CST 8221 – JAP - Assignment #2, Part 2

Due Date: prior or on 4 January 2018

Earnings: 7% of your total course mark (plus up to 3% bonus)

Purpose: Writing the Client and Server code
The purpose of Assignment 2 is to build a simple socket based client/server application. In Part 1 of the assignment you have built the client GUI. In Part 2 of the assignment you are to write the client and the server code. The server will be a TCP/IP multithreaded console based application. The application must behave exactly as specified below and shown in the **ClientServerScreenCaptures_F17 .pdf.**

Requirements Specification:

Server Implementation

The server implementation must satisfy the following requirements and must behave as shown in **ClientServerScreenCaptures_F17 .pdf**.

Requirements:

The server implementation consists of two separate classes: **Server** and **ServerSocketRunnable**.

Class Server

The ***Server*** class is responsible for creating a server socket and starting a service thread responsible for serving each individual client.

The ***main()*** method should perform the following tasks:

- If command line string is supplied at launch, the method converts the string to an integer port number. Otherwise, the server must use 65535 as a default port number.
- The method creates a TCP/IP server socket bound to the specified port.
- In an endless loop the method calls ***accept()*** on the server socket instance. Once a connection with the client is established, the *accept()* method will return a Socket instance. The main() method prints an appropriate message on the console screen and proceeds (see **ClientServerScreenCaptures_F17 .pdf**).
- In the same loop the main() method creates an instance of *ServerSocketRunnable* class passing the socket instance to its constructor.
- Finally, in the same loop the main method uses the *ServerSocketRunnable* object to create a ***Thread*** instance and to start a service thread.

The server does not have a GUI. It must started at the command prompt. In order to stop the server the user must terminate the Java Virtual Machine (JVM).

Class ServerSocketRunnable

The ***ServerSocketRunnable*** class is responsible for communicating with the client and responding to the command strings sent by the client. The class must implement the Runnable interface. The class must have a single constructor taking a reference to a Socket object as a parameter.

The server request line (string) sent by the client must have the following syntax:

```
-COMMAND-optional command string
```

In this implementation, the only supported COMMANDs are: `quit`, `echo`, `time`, `date`, `help` and `clrs`**.** The commands are case sensitive. Any command can have an optional string. Allcommand must start with a dush `-` (minus sign). If the command conatins an optional string the command must end with a dash `-`**.**

Command examples (see **ClientServerScreenCaptures_F17.pdf**):

```
-time
-echo- Testing connection
-help-asking for help
```

The ***run()*** method should perform the following tasks:

- It opens input and output streams.
  Note: I have used ObjectOutputStream and ObjectInputStream in my implementation, but you are free to use any stream of your choosing.
- In a loop that repeats until the `quit` command is sent by the client, the run() method first reads the command string sent by the client.
- The method extracts the COMMAND from the command string and depending on the COMMAND it writes back to the client.
  (See **ClientServerScreenCaptures_F17.pdf**).
- If the COMMAND is `echo`, the method writes back the following string:

  ```
  ECHO:  optional command parameter string
  ```

- If the COMMAND is `time`, the method writes back the following string:

  ```
   TIME: current hour:current minutes:current seconds AM or PM
  ```

- If the COMMAND is `date`, the method writes back the following string:

  ```
  DATE: day number month name year number
  ```

- If the COMMAND is `help`, the method writes back the following string:

  ```
  Available Services:
  quit
  echo
  time
  date
  help
  clrs
  ```

- If the COMMAND is `clrs`, the method writes back the string `clrs`. If the Client receives the `clrs` string from the server, it must clear the terminal screen.
- If the COMMAND is `quit`, the method exits the loop, prints a message, notifies the client, closes the streams and the client socket, and terminates
- At the end of each of the loop iterations, the tread sleeps for 100 milliseconds.

The ***run()*** method **must** handle all possible exceptions and display an appropriate message on the console screen. The Server and the ***run()*** method **must not crash**.

Client Implementation

In this part of the assignment you must implement the event handling part the client GUI application. Before writhing the code, read the requirements and see the Client screen captures in **ClientServerScreenCaptures_F17.pdf.**

The server implementation must satisfy the following requirements and must behave as shown in **ClientServerScreenCaptures_F17.pdf**.

Requirements:

- All messages generated by the client must be displayed in the display area of the GUI. No messages must be displayed on the Console screen.
- At launch, the Connect button must be red and enabled, and the Send button must be disabled. The display area must be empty.
- When the user clicks the Connect button, the client should try to connect to the server using the specified host and port number. To avoid making your GUI irresponsive **a timeout socket connection** must be used.
- If the connection is successful, input and output streams should be open, an appropriate message should be displayed **(ClientServerScreenCaptures_F17 .pdf**), the Connect button should be disabled and changed to blue in color, and the Send button should be enabled.
- If for some reason the connection to the server is unavailable or becomes unavailable (for example, the server receives the `quit` command), an appropriate message should be displayed (**ClientServerScreenCaptures_F17 .pdf**), the Connect button should be enabled, and the Send button should be disabled.
- When the user types a command line string (see above the expected syntax of the command string) and clicks the Send button, the command line string must be sent to the server, and the server response string must be displayed (**ClientServerScreenCaptures_F17.pdf)** in the display area of the GUI.
- When the connection session is terminated, the Connect button should be enabled, and the Send button should be disabled.
- The ***event handler*** must handle all possible exceptions and display an appropriate message in the display area of the GUI. (**ClientServerScreenCaptures_F17.pdf)**. The Client must not crash and the event handler generate exceptions in the Console window.

Note: If you use only one event handler that implements the ActionListener interface, you should use getSelectedItem() to get the port number from the combo box.

Bonus Task 1 (1% of your course mark):

Use an appropriate thread pool and the Executor Framework to manage the server threads.
Reference: Textbook, Chapter 23, Section 26.3

Bonus Task 2 (2.0% of your course mark):

Use a thread to manage the client part of the connection. The implementation must comply with Rule #1 and Rule #2 outlined in HybridAct 11.
For full bonus mark you must use a *BlockingQueue* to send the client's command to the communication thread.
Reference: Textbook, Chapter 23, Section 23.11, HybridAct 11, and Lab 12.

Note: In order to have the bonus marks added to your assignment mark your application must work as specified and must not crash or generate uncaught exceptions. The bonus implementation must be documented in a text file named README_FIRST.txt

What to Submit:

Paper submission:

No paper submission is required for this assignment.

Code submission:

Compress in one **.zip** file all relevant to the assignment **.java** and **.class** files (plus the README_FIRST.txt for the bonus(es)). Upload the assignment **zip** file to Blackboard prior to or on the due date. The name of the zip file must have the following structure: Student's family name followed by the last three digits of the student ID number followed by _JAP_A2P2. For example, *Ranev_JAP_A2P2.zip*.

The submission must follow the course submission standards. The ***Assignment Submission Standard*** and the ***Assignment Marking Guide*** are posted on Blackboard.
.

Enjoy the assignment. And do not forget that:

"To serve a client you ought to have a server first."   Business Rule #2

CST8221 – JAP, 27 November 2017, S^R