

```
1 package chat.client;
2
3 import chat.protocol.Message;
4 import java.io.BufferedReader;
5 import java.io.IOException;
6 import java.io.InputStreamReader;
7 import java.io.PrintStream;
8 import java.net.Socket;
9 import java.net.UnknownHostException;
10
11 /**
12  * Classe representant un client d'un chat utilisant les sockets.
13  * @author Nico
14  */
15 public class Client{
16     /**
17      * Etat du client, connecté : true, Déconnecté : false.
18      */
19     boolean state;
20
21     /**
22      * Adresse du serveur.
23      */
24     protected String serverAddress;
25
26     /**
27      * Port de communication.
28      */
29     protected int serverPort;
30
31     /**
32      * Nom d'utilisateur.
33      */
34     protected String userName;
35
36     /**
37      * Vue du client.
38      */
39     ClientChatView clientChatView;
40
41     /**
42      * Socket de communication.
43      */
44     Socket socket;
45
46     /**
47      * Flux d'écriture de message.
48      */
49     PrintStream socOut;
50
51     /**
52      * Flux de lecture de message.
```

- 1 -

```
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             serverAddress = 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 getServerAddress() {
92         return serverAddress;
93     }
94
95     /**
96      * Getter.
97      * @return Port de communication.
98      */
99     public int getServerPort() {
100         return serverPort;
101     }
102
103     /**
104      * Setter.
```

- 2 -

```
105     * @param name Nom d'utilisateur.
106     */
107     public void setUsername(String name) {
108         userName = name;
109     }
110
111     /**
112      * Setter.
113      * @param address Adresse du serveur.
114      */
115     public void setServerAddress(String address) {
116         serverAddress = address;
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
```

- 3 -

```
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 : "+
168                 e.toString());
169                 System.exit(1);
170             }
171             clientThread = null;
172             state = false;
173             clientChatView.refresh();
174         } else {
175             displayMessage(new Message("server","all",client+"
176             signed in ..."));
177         }
178     }
179
180     /**
181      * Methode permettant d'envoyer un message au serveur et qui
182      * sera retransmi
183      * au "receiver".
184      * @param receiver Destinataire. ("all" si destiné à tous les
185      * utilisateurs)
186      * @param m Texte du message.
187      */
188     public void sendMessage(String receiver,String m) {
189         socOut.println("SENDTO " + receiver + " CONTENT " + m);
190     }
191
192     /**
193      * Methode permettant de se connecter au serveur. Créer un
194      * ClientThread
195      * permettant la reception de messages.
196      * @param sAddress Adresse du serveur.
197      * @param sPort Port de communication.
198      * @param name Nom de l'utilisateur.
199      * @return Retourne true si réussi, false sinon.
200     */
201     public boolean connect(String sAddress, String sPort, String name) {
202         try {
203             serverAddress = sAddress;
204             state = false;
205             userName = name;
206             serverPort = Integer.parseInt(sPort); //gerer exception
```

- 4 -

```

/Users/Nico/Desktop/Code/Socket/Client.java
Page 5 of 5
Mon Jan 25 20:49:10 2016

203     socket = new Socket(serverAddress,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
212             Message("server",userName,"username \\"
213                 + userName + "\" already used."));
214     } else {
215         clientThread = new ClientThread(this, socIn);
216         clientThread.start();
217         state = true;
218         clientChatView.refresh();
219     } catch (UnknownHostException e) {
220         displayMessage("Don't know about host:" + serverAddress);
221         System.err.println();
222     } catch (IOException e) {
223         displayMessage("Client exception : Couldn't get I/O for "
224             + "the connection to:" + serverAddress);
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
243  * serveur.
244  * @return True si oui, false sinon.
245  */
246 public Boolean isConnected() {
247     return state;
248 }
249 }

```

- 5 -

```

/Users/Nico/Desktop/Code/Socket/ClientThread.java
Page 1 of 2
Mon Jan 25 20:49:17 2016

1  /**
2  * To change this license header, choose License Headers in Project
3  * Properties.
4  * To change this template file, choose Tools | Templates
5  * and open the template in the editor.
6  */
7  package chat.client;
8
9  import chat.protocol.Message;
10 import java.io.BufferedReader;
11
12
13 /**
14  * Thread client d'écoute de socket.
15  * @author Nico
16  */
17 public class ClientThread extends Thread{
18     /**
19      * Référence sur le client correspondant.
20      */
21     Client client;
22
23     /**
24      * Flux de lecture de message.
25      */
26     BufferedReader socIn;
27
28     /**
29      * Constructeur.
30      * @param c Référence sur le client correspondant.
31      * @param br Flux de lecture de message.
32      */
33     public ClientThread(Client c,BufferedReader br) {
34         client = c;
35         socIn = br;
36     }
37
38     /**
39      * Méthode exécutée par le thread. Lit les messages provenant du
40      * serveur et
41      * en notifie le "client".
42      * Traite les lignes que le BufferedReader socIn (de la socket)
43      * reçoit du
44      * serveur et analyse ce qu'elle reçoit et fait le traitement
45      * approprié
46      * selon si c'est un message informatif(quelqu'un s'est connecté
47      * ou
48      * déconnecté) ou un message d'un client.
49      */
50     @Override
51     public void run(){

```

- 1 -

```

/Users/Nico/Desktop/Code/Socket/ClientThread.java
Page 2 of 2
Mon Jan 25 20:49:17 2016

48     try {
49         String line;
50         while (true) {
51             line = socIn.readLine();
52             treatLine(line);
53         }
54     } catch (Exception e) {
55         System.err.println("ClientThread exception : " +
56             e.toString());
57     }
58 }
59
60 /**
61  * Methode traitant une ligne recue par le flux d'entrée "socIn".
62  * @param line Ligne lue.
63  */
64 private void treatLine(String line){
65     if(line.contains("SIGNIN")) {
66         String[] splits = line.split("SIGNIN ");
67         client.newUserConnected(splits[1]);
68     } else if(line.contains("SIGNOUT")) {
69         String[] splits = line.split("SIGNOUT ");
70         client.userDisconnected(splits[1]);
71     } else if(line.contains("MESSAGE FROM") &&
72         line.contains("TO") && line.
73         contains("CONTENT")) {
74         String[] splits1 = line.split("MESSAGE FROM ");
75         String[] splits2 = splits1[1].split(" TO ");
76         String[] splits3 = splits2[1].split(" CONTENT ");
77         String sender = splits2[0];
78         String receiver = splits3[0];
79         String textMessage = splits3[1];
80         client.displayMessage(new
81             Message(sender,receiver,textMessage));
82     }
83 }
84 }

```

- 2 -

```

/Users/Nico/Desktop/Code/Socket/Message.java
Page 1 of 2
Mon Jan 25 20:49:23 2016

1  package chat.protocol;
2
3  /**
4  * Classe permettant de modéliser un message.
5  * @author Nico
6  */
7  public class Message{
8      /**
9      * Nom de l'expéditeur du message.
10     */
11     private final String sender;
12
13     /**
14      * Nom du destinataire du message.
15      */
16     private final String receiver;
17
18     /**
19      * Texte du message.
20      */
21     private final String textMessage;
22
23     /**
24      * Constructeur.
25      * @param s Nom de l'expéditeur.
26      * @param r Nom du destinataire.
27      * @param tm Texte du message.
28      */
29     public Message(String s, String r, String tm) {
30         receiver = r;
31         sender = s;
32         textMessage = tm;
33     }
34
35     /**
36      * Getter.
37      * @return Nom de l'expéditeur du message.
38      */
39     public String getSender() {
40         return sender;
41     }
42
43     /**
44      * Getter.
45      * @return
46      */
47     public String getReceiver() {
48         return receiver;
49     }
50
51     /**
52      * Getter.

```

- 1 -

```
53      * @return Texte du message.
54      */
55      public String getTextMessage() {
56          return textMessage;
57      }
58  }
59
```

- 2 -

```
1
2  /*
3   * To change this license header, choose License Headers in Project
4   * Properties.
5   * To change this template file, choose Tools | Templates
6   * and open the template in the editor.
7   */
8  package chat.server;
9
10 import java.net.InetAddress;
11
12 /**
13  * Classe representant un serveur d'un chat utilisant les sockets.
14  * @author Nico
15  */
16 public class Server {
17
18     /**
19      * Nom du serveur.
20      */
21     private String serverName;
22
23     /**
24      * Port de communication.
25      */
26     private int serverPort;
27
28     /**
29      * Etat du serveur. True : connecté, false sinon.
30      */
31     private boolean state;
32
33     /**
34      * Thread d'écoute permettant la connexion d'utilisateurs et de
35      * transmettre
36      * les messages à tous les utilisateurs.
37      */
38     private ServerMultiThread serverMultiThread;
39
40     /**
41      * Vue du serveur.
42      */
43     private ServerRunningView serverRunningView;
44
45     /**
46      * Constructeur.
47      */
48     public Server() {
49         initialize("server",1099);
50     }
51
52     /**
53      * Getter.
54
```

- 1 -

```
51      * @return Port de communication.
52      */
53      public int getServerPort() {
54          return serverPort;
55      }
56
57      /**
58      * Getter.
59      * @return Nom du serveur.
60      */
61      public String getServerName() {
62          return serverName;
63      }
64
65      /**
66      * Methode permettant d'initialiser le serveur.
67      * @param sName Nom du serveur.
68      * @param sPort Port de communication.
69      */
70      private void initialize(String sName, int sPort) {
71          serverName = sName;
72          serverPort = sPort;
73          state = false;
74          serverMultiThread = null;
75          serverRunningView = new ServerRunningView(this);
76          serverRunningView.setLocationRelativeTo(null);
77          serverRunningView.setVisible(true);
78      }
79
80      /**
81      * Methode permettant de lancer le serveur.
82      * @param port
83      */
84      public void start(String port)
85      {
86          try {
87              serverPort = Integer.parseInt(port);
88              state = true;
89              serverMultiThread = new ServerMultiThread(this);
90              serverMultiThread.start();
91
92              serverRunningView.displayServerLocalAdress(InetAddress.getL
93                  ocalHost()
94                      .getHostAddress());
95          } catch (Exception e) {
96              System.err.println("Server exception : " + e.toString());
97          }
98      }
99
100     /**
101      * Methode permettant d'arreter le serveur.
102
```

- 2 -

```
101     */
102     public void stop() {
103         state = false;
104         if (serverMultiThread!=null) {
105             serverMultiThread.interrupt();
106         }
107         refreshClientsNumber();
108     }
109
110     /**
111      * Methode permettant de rafraichir le nombre de client sur la
112      * vue.
113      */
114     public void refreshClientsNumber() {
115         if(serverMultiThread!=null) {
116             serverRunningView.setClientNumber(serverMultiThread.
117                 getClientsNumber());
118         } else {
119             serverRunningView.setClientNumber(0);
120         }
121     }
122
123     /**
124      * Methode permettant de savoir si le serveur est en train de
125      * tourner.
126      * @return True si oui, false sinon.
127      */
128     public boolean isRunning() {
129         return state;
130     }
131 }
```

- 3 -

```
1  /*
2  * To change this license header, choose License Headers in Project
3  * Properties.
4  * To change this template file, choose Tools | Templates
5  * and open the template in the editor.
6  */
7  package chat.server;
8
9  import chat.protocol.Message;
10 import java.net.ServerSocket;
11 import java.net.Socket;
12 import java.util.LinkedList;
13
14 /**
15 * Thread d'écoute permettant la connexion d'utilisateurs et de
16 * transmettre
17 * les messages à tous les utilisateurs.
18 * @author Nico
19 */
20 public class ServerMultiThread extends Thread {
21
22     private Server server;
23     ServerSocket listenSocket;
24     private LinkedList<ServerThread> serverThreads;
25
26     /**
27     * Constructeur.
28     * @param s Reference sur le serveur.
29     */
30     public ServerMultiThread(Server s) {
31         server = s;
32         serverThreads = new LinkedList<>();
33         try {
34             listenSocket = new ServerSocket(server.getServerPort());
35         } catch (Exception e) {
36             System.err.println("ServerMultiThread exception : " +
37                 e.toString());
38         }
39     }
40
41     /**
42     * Getter.
43     * @return Nombre d'utilisateurs connectés.
44     */
45     public int getClientsNumber() {
46         return serverThreads.size();
47     }
48
49     /**
50     * Méthode exécutée par le thread. Boucle tant que
51     * server.isRunning()
52     * est true. Attend la connexion d'utilisateur, créer un
53     */
54 }
```

- 1 -

```
55     ServerThread à
56     * la connexion d'un utilisateur et lance ce ServerThread.
57     */
58     public void run() {
59         try {
60             while (server.isRunning()) {
61                 Socket clientSocket = listenSocket.accept();
62
63                 ServerThread serverThread = new
64                     ServerThread(server, this,
65                         clientSocket);
66                 serverThread.start();
67             } catch (Exception e) {
68                 System.err.println("ServerMultiThread exception : " +
69                     e.toString());
70             }
71         }
72     }
73
74     /**
75     * Méthode d'interruption du thread. Ferme la socket server et
76     * appelle la
77     * méthode interrupt() de chaque ServerThreads (d'écoute côté
78     * serveur de
79     * chaque utilisateurs).
80     */
81     @Override
82     public void interrupt() {
83         try {
84             super.interrupt();
85             listenSocket.close();
86             for (ServerThread st : serverThreads) {
87                 st.interrupt();
88             }
89             serverThreads.removeAll(serverThreads);
90         } catch (Exception e) {
91             System.err.println("ServerMultiThread exception : " +
92                 e.toString());
93         }
94     }
95
96     /**
97     * Methode permettan d'envoyer un message.
98     * @param m Message.
99     */
100    public void sendMessage(Message m) {
101        if (m.getReceiver().compareTo("all") == 0) {
102            for (ServerThread st : serverThreads) {
103                st.sendMessage(m.getTextMessage());
104            }
105        } else {
106            for (ServerThread st : serverThreads) {
107                st.sendMessage(m.getTextMessage());
108            }
109        }
110    }
111 }
```

- 2 -

```
102    }
103
104    /**
105     * Methode permettant de notifier la connexion d'un utilisateur.
106     * @param st Thread d'écoute côté serveur de l'utilisateur qui
107     * se déconnecte.
108     */
109     public void signout(ServerThread st) {
110         Message message;
111         String text;
112         if (server.getServerName().compareTo(st.getUserName()) == 0 ||
113             contains(st.getUserName())) {
114             text = "SIGNOUT ";
115             text += st.getUserName();
116             st.sendMessage(text);
117             st.interrupt();
118         } else {
119             //On informe au nouveau client que la connexion est
120             //acceptée
121             text = "SIGNIN ";
122             text += st.getUserName();
123             st.sendMessage(text);
124
125             //On envoie la liste des utilisateurs déjà connecté au
126             //client
127             for (ServerThread s : serverThreads) {
128                 text = "SIGNIN ";
129                 text += s.getUserName();
130                 st.sendMessage(text);
131             }
132
133             serverThreads.add(st);
134
135             //On informe tous utilisateurs de la connexion d'un
136             //nouveau client
137             text = "SIGNIN ";
138             text += st.getUserName();
139             message = new Message(server.getServerName(), "all", text);
140             sendMessage(message);
141
142             server.refreshClientsNumber();
143         }
144     }
145
146     /**
147     */
148 }
```

- 3 -

```
149     * Methode permettant de notifier la déconnexion d'un utilisateur.
150     * @param st Thread d'écoute côté serveur de l'utilisateur qui
151     * se déconnecte.
152     */
153     public void signout(ServerThread st) {
154         String text;
155         text = "SIGNOUT ";
156         text += st.getUserName();
157         Message message = new Message(server.getServerName(), "all",
158             text);
159         sendMessage(message);
160         st.interrupt();
161         serverThreads.remove(st);
162         server.refreshClientsNumber();
163     }
164
165     /**
166     * Getter permettant de recuperer le thread d'écoute côté
167     * serveur de
168     * l'utilisateur ayant le nom "username".
169     * @param username Nom de l'utilisateur.
170     * @return Thread d'écoute côté serveur du client ayant le nom
171     * "username".
172     */
173     public ServerThread getServerThread(String username) {
174         ServerThread st = null;
175         for (ServerThread s : serverThreads) {
176             if (username.compareTo(s.getUserName()) == 0) {
177                 st = s;
178                 break;
179             }
180         }
181         return st;
182     }
183
184     /**
185     * Methode permettant de savoir si un utilisateur ayant le nom
186     * d'utilisateur
187     * "username" est déjà connecté.
188     * @param username Nom de l'utilisateur.
189     * @return True si oui, false sinon.
190     */
191     public boolean contains(String username) {
192         return getServerThread(username) != null;
193     }
194 }
```

- 4 -

```
1  /*
2  * To change this license header, choose License Headers in Project
3  * Properties.
4  * To change this template file, choose Tools | Templates
5  * and open the template in the editor.
6  */
7  package chat.server;
8
9  import chat.protocol.Message;
10 import java.io.BufferedReader;
11 import java.io.InputStreamReader;
12 import java.io.PrintStream;
13 import java.net.Socket;
14
15 /**
16 *
17 * @author Nico
18 */
19 public class ServerThread extends Thread {
20     /**
21      * Référence sur le serveur.
22      */
23     private final Server server;
24
25     /**
26      * Référence sur la socket de communication.
27      */
28     private final Socket clientSocket;
29
30     /**
31      * Référence sur le thread parent.
32      */
33     private final ServerMultiThread serverMultiThread;
34
35     /**
36      * Flux d'écriture de message.
37      */
38     PrintStream socOut;
39
40     /**
41      * Flux de lecture de message.
42      */
43     BufferedReader socIn;
44
45     /**
46      * Nom de l'utilisateur relié.
47      */
48     private String userName;
49
50     /**
51      * Constructeur.
52      * @param s Référence sur le serveur.
53      */
54 }
```

- 1 -

```
52     * @param smt Référence sur le thread parent.
53     * @param c Socket de communication.
54     */
55     public ServerThread(Server s, ServerMultiThread smt, Socket c) {
56         server = s;
57         clientSocket = c;
58         serverMultiThread = smt;
59         try {
60             socIn = new BufferedReader(new
61                 InputStreamReader(clientSocket
62                     .getInputStream()));
63             socOut = new PrintStream(clientSocket.getOutputStream());
64         } catch (Exception e) {
65             System.err.println("ServerThread exception : " +
66                 e.toString());
67         }
68     }
69
70     /**
71      * Getter.
72      * @return Nom de l'utilisateur.
73      */
74     public String getUserName() {
75         return userName;
76     }
77
78     /**
79      * Méthode exécutée par le thread. Boucle tant que
80      * server.isRunning() est
81      * true. Attend que le client envoie une ligne et la traite.
82      */
83     @Override
84     public void run() {
85         try {
86             String line;
87             while (server.isRunning()) {
88                 line = socIn.readLine();
89                 treatLine(line);
90             } catch (Exception e) {
91                 System.err.println("ServerThread exception : " +
92                     e.toString());
93             }
94         }
95     }
96
97     /**
98      * Méthode d'interruption du thread. Deconnecte l'utilisateur en
99      * lui envoyant
100     * un message de deconnexion et ferme la sockets.
101     */
102     @Override
103     public void interrupt() {
```

- 2 -

```
99         super.interrupt();
100         try {
101             sendMessage("SIGNOUT "+userName);
102             socOut.close();
103             socIn.close();
104             clientSocket.close();
105         } catch (Exception e) {
106             System.err.println("Client exception : "+ e.toString());
107         }
108     }
109
110     /**
111      * Methode traitant une ligne recue par le flux d'entrée "socIn".
112      * @param line Ligne lu.
113      */
114     private void treatLine(String line) {
115         if (line.contains("CONNECT")) {
116             String[] splits = line.split("CONNECT ");
117             userName = splits[1];
118             serverMultiThread.signin(this);
119         } else if (line.compareTo("QUIT")==0) {
120             serverMultiThread.signout(this);
121         } else if (line.contains("SENDTO") &&
122             line.contains("CONTENT")) {
123             String[] splits1 = line.split("SENDTO ");
124             String[] splits2 = splits1[1].split(" CONTENT ");
125             String sender = userName;
126             String receiver = splits2[0];
127             String textMessage = "MESSAGE FROM " + sender + " TO " +
128                 receiver
129                 + " CONTENT " + splits2[1];
130             serverMultiThread.sendMessage(
131                 new Message(sender, receiver, textMessage));
132         }
133     }
134
135     /**
136      * Méthode permettant d'envoyer un message au client.
137      * @param m
138      */
139     public void sendMessage(String m) {
140         try {
141             socOut.println(m);
142         } catch (Exception e) {
143             System.err.println("ServerThread exception : " +
144                 e.toString());
145         }
146     }
147 }
```

- 3 -