```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.
```



```java
     */
    BufferedReader socIn;

    /**
     * Thread de lecture du client.
     */
    ClientThread clientThread;

    /**
     * Constructeur.
     */
    public Client() {
        try {
            state = false;
            clientChatView = new ClientChatView(this);
            serverAdress = null;
            socket = null;
            socIn = null;
            socOut = null;
            serverPort = 1099;
            clientChatView.setVisible(true);
        } catch (Exception e) {
            System.err.println("Client exception : " + e.toString());
        }
    }

    /**
     * Getter.
     * @return Non d'utilisateur.
     */
    public String getUserName() {
        return userName;
    }

    /**
     * Getter.
     * @return Adresse du serveur.
     */
    public String getServerAdress() {
        return serverAdress;
    }

    /**
     * Getter.
     * @return Port de communication.
     */
    public int getServerPort() {
        return serverPort;
    }

    /**
     * Setter.
```



```java
     * @param name Nom d'utilisateur.
     */
    public void setUserName(String name) {
        userName = name;
    }

    /**
     * Setter.
     * @param adress Adresse du serveur.
     */
    public void setServerAdress(String adress) {
        serverAdress = adress;
    }

    /**
     * Setter.
     * @param port Port de communication.
     */
    public void setServerPort(int port) {
        serverPort = port;
    }

    /**
     * Methode permettant d'afficher un message sur la vue.
     * @param m Message.
     */
    public void displayMessage(Message m){
        clientChatView.displayMessage(m);
    }

    /**
     * Methode permettant d'afficher un message sur la vue.
     * @param m Message.
     */
    public void displayMessage(String m){
        clientChatView.displayMessage(m);
    }

    /**
     * Methode notifiant la connexion d'un autre utilisateur.
     * Ajoute le nom du client à la vue
     * @param client Client
     */
    public void newUserConnected(String client) {
        clientChatView.addUser(client);
        displayMessage(new Message("server","all",client+" signed in ...."));
    }

    /**
     * Methode notifiant la déconnexion d'un autre utilisateur.
     * Enlève le nom du client à la vue
```



```java
     * @param client Client
     */
    public void userDisconnected(String client) {
        clientChatView.removeUser(client);
        if(client.compareTo(userName)==0) {
            clientThread.interrupt();
            try {
                socOut.close();
                socIn.close();
                socket.close();
            } catch (Exception e) {
                System.err.println("Client exception : "+
                e.toString());
                System.exit(1);
            }
            clientThread = null;
            state = false;
            clientChatView.refresh();
        } else {
            displayMessage(new Message("server","all",client+"
            signed in ..."));
        }
    }

    /**
     * Methode permettant d'envoyer un message au serveur et qui
     serra retransmi
     * au "receiver".
     * @param receiver Destinataire. ("all" si destiné à tous les
     utilisateurs)
     * @param m Texte du message.
     */
    public void sendMessage(String receiver,String m) {
        socOut.println("SENDTO " + receiver + " CONTENT " + m);
    }

    /**
     * Methode permettant de se connecter au serveur. Créer un
     ClientThread
     * permettant la recepetion de messages.
     * @param sAdress Adresse du serveur.
     * @param sPort Port de communication.
     * @param name Nom de l'utilsateur.
     * @return Retourne true si reussi, false sinon.
     */
    public boolean connect(String sAdress, String sPort, String name) {
        try {
//            serverAdress = sAdress;
            state = false;
            userName = name;
            serverPort = Integer.parseInt(sPort);//gerer exception
```

```
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
```



```
1   /*
2    * To change this license header, choose License Headers in Project
     Properties.
3    * To change this template file, choose Tools | Templates
4    * and open the template in the editor.
5    */
6   package chat.client;
7
8   import chat.protocol.Message;
9   import java.io.BufferedReader;
10
11
12
13  /**
14   * Thread client d'écoute de socket.
15   * @author Nico
16   */
17  public class ClientThread extends Thread{
18      /**
19       * Réference 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 Réference 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
         serveur et
40       * en notifie le "client".
41       * Traite les lignes que le BufferReader socIn (de la socket)
         reçoit du
42       * serveur et analyse ce qu'elle reçoit et fait le traitement
         approprié
43       * selon si c'est un message informatif(quelqu'un s'est connecté
         ou
44       * deconnecté) ou un message d'un client.
45       */
46      @Override
47      public void run(){
```



```
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 : " +
                e.toString());
56          }
57      }
58
59      /**
60       * Methode traitant uns ligne recus par le flux d'entrée "socIn".
61       * @param line Ligne lu.
62       */
63      private void treatLine(String line){
64          if(line.contains("SIGNIN")) {
65              String[] splits = line.split("SIGNIN ");
66              client.newUserConnected(splits[1]);
67          } else if(line.contains("SIGNOUT")) {
68              String[] splits = line.split("SIGNOUT ");
69              client.userDisconnected(splits[1]);
70          } else if(line.contains("MESSAGE FROM") &&
            line.contains("TO") && line.
71                  contains("CONTENT")) {
72              String[] splits1 = line.split("MESSAGE FROM ");
73              String[] splits2 = splits1[1].split(" TO ");
74              String[] splits3 = splits2[1].split(" CONTENT ");
75              String sender = splits2[0];
76              String receiver = splits3[0];
77              String textMessage = splits3[1];
78              client.displayMessage(new
                Message(sender,receiver,textMessage));
79          }
80      }
81  }
82
```



```
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'expediteur 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 meassage.
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 meassage.
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'envoyeur 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.
```

```
     * @return Texte du meassage.
     */
    public String getTextMessage() {
        return textMessage;
    }
}
```



```
/*
 * 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 java.net.InetAddress;

/**
 * Classe representant un serveur d'un chat utilisant les sockets.
 * @author Nico
 */
public class Server {

    /**
     * Nom du serveur.
     */
    private String serverName;

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

    /**
     * Etat du serveur. True : connecté, false sinon.
     */
    private boolean state;

    /**
     * Thread d'écoute permettant la connexion d'utilisateurs et de transmettre
     * les messages à tous les utilisateurs.
     */
    private ServerMultiThread serverMultiThread;

    /**
     * Vue du serveur.
     */
    private ServerRunningView serverRunningView;

    /**
     * Constructeur.
     */
    public Server() {
       initialize("server",1099);
    }

    /**
     * Getter.
```



```
     * @return Port de communication.
     */
    public int getServerPort() {
        return serverPort;
    }

    /**
     * Getter.
     * @return Nom du serveur.
     */
    public String getServerName() {
        return serverName;
    }

    /**
     * Methode permettant d'initialiser le serveur.
     * @param sName Nom du serveur.
     * @param sPort Port de communication.
     */
    private void initialize(String sName, int sPort) {
        serverName = sName;
        serverPort = sPort;
        state = false;
        serverMultiThread = null;
        serverRunningView = new ServerRunningView(this);
        serverRunningView.setLocationRelativeTo(null);
        serverRunningView.setVisible(true);
    }

    /**
     * Methode permettant de lancer le serveur.
     * @param port
     */
    public void start(String port)
    {
        try {
            serverPort = Integer.parseInt(port);
            state = true;
            serverMultiThread = new ServerMultiThread(this);
            serverMultiThread.start();

            serverRunningView.diplayServerLocalAdress(InetAddress.getLocalHost()
                    .getHostAddress());

        } catch (Exception e) {
            System.err.println("Server exception : " + e.toString());
        }
    }

    /**
     * Methode permettant d'arreter le serveur.
```



```
     */
    public void stop() {
        state = false;
        if (serverMultiThread!=null) {
            serverMultiThread.interrupt();
        }
        refreshClientsNumber();
    }

    /**
     * Methode permettant de raffraichir le nombre de client sur la vue.
     */
    public void refreshClientsNumber() {
        if(serverMultiThread!=null) {
            serverRunningView.setClientNumber(serverMultiThread.
                    getClientsNumber());
        } else {
            serverRunningView.setClientNumber(0);
        }
    }

    /**
     * Methode permettant de savoir si le serveur est en train de tourner.
     * @return True si oui, false sinon.
     */
    public boolean isRunning() {
        return state;
    }

}
```

```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
```



```java
     ServerThread à
     * la connexion d'un utilisateur et lance ce ServerThread.
     */
    public void run(){
        try {
            while (server.isRunning()) {
                Socket clientSocket = listenSocket.accept();

                ServerThread serverThread = new ServerThread(server,this,
                        clientSocket);
                serverThread.start();
            }
        } catch (Exception e) {
            System.err.println("ServerMultiThread exception : " +
            e.toString());
        }
    }

    /**
     * Méthode d'intéruption du thread. Ferme la socket server et appelle la
     * méthode interrupt() de chaque ServerThreads (d'écoute côté serveur de
     * chaque utilisateurs).
     */
    @Override
    public void interrupt(){
        try {
            super.interrupt();
            listenSocket.close();
            for(ServerThread st : serverThreads) {
                st.interrupt();
            }
            serverThreads.removeAll(serverThreads);
        } catch (Exception e) {
            System.err.println("ServerMultiThread exception : " +
            e.toString());
        }
    }

    /**
     * Methode permettan d'envoyer un message.
     * @param m Message.
     */
    public void sendMessage(Message m) {
        if (m.getReceiver().compareTo("all") == 0) {
            for(ServerThread st : serverThreads) {
                st.sendMessage(m.getTextMessage());
            }
        } else {
            for(ServerThread st : serverThreads) {
```



```java
                if(st.getUserName().compareTo(m.getReceiver())==0 ||
                        st.getUserName().compareTo(m.getSender())==0) {
                    st.sendMessage(m.getTextMessage());
                }
            }
        }
    }

    /**
     * Methode permettant de notifier la connexion d'un utilisateur.
     * @param st Thread d'écoute côté serveur de l'utilisateur qui se déconnecte.
     */
    public void signin(ServerThread st) {
        Message message;
        String text;
        if(server.getServerName().compareTo(st.getUserName()) == 0 ||
                contains(st.getUserName())){
            text = "SIGNOUT ";
            text += st.getUserName();
            st.sendMessage(text);
            st.interrupt();
        } else {
            //On informe au nouveau client que la connexion est acceptée
            text = "SIGNIN ";
            text += st.getUserName();
            st.sendMessage(text);

            //On envoie la liste des utilisateurs deja connecté au client
            for (ServerThread s : serverThreads) {
                text = "SIGNIN ";
                text += s.getUserName();
                st.sendMessage(text);
            }


            serverThreads.add(st);

            //On informe tous utilisateurs de la connexion d'un nouveau client
            text = "SIGNIN ";
            text += st.getUserName();
            message = new Message(server.getServerName(),"all",text);
            sendMessage(message);

            server.refreshClientsNumber();
        }
    }

    /**
```



```java
     * Methode permettant de notifier la déconnexion d'un utilisateur.
     * @param st Thread d'écoute côté serveur de l'utilisateur qui se déconnecte.
     */
    public void signout(ServerThread st) {
        String text;
        text = "SIGNOUT ";
        text += st.getUserName();
        Message message = new Message(server.getServerName(), "all", text);
        sendMessage(message);
        st.interrupt();
        serverThreads.remove(st);
        server.refreshClientsNumber();
    }

    /**
     * Getter permettant de recuperer le thread d'écoute côté serveur de
     * l'utilisateur ayant le nom "username".
     * @param username Nom de l'utilisateur.
     * @return Thread d'écoute côté serveur du client ayant le nom "username".
     */
    public ServerThread getServerThread(String username) {
        ServerThread st = null;
        for(ServerThread s : serverThreads) {
            if (username.compareTo(s.getUserName())==0) {
                st = s;
                break;
            }
        }
        return st;
    }

    /**
     * Méthode permettant de savoir si un utilisateur ayant le nom d'utilisateur
     * "username" est déjà connecté.
     * @param username Nom de l'utilisateur.
     * @return True si ouiu, false sinon.
     */
    public boolean contains(String username) {
        return getServerThread(username)!=null;
    }

}
```

```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.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;

/**
 *
 * @author Nico
 */
public class ServerThread extends Thread {
    /**
     * Référence sur le serveur.
     */
    private final Server server;

    /**
     * Référence sur la socket de communication.
     */
    private final Socket clientSocket;

    /**
     * Référence sur le thread parent.
     */
    private final ServerMultiThread serverMultiThread;

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

    /**
     * Flux de lecture de message.
     */
    BufferedReader socIn;

    /**
     * Nom de l'utilisateur relié.
     */
    private String userName;

    /**
     * COnstructeur.
     * @param s Référence sur le serveur.
```



```java
     * @param smt Référence sur le thread parent.
     * @param c Socket de communication.
     */
    public ServerThread(Server s,ServerMultiThread smt, Socket c) {
        server = s;
        clientSocket = c;
        serverMultiThread = smt;
        try {
            socIn = new BufferedReader(new InputStreamReader(clientSocket
                    .getInputStream()));
            socOut = new PrintStream(clientSocket.getOutputStream());
        } catch (Exception e) {
            System.err.println("ServerThread  exception : " + e.toString());
        }
    }

    /**
     * Getter.
     * @return Nom de l'utilisateur.
     */
    public String getUserName() {
        return userName;
    }

    /**
     * Méthode exécutée par le thread. Boucle tant que server.isRunning() est
     * true. Attend que le client envoi une ligne et la traite.
     */
    @Override
    public void run(){
        try {
            String line;
            while (server.isRunning()) {
                line = socIn.readLine();
                treatLine(line);
            }
        } catch (Exception e) {
            System.err.println("ServerThread  exception : " + e.toString());
        }
    }

    /**
     * Méthode d'intéruption du thread. Deconnecte l'utilisateur en lui envoyant
     * un message de deconnexion et ferme la sockets.
     */
    @Override
    public void interrupt() {
```



```java
        super.interrupt();
        try {
            sendMessage("SIGNOUT "+userName);
            socOut.close();
            socIn.close();
            clientSocket.close();
        } catch (Exception e) {
            System.err.println("Client exception : "+ e.toString());
        }
    }

    /**
     * Methode traitant une ligne recue par le flux d'entrée "socIn".
     * @param line Ligne lu.
     */
    private void treatLine(String line){
        if(line.contains("CONNECT")) {
            String[] splits = line.split("CONNECT ");
            userName = splits[1];
            serverMultiThread.signin(this);
        } else if(line.compareTo("QUIT")==0) {
            serverMultiThread.signout(this);
        } else if(line.contains("SENDTO") && line.contains("CONTENT")) {
            String[] splits1 = line.split("SENDTO ");
            String[] splits2 = splits1[1].split(" CONTENT ");
            String sender = userName;
            String receiver = splits2[0];
            String textMessage = "MESSAGE FROM " + sender + " TO " + receiver
                    + " CONTENT " + splits2[1];
            serverMultiThread.sendMessage(
                    new Message(sender,receiver,textMessage));
        }
    }

    /**
     * Méthode permettant d'envoyer un message au client.
     * @param m
     */
    public void sendMessage(String m) {
        try{
            socOut.println(m);
        } catch (Exception e) {
            System.err.println("ServerThread  exception : " + e.toString());
        }
    }
}
```