

Mestrado integrado em Engenharia Telecomunicações e Informática

Sistemas Distribuídos

Docente Carlos Miguel Ferraz Baquero Moreno

Alexandre Teixeira da Silva Vieira A76479

André Filipe Marques da Silva Machado AE3716

João Pedro Pinho Costa A76451

índice

[Introdução 2](#_Toc516431071)

[Desenvolvimento 2](#_Toc516431072)

[- Servidor 2](#_Toc516431073)

[- Cliente 4](#_Toc516431074)

Índice de Figuras

[Figura 1 - Fluxograma da classe 'Servidor'. 3](#_Toc516431635)

[Figura 2 - Fluxograma da classe "PairtoG" e "PairtoU". 4](#_Toc516431636)

[Figura 3 - Fluxograma da classe cliente. 5](#_Toc516431637)

[Figura 4 - Fluxograma do Método "emparelhar". 6](#_Toc516431638)

# Introdução

Na unidade curricular de sistemas distribuídos foi-nos proposto um projeto que visa consolidar e por em prática todos conhecimentos adquiridos ao longo do semestre.

Deste modo foi proposto a realização de um trabalho prático onde é implementado uma arquitetura cliente-servidor de um chat de atendimento. Quando cada cliente efetua uma conexão introduz se é “user” ou “genius”. Após a conexão de um utilizador de cada tipo, este faz match dando origem a um “chat”.

Neste relatório vamos explicar como solucionamos e estruturamos o problema.

# Desenvolvimento

### - Servidor

O sistema está dividido em dois programas em que um desempenha o papel de servidor e o outro o de cliente.

O programa do servidor é constituído por 4 classes **RequestHandler**, **Serv**, **PeerToPeer** e **Servidor**.

Na classe **RequestHandler** é definido o comportamento geral do Servidor em relação às conexões que recebe. Um objeto desta classe tem como variáveis de instância uma List<Socket> para genius e outra para users onde são registados todos os clientes que se conectaram até à data assim como um ServerSocket e um Socket onde é guardado o socket desse servidor e o socket do cliente que se tenta conectar e que posteriormente é adicionado [a uma](https://www.facebook.com/) das listas de clientes. Consoante o novo cliente declara ser um user ou um genius ele é adicionado à sua respetiva lista, de seguida averigua-se qual é a posição que ele ocupa nessa mesma lista, se existir na outra lista um cliente que ocupe a mesma posição ele pode conectar-se a ele, caso contrario terá de esperar que outro cliente se ligue.

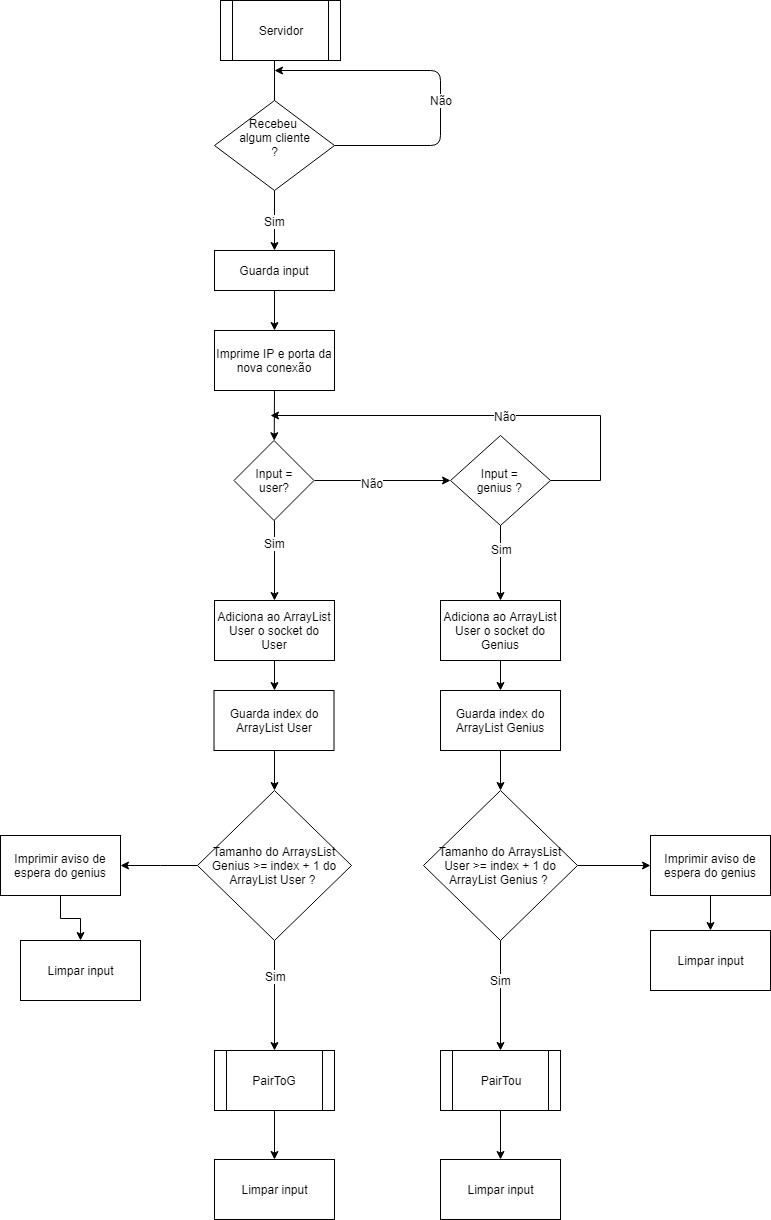


Figura - Fluxograma da classe 'Servidor'.

A classe PeerToPeer contém os métodos que tratam do estabelecimento das conexões por parte do servidor, os métodos PaitToU e PairtToG tem o mesmo funcionamento só que direcionados para os dois tipos de clientes, estes metodos passam para uma String o endereço ip e a porta do socket do cliente que está a aguardar por uma conexão, e retorna essa mesma String que depois é enviada para o cliente que vai estabelecer o emparelhamento. A classe Serv apenas contem o uma variável de instância do tipo ServerSocket e um construtor para um objeto deste tipo.

A classe Servidor é a classe principal onde se encontra a main, onde é declarado um objeto do tipo Serv criando assim o ServerSocket para o servidor, de de seguida é criada uma Thread que executa o método run da classe RequestHandler.

O programa do cliente por sua vez é constituído por 3 classes: **Cliente, ReadThread e WriteThread.**

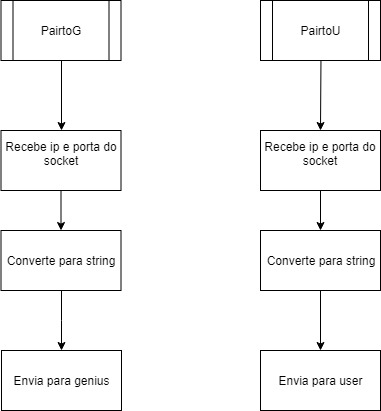


Figura - Fluxograma da classe "PairtoG" e "PairtoU".

### - Cliente

Na classe **Cliente** tem como principal função tentar-se conectar ao servidor através de uma porta e um ip fornecidos por parâmetro a um método desta classe. Após a sua conexão é criado um socket com esse IP e essa porta. De seguida implementou-se um *BufferedReader* que através de um *PrintWritter* é armazenado o que o cliente escreve no terminal. Deste modo o utilizador pode definir se é um “user” / “USER” ou um utilizador do tipo “genius”/ “GENIUS”.

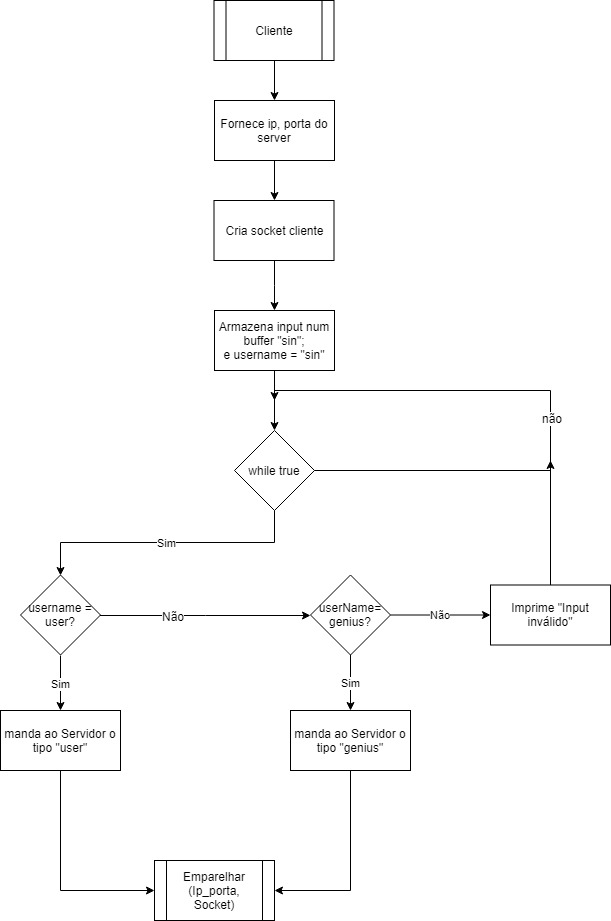
Após esta verificação é enviado essa string para o servidor. E é invocado o método “emparelhar”. 

Figura - Fluxograma da classe cliente.

O método “emparelhar” recebe como parâmetro uma string str e o seu socket. Caso essa string contenha “Aguarde” esse socket é definido como host da ligação e fica a aguardar o par para se comunicar e são invocados as Classes **ReadThread** e **WriteThread**. Caso a string é diferente de “Aguarde” esta cria novo socket e invoca as classes **ReadThread** e **WriteThread.**

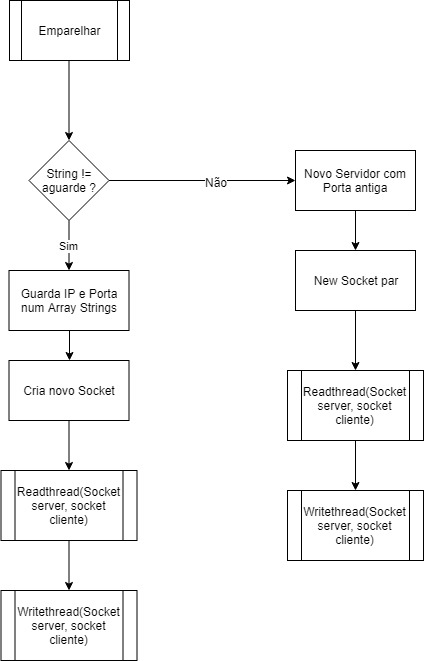
****

Figura - Fluxograma do Método "emparelhar".

A classe **WriteThread** tem como principal funcionalidade a escrita de mensagens nos diferentes pares efetuados. Decidiu-se que este troque mensagens enquanto não se verificar a mensagem “bye” através de Printwriter do output de cada Socket.

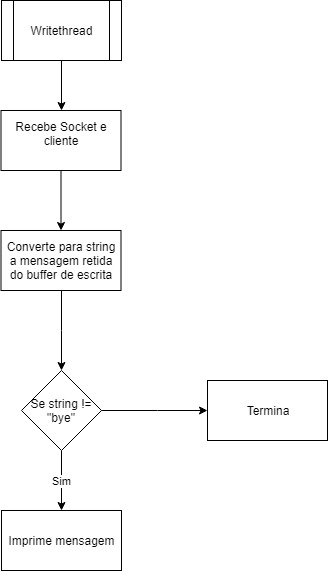


Figura - Fluxograma da classe "writethread".

A classe **Readthread** é responsável por efetuar a leitura de cada par de sockets. Esta lê as mensagens de cada socket e por sua vez armazena o seu input num *BuffereReader reader*.

Decidiu-se por sua vez efetuar isto enquanto não for obtida a mensagem “bye”.

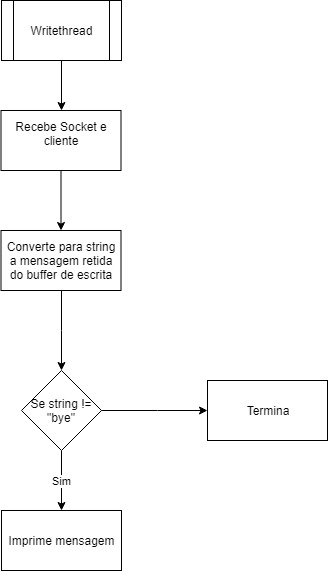


Figura - Fluxograma da classe "Writethread".

# Conclusão

O projeto dá-se assim por finalizado. O grupo faz uma apreciação positiva deste projeto, pois acha que cumpriu todos os requisitos, apesar de ter que lidar com tempo limitado devido a outras entregas de outras unidades curriculares. Sente deste modo que conseguiu transpor no projeto que ficou a entender os conhecimentos adquiridos nesta unidade curricular e ainda demonstrou ter entendido a arquitetura “Cliente-Servidor”.