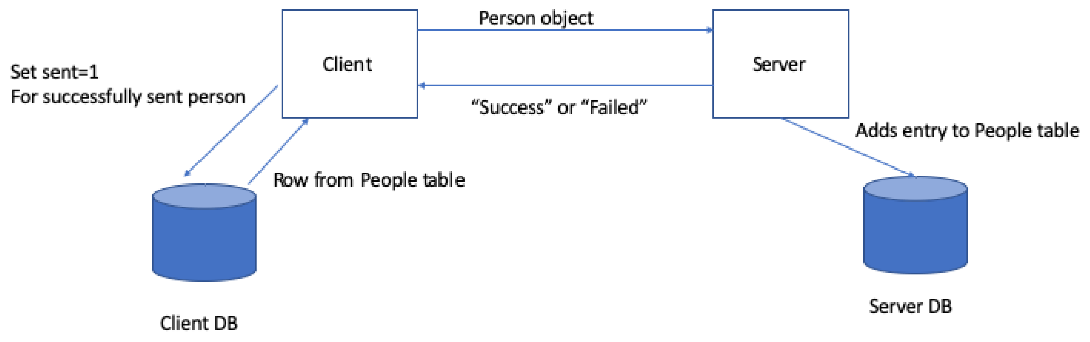
CS9053
Final Project
Dr. Dean Christakos
Due: May 19, 2020

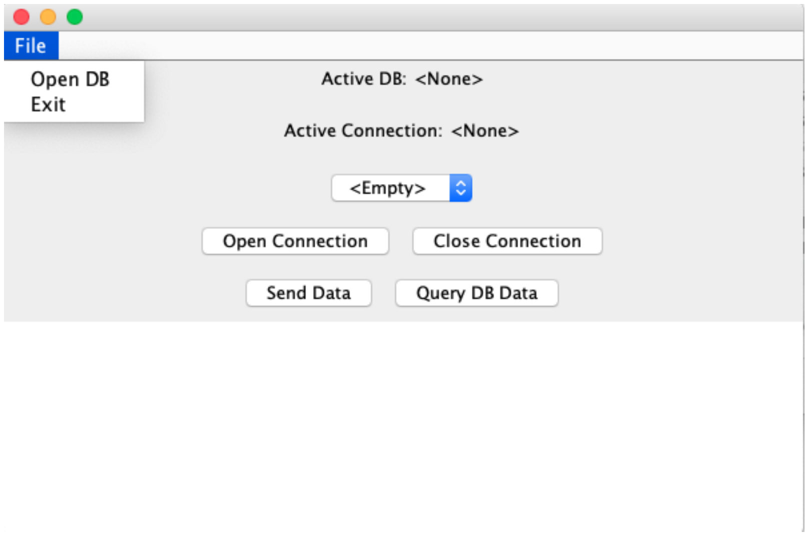
The project is to build a client/server system that reads data from a DB into an object and sends the object to the server. The server then writes that data into its DB.



Here is an overall structure of how it works. What follows will be a specification, starting with the UI.

Client UI

Remember that you **do not** need to duplicate the UI **exactly**. Functional implementation is fine. Layouts can vary according to your preferences and what is easiest for you.



The Client UI has a menu that allows you to Exit and open a database, presumably the client.db database. The JFileChooser is sufficient for selecting it, seen in the OpenDBListener inner class in ClientInterface.java

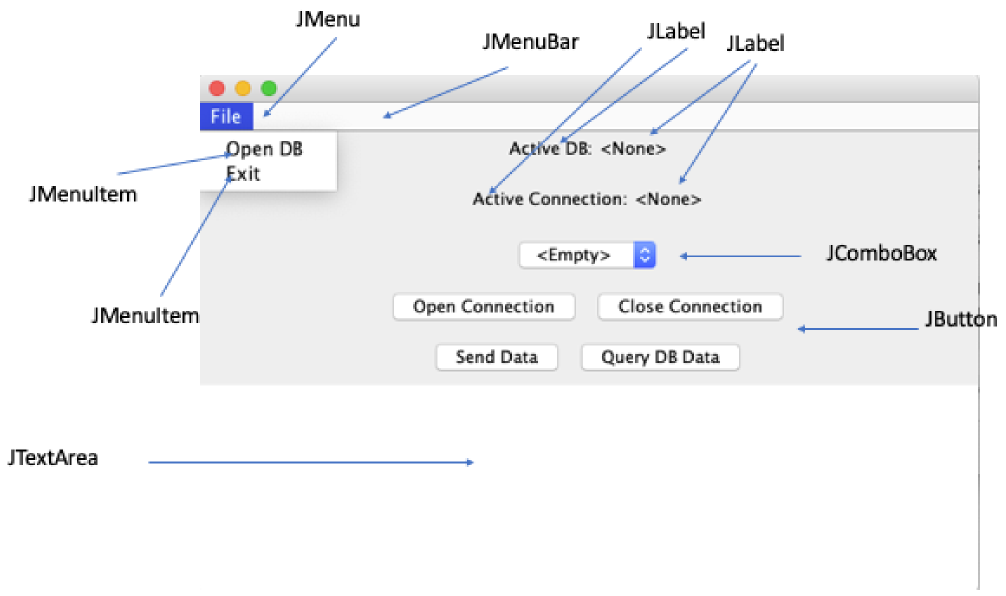
The dropdown box is empty (with one entry, "Empty") when not connected to the database. When it is connection to the database, it should be populated with all names whose value of "sent" in the People table is "0" (or false. Sqlite does not support Boolean types explicitly).

The "open connection" button should open a connection to the server. The Server location should be "localhost" and the port should default to "8001." The "Close Connection" button should close the connection. You should be able to Open the connection, send data, close the connection, re-open the connection, and send data again without any errors.

"Send Data" should get the entry in the people table that corresponds to what is selected in the dropdown box, create a Person object, and sent the person object to the server. If the client is not connected to the server, then you may raise an error, but the client should not terminate or otherwise fail to continue working.

"Query DB Data" should show the current Contents of the People table in the text area, with Row names (see the video).

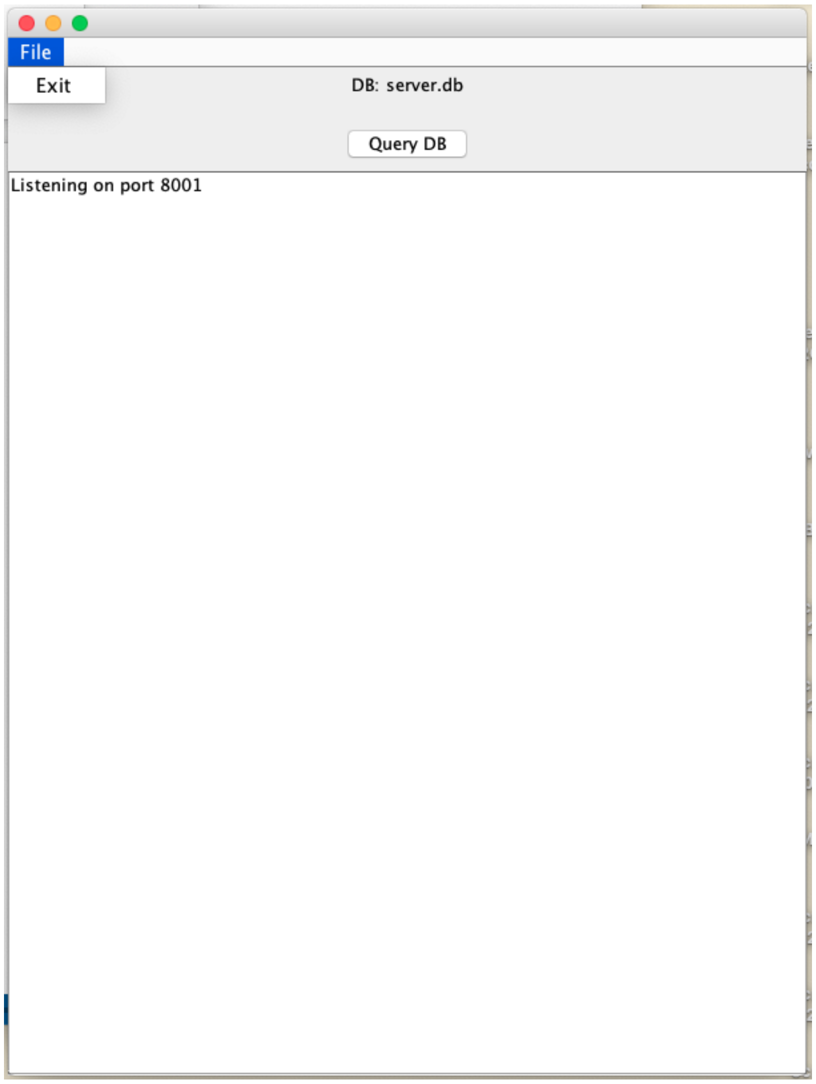
By way of some hints, here are the Java swing objects in the client:


JMenu
JMenuBar
JLabel
JLabel
File
Open DB
Exit
Active DB: <None>
Active Connection: <None>
JMenuItem
<Empty>
JComboBox
JMenuItem
Open Connection
Close Connection
JButton
Send Data
Query DB Data
JTextArea

Server UI

The server is a little simpler. The server should connect to server.db on startup and indicate such. It should be listening on port 8001. There should be a File menu with an Exit menu item that quits the application.

The “Query DB” button should show the contents of the People table. Any messages you want (status messages, indications that data has been received, success status, etc.) can appear in the JTextArea, and be appended over time. You probably want to use a JScrollPane for this so that you can scroll through the messages.



```
Listening on port 8001
```

Client Network Connections

A client should open a connection to the server on port 8001. This was demonstrated in the lecture and the code from the last lecture. Feel free to experiment with an use any of that code.

The connection should be persistent but can be closed and reopened.

The client should send over an object to the server and then wait for a reply from the server.

The server will reply either “Success” or “Failed.” In the SendButtonListener inner class in ClientInterface.java, there are a few lines of code that create the BufferedReader from a socket and wait for a response, expected after you write the close to send off the object.

If it receives “success”, it should update the People table to set sent=1 for the Person who was just sent to the Server.

Server Network Connections

The server will run a thread that listens for connections. When it receives a connection from a client, it should spawn a thread to handle that connection from the client. **To get full credit the listener thread must spawn a “Handler” thread that listens for objects and sends replies**.

Just as a reminder from the lecture, the line `Socket socket = serverSocket.accept()`. That socket can be used to read and write ObjectInputStream and DataOutputStream objects. The thread loop in the “Handler” thread will continually wait on the ObjectInputStream for an incoming object.

If there are IO errors, in the Streams or the Sockets, assume there is some kind of network problem and exit the thread by having a using a “break” inside the while loop.

Any errors should also result in sending “Failed” back to the Client. And the client expects a newline, so it will have to be “Failed\n”.

After receiving the Person object, the Server should insert the data from that object into the People table of the Server DB. And then after it success, it should sent “Success\n” back to the Client.

Miscellaneous, Hints, and Notes

This is hard, and partial credit will be generous.

Break this into pieces:

Client:

The client gets all of the rows of the People table where sent = 0 and puts the resulting names in the JComboBox. See the class ComboBoxItem for how to associate Names with Ids in a combo box.

The client connects to the server using `new Socket(host, port)`

You get `InputStream` and `OutputStream` objects by executing `socket.getInputStream` and `socket.getOutputStream`

The client gets an individual row from the People table according to Id or Name, based on the current selected item in the JComboBox. With the data in that row, create a Person object.

Send the Person object to the server over an `ObjectOutputStream`.

Wait for a String reply with BufferedReader.

If success, update the People table in the DB to indicate that the Person you created was "sent". This would be a good time to rebuild the JComboBox contents with the latest results in the DB

Server:

The server connects to its DB, server.db, on startup and waits for connections on port 8001.

Pressing "query DB" should just do a "SELECT * FROM People" and display the results, including the column names from the ResultSetMetaData. Display the results in the text area.

The structure for a multithreaded server is in the MultiThreadServer.java file from the lecture.

In the "Handler" thread, pass in the socket that you received when the client made a connection. From the socket, get the InputStream and OutputStream objects. Once you've done that, then start the while loop in the Handler thread that listens for data, receives the data, and sends a reply back.

Maximize Your Partial Credit and Chances of Success

Build the Server first. It has few UI components. Connect to the DB on startup. Create a basic UI, and create an ActionListener for the QueryDB button that displays the contents of the People table in the TextArea (it will be empty, of course, but you can show the column names, or you can experiment by connecting to the client.db file).

Next, create a listener thread that waits for a socket. Start up the server and make sure it listens.

When it comes to the client, your strategy might be to either build the entire UI first and write the code for the buttons later or it might be to implement each button/component one at a time and add them as you go.

Before you deal with any network connections, put all the database code in place. You need a JMenuItem that connects to the DB. There is an inner class OpenDBListener that handles this, you just have to make a database connection out of that.

Next is implementing the "Query DB Data" button. This should send a "SELECT * FROM People" query to the DB and get the results back. You should ultimately display the results in the JTextArea component, but you might want to worry about getting the DB part, first.

Next, go back and figure out how to populate the JComboBox right after to connect to the client.db database. This should get the entries where sent = 0 and put their names in the JComboBox. I have two methods in ClientInterface—fillComboBox and getNames – that should make it clear how to do that. The order of events should be Connect to Db -> Fill in comboBox. In getNames you will have to query the People table, read in the results, and create a list of ComboBoxItem objects.

Next, write the code for "Open Connection" button and see if the server can acknowledge your connection. Now you're on your way!

<u>More Hints</u>

The code you need is generally all found in the Thread, JDBC, GUI, and Networking lectures.

There are no concurrency issues you have to worry about.

I've gotten a few fairly consistent questions about the final project, so I want to address them all here:

How do we handle misuse cases/errors?

The errors should be handled in such a manner that the program continues operating.

If there’s an error in the DB on the server side, it should send a “Failed” message to the client without terminating the connection.

If there’s an error in the network connection, the connection should terminate and the thread that serves that connection should terminate, but the client should be able to reconnect to the server and continue as normal.

If the client cannot connect at all, the client should still operate and at least be able to try again. You don’t need any particular GUI-based error messages, though. That’s nice but not a requirement.

If there are multiple clients, how do we handle any SQLite errors of multiple clients trying to read or write to the client.db?

SQLite isn't helpful when it comes to simultaneous access to a SQLite file. That issue isn't part of the scope of the project. What is important and necessary is that your server should be able to handle multiple simultaneous client connections in their own threads, and those threads should terminate when the connection is terminated.

What about if the client is using the same DB as the server?

You should not be doing that. The client(s) should be using client.db and the server should be using server.db