



**Ciências
ULisboa**

SEGURANÇA E CONFIABILIDADE

Relatório Projeto 1

Engenharia Informática – 2018/2019

Grupo 17

Diogo Nogueira, Nº 49435

Filipe Capela, Nº 50296

Filipe Silveira, Nº 49506

Índice

Balanço dos objetivos cumpridos	2
Mensagens e formato das mesmas.....	2
Configuração da Sandbox.....	3
Organização do Software	4
Requisitos de Segurança	6

Balanço dos objetivos cumpridos

Para a primeira fase do projeto da disciplina de Segurança e Confiabilidade, foi-nos pedido para criar e desenvolver uma aplicação do tipo cliente-servidor, em que fossem permitidas ligações de múltiplos clientes a um servidor, onde estes pudessem comunicar e fazer partilha de ficheiros e de mensagens entre si, e com o servidor.

Durante o tempo estipulado para elaboração da primeira fase, conseguimos desenvolver com sucesso todos os aspetos da aplicação, podendo então os clientes comunicar com o servidor, apresentando os resultados esperados e, em caso de erro, este mesmo é apresentado para que o cliente consiga aperceber-se do mesmo.

Por outro lado, foi-nos pedido para, utilizando ficheiros .policy, restringir o acesso a diversos recursos (escrita de ficheiros, leitura de sockets, entre outros). Para este ponto criámos dois ficheiros, um para o servidor e outro para o cliente. Tentámos correr a nossa aplicação utilizando estes ficheiros, mas esta apresentava diversos erros, como o apresentado em seguida:

```
<terminated> MsgFileServer [Java Application] C:\Program Files\AdoptOpenJDK\jdk-11.0.2+9\bin\javaw.exe (23/03/2019, 23:10:10)
servidor: main
java.security.policy: error adding Permission, java.io.FilePermission:
    java.lang.Exception: substitution value, java.class.path, unsupported
Exception in thread "main" java.security.AccessControlException: access denied ("java.io.FilePermission" "servidor" "write")
    at java.base/java.security.AccessControlContext.checkPermission(AccessControlContext.java:472)
    at java.base/java.security.AccessController.checkPermission(AccessController.java:895)
    at java.base/java.lang.SecurityManager.checkPermission(SecurityManager.java:322)
    at java.base/java.lang.SecurityManager.checkWrite(SecurityManager.java:752)
    at java.base/java.io.File.mkdir(File.java:123)
    at MsgFileServer.startServer(MsgFileServer.java:238
    at MsgFileServer.main(MsgFileServer.java:223)
```

Em suma, conseguimos criar a aplicação, mas não conseguimos restringir utilizando ficheiros .policy o acesso a recursos do sistema.

Mensagens e formato das mesmas

As mensagens trocadas pelos clientes têm um formato bastante simples. Visto que, na interpretação do grupo estas seriam apenas mensagens de texto, estas passar-se-iam pelo terminal do seguinte modo: <msg Utilizador1 ola, eu chamo-me João>, sendo Utilizador1 o nome do utilizador ao qual nos queremos dirigir, e “ola, eu chamo-me João” um exemplo de mensagem enviada para o mesmo.

Tomámos liberdade de definir deste modo o formato das mensagens pois se fosse necessário enviar mensagens de outro tipo (p. ex. um ficheiro de imagem ou semelhante), esta seria enviada utilizando a função download presente no programa.

O carácter destas mensagens pode ser informativo, tal como no caso em que o Utilizador1 deseja informar o Utilizador2 de que fez download de um ficheiro da área do servidor do Utilizador2. O primeiro pode assim enviar uma mensagem de texto a informar a ocorrência. Estas mensagens podem também ser apenas usadas para comunicação “tipo chat”, ou seja, podemos deixar uma mensagem e esperar por uma resposta passado um tempo.

De realçar que, quando se executa o comando “collect”, este vai apresentar o utilizador que enviou a mensagem, seguido da mensagem que este mesmo enviou, como segue no exemplo da imagem:

```
collect
|joao disse: ola tio tudo bem?

Não tem mais mensagens para ler!
```

De modo a armazenar as mensagens, e quem as enviou, cada utilizador tem, na sua área do servidor, um ficheiro que contém as mensagens a serem apresentadas no terminal, para que, assim que o utilizador execute o comando “collect”, o ficheiro seja limpo e as mensagens não sejam novamente guardadas após aparecerem no terminal).

Configuração da Sandbox

De forma a configurar a nossa sandbox criámos então dois ficheiros, que foram entregues juntamente com o zip, denominados client.policy e server.policy. Nestes consta um conjunto de instruções que restringem o acesso a recursos como os sockets, os ficheiros e as threads, tornando a nossa aplicação mais segura. Infelizmente não conseguimos utilizar estes ficheiros pois não conseguimos perceber o path correto a utilizar para poder restringir o acesso aos ficheiros MsgFileServer.java e MsgFile.java.

Não obstante, segue uma apresentação das permissões que escrevemos nestes ficheiros, e uma explicação das mesmas:

Do lado do servidor:

- java.net.SocketPermission "*", "listen, accept";
 - De modo a poder receber pedidos do socket e a aceitar a ligação feita pelo cliente.
- java.io.FilePermission "file:\SegC-grupo17-proj1*", "read, write, delete, execute";
 - Para podermos criar as pastas “Servidor” e “Clientes”, seria então necessário dar ao servidor permissões de escrita, leitura, remoção e execução para poder satisfazer os pedidos dos clientes.
- java.io.FilePermission "file:\SegC-grupo17-proj1\servidor*", "read, write, delete, execute";
 - Visando poder criar os ficheiros do lado do servidor (p. ex. dar store a um ficheiro) seria então necessário dar ao servidor permissões de escrita, leitura, remoção e execução para poder satisfazer os pedidos dos clientes.
- java.io.FilePermission "file:\SegC-grupo17-proj1\clientes*", "read, write, delete, execute";
 - De forma a poder criar os ficheiros do lado do servidor (neste caso apenas seria para armazenar os ficheiros resultantes do comando “Download”) seria então necessário dar ao servidor permissões de escrita, leitura, remoção e execução para poder satisfazer os pedidos dos clientes.
- java.lang.RuntimePermission "modifyThread";
 - É necessário dar estas permissões para o servidor poder manipular threads.

Do lado do cliente:

- java.io.FilePermission "file:\SegC-grupo17-proj1\clientes*", "read, write";
 - Necessário que o cliente consiga escrever ficheiros do lado do mesmo para poder armazenar os ficheiros que deseja transferir do servidor.
- java.net.SocketPermission "*", "connect, resolve";
 - Necessário para que o cliente se consiga conectar ao socket do servidor.

Organização do Software

A organização do programa rege-se do seguinte modo:

- No ficheiro MsgFileServer foram criadas três classes, nomeadamente:
 - A classe Utilizador, que contém um construtor de um utilizador, sendo que este tem três parâmetros, o seu ID (ou seja, o seu nome de utilizador), uma lista com os ficheiros que o mesmo tem armazenados no servidor, e ainda uma lista com os utilizadores que este utilizador tem como trusted. Ainda nesta classe Utilizador existem vários métodos para poder operar com estes mesmos parâmetros.
 - A classe MsgFileServer é então responsável pela componente main do nosso programa, sendo esta muito simples, pois apenas é necessário que esta initialize o nosso servidor, inicializando a ligação feita por um socket e criando por fim threads para cada cliente com a ajuda da nossa classe ServerThread.
 - A classe ServerThread cria uma thread para cada cliente, podendo assim fazer conexão com um cliente que se liga. É nesta classe que lemos a informação passada pela inputStream do socket para o servidor, e desta maneira executar as funções pedidas (store, list, users, etc...). Para tal, o mais lógico seria então criar um switch-case de modo a, dependendo daquilo que o cliente passasse para o servidor, este faria operações diferentes.

Para desempenhar as funções pedidas, seguiu-se a seguinte metodologia.

Para cada utilizador cria-se uma lista com os ficheiros que este tem no servidor e uma outra lista que contém os trusted users deste utilizador.

Store: Enviar o ficheiro a partir da área do cliente, para o servidor, escrevendo bytes que serão lidos pelo mesmo, escrevendo um novo ficheiro na área do cliente no servidor.

List: Passar pela rede o nome de ficheiros guardados no servidor, contidos na lista de ficheiros do utilizador corrente.

Users: Transmitir pela rede o conteúdo de uma lista de users, que contém a informação do ficheiro UtilizadoresRegistados.txt.

Trusted: Ler a lista dos trusted users do utilizador corrente, imprimindo o seu conteúdo no terminal (Verificando se o utilizador existe, que ainda não é amigo e que não é dado o nome do próprio).

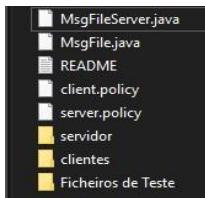
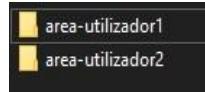
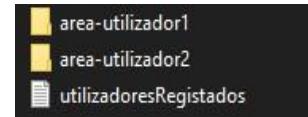
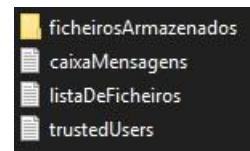
Untrusted: Apenas é necessário remover o nome do utilizador passado da lista de trusted users do cliente (Tem de ser amigo previamente).

Download: Fazer a verificação dos parâmetros (Utilizador tem de existir e ser trusted do utilizador cujo ID foi passado) e ler o ficheiro da área do utilizador no servidor e transmitir pela rede para o cliente que requisitou o download.

Msg: Fazer a verificação dos parâmetros (Utilizador tem de existir e ser trusted user) e escrever o conteúdo da mensagem para o ficheiro caixaMensagens.txt do destinatário da mensagem.

Collect: Passar pela rede o conteúdo do ficheiro caixaMensagens.txt para o cliente, limpando o mesmo.

- No ficheiro MsgFile foi criada apenas uma classe, de nome próprio:
 - Nesta temos a função main, onde tudo acontece. Após uma primeira fase de autenticação o cliente vai enviar comandos para o servidor, e recebe posteriormente a sua resposta, tratando de notificar o utilizador das mudanças que possam ter ocorrido ou dos erros que eventualmente possam ter sido despoletados pelo programa.
- Os diretórios gerados pelo nosso programa são os seguintes:
 - Pasta servidor:
 - Dentro desta pasta serão criadas tantas pastas quantos utilizadores estiverem registados no sistema, denominadas “area-XXXX” onde XXXX corresponde ao nome do utilizador, sendo esta gerada aquando do registo do utilizador.
 - Dentro de cada uma das pastas da área de utilizador serão gerados três ficheiros, caixaMensagens.txt, listaDeFicheiros.txt e trustedUsers.txt, onde são guardadas as mensagens recebidas, onde estão contidos os nomes dos ficheiros armazenados no servidor (por este utilizador) e onde estão contidos os nomes dos trusted users deste utilizador, respetivamente. É ainda criada uma pasta ficheirosArmazenados, onde ficarão guardados os ficheiros stored pelo cliente através do comando “store <files>”.
 - Ficheiro utilizadoresRegistados.txt que contém uma listagem dos utilizadores registados bem como as suas palavras-passe.
 - Pasta clientes:
 - Nesta pasta serão colocados os ficheiros que se querem armazenar no servidor manualmente, de modo a simular a área local do cliente.
 - Será apenas criada a pasta ficheirosTransferidos onde serão guardados os ficheiros transferidos da área do servidor de outros utilizadores.
 - Pasta Root do programa:
 - Criação das pastas “servidor” e “Clientes” aquando do primeiro arranque do servidor



De modo a informar o cliente dos erros que possam ocorrer (p. ex. enviar uma mensagem a um utilizador que não existe) o servidor trata de enviar essa mensagem pelo socket, visando abranger e tratar todos os erros que possam ocorrer ao inserir determinado comando. Esta mensagem é lida pelo cliente através do socket e impressa no terminal.

Para garantir o bom funcionamento da nossa aplicação, tomámos a liberdade de, primeiramente, tornar o acesso aos ficheiros algo sincronizado, prevenindo que a informação fosse corrompida ao ler dos ficheiros (p. ex. lista de utilizadores registados no sistema). Para tal criámos e utilizámos semáforos, para dar permissão de escrita ou leitura de um ficheiro apenas a um utilizador de cada vez.

Requisitos de Segurança

Para proteger a informação dos utilizadores, a palavra-passe dos mesmos não está guardada dentro da classe Utilizador ou em qualquer uma das outras do ficheiro MsgFileServer, não permitindo assim que, um cliente ao estar ligado ao servidor, tenha acesso à palavra passe deste utilizador se este assim o pretender. Sempre que um utilizador se autentique no servidor, este não tem maneira de conseguir, através do mesmo, informação sobre a palavra-passe. Assim, o cliente é protegido de um possível ataque à sua conta, pois assim um indivíduo “mal-intencionado” não poderia obter a palavra-passe do cliente através do servidor. Para garantir isto apenas foi necessário não incluir a informação da palavra-passe no construtor ou em qualquer outro método da classe.

De maneira a tornar a troca de mensagens mais seguras, estas poderiam ser encriptadas, para que, se alguém estivesse “à escuta” do canal de troca de mensagens, não pudesse saber o conteúdo das mesmas facilmente.

Para aumentar a confidencialidade do nosso sistema e dos seus utilizadores, poder-se-ia encriptar os ficheiros no servidor, e as respetivas áreas remotas, para que, se alguma entidade acedesse à base de dados do servidor, não pudesse obter informação dos ficheiros armazenados.

A troca de ficheiros pela rede (caso do store ou download), deveria ser segmentada, e não enviada como um todo, para que, se alguém intercetasse pacotes trocados pelo cliente e o servidor, este alguém não poderia ter acesso aos ficheiros trocados, ficando apenas com alguns bytes deste ficheiro. Nesta fase ainda não foi implementado esta maneira de segmentação dos ficheiros, pois pensamos que no próximo seria mais pertinente colocar isto em ação.

Considera-se um ponto de falha do nosso sistema o ficheiro UtilizadoresRegistados, presente no diretório de raiz do nosso servidor. Este contém a informação de todos os utilizadores e das suas palavras passes, tornando a nossa aplicação menos segura. Isto apenas acontece pois seguimos a implementação pedida pelos professores.

Exemplo de funcionamento da aplicação

```
Registo bem sucedido! Bem vindo "utilizador1"!
Por favor introduza um dos seguintes comandos:
Comandos disponíveis:
store <files>
list
remove <files>
users
trusted <trustedUserIDs>
untrusted <untrustedUserIDs>
download <userID> <file>
msg <userID> <msg>
collect
quit

trusted utilizador2
Utilizador: utilizador2
Adicionado ah trusted list com sucesso!
Por favor introduza um dos seguintes comandos:
Comandos disponíveis:
store <files>
list
remove <files>
users
trusted <trustedUserIDs>
untrusted <untrustedUserIDs>
download <userID> <file>
msg <userID> <msg>
collect
quit
```

```
servidor: main
A aceitar ligacoes

Utilizador "utilizador1" registou-se e autenticou-se!
Aguardando instrucoes...

Utilizador "utilizador2" registou-se e autenticou-se!
Aguardando instrucoes...
Utilizador "utilizador2" fez um pedido do tipo "trusted"
Aguardando instrucoes...
Utilizador "utilizador1" fez um pedido do tipo "trusted"
Aguardando instrucoes...
Utilizador "utilizador1" fez um pedido do tipo "msg"
Aguardando instrucoes...
Utilizador "utilizador2" fez um pedido do tipo "collect"
Aguardando instrucoes...
Utilizador "utilizador2" fez um pedido do tipo "msg"
Aguardando instrucoes...
```

Figura 3 - Terminal do Cliente

Figura 4 - Terminal do Servidor

```

1  ****
2  *
3  *  Seguranca e Confiabilidade 2018/19
4  *  Grupo SegC-017
5  *  Diogo Nogueira 49435
6  *  Filipe Silveira 49506
7  *  Filipe Capela 50296
8  *
9  ****
10
11 import java.io.BufferedInputStream;
12 import java.io.BufferedOutputStream;
13 import java.io.File;
14 import java.io.FileInputStream;
15 import java.io.FileOutputStream;
16 import java.io.IOException;
17 import java.io.ObjectInputStream;
18 import java.io.ObjectOutputStream;
19 import java.net.Socket;
20 import java.net.SocketException;
21 import java.util.Scanner;
22
23 /**
24  * Class MsgFile do cliente
25  * @author Diogo Nogueira 49435 Filipe Silveira 49506 Filipe Capela 50296
26  *
27 */
28 public class MsgFile {
29
30 /**
31  * Funcao que verifica se os argumentos sao validos
32  * @param args - argumentos
33  * @return true se forem validos
34  */
35 public static boolean notargs(String[] args){
36     String[] s = args[0].split(":", 2);
37     try{
38         Integer.parseInt(s[1]);
39     }catch(NumberFormatException e){
40         return true;
41     }
42     if(args.length < 2 || args.length > 3) {
43         return true;
44     }
45     return false;
46 }
47
48 /**
49  * Funcao main do cliente
50  * @param args - argumentos
51  */
52 public static void main(String[] args) {
53     String pw = null;
54     if(notargs(args)){
55         System.out.println("Argumentos invalidos");

```

```

56     System.out.println("Deve ser: MsgFile <serverAddress> <localUserID>
•      <password>");
57     System.out.println("Exemplo: MsgFile 127.1.1.1:12345 utilizador
•      palavrapass");
58 }
59 else {
60     Scanner reader = new Scanner(System.in);
61     Boolean faltaPass = args.length == 2;
62     if(!faltaPass) {
63         pw = args[2];
64     }
65     while (faltaPass) {
66         System.out.println("Por favor introduza a palavra-passe");
67         pw = reader.nextLine();
68         if (pw != null) {
69             faltaPass = false;
70         }
71     }
72     Socket soc = null;
73     try {
74
75         String[] s = args[0].split(":", 2);
76         soc = new Socket(s[0], Integer.parseInt(s[1]));
77         ObjectOutputStream outStream = new
78         •          ObjectOutputStream(soc.getOutputStream());
79         ObjectInputStream inStream = new ObjectInputStream(soc.getInputStream());
80         outStream.writeObject(args[1]);
81         outStream.writeObject(pw);
82
83         Boolean session = (boolean) inStream.readObject();
84         String username = args[1];
85         if(!session){
86             String erro = (String)inStream.readObject();
87             System.out.println(erro);
88             System.out.println("Sessao Terminada");
89         }
90         else {
91             String sucesso = (String)inStream.readObject();
92
93             File pastaCliente = new File("clientes/area-" + username);
94             if (!pastaCliente.isDirectory()) {
95                 pastaCliente.mkdir();
96             }
97             System.out.println(sucesso);
98         }
99
100        while(session){
101            System.out.println("Por favor introduza um dos seguintes comandos.");
102            System.out.println("Comandos disponiveis:");
103            System.out.println("store <files>\nlist\nremove
•      <files>\nusers\ntrusted <trustedUserIDs>");
104            System.out.println("untrusted <untrustedUserIDs>\ndownload <userID>
•      <file>\nmsg <userID> <msg>\ncollect\nquit\n");
105            String leitor = reader.nextLine();
106            String[] parametros = leitor.split(" ");

```

```

106     String comando = parametros[0];
107     switch(comando) {
108
109         case "store":
110             if(parametros.length < 2){
111                 System.out.println("Argumentos invalidos.");
112             }
113             else{
114                 outStream.writeObject(comando);
115                 outStream.writeObject(parametros.length-1);
116                 for (int i = 1; i < parametros.length; i++) {
117                     String nomeDoFicheiro = parametros[i];
118                     File fileToStore = new File("clientes/area-" + username + "/" +
119                     •
119                     nomeDoFicheiro);
120                     if(!fileToStore.exists()) {
121                         System.out.println("Erro: Ficheiro " + "\"" + nomeDoFicheiro
122                         + "\"" + " nao existe\nNao se esqueca de indicar a
123                         •
123                         extensão do ficheiro!\n");
124                         outStream.writeObject(false);
125                     }else {
126                         outStream.writeObject(true);
127                         outStream.writeObject(nomeDoFicheiro);
128                         if ((boolean) inStream.readObject()) {
129                             int fileLength = (int) fileToStore.length();
130                             outStream.writeObject(fileLength);
131                             byte[] mybytearray = new byte[fileLength];
132                             FileInputStream fis = new FileInputStream(fileToStore);
133                             BufferedInputStream bis = new BufferedInputStream(fis);
134                             bis.read(mybytearray,0,mybytearray.length);
135                             outStream.write(mybytearray, 0, mybytearray.length);
136                             outStream.flush();
137                             bis.close();
138                             if ((boolean)inStream.readObject()) {
139                                 System.out.println("Ficheiro: " + parametros[i] +
139                                 •
139                                 " guardado com sucesso!\n");
140                             }else {
141                                 System.out.println("Erro ao guardar o ficheiro " +
141                                 •
141                                 parametros[i] + "\n");
142                             }
143                         }
144                         System.out.println("Ficheiro " + parametros[i] + " jah estah
144                         •
144                         guardado no servidor \n");
145                     }
146                 }
147             }
148         }
149     }
150     break;
151
152     case "list":
153         outStream.writeObject(comando);
154         String nomeFicheiro;
155         int contador = 1;

```

```
156         while(((boolean) inStream.readObject())) {
157             nomeFicheiro = (String) inStream.readObject();
158             System.out.println("Ficheiro" + contador + " : " + nomeFicheiro +
159             "\n");
160             contador++;
161         }
162         System.out.println((String)inStream.readObject());
163         break;
164
165     case "remove":
166         if(parametros.length < 2){
167             System.out.println("Argumentos invalidos.");
168         }
169         else{
170             outStream.writeObject(comando);
171             outStream.writeObject(parametros.length-1);
172             for (int i = 1; i < parametros.length; i++) {
173                 outStream.writeObject(parametros[i]);
174                 if ((boolean)inStream.readObject()) {
175                     System.out.println("Ficheiro: " + parametros[i] + "\nRemovido
176                     com sucesso!");
177                 }
178             }
179         }
180         break;
181
182     case "users":
183         outStream.writeObject(comando);
184         String nomeUtilizadores;
185         int count = 1;
186         while(((boolean) inStream.readObject())) {
187             nomeUtilizadores = (String) inStream.readObject();
188             System.out.println("Utilizador " + count + ":" + +
189             nomeUtilizadores);
190             count++;
191         }
192         System.out.print("\n");
193         break;
194
195     case "trusted":
196         if(parametros.length < 2){
197             System.out.println("Argumentos invalidos.");
198         }
199         else{
200             outStream.writeObject(comando);
201             outStream.writeObject(parametros.length-1);
202             for (int i = 1; i < parametros.length; i++) {
203                 outStream.writeObject(parametros[i]);
204                 if ((boolean)inStream.readObject()) {
205                     System.out.println("Utilizador: " + parametros[i] +
206                     "\nAdicionado ah trusted list com sucesso!");
207                 }
208             }
209         }
210     }
```

```
207             }else {
208                 System.out.println((String) inStream.readObject());
209             }
210         }
211     }
212     break;
213
214     case "untrusted":
215         if(parametros.length < 2){
216             System.out.println("Argumentos invalidos.");
217         }
218     else{
219         outStream.writeObject(comando);
220         outStream.writeObject(parametros.length-1);
221         for (int i = 1; i < parametros.length; i++) {
222             outStream.writeObject(parametros[i]);
223
224             if ((boolean)inStream.readObject()) {
225                 System.out.println("Utilizador: " + parametros[i] +
226                     "\nremovido da trusted list com sucesso!");
227             }else {
228                 System.out.println((String) inStream.readObject());
229             }
230         }
231     }
232     break;
233
234     case "download":
235         if(parametros.length != 3){
236             System.out.println("Argumentos invalidos.");
237         }
238     else{
239         outStream.writeObject(comando);
240         outStream.writeObject(parametros[1]);
241         String nomeFicheiroDownloaded = parametros[2];
242         outStream.writeObject(nomeFicheiroDownloaded);
243
244         if ((boolean)inStream.readObject()) {
245             @SuppressWarnings("unused")
246             int fileLength = (int) inStream.readObject();
247             byte[] mybytearray = new byte[8192];
248             File pastaDownloads = new File("clientes/area-" + username +
249                 "/ficheirosTransferidos");
250             if (!pastaDownloads.isDirectory()) {
251                 pastaDownloads.mkdir();
252             }
253             FileOutputStream fos = new FileOutputStream("clientes/area-" +
254                 username +
255                 "/ficheirosTransferidos/" + nomeFicheiroDownloaded);
256             BufferedOutputStream bos = new BufferedOutputStream(fos);
257             int bytesRead = inStream.read(mybytearray, 0,
258                 mybytearray.length);
259             bos.write(mybytearray, 0, bytesRead);
260             bos.flush();
261             System.out.println("Ficheiro " + "\"" + nomeFicheiroDownloaded +
```

```

•
260           "\\" +  

261           " transferido com sucesso! " );  

262       bos.close();  

263   }else {  

264       System.out.println((String)inStream.readObject());  

265   }  

266   break;  

267  

268 case "msg":  

269 if(parametros.length < 3){  

270     System.out.println("Argumentos invalidos.");  

271 }  

272 else{  

273     outStream.writeObject(comando);  

274     outStream.writeObject(parametros[1]);  

275     if ((boolean)inStream.readObject()) {  

276         StringBuilder sb = new StringBuilder();  

277         for (int i = 2; i < parametros.length-1; i++) {  

278             sb.append(parametros[i] + " ");  

279         }  

280         sb.append(parametros[parametros.length-1]);  

281  

282         outStream.writeObject(sb.toString());  

283         System.out.println("Mensagem enviada com sucesso");  

284     }else {  

285         System.out.println((String)inStream.readObject());  

286     }  

287 }  

288 break;  

289 case "collect":  

290 if(parametros.length != 1){  

291     System.out.println("Argumentos invalidos.");  

292 }  

293 else{  

294     outStream.writeObject(comando);  

295     while ((boolean)inStream.readObject()) {  

296         String mensagem = (String) inStream.readObject();  

297         System.out.println(mensagem);  

298     }  

299     System.out.println((String)inStream.readObject());  

300 }  

301 break;  

302  

303 case "quit":  

304     outStream.writeObject(comando);  

305     session = false;  

306     if ((boolean) inStream.readObject()) {  

307         System.out.println("Sessão terminada");  

308     }  

309     break;  

310  

311 default:  

312     System.out.println("Comando introduzido estah errado!\n");  

313     break;  

...

```

```
314         }
315     }
316     soc.close();
317
318 } catch (SocketException e) {
319     System.out.println("Ligação perdida");
320     System.out.println("A desconectar");
321 }
322 catch (IOException e) {
323     System.err.println(e.getMessage());
324     System.exit(-1);
325 } catch (ClassNotFoundException e1) {
326     e1.printStackTrace();
327 }
328
329 }
330 }
331 }
332 }
```

```
1  ****
2  *
3  *  Seguran a e Confiabilidade 2018/19
4  *  Grupo SegC-017
5  *  Diogo Nogueira 49435
6  *  Filipe Silveira 49506
7  *  Filipe Capela 50296
8  *
9  ****
10
11 import java.io.BufferedInputStream;
12 import java.io.BufferedOutputStream;
13 import java.io.BufferedReader;
14 import java.io.BufferedWriter;
15 import java.io.File;
16 import java.io.FileInputStream;
17 import java.io.FileNotFoundException;
18 import java.io.FileOutputStream;
19 import java.io.FileReader;
20 import java.io.FileWriter;
21 import java.io.IOException;
22 import java.io.ObjectInputStream;
23 import java.io.ObjectOutputStream;
24 import java.io.Writer;
25 import java.net.ServerSocket;
26 import java.net.Socket;
27 import java.net.SocketException;
28 import java.util.Scanner;
29 import java.util.concurrent.Semaphore;
30 import java.util.ArrayList;
31 import java.util.List;
32
33 /**
34  * Class Utilizador
35  * @author Diogo Nogueira 49435 Filipe Silveira 49506 Filipe Capela 50296
36  *
37 */
38 class Utilizador{
39
40     private String id;
41     private List<String> files;
42     private List<String> trustedIDs;
43
44     /**
45      * Construtor que cria um utilizador
46      * @param id - id do cliente
47      * @param password - password do cliente
48      */
49     Utilizador(String id, String password){
50         this.id = id;
51         this.files = new ArrayList<>();
52         this.trustedIDs = new ArrayList<String>();
53
54         File areaUtilizador = new File("servidor/area-"+id);
55         if(areaUtilizador.isDirectory()) {
```

```

56     try {
57         Scanner allFilesScanner = new Scanner(new File("servidor/area-" + id +
58             "/listaDeFicheiros.txt"));
59         while (allFilesScanner.hasNext()){
60             files.add(allFilesScanner.next());
61         }
62         allFilesScanner.close();
63
64         Scanner trustedScanner;
65         trustedScanner = new Scanner(new File("servidor/area-" + id + "/"
66             "trustedUsers.txt"));
67         while (trustedScanner.hasNext()){
68             trustedIDs.add(trustedScanner.next());
69         }
70         trustedScanner.close();
71     } catch (FileNotFoundException e) {
72         e.printStackTrace();
73     }
74     else{
75         try {
76             areaUtilizador.mkdir();
77             File ficheiros = new File("servidor/area-" + id + "/"
78                 "listaDeFicheiros.txt");
79             ficheiros.createNewFile();
80             File trusted = new File("servidor/area-" + id + "/trustedUsers.txt");
81             trusted.createNewFile();
82             File caixaMensagens = new File("servidor/area-" + id + "/"
83                 "caixaMensagens.txt");
84             caixaMensagens.createNewFile();
85             File pastaFicheirosRecebidos = new File("servidor/area-" + id + "/"
86                 "ficheirosArmazenados");
87             pastaFicheirosRecebidos.mkdir();
88         } catch (IOException e) {
89             e.printStackTrace();
90         }
91     }
92     /**
93      * @return o id do utilizador
94      */
95     public String getId() {
96         return this.id;
97     }
98     /**
99      * @return os ficheiros do utilizador
100     */
101    public List<String> getFiles() {
102        return this.files;
103    }
104    /**
105     * @return a Lista de amigos do utilizador

```

```

107     */
108     public List<String> getTrustedIDs() {
109         return this.trustedIDs;
110     }
111
112     /**
113      * Funcao que verifica se um ficheiro esta na lista de ficheiros do utilizador
114      * @param nomeDoFicheiro - o nome do ficheiro a verificar
115      * @return true se o ficheiro estiver na lista de ficheiros do utilizador
116      */
117     public boolean containsFile(String nomeDoFicheiro) {
118         if(files.contains(nomeDoFicheiro)){
119             return true;
120         }
121         return false;
122     }
123
124     /**
125      * Funcao que adiciona um ficheiro na lista de ficheiros do utilizador
126      * @param nomeDoFicheiro - o nome do ficheiro a adicionar
127      * @return true se adicionar o ficheiro com sucesso
128      */
129     public boolean addFile(String nomeDoFicheiro){
130         try {
131             Writer output;
132             output = new BufferedWriter(new FileWriter("servidor/area-" + id +
133                     "/listaDeficheiros.txt", true));
134             output.append(nomeDoFicheiro + "\n");
135             output.close();
136         } catch (IOException e) {
137             e.printStackTrace();
138             return false;
139         }
140         files.add(nomeDoFicheiro);
141         return true;
142     }
143
144     /**
145      * Funcao que adiciona um utilizador ah lista de amigos do utilizador
146      * @param nome - o nome do utilizador a adicionar ah Lista
147      * @return true se adiciona o utilizador ah lista de amigos
148      */
149     public boolean addTrust(String nome){
150         if(trustedIDs.contains(nome)){
151             return false;
152         }
153         else{
154             trustedIDs.add(nome);
155             return true;
156         }
157     }
158
159     /**
160      * Funcao que remove um utilizador da lista de amigos do utilizador
161      * @param nome - o nome do utilizador a remover ah lista

```

```

162     * @return true se o utilizador foi removido com sucesso da lista de amigos
163     */
164     public boolean removeTrust(String nome){
165         if(trustedIDs.contains(nome)){
166             trustedIDs.remove(nome);
167             return true;
168         }
169         else{
170             return false;
171         }
172     }
173
174 /**
175 * Funcao que adiciona uma mensagem ah caixa de mensagens de um utilizador
176 * @param msg - a mensagem
177 * @param id - o id do utilizador
178 */
179 public void addMsg(String msg, String id) {
180
181     Writer output;
182     try {
183         output = new BufferedWriter(new FileWriter("servidor/area-"+ this.id +
184             "/caixaMensagens.txt", true));
185         output.append(id + " disse: " + msg + "\n");
186         output.close();
187     } catch (IOException e) {
188         e.printStackTrace();
189     }
190 }
191
192 /**
193 * Altera o id do utilizador
194 * @param id - o id do utilizador
195 */
196 public void setId(String id) {
197     this.id = id;
198 }
199 }
200
201
202 /**
203 * Class MsgFileServer
204 * @author Diogo Nogueira 49435 Filipe Silveira 49506 Filipe Capela 50296
205 *
206 */
207 public class MsgFileServer{
208
209     static Semaphore mutex = new Semaphore(1);
210
211     public ArrayList<Utilizador> users = new ArrayList<>();
212
213 /**
214 * Funcao main do MsgFileServer
215 * @param args - os argumentos
216 */

```

```

217 public static void main(String[] args) {
218
219     try{
220         int soc = Integer.parseInt(args[0]);
221         System.out.println("servidor: main");
222         MsgFileServer server = new MsgFileServer();
223         server.startServer(soc);
224     }catch(IOException e){
225         System.out.println("Argumentos invalidos");
226         System.out.println("Deve ser: MsgFileServer <port>");
227         System.out.println("Exemplo: MsgFileServer 12345");
228     }
229 }
230
231 /**
232 * Funcao que inicia o servidor
233 * @param soc - o numero do socket
234 * @throws FileNotFoundException - ficheiro nao encontrado
235 */
236 public void startServer (int soc) throws FileNotFoundException{
237
238     new File("servidor").mkdir();
239     new File("clientes").mkdir();
240     File utilizadoresRegistados = new File("servidor/
241     • utilizadoresRegistados.txt");
242     try {
243         utilizadoresRegistados.createNewFile();
244     } catch (IOException e1) {
245         e1.printStackTrace();
246     }
247     try {
248         FileReader fs = new FileReader(new File("servidor/
249     • utilizadoresRegistados.txt"));
250         BufferedReader br = new BufferedReader(fs);
251         String line;
252         while ((line = br.readLine()) != null) {
253             String[] up = line.split(":");
254             users.add(new Utilizador(up[0], up[1]));
255         }
256         br.close();
257     }
258     catch(IOException e) {
259         System.err.println(e.getMessage());
260         System.exit(-1);
261     }
262
263     ServerSocket sSoc = null;
264
265     try {
266         sSoc = new ServerSocket(soc);
267     } catch (IOException e) {
268         System.err.println(e.getMessage());
269         System.exit(-1);

```

```

270     }
271     System.out.println("A aceitar ligacoes");
272     Boolean b = true;
273     while(b){
274         try {
275             Socket inSoc = sSoc.accept();
276             ServerThread newServerThread = new ServerThread(inSoc);
277             newServerThread.start();
278         }
279         catch (IOException e) {
280             e.printStackTrace();
281             b = false;
282         }
283     }
284     try {
285         sSoc.close();
286     } catch (IOException e) {
287         e.printStackTrace();
288     }
289 }
290
291
292 /**
293 * Threads utilizadas para comunicacao com os clientes
294 * @author Diogo Nogueira 49435 Filipe Silveira 49506 Filipe Capela 50296
295 *
296 */
297 class ServerThread extends Thread {
298
299     private Socket socket = null;
300     public Utilizador utilizador;
301
302     /**
303      * @param inSoc - o socket
304      */
305     public ServerThread(Socket inSoc) {
306         socket = inSoc;
307     }
308
309
310     /*
311      * Funcao run da thread
312      */
313     public void run(){
314         try {
315             ObjectOutputStream outStream = new
316                 ObjectOutputStream(socket.getOutputStream());
317             ObjectInputStream inStream = new
318                 ObjectInputStream(socket.getInputStream());
319
320             String user = (String)inStream.readObject();
321             String passwd = (String)inStream.readObject();
322
323             FileReader fs = new FileReader(new File("servidor/

```

```

•     utilizadoresRegistados.txt"));
323    BufferedReader br = new BufferedReader(fs);
324    String line;
325    boolean autenticacao = true;
326    boolean alive = true;
327    try {
328        mutex.acquire();
329    } catch (InterruptedException e) {
330        e.printStackTrace();
331    }
332    while ((line = br.readLine()) != null && autenticacao) {
333        String[] userPass = line.split(":");
334        //User pertence ao sistema
335        if (userPass[0].equals(user)) {
336            //password coincide
337            if(userPass[1].equals(passwd)) {
338                outStream.writeObject(true);
339                outStream.writeObject("Autenticacao bem sucedida! Bem vindo "
340                    + "\"" + user + "\"" + "!\n");
341                System.out.println("Utilizador " + "\"" + user + "\"" + " "
342                    + "autenticou-se!");
343
344                for (Utilizador utilizadorCorrente : users) {
345                    if (utilizadorCorrente.getId().equals(user)) {
346                        utilizador = utilizadorCorrente;
347                    }
348                    autenticacao = false;
349
350                }
351                //password nao coincide
352            else {
353                //enquanto a palavra passe estiver errada
354                outStream.writeObject(false);
355                outStream.writeObject("Password errada!");
356                //fim da autenticacao
357                alive = false;
358                autenticacao = false;
359            }
360        }
361    }
362    mutex.release();
363    br.close();
364    //registo de novo utilizador no sistema
365    if (autenticacao) {
366        utilizador = new Utilizador(user,passwd);
367        users.add(utilizador);
368        Writer output;
369        output = new BufferedWriter(new FileWriter("servidor/
•         utilizadoresRegistados.txt"
370            , true));
371        output.append(user + ":" + passwd + "\n");
372        output.close();
373        outStream.writeObject(true);
374        outStream.writeObject("Registo bem sucedido! Bem vindo "

```

```

375         + "\"" + user + "\"" + "!\n");
376     System.out.println("\nUtilizador " + "\"" + user + "\"" + " registrou-
•         se e autenticou-se!");
377 }
378
379     while(alive){ //para manter vivo o servidor
380         System.out.println("Aguardando instrucoes...");
381         String comando = (String)inStream.readObject();
382         System.out.println("Utilizador " + "\"" + user + "\"" + " fez um
•             pedido do tipo \""
383             + comando + "\" \n");
384
385         switch (comando) {
386             case "store":
387                 int numElemToStore = (int) inStream.readObject();
388                 for (int i = 0; i < numElemToStore; i++) {
389                     if ((boolean) (inStream.readObject())){
390                         String nomeDoFicheiro = (String) inStream.readObject();
391                         if (!this.utilizador.containsFile(nomeDoFicheiro)){
392                             outStream.writeObject(true);
393                             @SuppressWarnings("unused")
394                             int fileLength = (int) inStream.readObject();
395                             byte[] mybytearray = new byte[8192];
396                             FileOutputStream fos = new FileOutputStream(
397                                 "servidor/area-"
+ this.utilizador.getId() +
398                                 "/ficheirosArmazenados/" + nomeDoFicheiro);
399                             BufferedOutputStream bos = new BufferedOutputStream(fos);
400                             int bytesRead =
401                                 inStream.read(mybytearray,0,mybytearray.length);
402                             int current = bytesRead;
403
404                             bos.write(mybytearray, 0 , current);
405                             bos.flush();
406
407                             outStream.writeObject(this.utilizador.addFile(nomeDoFicheiro));
408                             bos.close();
409                         }
410                         else {
411                             outStream.writeObject(false);
412                         }
413                     }
414                 }
415                 break;
416             case "list":
417                 int count = 0;
418                 for (String ficheiro : this.utilizador.getFiles()) {
419                     outStream.writeObject(true);
420                     outStream.writeObject(ficheiro);
421                     count++;
422                 }
423                 if (count == 0) {
424                     outStream.writeObject(false);
425                     outStream.writeObject("Nao tem ficheiros no servidor!\n");
426                 }
427             }

```

```

421
422     else {
423         outStream.writeObject(false);
424         outStream.writeObject("List terminado\n");
425     }
426     break;
427 case "remove":
428     int numElemToRemove = (int) inStream.readObject();
429     for (int i = 0; i < numElemToRemove; i++) {
430         String nomeFicheiroARemover = (String) inStream.readObject();
431         if (utilizador.containsFile(nomeFicheiroARemover)){
432             File ficheiroARemover = new File("servidor/area-
433             •
434                 "+this.utilizador.getId()+
435                     "/ficheirosArmazenados/" + nomeFicheiroARemover);
436             if(ficheiroARemover.delete()){
437                 this.utilizador.getFiles().remove(nomeFicheiroARemover);
438
439                 File inputFile = new File("servidor/area- " +
440                     this.utilizador.getId()
441                         + "/listaDeFicheiros.txt");
442                 File tempFile = new File("servidor/area- " +
443                     this.utilizador.getId()
444                         + "/myTempFile.txt");
445                 tempFile.createNewFile();
446
447
448                 BufferedReader reader = new BufferedReader(new
449                     FileReader(inputFile));
450                 BufferedWriter writer = new BufferedWriter(new
451                     FileWriter(tempFile));
452
453
454                 String lineToRemove = nomeFicheiroARemover;
455                 String currentLine;
456
457                 while((currentLine = reader.readLine()) != null) {
458                     String trimmedLine = currentLine.trim();
459                     if(trimmedLine.equals(lineToRemove)) continue;
460                     writer.write(currentLine +
461                         System.getProperty("line.separator"));
462                 }
463                 writer.close();
464                 reader.close();
465                 inputFile.delete();
466                 tempFile.renameTo(inputFile);
467                 outStream.writeObject(true);
468             }
469             else{
470                 outStream.writeObject(false);
471             }
472         }
473         break;
474 case "users":
475     for (Utilizador utilizador : users) {
476         outStream.writeObject(true);

```

```

470         outStream.writeObject(true);
471         outStream.writeObject(utilizador.getId());
472     }
473     outStream.writeObject(false);
474     break;
475   case "trusted":
476     int numElemToTrust = (int) inStream.readObject();
477     boolean done = false;
478     for (int i = 0; i < numElemToTrust; i++) {
479       done = false;
480       String nomeDaPessoa = (String) inStream.readObject();
481       if (nomeDaPessoa.equals(this.utilizador.getId())) {
482         outStream.writeObject(false);
483         outStream.writeObject("O utilizador eh igual ao seu!");
484       }
485     }
486   else {
487     for (Utilizador utilizador : users) {
488       if (utilizador.getId().equals(nomeDaPessoa)){
489         boolean resultado = this.utilizador.addTrust(nomeDaPessoa);
490         if (resultado){
491           Writer output;
492           output = new BufferedWriter(new FileWriter("servidor/area-
493           * 
494           this.utilizador.getId() + "/trustedUsers.txt", true));
495           output.append(nomeDaPessoa + "\n");
496           output.close();
497           outStream.writeObject(true);
498           done = true;
499           break;
500         }
501       }
502     }
503   }
504   if (!done) {
505     outStream.writeObject(false);
506     outStream.writeObject("Utilizador jah eh seu amigo!");
507     done = true;
508     break;
509   }
510 }
511 }
512 }
513 if (!done) {
514   outStream.writeObject(false);
515   outStream.writeObject("Utilizador nao estah registado no
516   * 
517   sistema!");
518 }
519 }
520 }
521 break;
522 case "untrusted":
523   int numElemToUntrust = (int) inStream.readObject();
524   for (int i = 0; i < numElemToUntrust; i++) {
525     String nomeDaPessoa = (String) inStream.readObject();
526     boolean resultado = utilizador.removeTrust(nomeDaPessoa);
527     if (resultado)
528

```

```
527
530     *           */
531     File inputFile = new File("servidor/area-" +
532         •           this.utilizador.getId() +
533             •           "/trustedUsers.txt");
534     File tempFile = new File("servidor/area-" +
535         •           this.utilizador.getId() +
536             •           "/myTempFile.txt");
537     tempFile.createNewFile();
538
539     BufferedReader reader = new BufferedReader(new
540         •           FileReader(inputFile));
541     BufferedWriter writer = new BufferedWriter(new
542         •           FileWriter(tempFile));
543
544     String lineToRemove = nomeDaPessoa;
545     String currentLine;
546
547     while((currentLine = reader.readLine()) != null) {
548         String trimmedLine = currentLine.trim();
549         if(trimmedLine.equals(lineToRemove)) continue;
550         writer.write(currentLine +
551             System.getProperty("line.separator"));
552     }
553     writer.close();
554     reader.close();
555     inputFile.delete();
556     tempFile.renameTo(inputFile);
557
558     outStream.writeObject(resultado);
559     if (!resultado) {
560         if (this.utilizador.getId().equals(nomeDaPessoa)) {
561             outStream.writeObject("Impossivel adicionar o nome " +
562                 •           nomeDaPessoa +
563                     ", ♦ igual ao seu!");
564         }
565         outStream.writeObject("O utilizador " + nomeDaPessoa + " nao
566         •           existe na sua lista de trustedUsers");
567     }
568
569     break;
570   case "download":
571     String userIDDownload = (String) inStream.readObject();
572     List<String> trustedUserIDDownload = null;
573
574     if (userIDDownload.equals(this.utilizador.getId())) {
575       outStream.writeObject(false);
576       outStream.writeObject("O utilizador eh igual ao seu!");
577     }else {
578       for (Utilizador utilizador : users) {
579         if (utilizador.getId().equals(userIDDownload)) {
580           trustedUserIDDownload = utilizador.getTrustedIDs();
581         }
582       }
583       if (trustedUserIDDownload == null) {
584         outStream.writeObject(false);
585         outStream.writeObject("O utilizador nao existel");
586       }
587     }
588   }
589 }
```

```
    ...
578    outStream.writeObject("O ficheiro nao existe!");
579 }else {
580     if (trustedUserIDDownload.contains(this.utilizador.getId())) {
581         String nomeFicheiroDownload = (String) inStream.readObject();
582         File fileToDownload = new File("servidor/area-" +
583             userIDDownload
584             + "/ficheirosArmazenados/" + nomeFicheiroDownload);
585         if(!fileToDownload.exists()) {
586             outStream.writeObject(false);
587             outStream.writeObject("O ficheiro nao existe!");
588         else {
589             outStream.writeObject(true);
590             FileInputStream fis = new FileInputStream(fileToDownload);
591             BufferedInputStream bis = new BufferedInputStream(fis);
592             int tamanhoFicheiro = (int) fileToDownload.length();
593             outStream.writeObject(tamanhoFicheiro);
594             byte[] mybytearray = new byte[tamanhoFicheiro];
595             bis.read(mybytearray, 0, tamanhoFicheiro);
596             outStream.write(mybytearray, 0, tamanhoFicheiro);
597             outStream.flush();
598             bis.close();
599         }
600     }
601     else {
602         outStream.writeObject(false);
603         outStream.writeObject("Nao eh amigo deste utilizador!");
604     }
605     break;
606 case "msg":
607     String userIDMsg = (String) inStream.readObject();
608     List<String> trustedUserIDMsg = null;
609     Utilizador amigo = null;
610     if (userIDMsg.equals(this.utilizador.getId())) {
611         outStream.writeObject(false);
612         outStream.writeObject("O utilizador eh igual ao seu!");
613     else {
614         for (Utilizador utilizador : users) {
615             if (utilizador.getId().equals(userIDMsg)) {
616                 trustedUserIDMsg = utilizador.getTrustedIDs();
617                 amigo = utilizador;
618             }
619         }
620         if (trustedUserIDMsg == null) {
621             outStream.writeObject(false);
622             outStream.writeObject("O utilizador nao existe!");
623         else {
624             if (trustedUserIDMsg.contains(this.utilizador.getId())) {
625                 outStream.writeObject(true);
626                 String mensagem = (String) inStream.readObject();
627                 amigo.addMsg(mensagem, this.utilizador.getId());
628             }
629             else {
630                 outStream.writeObject(false);
631                 outStream.writeObject("Nao eh amigo deste utilizador!");
632             }
633         }
634     }
635 }
```

```
632         }
633     }
634 }
635 break;

636 case "collect":
637     BufferedReader reader;
638     try {
639         reader = new BufferedReader(
640             new FileReader("servidor/area-"+
641                 this.utilizador.getId() + "/caixaMensagens.txt"));
642         String msg = reader.readLine();
643         if(msg == null) {
644             outStream.writeObject(false);
645             outStream.writeObject("\nA caixa de mensagens esta
646             vazia!\n");
647         }
648         else {
649             outStream.writeObject(true);
650             outStream.writeObject(msg);
651             while ((msg = reader.readLine()) != null) {
652                 outStream.writeObject(true);
653                 outStream.writeObject(msg);
654             }
655             outStream.writeObject(false);
656             outStream.writeObject("\nNao tem mais mensagens para ler!\n");
657         }
658         reader.close();
659     }

660     File inputFile = new File("servidor/area- +
661         this.utilizador.getId()
662         + "/caixaMensagens.txt");
663     File tempFile = new File("servidor/area- +
664         this.utilizador.getId()
665         + "/myTempFile.txt");
666     tempFile.createNewFile();
667     inputFile.delete();
668     tempFile.renameTo(inputFile);

669 } catch (IOException e) {
670     e.printStackTrace();
671 }
672 break;

673 case "quit":
674     alive = false;
675     outStream.writeObject(true);
676     System.out.println("Utilizador " + "\"" + this.utilizador.getId() +
677         "\"" + " terminou a sessao");
678     break;
679 }
680 outStream.close();
681 inStream.close();
682 socket.close();
683 } catch (SocketException e) {
```

```
--  
683     System.out.println("Cliente " + this.utilizador.getId() + " desconectado  
•         abruptamente");  
684     System.out.println("A aceitar liga??es");  
685 }  
686 catch (IOException e) {  
687     e.printStackTrace();  
688 } catch (ClassNotFoundException e) {  
689     e.printStackTrace();  
690 }  
691 }  
692 }  
693 }  
694 }
```