

**Relatório de Segurança e Confiabilidade**

Trabalho 1

Engenharia Informática

Grupo 51

João David n49448

João Marques n49038

Luís Moreira n49531

Indice

[Objectivos concretizados 3](#_Toc4405585)

[Organização do Software 3](#_Toc4405586)

[Segurança 4](#_Toc4405587)

[Formato das mensagens 5](#_Toc4405588)

[Configuração da sandbox 7](#_Toc4405589)

[Código do Projecto 9](#_Toc4405590)

# Objectivos concretizados

Todos os temas propostos no enunciado do projecto foram concretizados. Os seguintes tópicos no relatório explicam com melhor detalhe o que foi concretizado.

# Organização do Software

Na implementação do projecto, foram criados dois programas, o MsgFile (cliente) que recorre a uma linha de comandos para comunicar com o outro programa, o MsgFileServer (servidor). Para estabelecer um protocolo de comunicação, foi criada uma class Message que “carrega” o conteúdo das mensagens pela rede (explicada em maior detalhe mais à frente). De forma a organizar todas as classes, foram criados três packages, o package ‘client’ que contem as classes que implementam o MsgFile (inclusive), o package ‘server’ que contem todas as classes que implementam o MsgFileServer(inclusive), nestes dois packages, as classes contidas não “conhecem” outras classes fora do seu package, excepto as classes do terceiro package, ‘communication’, que contém a class Message referida anteriormente e o enumerado OpCode que define os códigos das operações.

Package client

* Client
* ClientConst
* MsgFile (contém o main)

O ‘MsgFile’ corresponde ao programa cliente, o objecto ‘Client’ é responsável por comunicar com o servidor, após o utilizador se ter conectado com sucesso é disponibilizado a linha de comandos, o parsing é feito pelo ‘MsgFile’. No ‘ClientConst’ temos os caminhos dos directórios definidos para guardar os ficheiros transferidos e mensagens colectadas do servidor.

Package communication

* Message
* OpCode

O ‘Message’ corresponde ao objecto que é enviado pela rede entre o cliente e o servidor, sempre que é enviada uma mensagem, um dos atributos de Message é sempre inicializado com um ‘OpCode’, trata-se de um enumerável que permite ao receptor da mensagem saber que tipo de dados estão disponíveis na mensagem, e no caso do servidor saber que operação o cliente está a tentar executar.

Package server

* ConcurrentManager
* Manager
* MsgFileServer (contém o main)
* ServerConst
* Skel

A classe ConcurrentManager é responsável por gerir os semáforos que permitem fazer exclusão mútua de ficheiros que podem ser acedidos em simultâneo por dois utilizadores conectados ao servidor. Na class Manager estão os métodos responsáveis por concretizar as operações requisitadas pelo cliente (escrever,ler,criar ficheiros). O MsgFileServer é o programa servidor, sempre que um cliente se conecta, é criado um thread para satisfazer os pedidos desse cliente. O ServerConst possui constantes com os nomes dos directórios definidos para guardar os ficheiros no lado do servidor. O Skel implementa a função invoke, que ao receber uma Message do cliente, com base no OpCode que vem lá dentro, chama a função respectiva implementada pelo ‘Manager’, com base no sucesso ou insucesso dessa operação, o invoke constrói uma nova Message de resposta para o cliente com o OpCode respectivo.

# Segurança

Para impedir múltiplos logins no servidor pelo mesmo utilizador, foi realizado o seu bloqueio, ou seja, apenas pode permanecer conectado ao servidor um utilizador para aquele “user name”.

O username e password são enviados pela rede como objectos String, isto é muito inseguro, alguém que consiga interceptar os pacotes enviados do cliente para o servidor quando faz login, consegue roubar os dados de acesso da conta. O mesmo acontece para as restantes mensagens, é enviado pela rede um objeto ‘Message’. As mensagens enviadas entre o cliente e servidor deviam estar cifradas, de forma a que o seu conteudo não possa ser acedido indevidamente.

A comunicação pode ser comprometida por um ataque Man in the Middle, visto que no programa concebido, não é utilizado nenhum protocolo de autenticação, como o TLC por exemplo.

# Formato das mensagens

Para concretizar o envio de mensagens foi criada uma class objecto ‘Message’ com diversos atributos que são usados dependendo do propósito da mensagem, para definir que conteúdo a mensagem contêm efectivamente foi criado um enum ‘OpCode’, sempre que uma mensagem é criada é definido um OpCode para que o receptor da mensagem saiba que atributos da mensagem foram inicializados, para que os possa aceder e concretizar a operação requerida, em certos casos a mensagem apenas contêm o OpCode inicializado visto não ser necessário mais informação para que seja feita a operação, por exemplo a operação “list” vinda do cliente para o servidor apenas leva o opCode com ‘LIST\_FILES’.

O OpCode também tem como função na resposta do servidor para o cliente definir se as operações foram bem concretizadas, ou em caso de erro informar o cliente o que aconteceu, por exemplo, ERR\_NOT\_REGISTERED, quando o utilizador introduzido como input numa operação não esteja registado no servidor.

Atributos de Message

* OpCode opCode
* OpCode[] arrCode
* String str
* String[] arrStr
* ArrayList<String> arrListStr
* ArrayList<Byte[]> arrListArrByte
* Byte[] arrByte

Operation Codes (OpCode)

* STORE\_FILES, LIST\_FILES, REMOVE\_FILES, USERS, TRUST\_USERS, UNTRUST\_USERS, DOWNLOAD\_FILE, SEND\_MSG, COLLECT\_MSG, END\_CONNECTION
* OP\_SUCCESSFUL: operação sobre no servidor bem-sucedida
* OP\_RES\_ARRAY: os resultados da operação vêm num array para cada parâmetro enviado para o servidor respectivamente, ex:”trusted fernando carlos rita” retorna um vetor de OpCode com length 3
* OP\_ERROR: operação no servidor não foi bem-sucedida, exception por exemplo
* ERR\_NOT\_REGISTERED: user enviado como parâmetro não está registado no servidor
* ERR\_ALREADY\_EXISTS: parâmetro introduzido já existe, por exemplo fazer “trusted fernando”, quando o fernando já tinha sido trusted anteriormente
* ERR\_NOT\_TRUSTED: o utilizador local não está na lista trusted do utilizador enviado como parâmetro
* ERR\_YOURSELF: utilizador enviado como parâmetro é o utilizador local
* ERR\_NOT\_FOUND: parâmetro enviado não encontrado, por exemplo, ao fazer download de um ficheiro inexistente, ou fazer untrusted de um utilizador que não foi trusted anteriormente

|  |  |  |
| --- | --- | --- |
| **Comando Utilizador** | **Mensagem de pedido** | **Mensagem de resposta** |
| store <files> | STORE\_FILES <arrListStr> <arrListArrBytes> | OP\_RES\_ARRAY <arrCode> |
|  |  | OP\_ERROR |
| list | LIST\_FILES | OP\_SUCCESSFUL <arrStr> |
|  |  | OP\_ERROR |
| remove <files> | REMOVE\_FILES <arrStr> | OP\_RES\_ARRAY <arrCode> |
|  |  |  |
| users | USERS | OP\_SUCCESSFUL <arrStr> |
|  |  | OP\_ERROR |
| trusted <trustedUserIDs> | TRUST\_USERS <arrStr> | OP\_RES\_ARRAY <arrCode> |
|  |  | OP\_ERROR |
| untrusted <untrustedUserIDs> | UNTRUST\_USERS <arrStr> | OP\_RES\_ARRAY <arrCode> |
|  |  | OP\_ERROR |
| download <userID> <file> | DOWNLOAD\_FILE <arrStr> | OP\_SUCCESSFUL <byteArray> |
|  | [0]=owner, [1]=fileName | ERR\_YOURSELF or ERR\_NOT\_FOUND or  ERR\_NOT\_TRUSTED or ERR\_NOT\_REGISTERED |
| msg <userID> <msg> | SEND\_MSG <arrStr> | OP\_SUCCESSFUL |
|  | [0]=receiver, [1]=textMsg | OP\_ERROR or ERR\_NOT\_TRUSTED  or ERR\_NOT\_REGISTERED |
| collect | COLLECT\_MSG <str> | OP\_SUCCESSFUL <arrListStr> |
|  |  | OP\_ERROR |
| exit | END\_CONNECTION | OP\_SUCCESSFUL |
|  |  |  |

A seguinte tabela apresenta os comandos e respectivas sintaxes, que os clientes podem requerer e a que o servidor pode responder.

Nas mensagens de pedido, a segunda linha corresponde à informação colocada no índex do array, enquanto que nas mensagens de resposta corresponde à mensagem de erro (a primeira linha corresponde à mensagem de resposta quando não há erro).

# Configuração da sandbox

## server.policy – Ficheiro de policy do servidor

As seguintes permissões permitem ao programa ler e escrever nas directorias que contêm os utilizadores registados no servidor, e os seus respectivos ficheiros e mensagens

permission java.io.FilePermission "${user.home}/MsgFileG51", "read,write";

permission java.io.FilePermission "${user.home}/MsgFileG51/server", "read,write";

Esta permissão também tem o parâmetro de delete, de forma a que os clientes ao usarem o comando remove, o servidor consiga apagar o respectivo ficheiro

permission java.io.FilePermission "${user.home}/MsgFileG51/server/-", "read,write,delete";

Corresponde ao endereço e porto que o servidor usa para estar à escuta de clientes que se queiram conectar, neste caso escolhemos o porto 23456, e o endereço de ip local

permission java.net.SocketPermission "127.0.0.1:23456", "listen,resolve";

Corresponde à permissão para aceitar uma conecção de clientes, tal como na permissão anterior, usa o endereço de ip local, e usa todas as portas a partir da 1024, corresponde as portas de aceitação de conecção

permission java.net.SocketPermission "127.0.0.1:1024-", "accept,resolve";

Como se pretende que o programa de servidor funcione em todos os sistemas operativos que suportem JVM, foram utilizadas propriedades do java para obter a home directory e o line separator, se isto não fosse utilizado, e fosse hardcoded, poderia ocorrer que em certos sistemas operativos, não fosse possível criar os directórios essências ao bom funcionamento do programa

permission java.util.PropertyPermission "user.home", "read";

permission java.util.PropertyPermission "line.separator", "read";

## client.policy – Ficheiro de policy do cliente

As seguintes permissões permitem ao programa ler e escrever nas directorias que contêm os downloads efectuados do servidor e mensagens que foram colectadas

permission java.io.FilePermission "${user.home}/MsgFileG51", "read,write";

permission java.io.FilePermission "${user.home}/MsgFileG51/client", "read,write";

Foi também dada a permissão de delete, visto que quando um utilizador faz download de um ficheiro do servidor com o mesmo nome de um outro ficheiro que já se encontre na sua directoria de download, este ficheiro é substituído pelo novo ficheiro acabado de transferir do servidor

permission java.io.FilePermission "${user.home}/MsgFileG51/client/-", "read,write,delete";

A aplicação cliente apenas tem permissão para se conectar ao ip correspondente à máquina local, e ao porto 23456, de igual forma ao que foi definido no servidor

permission java.net.SocketPermission "127.0.0.1:23456", "connect,resolve";

Analogamente ao que já foi explicado anteriormente no server.policy, estas permissões permitem que o programa funcione em diferentes sistemas operativos

permission java.util.PropertyPermission "user.home", "read";

permission java.util.PropertyPermission "line.separator", "read";

De forma a que o cliente possa fazer store no servidor de qualquer ficheiro na sua máquina, é necessário dar permissão de leitura a todos os ficheiros

permission java.io.FilePermission "<<ALL FILES>>", "read";

# Código do Projecto

## Package client

package client**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**ObjectInputStream**;**

**import** java**.**io**.**ObjectOutputStream**;**

**import** java**.**net**.**Socket**;**

**import** java**.**net**.**SocketException**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** communication**.**Message**;**

**import** communication**.**OpCode**;**

/\*\*

\* Responsible to connect to the Server!

\*/

public class Client **{**

private static final String CLASS\_NAME **=** Client**.**class**.**getName**();**

private final static Logger logger **=** Logger**.**getLogger**(**CLASS\_NAME**);**

private String username**;**

private String password**;**

private String host**;**

private int port**;**

private Socket echoSocket**;**

private ObjectInputStream in**;**

private ObjectOutputStream out**;**

private boolean isConnected**;**

/\*\*

\* Constructor

\* **@param** username

\* **@param** password

\* **@param** host

\* **@param** port

\* @requires all parameters != null

\*/

public Client**(**String username**,** String password**,** String host**,** int port**)** **{**

**this.**username **=** username**;**

**this.**password **=** password**;**

**this.**host **=** host**;**

**this.**port **=** port**;**

**this.**isConnected **=** **false;**

**}**

/\*\*

\* Get the UserName to Login

\* **@return** userName

\*/

public String getUsername**()** **{**

**return** **this.**username**;**

**}**

/\*\*

\* Connect to the Server

\* **@return** True if connected otherwise False

\*/

public boolean connect**()** **{**

//logger.log(Level.CONFIG, host + " " + port);

**try** **{**

echoSocket **=** **new** Socket**(**host**,** port**);**

in **=** **new** ObjectInputStream**(**echoSocket**.**getInputStream**());**

out **=** **new** ObjectOutputStream**(**echoSocket**.**getOutputStream**());**

logger**.**log**(**Level**.**CONFIG**,** "Connecting the user: " **+** username**);**

out**.**writeObject**(**username**);**

out**.**writeObject**(**password**);**

Message response **=** **(**Message**)** in**.**readObject**();**

**if(**response **!=** **null** **&&** response**.**getOpCode**()** **==** OpCode**.**OP\_SUCCESSFUL**){**

isConnected **=** **true;**

**return** isConnected**;**

**}else** **if(**response **!=** **null** **&&** response**.**getOpCode**()** **==** OpCode**.**ERR\_ALREADY\_EXISTS**){**

System**.**out**.**println**(**"You are already logged in: Existing Session"**);**

**}else** **if(**response **!=** **null** **&&** response**.**getOpCode**()** **==** OpCode**.**OP\_ERROR**)** **{**

System**.**out**.**println**(**"Username and password do not match"**);**

**}**

**}** **catch** **(** IOException **|** ClassNotFoundException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "Error to Connect the Client"**,** e**);**

**}**

**return** **false;**

**}**

/\*\*

\* Disconnect() from Socket and Streams

\* **@return** without errors True

\*/

public boolean disconnect**()** **{**

**try** **{**

**if(**out **!=** **null)** **{**

out**.**close**();**

**}**

**if(**in **!=** **null)** **{**

in**.**close**();**

**}**

**if(**echoSocket **!=** **null)** **{**

echoSocket**.**close**();**

**}**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "Error to close Socket/Streams"**,** e**);**

**return** **false;**

**}**

**return** **true;**

**}**

/\*\*

\* Connection with server

\* **@return** true if connected otherwise false

\*/

public boolean isConnected**()** **{**

**return** isConnected**;**

**}**

/\*\*

\* Send the message to Server

\* **@param** msgSent

\* **@return** if receive response return Message otherwise null

\* @requires msgSent != null

\*/

public Message sendMsg**(**Message msgSent**)** **{**

**try** **{**

out**.**writeObject**(**msgSent**);**

Object obj **=** in**.**readObject**();**

**if(**obj **instanceof** Message**){**

Message response **=** **(**Message**)** obj**;**

**return** response**;**

**}else{**

logger**.**log**(**Level**.**SEVERE**,** "Different object send by socket"**);**

**}**

**}catch** **(**SocketException e**)** **{**

System**.**out**.**println**(**"Connection lost with the server"**);**

**}catch** **(**IOException e**)** **{**

//logger.log(Level.SEVERE, "Error to resquest/response", e);

**}** **catch** **(**ClassNotFoundException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "Send message class not found"**,** e**);**

**}**

**return** **null;**

**}**

**}**

package client**;**

**import** java**.**io**.**File**;**

public class ClientConst **{**

public static final String FOLDER\_CLIENT **=** System**.**getProperty**(**"user.home"**)** **+** File**.**separator **+** "MsgFileG51" **+** File**.**separator **+** "client"**;**

public static final String FOLDER\_DOWNLOADS **=** "downloads"**;**

public static final String FILE\_NAME\_INBOX **=** "inbox.txt"**;**

public static final String FOLDER\_CLIENT\_USERS **=** FOLDER\_CLIENT **+** File**.**separator **+** "users"**;**

**}**

package client**;**

**import** java**.**io**.**File**;**

**import** java**.**io**.**FileOutputStream**;**

**import** java**.**io**.**FileWriter**;**

**import** java**.**io**.**IOException**;**

**import** java**.**nio**.**file**.**Files**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Arrays**;**

**import** java**.**util**.**Scanner**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** communication**.**Message**;**

**import** communication**.**OpCode**;**

public class MsgFile **{**

private static final String CLASS\_NAME **=** MsgFile**.**class**.**getName**();**

private final static Logger logger **=** Logger**.**getLogger**(**CLASS\_NAME**);**

//args <serverAddress> <localUserID> [password]

//127.0.0.1:23456 fernando pass123

public static void main**(**String**[]** args**)** **{**

Scanner sc **=** **new** Scanner**(**System**.**in**);**

**if(**args**.**length **==** 2 **||** args**.**length **==** 3**)** **{**

String**[]** hostPort **=** args**[**0**].**split**(**":"**);**

//System.out.print(hostPort[0] + hostPort[1]);

int port**;**

String passwd **=** **null;**

**if(**args**.**length **==** 3**)** **{**

passwd **=** args**[**2**];**

**}else** **{**

System**.**out**.**println**(**"Write your password please\n>>>"**);**

passwd **=** sc**.**nextLine**();**

**}**

**try** **{**

port **=** Integer**.**parseInt**(**hostPort**[**1**]);**

**}**

**catch** **(**NumberFormatException e**){**

System**.**out**.**println**(**"Client failed: Invalid serverAddress:" **+** e**.**toString**());**

sc**.**close**();**

**return;**

**}**

System**.**out**.**println**(**"Connecting to " **+** hostPort**[**0**]** **+** ":" **+** port **+** " ..."**);**

Client client **=** **new** Client**(**args**[**1**],** passwd**,** hostPort**[**0**],** port**);**

**if(**client**.**connect**())** **{**

System**.**out**.**println**(**"Connected to the server"**);**

System**.**out**.**println**(**"Welcome " **+** args**[**1**]);**

parser**(**client**);** //Process command lines

**}else{**

System**.**out**.**println**(**"Login failed"**);**

sc**.**close**();**

**return;**

**}**

client**.**disconnect**();**

System**.**out**.**println**(**"Disconnected"**);**

sc**.**close**();**

**}else** **{**

System**.**out**.**println**(**"Your args are not correct"**);**

System**.**out**.**println**(**"Valid args: <serverAddress> <localUserID> [password]"**);**

System**.**out**.**println**(**"example: 127.0.0.1:23456 fernando pass"**);**

**}**

**}**

/\*\*

\* @requires client != null && client.isConnected()

\* **@param** client

\*/

public static void parser**(**Client client**)** **{**

Scanner sc **=** **new** Scanner**(**System**.**in**);**

Message msgSent**;**

Message msgResponse**;**

boolean onLoop **=** **true;**

**while(**onLoop**)** **{**

System**.**out**.**print**(**">>>"**);**

String rawInput **=** sc**.**nextLine**();**

String**[]** parsedInput **=** rawInput**.**split**(**"(\\s)+"**);**

**switch** **(**parsedInput**[**0**])** **{**

**case** "store"**:** //store <files>

**if(**parsedInput**.**length **>** 1**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "store"**);**

File file**;**

ArrayList**<**String**>** nameFiles **=** **new** ArrayList**<>();**

ArrayList**<**Byte**[]>** byteFiles **=** **new** ArrayList**<>();**

ArrayList**<**String**>** nonExistent **=** **new** ArrayList**<>();**

**for(**int i **=** 1**;** i **<** parsedInput**.**length **;** i**++){**

file **=** **new** File**(**parsedInput**[**i**]);**

**if(**file**.**exists**()){**

**try** **{**

byteFiles**.**add**(**toObjects**(**Files**.**readAllBytes**(**file**.**toPath**())));**

nameFiles**.**add**(**file**.**getName**());**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "Não foi possivel converter para bytes"**,** e**);**

**}**

**}else** **{**

nonExistent**.**add**(**parsedInput**[**i**]);**

**}**

**}**

**if(**nameFiles**.**size**()** **!=** 0**)** **{**

msgSent **=** **new** Message**(**OpCode**.**STORE\_FILES**,** nameFiles**,** byteFiles**);**

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**}else** **{**

System**.**out**.**println**(**"Input files were not found : No files sent"**);**

**break;**

**}**

**if** **(**msgResponse **!=** **null** **&&** msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_RES\_ARRAY**)** **{**

OpCode**[]** arrCodes **=** msgResponse**.**getOpCodeArr**();**

int i **=** 0**;**

System**.**out**.**println**(**"--- Operation results ---"**);**

**for(**String str **:** nameFiles**)** **{**

**if(**arrCodes**[**i**]** **==** OpCode**.**OP\_SUCCESSFUL**)** **{**

System**.**out**.**println**(**str **+** " : " **+** "STORED"**);**

**}else** **{**

System**.**out**.**println**(**str **+** " : " **+** arrCodes**[**i**].**toString**());**

**}**

i**++;**

**}**

**if(**nonExistent**.**size**()** **!=** 0**)** **{**

**for(**String str **:** nonExistent**)** **{**

System**.**out**.**println**(**str**+** " : not found"**);**

**}**

**}**

System**.**out**.**println**(**"-------------------------"**);**

**}else** **{**

System**.**out**.**println**(**"ERROR: no answer from server"**);**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**case** "list"**:**

**if(**parsedInput**.**length **==** 1**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "list"**);**

msgSent **=** **new** Message**(**OpCode**.**LIST\_FILES**);**

//send message method

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**if** **(**msgResponse **!=** **null** **&&** msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_SUCCESSFUL**)** **{**

**if(**msgResponse**.**getArrStrParam**().**length **!=** 0**)** **{**

System**.**out**.**println**(**"--- Files in the server ---"**);**

**for(**String curr **:** msgResponse**.**getArrStrParam**())** **{**

System**.**out**.**println**(**curr**);**

**}**

System**.**out**.**println**(**"---------------------------"**);**

**}else** **{**

System**.**out**.**println**(**"You have no files in the server"**);**

**}**

**}else** **{**

System**.**out**.**println**(**"error: no answer from server"**);**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**case** "remove"**:** //remove <files>

**if(**parsedInput**.**length **>** 1**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "remove"**);**

String**[]** arrSent **=** Arrays**.**copyOfRange**(**parsedInput**,** 1**,** parsedInput**.**length**);**

msgSent **=** **new** Message**(**OpCode**.**REMOVE\_FILES**,**arrSent**);**

//send message method

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**if** **(**msgResponse **!=** **null** **&&** msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_RES\_ARRAY**)** **{**

OpCode**[]** arrCodes **=** msgResponse**.**getOpCodeArr**();**

**for(**int i **=** 0**;** i **<** arrSent**.**length**;** i**++)** **{**

**if(**arrCodes**[**i**]** **==** OpCode**.**OP\_SUCCESSFUL**)** **{**

System**.**out**.**println**(**arrSent**[**i**]** **+** " : " **+** "DELETED"**);**

**}else** **{**

System**.**out**.**println**(**arrSent**[**i**]** **+** " : " **+** arrCodes**[**i**].**toString**());**

**}**

**}**

**}else** **{**

System**.**out**.**println**(**"ERROR: no answer from server"**);**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**case** "users"**:**

**if(**parsedInput**.**length **==** 1**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "users"**);**

msgSent **=** **new** Message**(**OpCode**.**USERS**);**

//send message method

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**if** **(**msgResponse **!=** **null** **&&** msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_SUCCESSFUL**)** **{**

**if(**msgResponse**.**getArrStrParam**().**length **!=** 0**)** **{**

System**.**out**.**println**(**"--- Users registered in the server ---"**);**

**for(**String curr **:** msgResponse**.**getArrStrParam**())** **{**

System**.**out**.**println**(**curr**);**

**}**

System**.**out**.**println**(**"--------------------------------------"**);**

**}else** **{**

System**.**out**.**println**(**"No users registered in the server"**);**

**}**

**}else** **{**

System**.**out**.**println**(**"ERROR: no answer from server"**);**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**case** "trusted"**:** //trusted <trustedUserIDs>

**if(**parsedInput**.**length **>** 1**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "trusted"**);**

String**[]** arrSent **=** Arrays**.**copyOfRange**(**parsedInput**,** 1**,** parsedInput**.**length**);**

msgSent **=** **new** Message**(**OpCode**.**TRUST\_USERS**,**arrSent**);**

//send message method

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**if** **(**msgResponse **!=** **null** **&&** msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_RES\_ARRAY**)** **{**

OpCode**[]** arrCode **=** msgResponse**.**getOpCodeArr**();**

System**.**out**.**println**(**"--- Operation results ---"**);**

**for(**int i **=** 0**;** i **<** arrSent**.**length**;** i**++)** **{**

System**.**out**.**println**(**arrSent**[**i**]** **+** " : " **+** arrCode**[**i**]);**

**}**

System**.**out**.**println**(**"--------------------------------------"**);**

**}else** **{**

System**.**out**.**println**(**"ERROR: no answer from server"**);**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**case** "untrusted"**:** //untrusted <untrustedUserIDs>

**if(**parsedInput**.**length **>** 1**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "untrusted"**);**

String**[]** arrSent **=** Arrays**.**copyOfRange**(**parsedInput**,** 1**,** parsedInput**.**length**);**

msgSent **=** **new** Message**(**OpCode**.**UNTRUST\_USERS**,**arrSent**);**

//send message method

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**if** **(**msgResponse **!=** **null** **&&** msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_RES\_ARRAY**)** **{**

OpCode**[]** arrCode **=** msgResponse**.**getOpCodeArr**();**

System**.**out**.**println**(**"--- Operation results ---"**);**

**for(**int i **=** 0**;** i **<** arrSent**.**length**;** i**++)** **{**

System**.**out**.**println**(**arrSent**[**i**]** **+** " : " **+** arrCode**[**i**]);**

**}**

System**.**out**.**println**(**"--------------------------------------"**);**

**}else** **{**

System**.**out**.**println**(**"ERROR: no answer from server"**);**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**case** "download"**:** //download <userID> <file>

logger**.**log**(**Level**.**CONFIG**,** "download"**);**

**if(**parsedInput**.**length **==** 3**)** **{**

String**[]** arrSent **=** **new** String**[**2**];**

arrSent**[**0**]** **=** parsedInput**[**1**];**//users name account that has the file

arrSent**[**1**]** **=** parsedInput**[**2**];**//name of the file

//download file

msgSent **=** **new** Message**(**OpCode**.**DOWNLOAD\_FILE**,** arrSent**);**

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**if(**msgResponse **!=** **null)** **{**

**if(**msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_SUCCESSFUL**)** **{**

//File tempFile = new File("MsgFileResources" + File.separator + "client"

// + File.separator + client.getUsername() + File.separator + msgResponse.getStr());

File fileDown **=** **new** File**(**ClientConst**.**FOLDER\_CLIENT\_USERS **+** File**.**separator **+** client**.**getUsername**()** **+**

File**.**separator **+** ClientConst**.**FOLDER\_DOWNLOADS **+** File**.**separator **+** arrSent**[**1**]);**

**try{**

**if(**fileDown**.**exists**())** **{**

fileDown**.**delete**();**

System**.**out**.**println**(**"File with a same name already exists"**);**

System**.**out**.**println**(**"File downloaded will overwrite the old one"**);**

**}**

fileDown**.**getParentFile**().**mkdirs**();**

fileDown**.**createNewFile**();**

FileOutputStream fos **=** **new** FileOutputStream**(**fileDown**);**

fos**.**write**(**toPrimitives**(**msgResponse**.**getArrByte**()));**

fos**.**close**();**

System**.**out**.**println**(**"File downloaded to:"**);**

System**.**out**.**println**(**fileDown**.**getPath**());**

**}catch(**Exception e**){**

System**.**out**.**println**(**"error downloading file"**);**

e**.**printStackTrace**();**

**}**

**}else** **if(**msgResponse**.**getOpCode**()** **==** OpCode**.**ERR\_NOT\_FOUND**){**

System**.**out**.**println**(**arrSent**[**1**]** **+** " : " **+** msgResponse**.**getOpCode**());**

**}else** **if(**msgResponse**.**getOpCode**()** **==** OpCode**.**ERR\_NOT\_TRUSTED**){**

System**.**out**.**println**(**"error: you are not on " **+** arrSent**[**0**]** **+** "'s trusted list"**);**

**}else** **{**

System**.**out**.**println**(**msgResponse**.**getOpCode**());**

**}**

**}else** **{**

System**.**out**.**println**(**"ERROR: no answer from server"**);**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**case** "msg"**:** //msg <userID> <msg>

logger**.**log**(**Level**.**CONFIG**,** "msg"**);**

**if(**parsedInput**.**length **>=** 3**)** **{**

String userReceiver **=** parsedInput**[**1**];**

String msg **=** rawInput**.**substring**(**rawInput**.**indexOf**(**userReceiver**)** **+** userReceiver**.**length**());**

//send message

System**.**out**.**println**(**"To:" **+** userReceiver **+** " msg:" **+** msg**);**

String**[]** arrSent **=** **new** String**[**2**];**

arrSent**[**0**]** **=** userReceiver**;**//receiver

arrSent**[**1**]** **=** msg**;**//text message

**if(**client**.**getUsername**().**equals**(**userReceiver**))** **{**

System**.**out**.**println**(**"msg not sent : can't send a msg to yourself"**);**

**}else** **{**

msgSent **=** **new** Message**(**OpCode**.**SEND\_MSG**,** arrSent**);**

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**if(**msgResponse **!=** **null)** **{**

**if(**msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_SUCCESSFUL**)** **{**

System**.**out**.**println**(**"Message Sent Successfully to " **+** userReceiver**);**

**}else** **if(**msgResponse**.**getOpCode**()** **==** OpCode**.**ERR\_NOT\_REGISTERED**)** **{**

System**.**out**.**println**(**"error: " **+** userReceiver **+** " not registered in the server"**);**

**}else** **if(**msgResponse**.**getOpCode**()** **==** OpCode**.**ERR\_NOT\_TRUSTED**)** **{**

System**.**out**.**println**(**"error: you are not on " **+** userReceiver **+** "'s trusted list"**);**

**}else{**

System**.**out**.**println**(**"ERROR: msg not sent"**);**

**}**

**}else** **{**

System**.**out**.**println**(**"ERROR: no answer from server"**);**

**}**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**case** "collect"**:**

**if(**parsedInput**.**length **==** 1**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "collect"**);**

msgSent **=** **new** Message**(**OpCode**.**COLLECT\_MSG**);**

//send message method

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**if(**msgResponse **!=** **null)** **{**

**if(**msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_SUCCESSFUL**)** **{**

**if(**msgResponse**.**getArrListStr**().**size**()** **==** 0**)** **{**

System**.**out**.**println**(**"You don't have new messages in the server"**);**

**}else** **{**

File inboxFile **=** **new** File**(**ClientConst**.**FOLDER\_CLIENT\_USERS **+** File**.**separator **+** client**.**getUsername**()** **+**

File**.**separator **+** ClientConst**.**FILE\_NAME\_INBOX**);**

**try** **{**

inboxFile**.**getParentFile**().**mkdirs**();**

inboxFile**.**createNewFile**();**

FileWriter fileWriter **=** **new** FileWriter**(**inboxFile**,true);**

System**.**out**.**println**(**"--- Messages collected from the server ---"**);**

System**.**out**.**println**(**"--- and stored in your inbox file ---"**);**

**for** **(**String str **:** msgResponse**.**getArrListStr**())** **{**

fileWriter**.**write**(**str **+** System**.**getProperty**(**"line.separator"**));**

System**.**out**.**println**(**str**);**

**}**

fileWriter**.**close**();**

System**.**out**.**println**(**"------------------------------------------"**);**

**}** **catch** **(**IOException e**)** **{**

// TODO Auto-generated catch block

System**.**out**.**println**(**"Error saving messages in your inbox"**);**

**}**

**}**

**}else** **{**

System**.**out**.**println**(**msgResponse**.**getOpCode**());**

**}**

**}else** **{**

System**.**out**.**println**(**"ERROR: no answer from server"**);**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**case** "help"**:**

logger**.**log**(**Level**.**CONFIG**,** "help"**);**

System**.**out**.**println**(**"Commands available"**);**

System**.**out**.**println**(**"store <files>"**);**

System**.**out**.**println**(**"list"**);**

System**.**out**.**println**(**"remove <files>"**);**

System**.**out**.**println**(**"users"**);**

System**.**out**.**println**(**"trusted <trustedUserIDs>"**);**

System**.**out**.**println**(**"untrusted <untrustedUserIDs>"**);**

System**.**out**.**println**(**"download <userID> <file>"**);**

System**.**out**.**println**(**"msg <userID> <msg>"**);**

System**.**out**.**println**(**"collect"**);**

System**.**out**.**println**(**"exit"**);**

System**.**out**.**println**(**"-------------------"**);**

**break;**

**case** "exit"**:**

**if(**parsedInput**.**length **==** 1**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "exit"**);**

msgSent **=** **new** Message**(**OpCode**.**END\_CONNECTION**);**

msgResponse **=** client**.**sendMsg**(**msgSent**);**

**if(**msgResponse**.**getOpCode**()** **==** OpCode**.**OP\_SUCCESSFUL**)** **{**

client**.**disconnect**();**

onLoop **=** **false;**

**}**

**}else** **{**

incompleteCommand**();**

**}**

**break;**

**default:**

System**.**out**.**println**(**"Command is not recognized"**);**

**break;**

**}**

**}**

sc**.**close**();**

**}**

public static void incompleteCommand**()** **{**

System**.**out**.**println**(**"ERROR: incomplete command line"**);**

**}**

public static Byte**[]** toObjects**(**byte**[]** bytesPrim**)** **{**

Byte**[]** bytes **=** **new** Byte**[**bytesPrim**.**length**];**

int i **=** 0**;**

**for** **(**byte b **:** bytesPrim**){**

bytes**[**i**++]** **=** b**;**

**}**

**return** bytes**;**

**}**

/\*\*

\* Turn array of Byte[] to byte[]

\* **@param** oBytes

\* **@return**

\*/

private static byte**[]** toPrimitives**(**Byte**[]** oBytes**)** **{**

byte**[]** bytes **=** **new** byte**[**oBytes**.**length**];**

**for(**int i **=** 0**;** i **<** oBytes**.**length**;** i**++){**

bytes**[**i**]** **=** oBytes**[**i**];**

**}**

**return** bytes**;**

**}**

**}**

## Package communication

package communication**;**

public enum OpCode **{**

STORE\_FILES**(**"store"**),** LIST\_FILES**(**"list"**),** REMOVE\_FILES**(**"remove"**),**

USERS**(**"users"**),** TRUST\_USERS**(**"trusted"**),** UNTRUST\_USERS**(**"untrusted"**),**

DOWNLOAD\_FILE**(**"download"**),** SEND\_MSG**(**"msg"**),** COLLECT\_MSG**(**"collect"**),**

END\_CONNECTION**(**"exit"**),**

OP\_ERROR**(**"error"**),** OP\_SUCCESSFUL**(**"OK"**),** OP\_RES\_ARRAY**(**"op res in array"**),**

ERR\_NOT\_REGISTERED**(**"not registered in the server"**),** ERR\_ALREADY\_EXISTS**(**"already exists"**),**

ERR\_NOT\_TRUSTED**(**"not trusted"**),** ERR\_YOURSELF**(**"it's yourself"**),**

ERR\_NOT\_FOUND**(**"not found"**);**

private final String val**;**

/\*\*

\* **@param** text

\*/

OpCode**(**final String val**)** **{**

**this.**val **=** val**;**

**}**

@Override

public String toString**()** **{**

**return** val**;**

**}**

**}**

package communication**;**

**import** java**.**io**.**Serializable**;**

**import** java**.**util**.**ArrayList**;**

public class Message **implements** Serializable**{**

private static final long serialVersionUID **=** 1L**;**

private OpCode opCode**;**//operation code

private OpCode**[]** arrCode**;**//used for used operations that require different opcodes for every argument

private String str**;**//used to send a string as message

private String**[]** arrStr**;**

private ArrayList**<**String**>** arrListStr**;**

private ArrayList**<**Byte**[]>** arrListArrByte**;**

private Byte**[]** arrByte**;**

public Message**(**OpCode opCode**,** Byte**[]** arrByte**)** **{**

**this.**opCode **=** opCode**;**

**this.**arrByte **=** arrByte**;**

**}**

public Message **(**OpCode opCode**)** **{**

**this.**opCode **=** opCode**;**

**}**

public Message**(**OpCode**[]** arrCodeParam**)** **{**

**this.**arrCode **=** arrCodeParam**;**

**}**

public Message**(**OpCode opCode**,** ArrayList**<**String**>** param**)** **{**

**this.**opCode **=** opCode**;**

**this.**arrListStr **=** param**;**

**}**

public Message**(**OpCode opCode**,** OpCode**[]** arrCodeParam**)** **{**

**this.**opCode **=** opCode**;**

**this.**arrCode **=** arrCodeParam**;**

**}**

public Message **(**OpCode opCode**,** String strParam**)** **{**

**this.**opCode **=** opCode**;**

**this.**str **=** strParam**;**

**}**

public Message**(**OpCode opCode**,** String**[]** arrStrParam**)** **{**

**this.**opCode **=** opCode**;**

**this.**arrStr **=** arrStrParam**.**clone**();**

**}**

public Message**(**OpCode opCode**,** ArrayList**<**String**>** arrListStr**,** ArrayList**<**Byte**[]>** arrListArrBytes**)** **{**

**this.**opCode **=** opCode**;**

**this.**arrListStr **=** arrListStr**;**

**this.**arrListArrByte **=** arrListArrBytes**;**

**}**

public OpCode**[]** getArrCode**()** **{**

**return** arrCode**;**

**}**

public String getStr**()** **{**

**return** str**;**

**}**

public String**[]** getArrStr**()** **{**

**return** arrStr**;**

**}**

public Byte**[]** getArrByte**()** **{**

**return** **this.**arrByte**;**

**}**

public OpCode getOpCode**()** **{**

**return** **this.**opCode**;**

**}**

public OpCode**[]** getOpCodeArr**()** **{**

**return** **this.**arrCode**;**

**}**

public String getStrParam**()** **{**

**return** **this.**str**;**

**}**

public String**[]** getArrStrParam**()** **{**

**return** **this.**arrStr**.**clone**();**

**}**

public ArrayList**<**String**>** getArrListStr**()** **{**

**return** arrListStr**;**

**}**

public ArrayList**<**Byte**[]>** getArrListArrBytes**()** **{**

**return** arrListArrByte**;**

**}**

**}**

## Package server

package server**;**

**import** java**.**util**.**Hashtable**;**

**import** java**.**util**.**concurrent**.**locks**.**StampedLock**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

public class ConcurrentManager **{**

private Hashtable**<**String**,**StampedLock**>** concurrentFilesUsers**;**

//\*\* Log \*\*//

private static final String CLASS\_NAME **=** ConcurrentManager**.**class**.**getName**();**

private final static Logger logger **=** Logger**.**getLogger**(**CLASS\_NAME**);**

public ConcurrentManager**(){**

concurrentFilesUsers **=** **new** Hashtable**<>();**

**}**

/\*\*

\* Add file to hashtable

\* **@param** user

\* **@param** path

\* @requires user != null and path != null

\*/

public boolean addSem**(**String path**){**

**if(**path **==** **null** **||** path**.**isEmpty**()){**

logger**.**log**(**Level**.**SEVERE**,** "Error on parameters"**);**

**return** **false;**

**}else** **if(**concurrentFilesUsers**.**containsKey**(**path**)){**

logger**.**log**(**Level**.**CONFIG**,** "Already exists: " **+** path**);**

**return** **false;**

**}**

logger**.**log**(**Level**.**CONFIG**,** "Semaphore ADDED: " **+** path**);**

concurrentFilesUsers**.**put**(**path**,** **new** StampedLock**());**

**return** **true;**

**}**

/\*\*

\* Get semaphore

\* **@param** user

\* **@return** Tuple with file and semaphore

\* @requires user != null and path != null

\*/

public StampedLock getSem**(**String path**){**

**if(**path **==** **null** **||** path**.**isEmpty**()){**

logger**.**log**(**Level**.**SEVERE**,** "Error on parameters"**);**

**return** **null;**

**}** **else{**

StampedLock locker **=** concurrentFilesUsers**.**get**(**path**);**

**if(**locker **!=** **null** **){**

logger**.**log**(**Level**.**CONFIG**,** "Semaphore found! path: "**+** path**);**

**return** locker**;**

**}**

**}**

logger**.**log**(**Level**.**CONFIG**,** "Semaphore not found! path: "**+** path**);**

**return** **null;**

**}**

public void delSem**(**String user**,** String path**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "Delete Semaphore path: "**+** path**);**

**if(**path **==** **null** **||** path**.**isEmpty**()){**

logger**.**log**(**Level**.**SEVERE**,** "Error on parameters"**);**

**return;**

**}**

**if(**concurrentFilesUsers**.**remove**(**path**)** **!=** **null){**

logger**.**log**(**Level**.**CONFIG**,** "Semaphore deleted to path: "**+** path**);**

**}else{**

logger**.**log**(**Level**.**CONFIG**,** "Semaphore NOT deleted to path: "**+** path**);**

**}**

**}**

**}**

package server**;**

**import** java**.**io**.**BufferedReader**;**

**import** java**.**io**.**File**;**

**import** java**.**io**.**FileNotFoundException**;**

**import** java**.**io**.**FileOutputStream**;**

**import** java**.**io**.**FileReader**;**

**import** java**.**io**.**FileWriter**;**

**import** java**.**io**.**IOException**;**

**import** java**.**nio**.**file**.**Files**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**HashMap**;**

**import** java**.**util**.**concurrent**.**locks**.**StampedLock**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** communication**.**OpCode**;**

public class Manager **{**

//\*\* Log \*\*//

private static final String CLASS\_NAME **=** Manager**.**class**.**getName**();**

private final static Logger logger **=** Logger**.**getLogger**(**CLASS\_NAME**);**

//\*\* File of passwords \*\*//

private File usersFile**;**

//\*\* Object to Singleton \*\*//

private static Manager INSTANCE**;**

//\*\* ConcurrentManager \*\*//

private ConcurrentManager sempManager**;**

private HashMap**<**String**,** String**>** users**;**

private Manager**()** **{**

sempManager **=** **new** ConcurrentManager**();**

users **=** **new** HashMap**<>();**

usersFile **=** **new** File**(**ServerConst**.**FILE\_USERS\_PASSWORDS**);**

**if(!**usersFile**.**exists**())** **{**

**try** **{**

usersFile**.**getParentFile**().**mkdirs**();**

usersFile**.**createNewFile**();**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "ERROR to open the file " **,** e**);**

**}**

**}else{**

**try** **{**

synchronized **(**usersFile**)** **{**

BufferedReader br **=** **new** BufferedReader**(new** FileReader**(**usersFile**));**

String st**;**

**while** **((**st **=** br**.**readLine**())** **!=** **null)** **{**

String**[]** usersPass **=** st**.**split**(**":"**);**

users**.**put**(**usersPass**[**0**],** usersPass**[**1**]);**

**}**

br**.**close**();**

**}**

**}** **catch** **(**FileNotFoundException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "File of users not found"**,** e**);**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "IO Exception"**,** e**);**

**}**

**}**

File file **=** **new** File**(**ServerConst**.**FOLDER\_SERVER\_USERS**);**

File temp**;**

**if(**file**.**exists**()){**

**for(**String paths**:** file**.**list**()){**

temp **=** **new** File**(**file**.**getPath**()** **+** File**.**separator **+** paths**);**

**if(**file**.**exists**()** **&&** file**.**isDirectory**()** **&&** users**.**containsKey**(**paths**)){**

createSemaphore**(**temp**,** paths**);**

**}**

**}**

**}**

**}**

/\*\*

\* Recursion to add all semaphore for each file

\* ignore all folders only add semaphore only to files

\* **@param** file

\* **@param** user

\* @requires file.exists() and user != null

\*/

private void createSemaphore**(**File file**,** String user**){**

File temp**;**

**for(**String path**:** file**.**list**()){**

temp **=** **new** File**(**file**.**getPath**()** **+** File**.**separator **+**path**);**

**if(**temp**.**exists**()){**

**if(**ServerConst**.**FILE\_NAME\_TRUSTED**.**equals**(**temp**.**getName**())){**

**continue;**

**}else** **if(**temp**.**isFile**()){**

sempManager**.**addSem**(**temp**.**getPath**());**

**}else{**

createSemaphore**(**temp**,** user**);**

**}**

**}**

**}**

**}**

/\*\*

\* Singleton

\* **@return** Manager

\*/

public static Manager getInstance**()** **{**

**if(**INSTANCE **==** **null)** **{**

INSTANCE **=** **new** Manager**();**

**}**

**return** INSTANCE**;**

**}**

/\*\*

\* Create Account

\* **@param** username

\* **@param** password

\* **@return** true if got success

\*/

public boolean createAccount**(**String username**,** String password**)** **{**

logger**.**log**(**Level**.**INFO**,** "Creating account"**);**

**try** **{**

synchronized **(**usersFile**)** **{**

FileWriter fileWriter **=** **new** FileWriter**(**usersFile**,true);**

String newLine **=** System**.**getProperty**(**"line.separator"**);**

fileWriter**.**write**(**username **+** ":" **+** password **+** newLine**);**

users**.**put**(**username**,** password**);**

fileWriter**.**close**();**

**}**

String path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator

**+** username **+** File**.**separator **+** ServerConst**.**FOLDER\_FILES**;**

File userFiles **=** **new** File**(**path**);**

userFiles**.**getParentFile**().**mkdirs**();**

userFiles**.**mkdir**();**

//it is a folder do not need semaphore

//sempManager.addSem(username, path);

path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator

**+** username **+** File**.**separator **+** ServerConst**.**FILE\_NAME\_TRUSTED**;**

File userTrusted **=** **new** File**(**path**);**

userTrusted**.**getParentFile**().**mkdirs**();**

userTrusted**.**createNewFile**();**

//sempManager.addSem(username, path);

path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator

**+** username **+** File**.**separator **+** ServerConst**.**FILE\_NAME\_MSG**;**

File userMsg **=** **new** File**(**path**);**

userMsg**.**getParentFile**().**mkdirs**();**

userMsg**.**createNewFile**();**

sempManager**.**addSem**(**path**);**

**return** **true;**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "Error creating account"**,** e**);**

**}**

**return** **false;**

**}**

/\*\*

\* Login

\* **@param** username

\* **@param** password

\* **@return** true was a user with password valid

\*/

public boolean login**(**String username**,** String password**)** **{**

**try** **{**

synchronized **(**usersFile**)** **{**

BufferedReader br **=** **new** BufferedReader**(new** FileReader**(**usersFile**));**

String st**;**

**while** **((**st **=** br**.**readLine**())** **!=** **null)** **{**

//loginInfo[0] = username loginInfo[1] = password

String**[]** loginInfo **=** st**.**split**(**":"**);**

**if(**username**.**equals**(**loginInfo**[**0**]))** **{**

**if(**password**.**equals**(**loginInfo**[**1**]))** **{**

br**.**close**();**

**return** **true;**//user and pass match info on file

**}else** **{**

br**.**close**();**

**return** **false;**//password does not match

**}**

**}**

**}**

br**.**close**();**

**}**

**return** createAccount**(**username**,** password**);**//user is not on registered on file

**}** **catch** **(**FileNotFoundException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "File of users not found"**,** e**);**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "IO Exception"**,** e**);**

**}**

**return** **false;**

**}**

/\*\*

\* envia um ou mais ficheiros para o servidor, para a conta do utilizador local

\* (localUserID). Caso este utilizador já tenha algum ficheiro com o mesmo nome no

\* servidor, o servidor não substitui o ficheiro existente. O comando deve indicar, para cada

\* ficheiro, se o envio foi realizado com sucesso ou se o ficheiro já existia no servidor.

\*

\* **@param** nameFiles

\* **@param** byteFiles

\* **@param** connectedUser

\* @requires nameFiles != null && byteFiles != null && connectedUser != null

\* **@return** array of opcodes, with the same length as nameFiles and byteFiles, for each position has the respective

\* code indicating if the file was successfuly stored

\*/

public OpCode**[]** storeFiles**(**ArrayList**<**String**>** nameFiles**,** ArrayList**<**Byte**[]>** byteFiles**,** String connectedUser**){**

File file**;**

String path**;**

OpCode**[]** result **=** **new** OpCode**[**nameFiles**.**size**()];**

**for(**int i **=** 0**;** i **<** nameFiles**.**size**();**i**++){**

path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator **+** connectedUser

**+** File**.**separator **+** ServerConst**.**FOLDER\_FILES **+** File**.**separator **+** nameFiles**.**get**(**i**);**

file **=** **new** File**(**path**);**

**if(**file**.**exists**()){**

result**[**i**]** **=** OpCode**.**ERR\_ALREADY\_EXISTS**;**

**}else{**

**try{**

file**.**getParentFile**().**mkdirs**();**

file**.**createNewFile**();**

FileOutputStream fos **=** **new** FileOutputStream**(**file**);**

fos**.**write**(**toPrimitives**(**byteFiles**.**get**(**i**)));**

fos**.**close**();**

result**[**i**]** **=** OpCode**.**OP\_SUCCESSFUL**;**

sempManager**.**addSem**(**path**);**

**}catch(**Exception e**){**

logger**.**log**(**Level**.**SEVERE**,** "FAILED to store the file: " **+** file**.**getName**());**

result**[**i**]** **=** OpCode**.**OP\_ERROR**;**

**}**

**}**

**}**

**return** result**;**

**}**

/\*\*

\* Turn array of Byte[] to byte[]

\* **@param** oBytes

\* **@return**

\*/

private static byte**[]** toPrimitives**(**Byte**[]** oBytes**)** **{**

byte**[]** bytes **=** **new** byte**[**oBytes**.**length**];**

**for(**int i **=** 0**;** i **<** oBytes**.**length**;** i**++){**

bytes**[**i**]** **=** oBytes**[**i**];**

**}**

**return** bytes**;**

**}**

/\*\*

\* @requires localUser != null

\* **@param** localUser

\* **@return** lista os ficheiros que o utilizador local (localUserID) tem no servidor.

\*/

public String**[]** listFiles**(**String localUser**)** **{** //list

File userFiles **=** **new** File**(**ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator **+** localUser

**+** File**.**separator **+** ServerConst**.**FOLDER\_FILES**);**

**return** userFiles**.**list**();**

**}**

/\*\*

\* remove um ou mais ficheiros do servidor, da conta do utilizador local. O

\* comando deve indicar, para cada ficheiro, se a remoção foi realizada com sucesso ou um

\* erro no caso de o ficheiro não existir no servidor.

\* **@param** fileName

\* **@return**

\*/

public boolean deleteFile**(**String user**,** String fileName**)** **{**

String path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator **+** user **+**

File**.**separator **+** ServerConst**.**FOLDER\_FILES **+** File**.**separator **+** fileName**;**

logger**.**log**(**Level**.**CONFIG**,** "Patg to delete: " **+** path**);**

StampedLock sem **=** sempManager**.**getSem**(**path**);**

**if(**sem **==** **null){**

**return** **false;**

**}**

long stamp **=** sem**.**writeLock**();**

**try** **{**

File file **=** **new** File**(**path**);**

boolean isDeleted **=** file**.**delete**();**

**if(**isDeleted**){**

sempManager**.**delSem**(**user**,** path**);**

**}**

**return** isDeleted**;**

**}catch(**Exception e**){**

logger**.**log**(**Level**.**SEVERE**,** e**.**getMessage**(),** e**);**

**}finally** **{**

sem**.**unlockWrite**(**stamp**);**

**}**

**return** **false;**

**}**

/\*\*

\* **@return** List of Users

\*/

public String**[]** listUsers**()** **{** //users

**return** users**.**keySet**().**toArray**(new** String**[**users**.**size**()]);**

**}**

/\*\*

\* **@param** user

\* **@return** true if registered

\*/

public boolean isRegistered**(**String user**)** **{**

**return** users**.**containsKey**(**user**);**

**}**

/\*\*

\* Friends

\* **@param** localUser

\* **@param** otherUser

\* **@return** localUser is friend of otherUser

\*/

public boolean friends**(**String localUser**,** String otherUser**)** **{**

String path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator **+** otherUser **+** File**.**separator **+** ServerConst**.**FILE\_NAME\_TRUSTED**;**

File trustedFile **=** **new** File**(**path**);**

BufferedReader br**;**

**if(**trustedFile**.**exists**())** **{**

**try** **{**

br **=** **new** BufferedReader**(new** FileReader**(**trustedFile**));**

String st**;**

**while** **((**st **=** br**.**readLine**())** **!=** **null)** **{**

**if(**st**.**equals**(**localUser**))** **{**

br**.**close**();**

**return** **true;**

**}**

**}**

br**.**close**();**

**}** **catch** **(**FileNotFoundException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "File Not Found in Friends"**,** e**);**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "Method Friends fail"**,** e**);**

**}**

**}**

**return** **false;**

**}**

/\*\*

\* adiciona os utilizadores trustedUserIDs como amigos do

\* utilizador local. Se algum dos utilizadores já estiver na lista de amigos do utilizador local

\* deve ser assinalado um erro. Os restantes utilizadores são adicionados normalmente

\*

\* **@param** localUser

\* **@param** trustedUserID

\* **@return** vetor de opcode ou null em caso de erro

\*/

public OpCode**[]** trusted**(**String localUser**,** String**[]** trustedUserID**)** **{** //trusted <trustedUserIDs>

String path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator **+** localUser

**+** File**.**separator **+** ServerConst**.**FILE\_NAME\_TRUSTED**;**

File trustedFile **=** **new** File**(**path**);**

OpCode**[]** result **=** **new** OpCode**[**trustedUserID**.**length**];**

BufferedReader br**;**

ArrayList**<**String**>** fileContent **=** **new** ArrayList**<**String**>();**

**try** **{**

trustedFile**.**getParentFile**().**mkdirs**();**

trustedFile**.**createNewFile**();**

br **=** **new** BufferedReader**(new** FileReader**(**trustedFile**));**

String st**;**

**while** **((**st **=** br**.**readLine**())** **!=** **null)** **{**

fileContent**.**add**(**st**);**

**}**

br**.**close**();**

FileWriter fileWriter **=** **new** FileWriter**(**trustedFile**,true);**

**for** **(**int i **=** 0**;** i **<** trustedUserID**.**length**;** i**++)** **{**

**if(**localUser**.**equals**(**trustedUserID**[**i**]))** **{**

result**[**i**]** **=** OpCode**.**ERR\_YOURSELF**;**

**}else** **if(!**isRegistered**(**trustedUserID**[**i**]))** **{**

result**[**i**]** **=** OpCode**.**ERR\_NOT\_REGISTERED**;**

**}else** **if(**fileContent**.**contains**(**trustedUserID**[**i**]))** **{**

result**[**i**]** **=** OpCode**.**ERR\_ALREADY\_EXISTS**;**

**}else** **{**

fileWriter**.**write**(**trustedUserID**[**i**]** **+** System**.**getProperty**(**"line.separator"**));**

fileContent**.**add**(**trustedUserID**[**i**]);**

result**[**i**]** **=** OpCode**.**OP\_SUCCESSFUL**;**

**}**

**}**

fileWriter**.**close**();**

**return** result**;**

**}** **catch** **(**FileNotFoundException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "File not found: " **+** path**,** e**);**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "IOException in trusted"**,** e**);**

**}**

**return** **null;**

**}**

/\*\*

\*

\* **@param** localUser

\* **@param** untrustedUserID

\* **@return**

\*/

public OpCode**[]** untrusted**(**String localUser**,** String**[]** untrustedUserID**)** **{** //trusted <trustedUserIDs>

String path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator **+** localUser

**+** File**.**separator **+** ServerConst**.**FILE\_NAME\_TRUSTED**;**

OpCode**[]** result **=** **new** OpCode**[**untrustedUserID**.**length**];**

File trustedFile **=** **new** File**(**path**);**

BufferedReader br**;**

ArrayList**<**String**>** fileContent **=** **new** ArrayList**<**String**>();**

**try** **{**

trustedFile**.**getParentFile**().**mkdirs**();**

trustedFile**.**createNewFile**();**

br **=** **new** BufferedReader**(new** FileReader**(**trustedFile**));**

String st**;**

**while** **((**st **=** br**.**readLine**())** **!=** **null)** **{**

fileContent**.**add**(**st**);**

**}**

br**.**close**();**

**for(**int i **=** 0**;** i **<** untrustedUserID**.**length**;** i**++)** **{**

**if(!**isRegistered**(**untrustedUserID**[**i**]))** **{**

result**[**i**]** **=** OpCode**.**ERR\_NOT\_REGISTERED**;**

**}else** **if(**localUser**.**equals**(**untrustedUserID**[**i**]))** **{**

result**[**i**]** **=** OpCode**.**ERR\_YOURSELF**;**

**}else** **{**

boolean untrusted **=** fileContent**.**remove**(**untrustedUserID**[**i**]);**

**if(**untrusted**)** **{**

result**[**i**]** **=** OpCode**.**OP\_SUCCESSFUL**;**

**}else** **{**

result**[**i**]** **=** OpCode**.**ERR\_NOT\_FOUND**;**

**}**

**}**

**}**

FileWriter fileWriter **=** **new** FileWriter**(**trustedFile**);**

fileWriter**.**write**(**""**);**

fileWriter**.**close**();**

fileWriter **=** **new** FileWriter**(**trustedFile**,true);**

**for(**String str **:** fileContent**)** **{**

System**.**out**.**println**(**str**);**

fileWriter**.**write**(**str **+** System**.**getProperty**(**"line.separator"**));**

**}**

fileWriter**.**close**();**

**return** result**;**

**}** **catch** **(**FileNotFoundException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "File not found: " **+** path**,** e**);**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "IOException"**,** e**);**

**}**

**return** **null;**

**}**

/\*\*

\*

\* **@param** userOwner

\* **@param** userDownloading

\* **@param** nameFile

\* **@return**

\*/

public Byte**[]** sendFileToClient**(**String userOwner**,** String nameFile**)** **{**//download <userID> <file>

String path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator **+** userOwner

**+** File**.**separator **+** ServerConst**.**FOLDER\_FILES **+** File**.**separator **+** nameFile**;**

File file **=** **new** File**(**path**);**

**if(**file**.**exists**()){**

StampedLock sem **=** sempManager**.**getSem**(**path**);**

**if(**sem **==** **null){**

logger**.**log**(**Level**.**SEVERE**,** "Semaphore null"**);**

**return** **null;**

**}**

long stamp **=** sem**.**readLock**();**

**try** **{**

**return** toObjects**(**Files**.**readAllBytes**(**file**.**toPath**()));**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "Error to converte the file :" **+** nameFile **+** " to bytes"**,** e**);**

**}finally{**

sem**.**unlock**(**stamp**);**

**}**

**}**

**return** **null;**

**}**

private static Byte**[]** toObjects**(**byte**[]** bytesPrim**)** **{**

Byte**[]** bytes **=** **new** Byte**[**bytesPrim**.**length**];**

int i **=** 0**;**

**for** **(**byte b **:** bytesPrim**){**

bytes**[**i**++]** **=** b**;**

**}**

**return** bytes**;**

**}**

/\*\*

\* Write message on the file of user

\* **@param** userSender

\* **@param** userReceiver

\* **@param** msg

\* **@return** true if it is success

\*/

public boolean storeMsg**(**String userSender**,** String userReceiver**,** String msg**)** **{**//msg <userID> <msg>

String path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator **+** userReceiver

**+** File**.**separator **+** ServerConst**.**FILE\_NAME\_MSG**;**

File userMsgs **=** **new** File**(**path**);**

StampedLock sem **=** sempManager**.**getSem**(**path**);**

**if(**sem **==** **null){**

**return** **false;**

**}**

FileWriter fileWriter**;**

long stamp **=** sem**.**writeLock**();**

**try** **{**

fileWriter **=** **new** FileWriter**(**userMsgs**,true);**

fileWriter**.**write**(**userSender **+** ":" **+** msg **+** System**.**getProperty**(**"line.separator"**));**

fileWriter**.**close**();**

**return** **true;**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "Error to write in store Msg"**,** e**);**

**}finally{**

sem**.**unlock**(**stamp**);**

**}**

**return** **false;**

**}**

/\*\*

\* Collect Msg from user

\* **@param** user

\* **@return** list with all messages -> null if empty

\*/

public ArrayList**<**String**>** collectMsg**(**String user**)** **{**//collect

String path **=** ServerConst**.**FOLDER\_SERVER\_USERS **+** File**.**separator

**+** user **+** File**.**separator **+** ServerConst**.**FILE\_NAME\_MSG**;**

ArrayList**<**String**>** result **=** **new** ArrayList**<**String**>();**

File userMsgs **=** **new** File**(**path**);**

StampedLock sem **=** sempManager**.**getSem**(**path**);**

**if(**sem **==** **null){**

logger**.**log**(**Level**.**CONFIG**,** "Semaphore Empty"**);**

**return** **null;**

**}**

long stamp **=** sem**.**writeLock**();**

**try** **{**

BufferedReader br **=** **new** BufferedReader**(new** FileReader**(**userMsgs**));**

String st**;**

**while** **((**st **=** br**.**readLine**())** **!=** **null)** **{**

result**.**add**(**st**);**

**}**

br**.**close**();**

FileWriter fileWriter **=** **new** FileWriter**(**userMsgs**);**

fileWriter**.**write**(**""**);**

fileWriter**.**close**();**

//clear inbox

**return** result**;**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**SEVERE**,** "Impossible Colect Messages"**,** e**);**

**return** **null;**

**}finally{**

sem**.**unlock**(**stamp**);**

**}**

**}**

package server**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**ObjectInputStream**;**

**import** java**.**io**.**ObjectOutputStream**;**

**import** java**.**net**.**ServerSocket**;**

**import** java**.**net**.**Socket**;**

**import** java**.**net**.**SocketException**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Collections**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** communication**.**Message**;**

**import** communication**.**OpCode**;**

//Servidor MsgFileServer

public class MsgFileServer**{**

private static final String CLASS\_NAME **=** MsgFileServer**.**class**.**getName**();**

private final static Logger logger **=** Logger**.**getLogger**(**CLASS\_NAME**);**

private Manager accM **=** Manager**.**getInstance**();**

public static void main**(**String**[]** args**)** **{**

**if(**args**.**length **==** 1**)** **{**

int port**;**

**try** **{**

port **=** Integer**.**parseInt**(**args**[**0**]);**

**}**

**catch** **(**NumberFormatException e**){**

System**.**out**.**println**(** "Server failed: Invalid port"**);**

**return;**

**}**

System**.**out**.**println**(**"Initializing server on port: " **+** args**[**0**]);**

MsgFileServer server **=** **new** MsgFileServer**();**

server**.**startServer**(**port**);**

**}else** **{**

System**.**out**.**println**(**"Server failed: <port> is the only argument required"**);**

**}**

**}**

public void startServer **(**int port**){**

ServerSocket sSoc **=** **null;**

boolean isReady **=** **true;**

**try** **{**

sSoc **=** **new** ServerSocket**(**port**);**

**}** **catch** **(**IOException e**)** **{**

System**.**err**.**println**(**e**.**getMessage**());**

isReady **=** **false;**

System**.**exit**(-**1**);**

**}**

**while(**isReady**)** **{**

**try** **{**

Socket inSoc **=** sSoc**.**accept**();**

System**.**out**.**println**(**"Client connecting..."**);**

ServerThread newServerThread **=** **new** ServerThread**(**inSoc**);**

newServerThread**.**start**();**

**}**

**catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**if(**sSoc **!=** **null)** **{**

**try** **{**

sSoc**.**close**();**

**}** **catch** **(**IOException e**)** **{**

logger**.**log**(**Level**.**CONFIG**,** "Can not close the Socket Server"**);**

**}**

**}**

**}**

//save users logged

private static List**<**String**>** usersLogged **=** **(**List**<**String**>)** Collections**.**synchronizedList**(new** ArrayList**<**String**>());**

//Threads utilizadas para comunicacao com os clientes

class ServerThread **extends** Thread **{**

private Socket socket **=** **null;**

ServerThread**(**Socket inSoc**)** **{**

socket **=** inSoc**;**

logger**.**log**(**Level**.**CONFIG**,** "Thread created by server"**);**

**}**

public void run**(){**

**try** **{**

ObjectOutputStream outStream **=** **new** ObjectOutputStream**(**socket**.**getOutputStream**());**

ObjectInputStream inStream **=** **new** ObjectInputStream**(**socket**.**getInputStream**());**

String user **=** **null;**

String passwd **=** **null;**

**try** **{**

user **=** **(**String**)**inStream**.**readObject**();**

passwd **=** **(**String**)**inStream**.**readObject**();**

**}catch** **(**ClassNotFoundException e1**)** **{**

outStream**.**close**();**

inStream**.**close**();**

System**.**out**.**println**(**"Erro autenticacao" **+** e1**);**

**return;**

**}**

**if** **(**accM**.**login**(**user**,** passwd**)){**

boolean islogged **=** **false;**

synchronized **(**usersLogged**)** **{**

**if(!**usersLogged**.**contains**(**user**)){**

usersLogged**.**add**(**user**);**

System**.**out**.**println**(**"Client connected: " **+** user **+** " logged in"**);**

outStream**.**writeObject**(new** Message**(**OpCode**.**OP\_SUCCESSFUL**));**//envia true para o cliente a confirmar conecao

islogged **=** **true;**

**}else{**

System**.**out**.**println**(**"Error: client " **+** user **+** " already logged"**);**

outStream**.**writeObject**(new** Message**(**OpCode**.**ERR\_ALREADY\_EXISTS**));**//envia true para o cliente a confirmar conecao

**}**

**}**

**try** **{**

**while(**islogged**)** **{**

Object obj **=** inStream**.**readObject**();**

**if(**obj **==** **null** **||** **!(**obj **instanceof** Message**))** **{**

**break;**

**}**

Message msgReceived **=** **(**Message**)** obj**;**

**if(**OpCode**.**END\_CONNECTION **==** msgReceived**.**getOpCode**())** **{**

Message msgSent **=** **new** Message**(**OpCode**.**OP\_SUCCESSFUL**);**

outStream**.**writeObject**(**msgSent**);**

System**.**out**.**println**(**"Client disconnected: " **+** user **+** " logged out"**);**

**break;**

**}else** **{**

//processar msg

Message msgSent **=** Skel**.**invoke**(**msgReceived**,** user**);**

outStream**.**writeObject**(**msgSent**);**

**}**

**}**

**}** **catch** **(**ClassNotFoundException e**)** **{**

// TODO Auto-generated catch block

e**.**printStackTrace**();**//rever esta excecao

**}** **catch** **(**SocketException e**)** **{**//resolver lancamento de exception quando user e desligado abruptamente

System**.**out**.**println**(**"Client disconnected: Connection lost with " **+** user**);**

**}finally{**

**if(**islogged**){**

synchronized **(**usersLogged**)** **{**

usersLogged**.**remove**(**user**);**

**}**

**}**

**}**

**}**

**else** **{**

System**.**out**.**println**(**"Password errada " **+** user**);**

outStream**.**writeObject**(new** Message**(**OpCode**.**OP\_ERROR**));**//envia false para o cliente a rejeitar conexao

**}**

outStream**.**close**();**

inStream**.**close**();**

socket**.**close**();**

**}** **catch** **(**IOException e**)** **{**

e**.**printStackTrace**();**

**}**

**}**

**}**

**}**

package server**;**

**import** java**.**io**.**File**;**

public class ServerConst **{**

public static final String FILE\_NAME\_USERS\_PASSWORDS **=** "usersInfo.txt"**;**

public static final String FOLDER\_SERVER **=** System**.**getProperty**(**"user.home"**)** **+** File**.**separator **+** "MsgFileG51" **+** File**.**separator **+** "server"**;**

public static final String FOLDER\_FILES **=** "files"**;**

public static final String FILE\_NAME\_MSG **=** "msg.txt"**;**

public static final String FILE\_NAME\_TRUSTED **=** "trusted.txt"**;**

public static final String FOLDER\_SERVER\_USERS **=** FOLDER\_SERVER **+** File**.**separator **+** "users"**;**

public static final String FILE\_USERS\_PASSWORDS **=** FOLDER\_SERVER **+** File**.**separator **+** FILE\_NAME\_USERS\_PASSWORDS**;**

**}**

package server**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** communication**.**Message**;**

**import** communication**.**OpCode**;**

public class Skel **{**

private static final String CLASS\_NAME **=** Skel**.**class**.**getName**();**

private static Manager svM **=** Manager**.**getInstance**();**

private final static Logger logger **=** Logger**.**getLogger**(**CLASS\_NAME**);**

//fazer padrao singleton para o manager

//msg != null

public static Message invoke**(**Message msg**,** String connectedUser**)** **{** //alterar no fim para retornar uma message que depois

//e enviada no ciclo do msgFileServer

Message response **=** **null;**

String**[]** arrStrRes**;**

OpCode**[]** arrOpRes**;**

OpCode code **=** msg**.**getOpCode**();**

**switch** **(**code**)** **{**

**case** STORE\_FILES**:** //store <files>

logger**.**log**(**Level**.**CONFIG**,** "STORE\_FILES user: " **+** connectedUser**);**

arrOpRes **=** svM**.**storeFiles**(**msg**.**getArrListStr**(),** msg**.**getArrListArrBytes**(),** connectedUser**);**

**if(**arrOpRes **!=** **null)** **{**

response **=** **new** Message**(**OpCode**.**OP\_RES\_ARRAY**,** arrOpRes**);**

**}else** **{**

response **=** **new** Message**(**OpCode**.**OP\_ERROR**);**

**}**

**break;**

**case** LIST\_FILES**:**

logger**.**log**(**Level**.**CONFIG**,** "LIST\_FILES user: " **+** connectedUser**);**

arrStrRes **=** svM**.**listFiles**(**connectedUser**);**

**if(**arrStrRes **!=** **null)** **{**

response **=** **new** Message**(**OpCode**.**OP\_SUCCESSFUL**,** arrStrRes**);**

**}else** **{**

response **=** **new** Message**(**OpCode**.**OP\_ERROR**);**

**}**

**break;**

**case** REMOVE\_FILES**:** //remove <files>

logger**.**log**(**Level**.**CONFIG**,** "REMOVE\_FILES user: " **+** connectedUser**);**

String**[]** nameFile **=** msg**.**getArrStrParam**();**

arrOpRes **=** **new** OpCode**[**nameFile**.**length**];**

**for(**int i **=** 0**;** i **<** nameFile**.**length**;** i**++)** **{**

boolean deleted **=** svM**.**deleteFile**(**connectedUser**,**nameFile**[**i**]);**

**if(**deleted**)** **{**

arrOpRes**[**i**]** **=** OpCode**.**OP\_SUCCESSFUL**;**

**}else** **{**

arrOpRes**[**i**]** **=** OpCode**.**ERR\_NOT\_FOUND**;**

**}**

**}**

response **=** **new** Message**(**OpCode**.**OP\_RES\_ARRAY**,**arrOpRes**);**

**break;**

**case** USERS**:**

logger**.**log**(**Level**.**CONFIG**,** "USERS user: " **+** connectedUser**);**

arrStrRes **=** svM**.**listUsers**();**

**if(**arrStrRes **!=** **null)** **{**

response **=** **new** Message**(**OpCode**.**OP\_SUCCESSFUL**,** arrStrRes**);**

**}else** **{**

response **=** **new** Message**(**OpCode**.**OP\_ERROR**);**

**}**

**break;**

**case** TRUST\_USERS**:** //trusted <trustedUserIDs>

logger**.**log**(**Level**.**CONFIG**,** "TRUST\_USERS user: " **+** connectedUser**);**

String**[]** usersTrust **=** msg**.**getArrStrParam**();**

arrOpRes **=** svM**.**trusted**(**connectedUser**,** usersTrust**);**

**if(**arrOpRes **!=** **null)** **{**

response **=** **new** Message**(**OpCode**.**OP\_RES\_ARRAY**,** arrOpRes**);**

**}else** **{**

response **=** **new** Message**(**OpCode**.**OP\_ERROR**);**

**}**

**break;**

**case** UNTRUST\_USERS**:** //untrusted <untrustedUserIDs>

logger**.**log**(**Level**.**CONFIG**,** "UNTRUST\_USERS user: " **+** connectedUser**);**

String**[]** usersUntrust **=** msg**.**getArrStrParam**();**

arrOpRes **=** svM**.**untrusted**(**connectedUser**,** usersUntrust**);**

**if(**arrOpRes **!=** **null)** **{**

response **=** **new** Message**(**OpCode**.**OP\_RES\_ARRAY**,** arrOpRes**);**

**}else** **{**

response **=** **new** Message**(**OpCode**.**OP\_ERROR**);**

**}**

**break;**

**case** DOWNLOAD\_FILE**:** //download <userID> <file>

logger**.**log**(**Level**.**CONFIG**,** "DOWNLOAD\_FILE user: " **+** connectedUser**);**

//users name account that has the file and name of the file

String**[]** ownerFile **=** msg**.**getArrStrParam**();**

**if(**connectedUser**.**equals**(**ownerFile**[**0**])){**

//erro -> é o user local

response **=** **new** Message**(**OpCode**.**ERR\_YOURSELF**);**

//response.setStr("Erro: Utilizador é o mesmo do pedido.");

**}else** **if(!**svM**.**isRegistered**(**ownerFile**[**0**]))** **{**

//user is not registered

response **=** **new** Message**(**OpCode**.**ERR\_NOT\_REGISTERED**);**

**}else** **if(**svM**.**friends**(**connectedUser**,** ownerFile**[**0**])){**

//sao amigos

Byte**[]** byteArray **=** svM**.**sendFileToClient**(**ownerFile**[**0**],**ownerFile**[**1**]);**

**if(**byteArray **==** **null){**

//logger.log(Level.CONFIG, "File not found");

response **=** **new** Message**(**OpCode**.**ERR\_NOT\_FOUND**);**

**}else{**

response **=** **new** Message**(**OpCode**.**OP\_SUCCESSFUL**,** byteArray**);**

**}**

**}else{**

//não sao amigos

//logger.log(Level.CONFIG, "Error, they are not friends");

response **=** **new** Message**(**OpCode**.**ERR\_NOT\_TRUSTED**);**

**}**

**break;**

**case** SEND\_MSG**:** //msg <userID> <msg>

logger**.**log**(**Level**.**CONFIG**,** "SEND\_MSG user: " **+** connectedUser**);**

String**[]** receiverText **=** msg**.**getArrStrParam**();**

**if(**receiverText **==** **null** **||** receiverText**.**length **==** 0**){**

response **=** **new** Message**(**OpCode**.**OP\_ERROR**);**

**}else** **if(!**svM**.**isRegistered**(**receiverText**[**0**]))** **{**

response **=** **new** Message**(**OpCode**.**ERR\_NOT\_REGISTERED**);**

**}else** **if(!**svM**.**friends**(**connectedUser**,**receiverText**[**0**]))** **{**

response **=** **new** Message**(**OpCode**.**ERR\_NOT\_TRUSTED**);**

**}else{**

boolean saved **=** svM**.**storeMsg**(**connectedUser**,** receiverText**[**0**],** receiverText**[**1**]);**

**if(**saved**)** **{**

response **=** **new** Message**(**OpCode**.**OP\_SUCCESSFUL**);**

**}else** **{**

response **=** **new** Message**(**OpCode**.**OP\_ERROR**);**

**}**

**}**

**break;**

**case** COLLECT\_MSG**:**

logger**.**log**(**Level**.**CONFIG**,** "SHOW\_MSG user: " **+** connectedUser**);**

ArrayList**<**String**>** inbox **=** svM**.**collectMsg**(**connectedUser**);**

**if(**inbox **==** **null){**

response **=** **new** Message**(**OpCode**.**OP\_ERROR**);**

**}else{**

response **=** **new** Message**(**OpCode**.**OP\_SUCCESSFUL**,** inbox**);**

**}**

**break;**

**default:**

logger**.**log**(**Level**.**CONFIG**,** "Enum not recognized"**);**

**break;**

**}**

**return** response**;**

**}**

**}**