```java
package chat.client;

import chat.protocol.Message;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.net.UnknownHostException;

/**
 * Classe representant un client d'un chat utilisant les sockets.
 * @author Nico
 */
public class Client{
    /**
     * Etat du client, connecté : true, Déconnecté : false.
     */
    boolean state;

    /**
     * Adresse du serveur.
     */
    protected String serverAdress;

    /**
     * Port de communication.
     */
    protected int serverPort;

    /**
     * Nom d'utilisateur.
     */
    protected String userName;

    /**
     * Vue du client.
     */
    ClientChatView clientChatView;

    /**
     * Socket de communication.
     */
    Socket socket;

    /**
     * Flux d'écriture de message.
     */
    PrintStream socOut;

    /**
     * Flux de lecture de message.
```

```
53          */
54         BufferedReader socIn;
55
56         /**
57          * Thread de lecture du client.
58          */
59         ClientThread clientThread;
60
61         /**
62          * Constructeur.
63          */
64         public Client() {
65             try {
66                 state = false;
67                 clientChatView = new ClientChatView(this);
68                 serverAdress = null;
69                 socket = null;
70                 socIn = null;
71                 socOut = null;
72                 serverPort = 1099;
73                 clientChatView.setVisible(true);
74             } catch (Exception e) {
75                 System.err.println("Client exception : " + e.toString());
76             }
77         }
78
79         /**
80          * Getter.
81          * @return Non d'utilisateur.
82          */
83         public String getUserName() {
84             return userName;
85         }
86
87         /**
88          * Getter.
89          * @return Adresse du serveur.
90          */
91         public String getServerAdress() {
92             return serverAdress;
93         }
94
95         /**
96          * Getter.
97          * @return Port de communication.
98          */
99         public int getServerPort() {
100            return serverPort;
101        }
102
103        /**
104         * Setter.
```

```
105         * @param name Nom d'utilisateur.
106         */
107        public void setUserName(String name) {
108            userName = name;
109        }
110
111        /**
112         * Setter.
113         * @param adress Adresse du serveur.
114         */
115        public void setServerAdress(String adress) {
116            serverAdress = adress;
117        }
118
119        /**
120         * Setter.
121         * @param port Port de communication.
122         */
123        public void setServerPort(int port) {
124            serverPort = port;
125        }
126
127        /**
128         * Methode permettant d'afficher un message sur la vue.
129         * @param m Message.
130         */
131        public void displayMessage(Message m){
132            clientChatView.displayMessage(m);
133        }
134
135        /**
136         * Methode permettant d'afficher un message sur la vue.
137         * @param m Message.
138         */
139        public void displayMessage(String m){
140            clientChatView.displayMessage(m);
141        }
142
143        /**
144         * Methode notifiant la connexion d'un autre utilisateur.
145         * Ajoute le nom du client à la vue
146         * @param client Client
147         */
148        public void newUserConnected(String client) {
149            clientChatView.addUser(client);
150            displayMessage(new Message("server","all",client+" signed in
                ..."));
151        }
152
153        /**
154         * Methode notifiant la déconnexion d'un autre utilisateur.
155         * Enlève le nom du client à la vue
```

```
156         * @param client Client
157         */
158        public void userDisconnected(String client) {
159            clientChatView.removeUser(client);
160            if(client.compareTo(userName)==0) {
161                clientThread.interrupt();
162                try {
163                    socOut.close();
164                    socIn.close();
165                    socket.close();
166                } catch (Exception e) {
167                    System.err.println("Client exception : "+
                       e.toString());
168                    System.exit(1);
169                }
170                clientThread = null;
171                state = false;
172                clientChatView.refresh();
173            } else {
174                displayMessage(new Message("server","all",client+"
                   signed in ..."));
175            }
176        }
177
178        /**
179         * Methode permettant d'envoyer un message au serveur et qui
            serra retransmi
180         * au "receiver".
181         * @param receiver Destinataire. ("all" si destiné à tous les
            utilisateurs)
182         * @param m Texte du message.
183         */
184        public void sendMessage(String receiver,String m) {
185            socOut.println("SENDTO " + receiver + " CONTENT " + m);
186        }
187
188        /**
189         * Methode permettant de se connecter au serveur. Créer un
            ClientThread
190         * permettant la recepetion de messages.
191         * @param sAdress Adresse du serveur.
192         * @param sPort Port de communication.
193         * @param name Nom de l'utilsateur.
194         * @return Retourne true si reussi, false sinon.
195         */
196        public boolean connect(String sAdress, String sPort, String name) {
197            try {
198    //              serverAdress = sAdress;
199                state = false;
200                userName = name;
201                serverPort = Integer.parseInt(sPort);//gerer exception
202
```

```
203             socket = new Socket(serverAdress,serverPort);
204         socIn = new BufferedReader(new InputStreamReader(
205                     socket.getInputStream()));
206         socOut= new PrintStream(socket.getOutputStream());
207 
208             socOut.println("CONNECT " + userName);
209             String line = socIn.readLine();
210             if(line.contains("SIGNOUT")) {
211                 displayMessage(new
                    Message("server",userName,"username \""
212                         + userName + "\" already used."));
213             } else {
214                 clientThread = new ClientThread(this, socIn);
215                 clientThread.start();
216                 state = true;
217                 clientChatView.refresh();
218             }
219         } catch (UnknownHostException e) {
220             displayMessage("Don't know about host:" + serverAdress);
221             System.err.println();
222         } catch (IOException e) {
223             displayMessage("Client exception : Couldn't get I/O for "
224                                 + "the connection to:"+ serverAdress);
225         }
226 
227         return state;
228     }
229 
230     /**
231      * Methode permettant de se deconnecter du serveur.
232      */
233     public void disconnect() {
234         try {
235             socOut.println("QUIT");
236         } catch (Exception e) {
237             displayMessage("Client exception : "+ e.toString());
238         }
239     }
240 
241     /**
242      * Methode permettant de savoir si le client est connecté à un
      serveur.
243      * @return True si oui, false sinon.
244      */
245     public Boolean isConnected() {
246         return state;
247     }
248 }
249 
```