FCT Boleia

Algoritmos e Estruturas de Dados
Enunciado do Trabalho Prático, versão 1.0 – 2019-9-27

Notas importantes

Trabalho em Grupo de 2 alunos a desenvolver respeitando os princípios do *Código de Ética* do Departamento de Informática.

Trabalho com 3 Fases de entrega. A 1ª fase consta dum relatório com o desenho da solução proposta para a resolução do problema, o qual deve incluir o diagrama de classes da solução (10% da nota final na cadeira). A 2ª fase consta dum relatório e dum programa submetido na plataforma Mooshak, o qual pode utilizar as interfaces e classes da API do Java. (10% da nota final na cadeira). A 3ª fase consta dum relatório e dum programa submetido na plataforma Mooshak. Neste programa todas as interfaces e classes usadas devem ser implementadas pelos alunos (20% da nota final na cadeira); a utilização de qualquer classe ou interface do package java.util (excepto a classe Scanner), levará à anulação da entrega da 3ªfase (0 valores na fase).

Prazos de entrega das 3 fases. A 1ª fase deve ser entregue até às 17:00 horas do dia 15 de outubro de 2019. A 2ª fase deve ser entregue até às 17:00 horas do dia 12 de novembro de 2019. A 3ª fase deve ser entregue até às 17:00 horas do dia 6 de dezembro de 2019.

Frequência na cadeira de AED é dada pela média ponderada das três fases do trabalho. Para obter frequência, essa nota deve ser superior ou igual a 9 valores em 20.

Submissão e aceitação do código fonte pelo sistema de avaliação automática Mooshak. Os relatórios escritos de todas as fases devem ser entregues no moodle nas datas indicadas acima.

1 Introdução

Este documento descreve o trabalho prático da disciplina de Algoritmos e Estruturas de Dados do 2º ano do Mestrado Integrado em Engenharia Informática. Os alunos podem e devem tirar todas as dúvidas com os docentes da disciplina.

2 Desenvolvimento da Aplicação *FCT Boleia*

O objectivo deste trabalho é o desenvolvimento de um sistema de partilha de carros, em que se oferecem e pedem "boleias".

2.1 Descrição do Problema

Dar e pedir boleia é uma coisa que já provavelmente fez com alguns dos seus amigos: é simplesmente o facto de deslocar-se com pessoas partilhando um carro e os custos da deslocação (combustível e portagens).

Pretende-se criar uma aplicação para facilitar o uso desta forma de viajar. A aplicação deve permitir que as pessoas encontrem e/ou partilhem um carro para se deslocarem juntos, "à boleia". Esta aplicação tem como funcionalidades principais:

- registar uma dada deslocação, que poderá ser partilhada;
- remover uma dada deslocação registada;
- registar uma boleia para uma dada deslocação;
- desistir de uma boleia;
- consultar uma deslocação registada (locais de origem e destino, data, hora, duração prevista, lugares vagos, emails de utilizadores em boleia, número de utilizadores em espera para essa boleia,...);
- listar as diferentes deslocações registadas por vários critérios.

Para poder usar estas funcionalidades, o utilizador deve registar-se na aplicação com o seu email. Assim sendo, o programa pode encontrar-se em dois modos:

- inicial: aqui é possível fazer registos de novos utilizadores, iniciar uma sessão com um utilizador e terminar a execução do programa;
- sessão: aqui é possível registar, partilhar e consultar viagens.

A interação do utilizador com o programa é feita sempre através de comandos, descritos na secção seguinte. Sempre que se inicia a execução da aplicação, esta fica no modo inicial, e o prompt é ">". Quando um utilizador já registado faz *login* na aplicação, o programa passa a modo "sessão", e o prompt é "emailUtilizador >"(sendo emailUtilizador o email do utilizador que entrou na aplicação.

Prevê-se que existam mais de 10000 utilizadores registados, e que cada utilizador tenha quase sempre 1 deslocação diária registada. Cada deslocação poderá ter no máximo 10 boleias. No sistema assuma que só existem deslocações de no máximo 3 anos diferentes.
As funcionalidades mais utilizadas são: as listagens (comando **lista**), o registo de boleia (comando **boleia**), e a remoção de deslocação (comando **remove**). Destas funcionalidades a mais frequente é a listagem por data, a segunda mais frequente é a remoção de deslocação, a terceira mais frequente é o registo de boleia, e por último as restantes listagens.

A persistência dos dados deve ser assegurada entre execuções consecutivas.
Isto significa que, antes de terminar a execução, o sistema deve guardar o estado da base de dados em disco, utilizando as funcionalidades de serialização do Java. Na próxima execução do programa, os dados armazenados deverão ser carregados do disco e o estado do sistema reconstituído.

3 Comandos

Nesta secção apresentam-se os vários comandos que o sistema deve ser capaz de interpretar e executar. Nos exemplos apresentados, diferenciamos o `texto escrito pelo utilizador` da retroacção escrita pelo programa na consola, ao executar o comando.

Vários comandos têm argumentos. A entrada e a saída deverão respeitar o formato rígido que se indica neste documento. Pode assumir que o utilizador não cometerá erros na introdução de argumentos nos comandos, para além dos descritos neste enunciado, ou seja, apenas tem de tratar as situações de erro descritas aqui, pela ordem que são descritas. Pode assumir que o utilizador apenas usa argumentos em número e de tipos correctos. No entanto, pode acontecer que os argumentos passados a um desses comandos tenham algum valor inválido no contexto do problema. Por esse motivo, teremos de testar esses argumentos exactamente pela ordem especificada neste enunciado.

Na leitura de comandos o interpretador não deve fazer distinção entre maiúsculas e minúsculas. Por exemplo, para o comando `sai`, o interpretador deve aceitar como válidas as alternativas `SAI`, `sai`, `Sai`, `sAI`, etc. Os caracteres permitidos são: abecedário em maiúscula ou minúscula, os digitos de 0 a 9, e o simbolo "@". Nos vários exemplos que se seguem, o símbolo ↵ denota a mudança de linha.

A tabela 1 apresenta os comandos do sistema, cujos detalhes são apresentados nas secções seguintes.

Tabela 1: Comandos

Comando	Modos	Descrição	Detalhes
`lista`	sessão	listagem de deslocações	Secção 3.1
`regista`	inicial	inserção de novo utilizador	Secção 3.2
`entrada`	inicial	inicia sessão de utilizador	Secção 3.3
`sai`	sessão	termina sessão de utilizador	Secção 3.4
`termina`	inicial	termina a execução do programa	Secção 3.5
`ajuda`	inicial, sessão	informação sobre comandos disponíveis	Secção 3.6
`nova`	sessão	inserção de nova deslocação	Secção 3.7
`remove`	sessão	remoção de uma deslocação	Secção 3.8
`boleia`	sessão	inserção de uma boleia	Secção 3.9
`retira`	sessão	remoção de uma boleia	Secção 3.10
`consulta`	sessão	informação sobre uma deslocação	Secção 3.11

Caso o utilizador introduza um comando inexistente ou um comando existente mas não aceite num dado modo do programa ("inicial"ou "sessão"), o programa escreve na consola a mensagem "Comando invalido.". Por exemplo o comando inexistente `desconhecido` teria este efeito:

```
> desconhecido↵
Comando invalido.↵
>
```

Caso o programa esteja em modo "inicial", o comando `sai` teria este efeito:

```
> sai↵
Comando invalido.↵
>
```

3.1 Comando `lista`

Lista todas ou algumas deslocações registadas. Este comando só terá sucesso se o programa estiver em modo sessão.

Este comando é um dos mais usados, pelo que a sua implementação deve ser cuidada de modo a ser eficiente.

Existem cinco modos de listagem. O comando `lista` tem um parâmetro, que aqui chamamos *`modoListagem`*, que serve para indicar qual o modo de listagem. Os valores possíveis de *`modoListagem`* são: (1) uma das palavras: `minhas` ou `boleias` ou `todas`; ou (2) uma data; ou (3) um email.

- Caso seja escrita a palavra `minhas`, deve-se listar a informação referente às deslocações do utilizador corrente (secção 3.1.1);
- Caso seja escrita a palavra `boleias`, deve-se listar a informação referente às boleias registadas do utilizador corrente (secção 3.1.2);
- Caso seja dado um email, deve-se listar a informação referente às deslocações do utilizador com o email dado (secção 3.1.3);
- Caso seja dado uma data, deve-se listar a informação referente às deslocações de todos os utilizadores na data indicada (secção 3.1.4);
- Caso seja escrita a palavra `todas`, deve-se listar a informação (data + email) referente às deslocações de todos os utilizadores (secção 3.1.5).

O formato geral do comando é o seguinte, onde cada listagem-deslocação-i representa a informação sobre uma deslocação:

```
emailUtilizador > lista modoListagem↵
listagem-deslocação-1↵
↵
listagem-deslocação-2↵
↵
...
listagem-deslocação-n↵
↵
emailUtilizador >
```

A listagem-deslocação-i pode conter:

1. email do utilizador que a registou, no caso de *modoListagem* ser *`diaDesloc-mesDesloc-anoDesloc`*;

2. data da deslocação e email do utilizador que a registou, no caso de *modoListagem* ser `todas`;

3. email do utilizador que a registou, local de origem e destino, data, hora e duração da deslocação, no caso de *modoListagem* ser `boleias` ou *`emailUtilizador`*;

4. informação dada em (3) mais número de lugares vagos, lista de emails de utilizadores cuja boleia foi aceite e número de utilizadores em espera para terem boleia, no caso de *modoListagem* ser `minhas`.

Serão dados mais detalhes sobre o formato da informação contida em *listagem-deslocação-i* nas secções seguintes.
As listagens das deslocações estão ordenadas por data, e em caso de empate na data pela ordem alfabética crescente do email do utilizador que as registou. As listagens de emails (das boleias numa dada deslocação) estão ordenadas por ordem de registo da boleia.

Caso não haja registo de nenhuma deslocação com os detalhes pretendidos, o sistema apresenta uma mensagem de erro. Ou seja, usa-se uma mensagem de erro no caso de *modoListagem* ser minhas e não existirem deslocações do utilizador corrente; no caso de ser boleias e não existirem boleias do utilizador corrente; no caso de ser todas e não existirem deslocações registadas para nenhum utilizador; e no caso de ser um email ou uma data e não existirem deslocações do utilizador com esse email, ou não existirem deslocações nessa data, respectivamente. Nestes casos obtemos a seguinte mensagem de erro:

```
emailUtilizador > lista modoListagem↵
Sem deslocacoes.↵
emailUtilizador >
```

3.1.1 Listar as `minhas` deslocações registadas

Quando se executa o comando `lista` com o argumento `minhas` obtém-se uma listagem de todas as deslocações do utilizador corrente.

Vejamos um exemplo em que o utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) tenha registado 2 deslocações:

```
emailUtilizador > lista minhas↵
aa@fct.unl.pt↵
Lisboa-Alvalade do Sado↵
20-12-2018 15:30 90↵
Lugares vagos: 5↵
Sem boleias registadas.↵
Em espera: 0↵
↵
aa@fct.unl.pt↵
Porto-Monte da Caparica↵
21-12-2018 10:45 120↵
Lugares vagos: 3↵
Boleias: bb@fct.unl.pt; cc@fct.unl.pt↵
Em espera: 0↵
↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando `lista minhas` é: (1) não estar em modo "sessão"(comando inválido).

3.1.2 Listar as minhas **boleias** registadas

Quando se executa o comando `lista` com o argumento `boleias`, obtém-se uma listagem de todas as boleias registadas do utilizador corrente.

No exemplo seguinte, suponha que o utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) registou 2 boleias. Ao executar o comando `lista` com o argumento `boleias`, o resultado é o seguinte:

```
aa@fct.unl.pt > lista boleias↵
bb@fct.unl.pt↵
Lisboa-Beja↵
12-12-2018 15:30 90↵
↵
dd@fct.unl.pt↵
Porto-Lisboa↵
15-12-2018 10:45 120↵
↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando `lista boleias` é: (1) não estar em modo "sessão"(comando inválido).

3.1.3 Listar as deslocações registadas por um dado email

Quando o argumento do comando `lista` é o email dum dado utilizador, obtém-se uma listagem de todas as deslocações registadas do utilizador com esse email.

Vejamos um exemplo. O utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) executa o comando `lista` com o argumento bb@fct.unl.pt, que é o email dum dado utilizador que tem uma deslocação registada, o que se obtem é o seguinte resultado:

```
aa@fct.unl.pt > lista bb@fct.unl.pt↵
bb@fct.unl.pt↵
Lisboa-Alvalade do Sado↵
20-12-2018 15:30 90↵
↵
aa@fct.unl.pt >
```

Por outro lado, caso o utilizador indicado, por exemplo *cccc@fct.unl.pt*, não exista, obtemos o seguinte:

```
aa@fct.unl.pt > lista cccc@fct.unl.pt↵
Nao existe o utilizador dado.↵
aa@fct.unl.pt >
```

Caso o utilizador indicado seja o último utilizador que entrou no programa com o comando `entrada`, ou seja, caso *emailUtilizadorR* seja o utilizador corrente, o comando é equiva-

lente ao comando `lista minhas`, ainda que a informação dada de cada deslocação não é a mesma.

A ordem de verificação de erros neste comando `lista` com um argumento que é um email, é: (1) não estar em modo "sessão"(comando inválido); (2) o email dado não é de um utilizador registado.

3.1.4 Listar as deslocações numa dada data

Quando se executa o comando `lista` com o argumento *`diaDesloc-mesDesloc-anoDesloc`*, obtém-se uma listagem de todas as deslocacões registadas para essa data. Nesta listagem só aparecem deslocações que ainda tem lugares vagos. Ou seja, quando o argumento do comando `lista` é uma data, obtém-se uma listagem das deslocações com lugares vagos, de todos os utilizadores, nessa data.

Vejamos um exemplo em que o utilizador com email *aa@fct.unl.pt* (utilizador com nome *Fernanda AA*, dado no registo de utilizador) executa o comando `lista`, dando uma data. Assuma que existem 4 deslocações registadas nessa data com lugares vagos. Os utilizadores que as registaram têm os seguintes emails *aa@fct.unl.pt*, *bb@fct.unl.pt*, *cc@fct.unl.pt* e *dd@fct.unl.pt*. Neste caso obtemos o seguinte resultado:

```
aa@fct.unl.pt > lista 20-12-2018↵
aa@fct.unl.pt↵
↵
bb@fct.unl.pt↵
↵
cc@fct.unl.pt↵
↵
dd@fct.unl.pt↵
↵
aa@fct.unl.pt >
```

Caso a data seja inválida, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > lista 0-12-2018↵
Data invalida.↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando `lista` com um argumento que é uma data, é: (1) não estar em modo "sessão"(comando inválido); (2) ter uma data inválida.

3.1.5 Listar todas as deslocações

Quando se executa o comando `lista` com o argumento `todas`, obtém-se uma listagem de todas as deslocacões registadas.

Vejamos um exemplo em que o utilizador com email *zz@fct.unl.pt* (utilizador com nome *ze*, dado no registo de utilizador) executa o comando `lista todas`. Assuma que existem 4 utilizadores que têm os seguintes emails *aa@fct.unl.pt*, *bb@fct.unl.pt*, *zz@fct.unl.pt* e *dd@fct.unl.pt*.

O utlizador *aa@fct.unl* não tem deslocações registadas. O utlizador *bb@fct.unl.pt* tem 2 deslocações registadas em 13-4-2020 e 1-12-2019. O utlizador *zz@fct.unl.pt* tem uma deslocação registada em 1-12-2019. O utlizador *dd@fct.unl.pt* tem 2 deslocações registadas em 13-5-2018 e 1-12-2019. Neste caso obtemos o seguinte resultado:

```
zz@fct.unl.pt > lista todas↵
13-5-2018 dd@fct.unl.pt↵
↵
1-12-2019 bb@fct.unl.pt↵
↵
1-12-2019 dd@fct.unl.pt↵
↵
1-12-2019 zz@fct.unl.pt↵
↵
13-4-2020 bb@fct.unl.pt↵
↵
zz@fct.unl.pt >
```

A ordem de verificação de erros neste comando `lista todas` é: (1) não estar em modo "sessão"(comando inválido).

3.2 Comando **regista**

Registra um novo utilizador no programa. Este comando só tem sucesso se o programa estiver em modo inicial, e não exisitir um utilizador com o email dado. Para realizar um registo deve ser dado sempre o email. Este email tem que ser único, e nesse caso é pedido o nome e a password do utilizador.

No caso do programa estar em modo inicial, o email ser único e a password ser válida, o formato do comando é o seguinte (parâmetros com estilo **bold**):

```
> regista emailUtilizador↵
nome (maximo 50 caracteres): nomeCompletoUtilizador↵
password (entre 4 e 6 caracteres - digitos e letras): passwordUtilizador↵
Registo N efetuado.↵
>
```

Note que *N* será o número de utilizadores registados até ao momento. Assuma que o nome do utilizador é uma String não vazia que não ultrapassa os 50 caracteres e é sempre válido. No entanto, a password deve ser verificada. A password para ser válida deve ter uma dimensão entre 4 e 6 caracteres, e estes devem ser letras (maiúsculas ou minúsculas) e dígitos. Caso não seja válida, o programa deve voltar a pedir a mesma um máximo de 3 vezes. Por exemplo:

```
> regista aa@fct.unl.pt↵
nome (maximo 50 caracteres): Fernanda AA↵
password (entre 4 e 6 caracteres - digitos e letras): aed↵
password (entre 4 e 6 caracteres - digitos e letras): aed_2↵
password (entre 4 e 6 caracteres - digitos e letras): aed18↵
Registo 1 efetuado.↵
>
```

Caso não seja dada uma password válida nas 3 tentativas o registo não será efetuado e escreve-se na consola Registo nao realizado.

```
> regista aa@fct.unl.pt↵
nome (maximo 50 caracteres): Fernanda AA↵
password (entre 4 e 6 caracteres - digitos e letras): aed↵
password (entre 4 e 6 caracteres - digitos e letras): aed2018↵
password (entre 4 e 6 caracteres - digitos e letras): aeda↵
Registo nao realizado.
>
```

Um exemplo da execução do comando `regista`, em que o email do utilizador **aa@fct.unl.pt** já existe. Neste caso, escreve-se na consola Utilizador ja existente.

```
> regista aa@fct.unl.pt↵
Utilizador ja existente.↵
>
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "inicial"(comando inválido); (2) o utilizador já existe e (3) não ter dado uma password válida nas 3 tentativas.

3.3 Comando `entrada`

Permite a entrada ("login") dum utilizador no programa. Este comando só tem sucesso se o programa estiver em modo inicial, e existir um utilizador com o email dado. Para realizar a entrada na aplicação deve ser dado sempre o email. Este email tem que ser de um utilizador já registado. Neste caso é pedido a password, tendo o utilizador 3 tentativas para a editar correctamente.

No caso do programa estar em modo inicial, o email e a password serem dum utilizador registado, o formato do comando é o seguinte:

```
> entrada emailUtilizador↵
password: passwordUtilizador↵
Visita N efetuada.
emailUtilizador >
```

Note que *N* é o número de visitas ("login") já efetuados pelo utilizador que acaba de entrar. Note também que, quando o comando `entrada` tem sucesso, o sistema passa para o modo sessão e o prompt seguinte tem o email do utilizador que fez login. Caso a password não seja correcta, o programa deve voltar a pedir (no máximo mais 2 vezes) até que seja dado um valor correcto. Por exemplo:

```
> entrada aa@fct.unl.pt↵
password: aed↵
password: aeda↵
password: aed18↵
Visita 1 efetuada.↵
aa@fct.unl.pt >
```

Caso não seja dada a password correcta, a entrada não é efetuada e escreve-se na consola Entrada nao realizada.

```
> entrada aa@fct.unl.pt↵
password: aed22↵
password: aed2018↵
password: aed1↵
Entrada nao realizada.↵
>
```

Um exemplo da execução do comando `entrada`, em que o email do utilizador **aa@fct.unl.pt** não existe. Neste caso, escreve-se na consola Utilizador nao existente.

```
> entrada aa@fct.unl.pt↵
Utilizador nao existente.↵
>
```

A ordem de verificação de erros neste comando é: (1) estar em modo "sessão"(comando inválido); (2) o utilizador não existe e (3) não ter dado a password válida nas 3 tentativas.

3.4 Comando `sai`

Termina a sessão do utilizador no programa. O comando não necessita de argumentos. Este comando só tem sucesso se o programa estiver em modo sessão. Note também que, quando o comando `sai` tem sucesso, o sistema passa para o modo inicial e o prompt seguinte já não tem o email do utilizador que estava em sessão.

No caso do programa estar em modo sessão, o formato do comando é o seguinte:

emailUtilizador > `sai`↵
Ate a proxima *nomeUtilizador*.↵

Um exemplo da execução do comando `sai`, quando o programa está em modo sessão com o utilizador **aa@fct.unl.pt**.

```
aa@fct.unl.pt > sai↵
Ate a proxima Fernanda AA.↵
>
```

O único erro neste comando é: estar em modo "inicial"(comando inválido).

3.5 Comando `termina`

Termina a execução do programa. O comando não necessita de argumentos. Este comando só tem sucesso se o programa estiver em modo inicial.

No caso do programa estar em modo inicial, o formato do comando é o seguinte:

```
> termina↵
Obrigado. Ate a proxima.↵
>
```

O único erro neste comando é: estar em modo "sessão"(comando inválido).

3.6 Comando `ajuda`

Informa sobre os comandos disponíveis no programa. O comando não necessita de argumentos e tem sempre sucesso. Este comando tem 2 formatos distintos, dependendo do modo em que o programa está. No caso de ainda não se ter realizado um login (modo inicial), o formato do comando é o seguinte:

```
> ajuda↵
ajuda - Mostra os comandos existentes↵
termina - Termina a execucao do programa↵
regista - Regista um novo utilizador no programa↵
entrada - Permite a entrada ("login") dum utilizador no programa↵
>
```

No caso do programa estar em modo sessão (já foi feito o comando `entrada`), o formato do comando é o seguinte:

```
emailUtilizador > ajuda↵
ajuda - Mostra os comandos existentes↵
sai - Termina a sessao deste utilizador no programa↵
nova - Regista uma nova deslocacao↵
lista - Lista todas ou algumas deslocacoes registadas↵
boleia - Regista uma boleia para uma dada deslocacao↵
consulta - Lista a informacao de uma dada deslocacao↵
retira - Retira uma dada boleia↵
remove - Elimina uma dada deslocacao↵
emailUtilizador >
```

Note que, quando o programa está em modo sessão, no prompt está o email do utilizador que fez login (comando `entrada`). Caso seja o utilizador com email **aa@fct.unl.pt** a executar o comando `ajuda`, o formato seria:

```
aa@fct.unl.pt > ajuda↵
ajuda - Mostra os comandos existentes↵
sai - Termina a sessao do utilizador no programa↵
nova - Regista uma nova deslocacao↵
lista - Lista todas ou algumas deslocacoes registadas↵
boleia - Regista uma boleia para uma dada deslocacao↵
consulta - Lista a informacao de uma dada deslocacao↵
retira - Retira uma dada boleia↵
remove - Elimina uma dada deslocacao↵
aa@fct.unl.pt >
```

3.7 Comando `nova`

Regista uma nova deslocação. Este comando só tem sucesso se o programa estiver em modo sessão. A deslocação é registada pelo utilizador da sessão.

Quando se regista a deslocação deve ser dada a seguinte informação: locais de origem e destino, data e horário da viagem (dia, mês, ano, hora e minutos estimados de partida e duração em minutos estimada de viagem) e número de lugares do carro disponíveis para boleia. Note que a hora estimada está no formato hh:mm (hh é um valor inteiro entre 0 e 23 e mm é um valor inteiro entre 0 e 59), e a duração é um valor inteiro positivo (maior que zero). Este registo tem sucesso se não existir uma deslocação ou boleia deste utilizador no mesmo dia, isto é, no sistema o utilizador só pode ter uma deslocação ou boleia por dia.

O formato geral do comando é o seguinte:

```
emailUtilizador > nova↵
localOrigem↵
localDestino↵
dia-mes-ano hora:minutos duracao numeroLugaresParaBoleia↵
Deslocacao N registada. Obrigado nomeUtilizador.↵
emailUtilizador >
```

Note que *N* é o número de deslocações já registadas pelo utilizador. Um exemplo do primeiro registo de deslocação do utilizador com email **aa@fct.unl.pt** e nome *Fernanda AA*, poderia ser:

```
aa@fct.unl.pt > nova↵
Lisboa↵
Alvalade do Sado↵
20-12-2018 15:00 90 5↵
Deslocacao 1 registada. Obrigado Fernanda AA.↵
aa@fct.unl.pt >
```

Por exemplo, suponha agora que o mesmo utilizador pretende registar uma deslocação entre Porto e Monte da Caparica no dia 20-12-2018 pelas 9:30 horas com uma duração estimada de 180 minutos, e 5 lugares disponíveis. Neste caso, o registo não é efetuado pois já existe **uma deslocação ou boleia** nessa data para este utilizador.

```
aa@fct.unl.pt > nova↵
Porto↵
Monte da Caparica↵
20-12-2018 9:30 180 5↵
Fernanda AA ja tem uma deslocacao ou boleia registada nesta data.↵
aa@fct.unl.pt >
```

Caso existam erros nos dados fornecidos pelo utilizador, o registo não é efetuado e escreve-se a mensagem de erro Dados invalidos. Os erros possíveis são: data inválida, horário inválido, duração, lugares e º de boleias disponíveis com valores não positivos, e nº de boleias superior ao valor máximo (10). Um exemplo desta situação é:

```
aa@fct.unl.pt > nova↵
Porto↵
Monte da Caparica↵
20-13-2018 15:78 180 5↵
Dados invalidos.↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"(comando inválido); (2) ter dados inválidos, e (3) ter uma deslocação ou boleia na data indicada.

3.8 Comando `remove`

Remove uma deslocação. Caso o utilizador tenha registado uma deslocação nesse dia, essa deslocação será eliminada se não houver registo de boleias para essa deslocação. Este comando só tem sucesso se o sistema estiver em modo sessão.

Este comando é um dos mais usados, pelo que a sua implementação deve ser cuidada de modo a ser eficiente.

O formato geral deste comando é o seguinte, em que diaDesloc-mesDesloc-anoDesloc é a data da deslocação:

```
emailUtilizador > remove diaDesloc-mesDesloc-anoDesloc↵
Deslocacao removida.↵
emailUtilizador >
```

Vejamos um exemplo. Assuma que o utilizador com email **aa@fct.unl.pt** e nome **Fernanda AA** registou duas deslocações: uma no dia 23-12-2019 e essa deslocação não tem boleias registadas, e uma no dia 25-12-2018 e essa deslocação já tem boleias registadas. Caso o utilizador trate de retirar a deslocação do dia 23-12-2019, esta é eliminada com sucesso:

```
aa@fct.unl.pt > remove 23-12-2019↵
Deslocacao removida.↵
aa@fct.unl.pt >
```

Caso o utilizador tente remover a deslocação do dia 25-12-2018, a deslocação não é eliminada porque tem boleias registadas:

```
aa@fct.unl.pt > remove 25-12-2018↵
Fernanda AA ja nao pode eliminar esta deslocacao.↵
aa@fct.unl.pt >
```

Caso o utilizador tente remover um registo de deslocação do dia 22-12-2020, e esse utilizador não tenha uma deslocação registada nesse dia, esse facto é indicado da seguinte forma:

```
aa@fct.unl.pt > remove 22-12-2020↵
Fernanda AA nesta data nao tem registo de deslocacao.↵
aa@fct.unl.pt >
```

Caso a data seja inválida, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > remove 0-12-2018↵
Data invalida.↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"(comando inválido); (2) data inválida; e (3) não existir uma deslocação nesse dia para o utilizador corrente; (4) a deslocação tem boleias já registadas.

3.9 Comando `boleia`

Regista uma boleia para uma dada deslocação. Regista mais um lugar ocupado para uma dada deslocação. Este comando só tem sucesso se o sistema estiver em modo sessão, e essa deslocação exista, tenha lugares vagos, não tenha sido registada pelo utilizador corrente, e não haja já uma boleia ou deslocação registada nesse dia para o utilizador corrente.

Caso não haja lugar vago, o utilizador corrente fica em fila de espera, para o caso de haver remoção de alguma boleia.

Este comando é um dos mais usados, pelo que a sua implementação deve ser cuidada de modo a ser eficiente.

O formato geral deste comando é o seguinte, em que *emailUtilizadorR* é o email do utilizador que registou a deslocação e *diaDesloc-mesDesloc-anoDesloc* é a data da deslocação:

emailUtilizador > boleia **emailUtilizadorR** *diaDesloc-mesDesloc-anoDesloc* ↵
Boleia registada.↵
emailUtilizador >

Vejamos um exemplo desta situação. Assuma que existe uma deslocação do utilizador **bb@fct.unl.pt** no dia 20-12-2018 com 1 lugar vago e que o utilizador corrente, **aa@fct.unl.pt**, não tem nenhuma boleia ou deslocação nesse dia. Se o utilizador ***aa@fct.unl.pt*** executar o comando boleia, indicando o email do utilizador que registou a deslocação (**bb@fct.unl.pt**), e a data da deslocação, a boleia é registada.

aa@fct.unl.pt >boleia **bb@fct.unl.pt 20-12-2018**↵
Boleia registada.↵
aa@fct.unl.pt >

Caso exista a deslocação mas já não tenha lugares vagos, o utilizador corrente (por exemplo o utilizador *mm@fct.unl.pt*) fica numa fila de espera e essa informação é dada da seguinte forma (onde *n* é a posição na fila do utilizador, e é um número entre 1 e tamanho da fila):

mm@fct.unl.pt >boleia **bb@fct.unl.pt 20-12-2018**↵
Ficou na fila de espera (posicao *n*).↵
mm@fct.unl.pt >

Quando houver remoções de boleias os utilizadores na fila vão entrando nas boleias, seguindo um comportamento FIFO.

Caso exista a deslocação mas tenha sido registada pelo próprio utilizador, esse facto é dito ao utilizador da seguinte forma:

aa@fct.unl.pt > boleia **aa@fct.unl.pt 20-12-2018**↵
Fernanda AA nao pode dar boleia a si proprio.↵
aa@fct.unl.pt >

Caso exista a deslocação mas o utilizador já registou antes uma boleia ou uma deslocação nessa data, esse facto é dito ao utilizador da seguinte forma:

aa@fct.unl.pt > boleia **bb@fct.unl.pt 20-12-2018**↵
Fernanda AA ja registou uma boleia ou deslocacao nesta data.↵
aa@fct.unl.pt >

Caso não exista a deslocação, esse facto é dito ao utilizador da seguinte forma:

aa@fct.unl.pt > boleia **bb@fct.unl.pt 21-12-2017**↵
Deslocacao nao existe.↵
aa@fct.unl.pt >

Caso não exista o utilizador dado, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > boleia naoexiste@gmail.com 21-12-2018↵
Utilizador inexistente.↵
aa@fct.unl.pt >
```

Caso a data seja inválida, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > boleia mtmp@fct.unl.pt 0-12-2018↵
Data invalida.↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"(comando inválido); (2) não existir o utilizador dado; (3) data inválida; (4) não existir a deslocação dada; (5) ser uma deslocação do próprio; e (6) já ter registado uma boleia ou deslocação nesse dia.

3.10 Comando `retira`

Retira uma dada boleia. Este comando só tem sucesso se o sistema estiver em modo sessão. Pode-se eliminar a boleia registada para o dia indicado, se essa boleia tiver sido registada pelo utilizador da sessão.

O formato geral deste comando é o seguinte, em que diaDesloc-mesDesloc-anoDesloc é a data da deslocação e nomeUtilizador é o nome do utilizador corrente:

```
emailUtilizador > retira diaDesloc-mesDesloc-anoDesloc↵
nomeUtilizador boleia retirada.↵
emailUtilizador >
```

Vejamos um exemplo em que o utilizador com email **aa@fct.unl.pt** e nome **Fernanda AA** tem uma boleia registada no dia 23-12-2018. Caso este utilizador retire o registo de boleia do dia 23-12-2018, esse facto é indicado da seguinte forma:

```
aa@fct.unl.pt > retira 23-12-2018↵
Fernanda AA boleia retirada.↵
aa@fct.unl.pt >
```

Caso o utilizador tente retirar um registo de boleia do dia 22-12-2018, e esse utilizador não tenha uma boleia registada nesse dia, esse facto é indicado da seguinte forma:

```
aa@fct.unl.pt > retira 22-12-2018↵
Fernanda AA nesta data nao tem registo de boleia.↵
aa@fct.unl.pt >
```

Caso a data seja inválida, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > retira 0-12-2018↵
Data invalida.↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"(comando inválido); (2) data inválida; e (3) não existir uma boleia nesse dia para o utilizador corrente.

3.11 Comando `consulta`

Consulta uma dada deslocação. Consulta uma dada deslocação dado o email do utilizador que a registou, que aqui chamamos *emailUtilizadorR* e a data da deslocação (*diaDesloc-mesDesloc-anoDesloc*). Este comando só tem sucesso se a deslocação existir e o programa estiver em modo sessão. Quando este comando é executado, na consola é escrita toda a informação da deslocação, os emails das pessoas que já registaram boleia e o número de pessoas em fila de espera.

O formato geral do comando é o seguinte:

```
emailUtilizador > consulta emailUtilizadorR diaDesloc-mesDesloc-anoDesloc↵
emailUtulizadorR↵
localOrigem-LocalDestino↵
diaDesloc-mesDesloc-anoDesloc hora:minutos duracao↵
Lugares vagos: V↵
Boleias: listaEmailsBoleia↵
Em espera: E↵
emailUtilizador >
```

Note que *V* é o número de lugares vagos nesta deslocação. *listaEmailsBoleia* é a listagem dos emails dos utilizadores que registaram boleia nesta deslocação separados por ';'. Note que *E* é o número de utilizadores que estão em espera para poder ir à boleia, caso algum utilizador em *listaEmailsBoleia* desista da boleia.

Vejamos um exemplo desta situação. Assuma que existe uma deslocação do utilizador **bb@fct.unl.pt** no dia 20-12-2018. Se o utilizador com email **aa@fct.unl.pt** e nome **Fernanda AA**, executar o comando `consulta`, indicando **bb@fct.unl.pt** como email do utilizador que registou a deslocação, e 20-12-2018 como a data da deslocação, a informação dada pelo sistema é a seguinte:

```
aa@fct.unl.pt > consulta bb@fct.unl.pt 20-12-2018↵
bb@fct.unl.pt↵
Braga-Monte da Caparica↵
20-12-2018 10:00 90↵
Lugares vagos: 1↵
Boleias: cc@fct.unl.pt; hh@fct.unl.pt; gg@fct.unl.pt↵
Em espera: 0↵
aa@fct.unl.pt >
```

Caso não exista a deslocação, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > consulta bb@fct.unl.pt 21-12-2017↵
Deslocacao nao existe.↵
aa@fct.unl.pt >
```

Caso não exista o utilizador dado, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > consulta naoexiste@gmail.com 21-12-2018↵
Utilizador inexistente.↵
aa@fct.unl.pt >
```

Caso a data seja inválida, esse facto é dito ao utilizador da seguinte forma:

```
aa@fct.unl.pt > consulta mtmp@fct.unl.pt 0-12-2018↵
Data invalida.↵
aa@fct.unl.pt >
```

A ordem de verificação de erros neste comando é: (1) não estar em modo "sessão"(comando inválido); (2) não existir o utilizador dado; (3) data inválida; e (4) não existir a deslocação dada.

4 A Entrega

O trabalho tem 3 fases, sendo que todas têm uma tarefa de análise e desenho do problema comúm. As duas últimas fases envolvem a entrega de um programa, o qual deve ser entregue nos concursos do Mooshak para o efeito.

Cada grupo de 2 alunos (equipa de trabalho) deve registar-se nesses concursos. O nome do utilizador deve ser o seu **número de aluno_número de aluno** (o primeiro número a colocar é o menor) no grupo correspondente ao seu turno (por exemplo, os alunos nº 3435 e nº 3434 do turno p1 deve ter como nome utilizador no mooshak **3434_3435** no grupo **P1**). Só serão aceites como entregas para avaliação, os programas cujo utilizador no concurso do Mooshak siga estas regras.

Para os programas e relatório serem avaliados, os alunos devem ter cumprido o código de ética do DI, e ter, no final do prazo, uma entrega dum programa no concurso do Mooshak associado a essa entrega. Pode submeter o seu código mais que uma vez. Será a última submissão que será avaliada.

4.1 1ª Fase

A 1ª fase consta de um relatório (máximo 5 páginas, incluindo a capa) onde estará o diagrama de classes da solução proposta pelos alunos. Esta fase tem um peso de 10% da nota final na cadeira.

Esta fase deve ser entregue na página da cadeira no Moodle até às 17:00 horas do dia 15 de Outubro de 2019.

4.2 2ª Fase

A 2ª fase consta dum relatório (máximo 3 páginas) e dum programa, que poderá usar as interfaces e classes da API do java (10% da nota final na cadeira).
Esta fase deve ser entregue até às 17:00 horas do dia 12 de Novembro de 2019. O concurso Mooshak correspondente a esta fase é "MIEI_AED_TP1_2".
A avaliação desta fase tem 3 componentes, sendo a nota final calculada como a soma das notas dessas componentes:

- Avaliação da correcção dos resultados produzidos: até 10 valores

  - Submissão ao concurso obtendo a pontuação máxima (100 pontos), terá 10 valores nesta componente.
  - Uma submissão com erros em algumas das funcionalidades terá uma classificação correspondente às funcionalidades correctamente implementadas. Por exemplo, um aluno que entregue uma aplicação que receba 90 pontos terá 90% dos 10 valores.

- Avaliação da qualidade do código: 7 valores. Esta avaliação tem em conta a estruturação da solução em TADs do problema (por exemplo, deslocação), a utilização correcta de TADs e classes no domínio das estruturas de dados, e a qualidade do código.

- Qualidade do relatório: 3 valores. Descrição da solução implementada e justificação das escolhas.

4.3 3ª Fase

A 3ª fase consta dum relatório e dum programa, no qual as interfaces e classes são todas implementadas pelos alunos (20% da nota final na cadeira). No programa entregue é **proibido** usar classes ou interfaces do java.util (excepto o Scanner). Entregas que não cumprem esta regra não serão avaliadas.
Esta fase deve ser entregue até às 17:00 horas do dia 6 de Dezembro de 2019. O concurso Mooshak correspondente a esta fase é "MIEI_AED_TP1_3".
A avaliação desta fase tem 3 componentes, sendo a nota final calculada como a soma das notas dessas componentes:

- Avaliação da correcção dos resultados produzidos: até 5 valores

  - Submissão ao concurso obtendo a pontuação máxima (100 pontos), terá 5 valores nesta componente.
  - Uma submissão com erros em algumas das funcionalidades terá uma classificação correspondente às funcionalidades correctamente implementadas. Por exemplo, um aluno que entregue uma aplicação que receba 90 pontos terá 90% dos 5 valores.

- Avaliação da qualidade do código: 12 valores. Esta avaliação tem em conta a utilização e implementação correcta das estruturas de dados nos TADs usados, a eficiência do programa, e a qualidade do código.

- Qualidade do relatório: 3 valores. Descrição da solução implementada e estudo da análise da complexidade do programa.