```java
package chatrmi.client;

import chatrmi.protocol.ClientInterface;
import chatrmi.protocol.Message;
import chatrmi.protocol.MessageInterface;
import chatrmi.protocol.ServerInterface;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.Date;

/**
 * Classe permettant de créer un objet Remote de type client pour un server de
 * chat
 * Implémente l'interface Remote "ClientInterface"
 * @author Nico
 */
public class Client implements ClientInterface{

    /**
     * Adresse du serveur sur lequel le client est connecté
     */
    protected String serverAdress;

    /**
     * Port par lequel le client et le serveur communiquent
     */
    protected int serverPort;

    /**
     * Nom de l'utilisateur
     */
    protected String userName;

    /**
     * Reference sur le serveur (Remote) sur lequel on est connecté
     */
    ServerInterface server;

    /**
     * Reference sur la fenetre du client
     */
    ClientChatView clientChatView;

    /**
     * Constructeur
     */
    public Client() {
        try {
            clientChatView = new ClientChatView(this);
```

```
52              serverAdress = null;
53              serverPort = 1099;
54              UnicastRemoteObject.exportObject(this, 0);
55              clientChatView.setVisible(true);
56          } catch (Exception e) {
57              System.err.println("Client exception : " + e.toString());
58                  e.printStackTrace();
59          }
60      }
61
62      /**
63       * Getter
64       * @return Le nom de l'utilisateur
65       */
66      @Override
67      public String getUserName() {
68          return userName;
69      }
70
71      /**
72       * Getter
73       * @return l'adresse du serveur
74       */
75      public String getServerAdress() {
76          return serverAdress;
77      }
78
79      /**
80       * Getter
81       * @return le port
82       */
83      public int getServerPort() {
84          return serverPort;
85      }
86
87      /**
88       * Setter
89       * @param name nom de l'utilisateur
90       */
91      public void setUserName(String name) {
92          userName = name;
93      }
94
95      /**
96       * Setter
97       * @param adress adress du serveur
98       */
99      public void setServerAdress(String adress) {
100         serverAdress = adress;
101     }
102
103     /**
```

```
104         * Setter
105         * @param port port de communication
106         */
107        public void setServerPort(int port) {
108            serverPort = port;
109        }
110
111        /**
112         * Methode permettant d'afficher un message sur la vue
113         * @param m message à afficher
114         * @throws RemoteException
115         */
116        @Override
117        public void displayMessage(MessageInterface m) throws
           RemoteException {
118            clientChatView.displayMessage(m);
119        }
120
121        /**
122         * Methode notifiant la connexion d'un autre utilisateur.
123         * Ajoute le nom du client à la vue
124         * @param c Client
125         * @throws RemoteException
126         */
127        @Override
128        public void newUserConnected(ClientInterface c) throws
           RemoteException {
129            clientChatView.addUser(c);
130        }
131
132        /**
133         * Methode notifiant la déconnexion d'un autre utilisateur.
134         * Enlève le nom du client à la vue
135         * @param c Client
136         * @throws RemoteException
137         */
138        @Override
139        public void userDisconnected(ClientInterface c) throws
           RemoteException {
140            if (c.getUserName().compareTo(userName)==0){
141                server = null;
142                clientChatView.refresh();
143            }
144            clientChatView.removeUser(c);
145        }
146
147        /**
148         * Methode permettant d'envoyer un message à tous le client
           connecté au
149         * serveur
150         * On crée un stub pour le message écrit par le client et on
           l'envoie au
```

```
151         * serveur.
152         * @param m texte du message
153         */
154        public void sendMessageToAll(String m) {
155            try {
156                Message msg = new Message(new Date(), userName, m);
157                MessageInterface stubMsg = (MessageInterface)
                   UnicastRemoteObject
158                        .exportObject(msg, 0);
159                server.sendMessageToAll(stubMsg);
160            } catch (Exception e) {
161                System.err.println("Client exception: " + e.toString());
162                e.printStackTrace();
163            }
164        }
165
166        /**
167         * Methode permettant de se connecter au serveur.
168         * Le client établit une connection au RMI registry crée dans la
           classe
169         * Server.java. Si la tentative de connexion réussi, le client
           est rajouté
170         * à la liste de Client du serveur.
171         * @param sAdress adresse du serveur
172         * @param sPort port de communication
173         * @param name nom du client
174         * @return
175         */
176        public Boolean connect(String sAdress, String sPort, String name) {
177            Boolean result = true;
178            try {
179                serverAdress = sAdress;
180                userName = name;
181                serverPort = Integer.parseInt(sPort);
182                Registry registry = LocateRegistry
183                        .getRegistry(serverAdress,serverPort);
184                server = (ServerInterface) registry.lookup("server");
185                if(server != null && !server.addClient(this)) {
186                    server = null;
187                    result = false;
188                }
189            } catch (Exception e) {
190                System.err.println("Client exception: " + e.toString());
191                try
192                {
193                    clientChatView.
194                            displayMessage(new Message(new Date(),
195                                    "Error", "Impossible to connect."));
196                } catch (Exception e2) {
197                    System.err.println("Client exception: " +
                       e2.toString());
198                }
```

```
199                 server = null;
200                 result = false;
201             }
202             if (result){
203                 clientChatView.refresh();
204             }
205             return result;
206         }
207 
208         /**
209          * Methode permettant de se deconnecter du serveur.
210          * Le client se déconnecte du serveur. On supprime le lien vers le serveur
211          * et on supprime le client de la liste Clients dans la classe Server.java .
212          */
213         public void disconnect() {
214             try {
215                 server.deleteClient(this);
216                 server = null;
217                 clientChatView.refresh();
218             } catch (Exception e) {
219                 System.err.println("Client exception: " + e.toString());
220                 e.printStackTrace();
221             }
222         }
223 
224         /**
225          * Methode permettant de savoir si le client est connecté a un serveur.
226          * On vérifie que le client est bien connecté au serveur (l'attribut "
227          * server" qui constitue la réference vers le serveur est non nul).
228          * @return Retourne true si connecté, false sinon
229          */
230         public Boolean isConnected() {
231             return null != server;
232         }
233     }
234 
```