```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package chat.server;

import chat.protocol.Message;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.LinkedList;

/**
* Thread d'écoute permettant la connexion d'utilisateurs et de transmettre
* les messages à tous les utilisateurs.
 * @author Nico
 */
public class ServerMultiThread extends Thread {

    private Server server;
    ServerSocket listenSocket;
    private LinkedList<ServerThread> serverThreads;

    /**
     * Constructeur.
     * @param s Reference sur le serveur.
     */
    public ServerMultiThread(Server s) {
        server = s;
        serverThreads = new LinkedList<>();
        try{
            listenSocket = new ServerSocket(server.getServerPort());
        } catch(Exception e) {
            System.err.println("ServerMultiThread exception : " + e.toString());
        }
    }

    /**
     * Getter.
     * @return Nombre d'utilisateurs connectés.
     */
    public int getClientsNumber() {
        return serverThreads.size();
    }

    /**
     * Méthode exécutée par le thread. Boucle tant que server.isRunning()
     * est true. Attend la connexion d'utilisateur, créer un
```

```
         ServerThread à
49       * la connexion d'un utilisateur et lance ce ServerThread.
50       */
51      public void run(){
52          try {
53              while (server.isRunning()) {
54                  Socket clientSocket = listenSocket.accept();
55
56                  ServerThread serverThread = new
                    ServerThread(server,this,
57                          clientSocket);
58                  serverThread.start();
59              }
60          } catch (Exception e) {
61              System.err.println("ServerMultiThread exception : " +
                e.toString());
62          }
63      }
64
65      /**
66       * Méthode d'intéruption du thread. Ferme la socket server et
         appelle la
67       * méthode interrupt() de chaque ServerThreads (d'écoute côté
         serveur de
68       * chaque utilisateurs).
69       */
70      @Override
71      public void interrupt(){
72          try {
73              super.interrupt();
74              listenSocket.close();
75              for(ServerThread st : serverThreads) {
76                  st.interrupt();
77              }
78              serverThreads.removeAll(serverThreads);
79          } catch (Exception e) {
80              System.err.println("ServerMultiThread exception : " +
                e.toString());
81          }
82      }
83
84      /**
85       * Methode permettan d'envoyer un message.
86       * @param m Message.
87       */
88      public void sendMessage(Message m) {
89          if (m.getReceiver().compareTo("all") == 0) {
90              for(ServerThread st : serverThreads) {
91                  st.sendMessage(m.getTextMessage());
92              }
93          } else {
94              for(ServerThread st : serverThreads) {
```

```
95                  if(st.getUserName().compareTo(m.getReceiver())==0 ||
96                          st.getUserName().compareTo(m.getSender())==0) {
97                      st.sendMessage(m.getTextMessage());
98                  }
99              }
100         }
101     }
102
103     /**
104      * Methode permettant de notifier la connexion d'un utilisateur.
105      * @param st Thread d'écoute côté serveur de l'utilisateur qui
         se déconnecte.
106      */
107     public void signin(ServerThread st) {
108         Message message;
109         String text;
110         if(server.getServerName().compareTo(st.getUserName()) == 0 ||
111                 contains(st.getUserName())){
112             text = "SIGNOUT ";
113             text += st.getUserName();
114             st.sendMessage(text);
115             st.interrupt();
116         } else {
117             //On informe au nouveau client que la connexion est
             acceptée
118             text = "SIGNIN ";
119             text += st.getUserName();
120             st.sendMessage(text);
121
122             //On envoie la liste des utilisateurs deja connecté au
             client
123             for (ServerThread s : serverThreads) {
124                 text = "SIGNIN ";
125                 text += s.getUserName();
126                 st.sendMessage(text);
127             }
128
129
130             serverThreads.add(st);
131
132             //On informe tous utilisateurs de la connexion d'un
             nouveau client
133             text = "SIGNIN ";
134             text += st.getUserName();
135             message = new Message(server.getServerName(),"all",text);
136             sendMessage(message);
137
138             server.refreshClientsNumber();
139         }
140     }
141
142     /**
```

```
143         * Methode permettant de notifier la déconnexion d'un utilisateur.
144         * @param st Thread d'écoute côté serveur de l'utilisateur qui se déconnecte.
145         */
146        public void signout(ServerThread st) {
147            String text;
148            text = "SIGNOUT ";
149            text += st.getUserName();
150            Message message = new Message(server.getServerName(), "all", text);
151            sendMessage(message);
152            st.interrupt();
153            serverThreads.remove(st);
154            server.refreshClientsNumber();
155        }
156
157        /**
158         * Getter permettant de recuperer le thread d'écoute côté serveur de
159         * l'utilisateur ayant le nom "username".
160         * @param username Nom de l'utilisateur.
161         * @return Thread d'écoute côté serveur du client ayant le nom "username".
162         */
163        public ServerThread getServerThread(String username) {
164            ServerThread st = null;
165            for(ServerThread s : serverThreads) {
166                if (username.compareTo(s.getUserName())==0) {
167                    st = s;
168                    break;
169                }
170            }
171            return st;
172        }
173
174        /**
175         * Méthode permettant de savoir si un utilisateur ayant le nom d'utilisateur
176         * "username" est déjà connecté.
177         * @param username Nom de l'utilisateur.
178         * @return True si ouiu, false sinon.
179         */
180        public boolean contains(String username) {
181            return getServerThread(username)!=null;
182        }
183
184    }
185
```