```java
package cmet.ac.sockets.servers;

import java.io.IOException;

/**
 * Class represents a Server component.
 *
 * @author thanuja
 * @version 20.11.2019
 */
public class SimpleServer extends AbstractServerComponent implements Runnable {

    // reference variable for server socket.
    private ServerSocket            serverSocket;

    // reference variable for ClientHandler for the server.
    private ClientManager           clientHandler;

    // boolean flag to indicate the server stop.
    private boolean                 stopServer;

    // reference variabale for the Thread
    private Thread                  serverListenerThread;

    // reference variable for ThreadGroup when handling multiple clients
    private ThreadGroup             clientThreadGroup;

    // variable to store server's port number
    int port;

    /**
     * Constructor.
     *
     */
    public SimpleServer() {

        this.stopServer = false;

        /**
         * Initializes the ThreadGroup.
         * Use of a ThreadGroup is easier when handling multiple clients, although it is not a must.
         */
        this.clientThreadGroup = new ThreadGroup("ClientManager threads");

    }

    /**
     * Initializes the server. Takes port number, creates a new serversocket instance.
     * Starts the server's listening thread.
     * @param port
     * @throws IOException
     */
    public void initializeServer(int port) throws IOException {

        this.port = port;
        if (serverSocket == null) {
            serverSocket = new ServerSocket(port);
        }

        stopServer = false;
        serverListenerThread = new Thread(this);
        serverListenerThread.start();

    }

    /**
     * handles messages from each client. In this case messages are simply displayed.
     * Modified to prepare a response and send back to the same client. Simply changes the input text to upper case.
     * This is a shared resource among all client threads, so it has to be synchronized.
```

```
72      *
73      *
74      * @param msg
75      * @param client
76      */
77     public synchronized void handleMessagesFromClient(String msg, ClientManager client) {
78
79         // format the client message before displaying in server's terminal output.
80         String formattedMessage = String.format("[client %d] : %s", client.getClientID(), msg);
81
82         display(formattedMessage);
83
84         //prepare a response for the client.
85         String response = "[server says]: " + msg.toUpperCase();
86         sendMessageToClient(response, client);
87
88     }
89
90     /**
91      * Handles displaying of messages received from each client.
92      * Called from handleMessagesFromClient()
93      * @param message
94      */
95     public void display(String message) {
96         System.out.println(">> " + message);
97     }
98
99
100    /**
101     * Handles, sending a message to client. In this case, it is a string.
102     * Each client will be calling this to send a message to the client, so it is made
    synchronized.
103     * However, this can be handled separately within the ClientManager.
104     *
105     * @param msg       Message
106     * @param client    Client to be sent
107     */
108    public synchronized void sendMessageToClient(String msg, ClientManager client) {
109        try {
110            client.sendMessageToClient(msg);
111        } catch (IOException e) {
112            System.err.println("[server: ] Server-to-client message sending failed...");
113        }
114    }
115
116    /**
117     *
118     * @return list of Thread[] pertaining to the clients connected to the server
119     */
120    public Thread[] getClientConnections() {
121
122        Thread[] clientThreadList = new Thread[clientThreadGroup.activeCount()];
123        clientThreadGroup.enumerate(clientThreadList);
124
125        return clientThreadList;
126    }
127
128    /**
129     * Close the server and associated connections.
130     */
131    public void close() {
132
133        if (this.serverSocket == null)
134            return;
135
136        try {
137            this.stopServer = true;
138            this.serverSocket.close();
139
140        } catch (IOException e) {
141            System.err.println("[server: ] Error in closing server connection...");
```

```
142         } finally {
143 
144             // Close the client sockets of the already connected clients
145             Thread[] clientThreadList = getClientConnections();
146             for (int i = 0; i < clientThreadList.length; i++) {
147                 try {
148                     ((ClientManager) clientThreadList[i]).closeAll();
149                 }
150                 // Ignore all exceptions when closing clients.
151                 catch (Exception ex) {
152 
153                 }
154             }
155             this.serverSocket = null;
156 
157         }
158 
159     }
160 
161     /**
162      * Represents the thread that listens to the port, and creates client connections.
163      * Here, each connection is treated as a separate thread, and each client is associated with
    the ThreadGroup.
164      *
165      */
166     @Override
167     public void run() {
168 
169         System.out.println("[server: ] starting server: listening @ port: " + port);
170 
171         // increments when a client connects.
172         int clientCount = 0;
173 
174         // loops until stopserver flag is set to true.
175         while (!this.stopServer) {
176 
177             Socket clientSocket = null;
178             try {
179                 clientSocket = serverSocket.accept();
180             } catch (IOException e1) {
181                 System.err.println("[server: ] Error when handling client connections on port " +
    port);
182             }
183 
184             ClientManager cm = new ClientManager(this.clientThreadGroup, clientSocket,
    clientCount, this);
185 
186             // new ClientManager(clientSocket, this);
187             try {
188                 Thread.sleep(1000);
189             } catch (InterruptedException e) {
190                 System.err.println("[server: ] server listner thread interruped..");
191             }
192 
193             clientCount++;
194 
195         }
196     }
197 
198     /**
199      * Main() to start the SimpleServer.
200      *
201      * @param args
202      */
203     public static void main(String[] args) {
204 
205         SimpleServer server = new SimpleServer();
206         // port number to listen
207         int port = 7777;
208 
209         try {
```

```
210             server.initializeServer(port);
211 
212         } catch (IOException e) {
213             System.err.println("[server: ] Error in initializing the server on port " + port);
214         }
215         // Main thread continues...
216 
217 
218     }
219 }
220 
```