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

import java.io.IOException;

/**
 * Class represents a handler for each Client for the Server. Each client to be treated as a separate thread.
 *
 * @author thanuja
 * @version 20.11.2019
 */
public class ClientManager extends Thread {

    // reference variable to store client socket
    private Socket                  clientSocket;

    // reference for the Sever
    private AbstractServerComponent server;

    // boolean flag to indicate whether to stop the connection
    private boolean                 stopConnection;

    // Input Output streams to communicate with the client using Serialized objects
    private ObjectOutputStream      out;
    private ObjectInputStream       in;

    // store an incrementing ID for the client.
    private int                     clientID;



    /**
     * Constructor to be called, when handling multiple clients. Requires a ThreadGroup instance from the Server
     *
     * @param threadgroup
     * @param socket
     * @param clientID
     * @param server
     */
    public ClientManager(ThreadGroup threadgroup, Socket socket, int clientID, AbstractServerComponent server) {
        super(threadgroup, (Runnable) null);

        this.clientSocket = socket;
        this.server = server;
        this.stopConnection = false;
        this.clientID = clientID;

        System.out.println("[ClientManager: ] new client request received, port "
                + socket.getPort());
        try {
            this.out = new ObjectOutputStream(this.clientSocket.getOutputStream());
            this.in = new ObjectInputStream(this.clientSocket.getInputStream());
        }
        catch(IOException e) {
            System.err.println("[ClientManager: ] error when establishing IO streams on client socket.");
            try {
                closeAll();
            } catch (IOException e1) {
                System.err.println("[ClientManager: ] error when closing connections..." + e1.toString());

            }
        }

        start();
    }

    /**
```

```java
     * Performs the function of sending a message from Server to remote Client#
     * Uses ObectOutputStream
     *
     * @param msg
     * @throws IOException
     */
    public void sendMessageToClient(String msg) throws IOException {
        if (this.clientSocket == null || this.out == null)
            throw new SocketException("socket does not exist");

        this.out.writeObject(msg);
    }

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

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

            // Close the input stream
            if (this.in != null)
                this.in.close();
        } finally {
            // Set the streams and the sockets to NULL no matter what.

            this.in = null;
            this.in = null;
            this.clientSocket = null;

        }
    }

    /**
     * Receive messages (String) from the client, passes the message to Sever's handleMessagesFromClient() method.
     * Works in a loop until the boolean flag to stop connection is set to true.
     */
    @Override
    public void run() {

        // The message from the client
        String msg = "";
        try {
            while (!this.stopConnection) {
                // This block waits until it reads a message from the client
                // and then sends it for handling by the server,
                // thread indefinitely waits at the following
                // statement until something is received from the server

                msg = (String)this.in.readObject();
                this.server.handleMessagesFromClient(msg, this);

                if(msg.equals("over")) {
                    this.stopConnection = true;
                }
            }

            System.out.println("[ClientManager: ] stopping the client connection ID: " + this.clientID);
        } catch (Exception e) {
            System.err.println("[ClientManager: ] error when reading message from client.." + e.toString());
            /**
```

```
             * If there is an error, while the connection is not stopped, close all.
             */
            if (!this.stopConnection) {
                try {
                    closeAll();
                }
                catch (Exception ex)
                {
                    System.err.println("[ClientManager: ] error when closing the connections.." + ex.toString());
                }
            }
        }
        finally {
            if(this.stopConnection) {
                try {
                    closeAll();
                } catch (IOException e) {
                    System.err.println("[ClientManager: ] error when closing the connections.." + e.toString());
                }
            }
        }



    }

    /**
     * @return a description of the client, including IP address and host name
     */
    public String toString() {
        return this.clientSocket == null ? null : this.clientSocket.getInetAddress().getHostName() + " ("
                + this.clientSocket.getInetAddress().getHostAddress() + ")";
    }


    //////// GETTERS AND SETTERS ////////////
    public int getClientID() {
        return this.clientID;
    }

}

```