

Abre: sábado, 15 de novembro de 2025 às 00:00

Data limite: sexta-feira, 12 de dezembro de 2025 às 23:59

Introdução

O trabalho consiste no desenvolvimento em Java de um servidor de *chat* e de um cliente simples para comunicar com ele. O servidor deve basear-se no modelo *multiplex*, aconselhando-se usar como ponto de partida o programa desenvolvido na [ficha de exercícios nº 5](#) das aulas práticas. Quanto ao cliente, deve partir [deste esqueleto](#), que implementa uma interface gráfica simples, e completá-lo com a implementação do lado cliente do protocolo. O cliente deve usar duas *threads*, de modo a poder receber mensagens do servidor enquanto espera que o utilizador escreva a próxima mensagem ou comando (caso contrário bloquearia na leitura da *socket*, tornando a interface inoperacional).

Linha de comando

O servidor deve estar implementado numa classe chamada `ChatServer` e aceitar como argumento da linha de comando o número da porta TCP na qual ficará à escuta, por exemplo:

```
java ChatServer 8000
```

O cliente deve estar implementado numa classe chamada `ChatClient` e aceitar como argumentos da linha de comando o nome DNS do servidor ao qual se quer conectar e o número da porta TCP em que o servidor está à escuta, por exemplo:

```
java ChatClient localhost 8000
```

Protocolo

O protocolo de comunicação é orientado à linha de texto, i.e., cada mensagem enviada pelo cliente ao servidor ou pelo servidor ao cliente deve terminar com uma mudança de linha, e a mensagem propriamente dita não pode conter mudanças de linha. Note que o TCP não faz delimitação de mensagens, pelo que é possível que uma operação de leitura da *socket* retorne apenas parte de uma mensagem ou várias mensagens (podendo a primeira e a última ser parciais). Cabe ao servidor fazer *buffering* por cliente de mensagens parcialmente recebidas¹.

As mensagens enviadas pelo cliente ao servidor podem ser comandos ou mensagens simples. Os comandos são do formato `/comando`, podendo levar argumentos separados por espaços. As mensagens simples apenas podem ser enviadas quando o utilizador está numa sala de *chat*; se começarem por um ou mais caracteres `'` é necessário fazer o seu escape, incluindo um carácter `'` adicional (o servidor deve interpretar este caso especial, enviando aos outros utilizadores da sala a mensagem sem esse carácter extra)²; ocorrências de `'` que não sejam no início da linha não precisam de escape. Este mecanismo de escape garante que, caso surja uma nova versão do protocolo com um comando adicional e um cliente da nova versão envie esse comando a um servidor que só suporta a antiga, o servidor não confunde o novo comando com uma mensagem de texto começada por `'`.

O servidor deve suportar os seguintes comandos:

`/nick nome`

Usado para escolher um nome ou para mudar de nome. O nome escolhido não pode estar já a ser usado por outro utilizador.

`/join sala`

Usado para entrar numa sala de *chat* ou para mudar de sala. Se a sala ainda não existir, é criada.

`/leave`

Usado para o utilizador sair da sala de *chat* em que se encontra.

`/bye`

Usado para sair do *chat*.

As mensagens enviadas pelo servidor ao cliente começam por uma palavra em maiúsculas, indicando o tipo de mensagem, podendo seguir-se um ou mais argumentos, separados por espaços. O servidor pode enviar as seguintes mensagens:

OK

Usado para indicar sucesso do comando enviado pelo cliente.

ERROR

Usado para indicar insucesso do comando enviado pelo cliente.

MESSAGE *nome mensagem*

Usado para difundir aos utilizadores numa sala a *mensagem* (simples) enviada pelo utilizador *nome*, também nessa sala.

NEWNICK *nome_antigo nome_novo*

Usado para indicar a todos os utilizadores duma sala que o utilizador *nome_antigo*, que está nessa sala, mudou de nome para *nome_novo*.

JOINED *nome*

Usado para indicar aos utilizadores numa sala que entrou um novo utilizador, com o nome *nome*, nessa sala.

LEFT *nome*

Usado para indicar aos utilizadores numa sala que o utilizador com o nome *nome*, que também se encontrava nessa sala, saiu.

BYE

Usado para confirmar a um utilizador que invocou o comando */bye* a sua saída.

O servidor mantém, associada a cada cliente, informação de estado, podendo cada cliente estar num dos seguintes estados:

init

Estado inicial de um utilizador que acabou de estabelecer a conexão ao servidor e, portanto, ainda não tem um nome associado.

outside

O utilizador já tem um nome associado, mas não está em nenhuma sala de *chat*.

inside

O utilizador está numa sala de *chat*, podendo enviar mensagens simples (para essa sala) e devendo receber todas as mensagens que os outros utilizadores nessa sala enviem.

O seguinte quadro ilustra as transições de estado possíveis para um utilizador, identificando os eventos que as despoletam e as acções a elas associadas.

Quadro 1: Estados e transições

Estado actual	Evento	Acção	Próximo estado	Notas
init	<i>/nick nome && !disponível(nome)</i>	ERROR	init	
init	<i>/nick nome && disponível(nome)</i>	OK	outside	<i>nome</i> fica indisponível para outros utilizadores
outside	<i>/join sala</i>	OK para o utilizador JOINED <i>nome</i> para os outros utilizadores na sala	inside	entrou na sala <i>sala</i> ; começa a receber mensagens dessa sala
outside	<i>/nick nome && !disponível(nome)</i>	ERROR	outside	mantém o nome antigo
outside	<i>/nick nome && disponível(nome)</i>	OK	outside	
inside	<i>mensagem</i>	MESSAGE <i>nome mensagem</i> para todos os utilizadores na sala	inside	necessário escape de / inicial, i.e., / passa a //, // passa a ///, etc.

Estado actual	Evento	Acção	Próximo estado	Notas
inside	/nick <i>nome</i> &&!disponível(<i>nome</i>)	ERROR	inside	mantém o nome antigo
inside	/nick <i>nome</i> &&disponível(<i>nome</i>)	OK para o utilizador NEWNICK <i>nome_antigo</i> <i>nome</i> para os outros utilizadores na sala	inside	
inside	/join <i>sala</i>	OK para o utilizador LEFT <i>nome</i> para os outros utilizadores na sala antiga JOINED <i>nome</i> para os outros utilizadores na sala nova	inside	entrou na sala <i>sala</i> ; começa a receber mensagens dessa sala; deixa de receber mensagens da sala antiga
inside	/Leave	OK para o utilizador LEFT <i>nome</i> para os outros utilizadores na sala	outside	deixa de receber mensagens
inside	/bye	BYE para o utilizador LEFT <i>nome</i> para os outros utilizadores na sala	—	servidor fecha a conexão ao cliente
inside	utilizador fechou a conexão	LEFT <i>nome</i> para os outros utilizadores na sala	—	servidor fecha a conexão ao cliente
qualquer excepto inside	/bye	BYE para o utilizador	—	servidor fecha a conexão ao cliente
qualquer excepto inside	utilizador fechou a conexão	—	—	servidor fecha a conexão ao cliente
qualquer excepto inside	<i>mensagem</i>	ERROR	mantém o estado	
qualquer	comando não suportado nesse estado	ERROR	mantém o estado	

De seguida, ilustra-se um diálogo entre o cliente e o servidor, onde **C→S** indica uma mensagem enviada do cliente para o servidor, **S→C** indica uma mensagem enviada do servidor para o cliente e **S→O** indica uma mensagem enviada do servidor para os outros clientes que estão na mesma sala.

(cliente estabelece conexão para o servidor)

```

C→S: /nick maria
S→C: ERROR
C→S: /nick miquinhas
S→C: OK
C→S: /join moda
S→C: OK
S→O: JOINED miquinhas
C→S: Olá a todos!
S→C: MESSAGE miquinhas Olá a todos!
S→O: MESSAGE miquinhas Olá a todos!
S→C: MESSAGE francisca Olá, miquinhas!
S→C: MESSAGE berto Olá, miquinhas!
C→S: /nick micas
S→C: OK
S→O: NEWNICK miquinhas micas
C→S: Tchau!
S→C: MESSAGE micas Tchau!
S→O: MESSAGE micas Tchau!

```

```

C→S: /join C++
S→C: OK
S→O: LEFT micas (na sala moda)
S→O: JOINED micas (na sala C++)
C→S: /// marca o início de um comentário em C++ (note que o utilizador escreveu // marca o
início de um comentário em C++)
S→C: MESSAGE micas // marca o início de um comentário em C++
S→O: MESSAGE micas // marca o início de um comentário em C++
C→S: /bye
S→C: BYE (servidor fecha a conexão à micas)
S→O: LEFT micas

```

IMPORTANTE: Dado que serão feitos testes automáticos, é muito importante que o serviço implementado cumpra escrupulosamente a especificação. As mensagens deverão ter exactamente o formato indicado, e o servidor não deve enviar nada mais para além delas (nem sequer uma mensagem inicial de boas-vindas).

Valorização

A implementação inteiramente correcta do servidor acima descrito será valorizada com 50% da cotação do trabalho.

A implementação inteiramente correcta do cliente acima descrito será valorizada com 35% da cotação do trabalho.

Se, adicionalmente, implementar no servidor o comando `/priv nome mensagem` (ver abaixo), obterá mais 10%.

Se processar as mensagens recebidas pelo cliente de modo a que na área de *chat* não apareça directamente o que foi recebido do servidor, mas sim o seu conteúdo num formato mais amigável (ver exemplo abaixo), obterá mais 5%.

O comando `/priv nome mensagem` serve para enviar ao utilizador *nome* (e apenas a ele) a *mensagem*. Se o utilizador *nome* não existir, o servidor deverá devolver `ERROR`, caso contrário deverá devolver `OK` e enviar ao utilizador *nome* a mensagem `PRIVATE emissor mensagem` (onde *emissor* é o nickname de quem enviou a mensagem). Note que podem estar em salas diferentes, ou mesmo em nenhuma.

Por formato mais amigável entende-se, por exemplo, que quando é recebida do servidor a mensagem `MESSAGE nome mensagem` seja mostrado na área de chat `nome: mensagem`, que quando é recebida do servidor a mensagem `NEWNICK nome_antigo nome_novo` seja mostrado na área de chat `nome_antigo mudou de nome para nome_novo`, etc.

Notas

1. É particularmente importante o servidor lidar correctamente com a delineação das mensagens. Para testar este aspecto pode usar como cliente o `ncat` (ou `netcat`, ou `nc`).
 - Para testar o envio de uma única mensagem partida em vários pacotes faça `ncat localhost 8000` e escreva `/ni<CTRL-D>ck bom<CTRL-D>rapaz<ENTER>`. O servidor não deve interpretar o comando ao fazer `<CTRL-D>`, apenas *bufferizar* os pedaços da mensagem (`/ni`, `ck bom` e `rapaz`). O comando completo (`/nick bomrapaz`) só deve ser processado quando fizer `<ENTER>`.
 - Para testar o envio de múltiplas mensagens num único pacote pode criar um ficheiro com as linhas

```

/nick bomrapaz
/join sala
Bom dia!

```

e fazer `ncat localhost 8000 < ficheiro`. O servidor deve interpretar o que recebe como dois comandos e uma mensagem de texto.

2. Por exemplo, se o utilizador `joe` escrever `/notacommand`, que não é nenhum dos comandos do protocolo, o cliente deve enviar ao servidor a mensagem `//notacommand`. O servidor detecta o mecanismo de escape, remove a `/` extra e envia aos clientes `MESSAGE joe`

`/notacommand`. Se o utilizador escrever `//comment`, o cliente envia ao servidor `///comment` e o servidor envia aos clientes `MESSAGE joe //comment`.

Entrega

O trabalho será desenvolvido em grupos de dois elementos. Cada grupo deverá entregar um arquivo .zip contendo exactamente:

- Um ficheiro `ChatServer.java` com o código-fonte do servidor.
- Um ficheiro `ChatClient.java` com o código-fonte do cliente.
- Um ficheiro `grupo.txt` com a identificação dos dois elementos do grupo no formato aqui exemplificado:

201012345 Ana Beatriz Carvalho Duarte
201054321 Eduardo Fernando Gonçalves Henriques

Se o cliente e/ou o servidor estiverem organizados em diferentes classes, deverão incluir ficheiros `.java` com o respectivo código-fonte (contudo, as classes principais do servidor e do cliente deverão ser, respectivamente, `ChatServer` e `ChatClient`). A submissão deverá ser feita no Moodle por apenas um dos elementos do grupo. O incumprimento das regras de submissão será penalizado.

Código de honra

A submissão de um trabalho implica a garantia, sob compromisso de honra, de que o trabalho submetido resultou exclusivamente do esforço dos elementos do respectivo grupo, não tendo havido qualquer partilha de código com pessoas externas ao grupo. É expressamente proibido usar código de outros grupos, bem como fornecer código a outros grupos.