that the server/client can write to.
   c. **os.writeObject(object)** – Writes object to the socket. Can be read on the other end

d. **is.readObject();** - Reads the object from the socket.

Server-Client Protocol:

The server would start up and listen for connections using a TCP/IP protocol. The server would block on a thread until a connection was created. Once the connection was established the server would wait for a request from the client and then respond in appropriate fashion. At this phase only **Contestmeister** clients can correctly communicate with the server. A thread is made for each **Contestmeister** that connects.

Requests would be done by sending **Request** objects through the **ObjectOutputStream** obtained from the client's socket. The server would read them it's **ObjectInputStream** obtained from the socket created at the start of the connection. For more information about the Request Object see Key Classes.

Response would be done by sending Response objects through the **ObjectOutputStream** obtained from the socket created at the start of the connection. The client would read them it's **ObjectInputStream** obtained from the socket created at the start of the connection. For more information about the Responses Object see Key Classes.

When a Contestmeister begins a contest, a new thread using a **RunningContest** as a runnable begins. This thread waits for one minute and accepts connections on a new port from Contestants. As connections are accepted, new threads are made for the Contestants. As soon as a Contestant's connection is accepted, a nickname is requested via a **NicknameResponse** object. The client replies with a **NicknameRequest** object. If the nickname is available, a **NicknameResponse** is sent with the **boolean** res set to true. If it is not available res is set to false, and the server waits for a **NicknameRequest** again indicating to the client it must send another request. This continues until the client obtains an available nickname. Once an available nickname has been successfully chosen by the client, the client waits for the next message from the server. When one minute has passed from the contest accepting connections and all clients have successfully selected a nickname, the first question can be sent to each client via **SendQuestion** object. This contains a Question object without the correct answer (stored as char '*'). The client prompts the user for a char answer. When entered the client responses with a **SendAnswer** object. Once all answers are received, the server thread sends an **AnswerResponse**, to the client. This cycle continues until all questions are asked. When this happens all clients are sent a **ContestOver** object, then the connections and threads are tore-down.

CNT4007C Michael Gordon
Project 2
Due: 4/18/2019

Question Bank:

The questions for *cserver* are stored in the *qbank*. The *qbank* is just a subdirectory that contains text files that are read as Java Objects.

The questionlist is an **ArrayList<Integer>** object that has the list of question numbers. Each Question object is stored in its own file

Files:

\questions\questionlist.txt

This is created by the server, stores the questions available to the server. Is an ArrayList of Integers.

.TP

\questions\question#.txt

1. The question # is stored in this file as a Question object. **ContestmeisterThread** – implements Runnable. Used to run a thread to manage interactions with a contestmeister client. (Note, it is not actually a Thread, it does not **extend** Thread it **implements** Runnable)

Contest Bank:

The questions for *cserver* are stored in the *cbank*. The *cbank* is just a subdirectory that contains text files that are read as Java Objects.

The questionlist is an **ArrayList<Integer>** object that has the list of question numbers. Each Question object is stored in its own file

Files:

\contests\contestist.txt

This is created by the server, stores the questions available to the server. Is an ArrayList of Integers.

\contests\contest#.txt

The contest # is stored in this file as a Contest object.

Some Error Handling:

If the client sends an object that is not an instance of a Request, then the server will always respond with a **BadResponse** object that contains an error message. The **responseType** obtained by getResponseType() will equal true in the case that it is returned by the server. **BadResponse** objects are also used on the client side, but uses false to indicate it was returned by the client.

CNT4007C Michael Gordon
Project 2
Due: 4/18/2019
Requests for questions that do not exist are checked by the server. In these cases the **getResponseType()** in the Response object returns false indicating failure.

"BadContestants" are removed from a Contest if the send a initial request that is not for a nickname. Contestmeisters are not allowed to append a question to a contest currently running.

How to Use:

1. Enter make into terminal
2. Start server with ./cserver
3. Start contestmeister client with ./contestmeisterclient hostname port [file]
4. Start contestant client with ./contestantclient hostname port