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

import java.io.BufferedReader;


/**
 * Represents a client.
 * Allows user inputs through keyboard and pass them to the server.
 * Receives responses from the user.
 *
 * @author thanuja
 * @version 20.11.2019
 */
public class SimpleClient implements Runnable {

    // reference variable for client socket
    private Socket                  clientSocket;

    // reference variable to store object IO streams, should be used when working with serialized objects.
    private ObjectOutputStream      output;
    private ObjectInputStream       input;

    // boolean variable to store stopclient flag.
    private boolean                 stopClient;

    // reference variable for Thread
    private Thread                  clientReader;

    // variables to store Host IP and port number
    private String                  host;
    private int                     port;

    /**
     * Constructor, initiates a client, and calls for openConnection.
     * @param host
     * @param port
     * @throws IOException
     */
    public SimpleClient(String host, int port) throws IOException {
        this.host = host;
        this.port = port;
        openConnection();
    }

    /**
     * opens a connection to the server
     * setup Object IO streams for the socket.
     *
     * @throws IOException
     */
    public void openConnection() throws IOException {

        // Create the sockets and the data streams
        try {

            this.clientSocket = new Socket(this.host, this.port);
            this.output = new ObjectOutputStream(this.clientSocket.getOutputStream());
            this.input = new ObjectInputStream(this.clientSocket.getInputStream());

        } catch (IOException ex) {
            try {
                closeAll();
            } catch (Exception exc) {
                System.err.println("[client: ] error in opening a connection to: " + this.host + " on port: " + this.port);
            }

            throw ex; // Rethrow the exception.
        }

```

```java
        // creates a Thread instance and starts the thread.
        this.clientReader = new Thread(this);
        this.stopClient = false;
        this.clientReader.start();

    }

    /**
     * Handles sending a message to server. In this case, it is a String.
     * @param msg
     * @throws IOException
     */
    public void sendMessageToServer(String msg) throws IOException {
        if (this.clientSocket == null || this.output == null)
            throw new SocketException("socket does not exist");

        this.output.writeObject(msg);
    }

    /**
     * Handle message from the server. In this case, simply display them.
     * @param msg
     */
    public void handleMessageFromServer(String msg) {
        display(msg);

    }

    /**
     * Simply display a String message in the terminal.
     * @param message
     */
    public void display(String message) {
        System.out.println("> " + message);
    }


    /**
     * Close all connections
     * @throws IOException
     */
    private void closeAll() throws IOException {
        try {
            // Close the socket
            if (this.clientSocket != null)
                this.clientSocket.close();

            // Close the output stream
            if (this.output != null)
                this.output.close();

            // Close the input stream
            if (this.input != null)
                this.input.close();

        } finally {
            // Set the streams and the sockets to NULL no matter what.
            this.output = null;
            this.input = null;
            this.clientSocket = null;
        }
    }

    /**
     * handles user inputs from the terminal.
     * This should run as a separate thread. In this case, main thread.
     *
     */
    public void runClient() {
        try {
            BufferedReader fromConsole = new BufferedReader(new InputStreamReader(System.in));
```

```
147             String message = null;
148 
149             while (true) {
150                 message = fromConsole.readLine();
151                 handleUserInput(message);
152                 if(message.equals("over"))
153                     break;
154             }
155 
156             System.out.println("[client: ] stopping client...");
157             this.stopClient = true;
158             fromConsole.close();
159             //closeAll();
160         } catch (Exception ex) {
161             System.out.println("[client: ] unexpected error while reading from console!");
162         }
163 
164     }
165 
166     /**
167      * Can perform any pre-processing or checking of the user input before sending it to server.
168      *
169      * @param userResponse
170      */
171     public void handleUserInput(String userResponse) {
172 
173         if (!this.stopClient) {
174             try {
175                 sendMessageToServer(userResponse);
176             } catch (IOException e) {
177                 System.err.println("[client: ] error when sending message to server: " +
    e.toString());
178 
179                 try {
180                     closeAll();
181                 } catch (IOException ex) {
182                     System.err.println("[client: ] error closing the client connections: " +
    ex.toString());
183                 }
184             }
185         }
186     }
187 
188     /**
189      * The thread that communicates with the server.
190      * receives a message from the server, passes it to handleMessageFromServer().
191      *
192      */
193     @Override
194     public void run() {
195 
196         String msg;
197 
198         // Loop waiting for data
199 
200         try {
201             while (!this.stopClient) {
202                 // Get data from Server and send it to the handler
203                 // The thread waits indefinitely at the following
204                 // statement until something is received from the server
205                 msg = (String) input.readObject();
206 
207                 // Concrete subclasses do what they want with the
208                 // msg by implementing the following method
209                 handleMessageFromServer(msg);
210             }
211 
212             System.out.println("[client: ] client stopped..");
213         } catch (Exception exception) {
214             if (!this.stopClient) {
215                 try {
```

```
216                     closeAll();
217                 } catch (Exception ex) {
218                     System.err.println("[client: ] error in closing the client connection...");
219                 }
220             }
221         } finally {
222             clientReader = null;
223         }
224 
225         System.out.println("[client: ] exiting thread...");
226     }
227 
228     /**
229      * Main() to initiate the client.
230      * @param args
231      */
232     public static void main(String[] args) {
233 
234         // hardcoded server IP and port number.
235         String ip = "127.0.0.1";
236         int port = 7777;
237 
238         SimpleClient chatclient = null;
239 
240         // thread to communicate with the server starts here.
241         try {
242             chatclient = new SimpleClient(ip, port);
243         } catch (IOException e) {
244             System.err.println("[client: ] error in openning the client connection to " + ip + " on
    port: " + port);
245         }
246 
247         // Main thread continues and in this case used to handle user inputs from the terminal.
248         chatclient.runClient();
249 
250     }
251 
252 }
253 
```