PortugalTravel

Programação Orientada pelos Objectos

Enunciado do 3º Trabalho Prático, versão 2.3 – 2013-05-24

Contacto: carla.ferreira@fct.unl.pt

Alterações em relação à versão 1.2: Adições e alterações de texto estão assinaladas a **verde**. Texto removido está assinalado com uma **anotação a vermelho** no ponto da remoção.

Simplificações em relação à versão 2.0:

1. Nas versões anteriores foi definido que não podiam existir utilizadores registados com o *username* ou *email* duplicados. De forma a simplificar a implementação, a aplicação deve permitir utilizadores com emails iguais. Assim, os utilizadores são identificados univocamente apenas pelo seu *username*.
2. Podem assumir que o número de reviews é relativamente reduzido (na ordem das dezenas para uma dada actividade).

Notas importantes:

Prazo de entrega: até às 23h55 do dia 27 de Maio de 2013.

Realização em grupos: grupos de 2 alunos inscritos no mesmo turno prático (ou do mesmo docente se este autorizar).

Material a entregar: Nos comandos registar

- **Moodle**: ficheiro zip submetido via Moodle contendo o **projecto eclipse completo** da aplicação desenvolvida, com código fonte devidamente comentado, o respectivo **Javadoc** e o **diagrama de classes e interfaces**. O nome do ficheiro compactado tem as mesmas regras que nos trabalhos práticos anteriores.
- **Mooshak**: submissão e aceitação do código fonte pelo sistema de avaliação automática Mooshak.

Recomendações: A boa documentação do seu código fonte será valorizada, tal como a utilização do melhor estilo de programação possível, para além, claro, do correcto funcionamento do projecto. Não se esqueça de comentar as declarações das classes e das interfaces com uma explicação do significado das mesmas.

1. Desenvolvimento de aplicação PortugalTravel

1.1. Descrição do problema

O objectivo deste trabalho é o desenvolvimento de uma aplicação online – PortugalTravel – que ajude um viajante a planificar a sua estadia em Portugal. O PortugalTravel deve permitir:

- manter informação geral sobre hotéis e restaurantes (preços, classificação, etc.);
- consultar e registar *reviews* sobre hotéis e restaurantes;
- gerir os utilizadores da aplicação.

O principal objectivo da aplicação é manter informação imparcial sobre a qualidade das diferentes empresas de restauração e de hotelaria. Assim, quem utilizar o site pode seriar hotéis e restaurantes com base nas classificações atribuídas pelas *reviews*, de acordo com vários critérios. Por exemplo, o site irá permitir seriar os hotéis de 4 estrelas em Évora **ou os hóteis com um custo inferior a 100 euros em Lisboa**.

A aplicação suporta dois tipos de utilizadores, *Author* e *Advisor*. Os utilizadores *Author* podem inserir novos hotéis e restaurantes no sistema, sendo que também podem inserir *reviews*. Os utilizadores *Advisor* apenas podem submeter *reviews* sobre as suas experiências num hotel ou restaurante.

A aplicação pode ser usada por utilizadores não registados para consultar informação sobre as actividades (alojamento e restauração) disponíveis no site (estes utilizadores **não** precisam de fazer login no sistema). Um utilizador que pretenda submeter *reviews* terá que se registar como *Advisor*. Para tal terá que seleccionar um login e indicar o seu nome, a morada e o email. Um utilizador também se pode registar como *Author* e nesse caso está autorizado a inserir informação sobre hotéis e restaurantes. Embora fora do contexto deste trabalho, pode considerar que a informação inserida pelos utilizadores *Author* seria posteriormente validada e que estes utilizadores seriam recompensados, por exemplo, com vales de desconto.

Dados sobre as actividades

As actividades a considerar são apenas alojamento e restauração, mas futuramente estas actividades poderão ser estendidas.

Actividade. Cada actividade tem os seguintes dados associados: (1) um nome; (2) cidade; (3) endereço; (4) descrição. Considere que cada actividade é univocamente identificada pelo nome e pela cidade.

Hotel. Além da informação comum a todas as actividades, cada hotel tem os seguintes dados: (1) classificação (de 1 a 5 estrelas); (2) preço médio por noite.

Restaurante. Além da informação comum a todas as actividades, cada restaurante tem os seguintes dados: (1) tipo de cozinha (e.g., portuguesa, italiana, etc.); (2) classificação por preço (de $ a $$$$).

Dados sobre as *reviews*

***Review*.** Cada *review* tem os seguintes dados associados: (1) a cidade; (2) o nome da actividade; (3) o utilizador que fez a *review*; (3) classificação geral (*Excellent, **Good**, Average, Poor, Terrible*); (4) o comentário.

***Review* de um hotel.** Além da informação comum a todas as *reviews*, cada *review* de um hotel classifica a qualidade seguintes aspectos: (1) serviço; e (2) localização. Os níveis a considerar são os usados na classificação geral (e.g., *Excellent, **Good**, Average, Poor, Terrible*)

***Review* de um restaurante.** Além da informação comum a todas as *reviews*, cada *review* de um restaurante classifica a qualidade dos seguintes aspectos: (1) comida; e (2) ambiente. Os níveis a considerar são os usados na classificação geral (e.g., *Excellent, **Good**, Average, Poor, Terrible*)

2. Comandos a suportar

A aplicação deve interagir com o utilizador por meio de menus de comandos. Caso seja introduzido um comando desconhecido, a aplicação não deve fazer rigorosamente nada. Ou seja, deve ser possível introduzir uma sequência de comandos desconhecidos sem que a aplicação devolva qualquer mensagem.

A aplicação e o seu menu de comandos prevêem três níveis de utilizadores: (1) utilizadores não registados que consultam informação sobre hotéis e restaurantes em Portugal (estes utilizadores não necessitam de efectuar o login para consultar a informação); (2) utilizadores registados (nome: *Advisors*), que podem inserir *reviews* sobre hotéis e restaurantes; (3) utilizadores registados (nome: *Authors*), que para além de poderem inserir *reviews*, podem também inserir novos hotéis e restaurantes no sistema.

Para simplificar, considera-se que somente um utilizador pode ter uma sessão aberta no sistema em cada instante – *Author* ou *Advisor*. O sistema disponibiliza opções diferentes a utilizadores diferentes – há pois três "modos" de utilização diferentes, que oferecem opções de menu diferentes: um para utilizadores não registados, outro para utilizadores registados como *Advisor*, e por último um para utilizadores registados como *Author*. **Aqui, texto da versão anterior foi removido.**

Para se executar um comando acessível a um tipo de utilizador diferente do actual, é necessário que um faça *logout* e outro faça *login*.

2.1. Comandos comuns a todos utilizadores inclusive utilizadores não registados

Nos comandos de seriação, as actividades são seriadas com base na classificação média das *reviews* e em caso de empate é usada a ordem alfabética do nome da actividade. **Para calcular a média das reviews atribua um valor inteiro a cada nível da classificação: *Excellent* – 5, *Good* – 4, *Average* – 3, *Poor* – 2, *Terrible* – 1.**

1. **Seriar hotéis** (comando `HOTELS ALL`)**.** É fornecido o nome da cidade. A operação falha se: (1) não existirem hotéis nessa cidade.
2. **Seriar hotéis por preço** (comando `HOTELS PRICE`)**.** É fornecido o nome da cidade e o preço máximo por noite. A operação falha se: (1) não existir nenhum hotel que satisfaça o critério.
3. **Seriar hotéis por classificação** (comando `HOTELS STARS`)**.** São fornecidos o nome da cidade e a classificação pretendida (por exemplo, 3 estrelas). A operação falha se: (1) não existir nenhum hotel que satisfaça o critério.
4. **Seriar restaurantes** (comando `RESTAURANTS ALL`)**.** É fornecido o nome da cidade. A operação falha se: (1) não existirem restaurantes nessa cidade.
5. **Seriar restaurantes por tipo de cozinha** (comando `RESTAURANTS CUISINE`)**.** São fornecidos o nome da cidade e o tipo de cozinha (por exemplo, portuguesa). A operação falha se: (1) não existir nenhum restaurante com o tipo de cozinha indicado.
6. **Seriar restaurantes por custo** (comando `RESTAURANTS COST`)**.** São fornecidos o nome da cidade e o tipo de preço pretendido (de $ a $$$$). A operação falha se: (1) não existir nenhum restaurante que satisfaça o critério de preço.
7. **Consultar as *reviews* de um hotel** (comando `REVIEWS HOTEL`)**.** São fornecidos o nome da cidade, o nome do hotel. A operação falha se: (1) o hotel não existir. ~~**(2) não existem *reviews* ao hotel.**~~
8. ~~**Consultar as *reviews* de um restaurante** (comando `REVIEWS RESTAURANT`)**.** São fornecidos o nome da cidade, o nome do restaurante. A operação falha se: (1) o restaurante não existir. **(2) não existem *reviews* ao restaurante.**~~
9. **Aprovar uma *review*** (comando `LIKE REVIEW`)**.** São fornecidos o nome da cidade, o nome da actividade e nome do utilizador que fez a *review*. A operação falha se: (1) a actividade (hotel ou restaurante) não existir; (2) o utilizador não existir; (3) o utilizador não tiver feito uma *review* a essa actividade.
10. **Registar utilizador *Author*** (comando `REGISTER AUTHOR`)**.** São fornecidos o *username* pretendido, o nome, a morada e o email. Em caso de sucesso a operação devolve a password

gerada pelo sistema para o novo utilizador. A operação falha se: (1) existir um utilizador *logged in*; (2) já existir um utilizador (*Author* ou *Advisor*) com o *username* ~~ou o email~~ dado.

11. **Registar utilizador *Advisor*** (comando `REGISTER ADVISOR`)**.** São fornecidos o *username* pretendido, o nome, a morada e o email. Em caso de sucesso a operação devolve a password gerada pelo sistema para o novo utilizador. A operação falha se: (1) existir um utilizador *logged in*; (2) já existir um utilizador (*Author* ou *Advisor*) com o *username* ~~ou o email~~ dado.
12. **Sair da aplicação** (comando `EXIT`)**.** A operação tem sempre sucesso.

2.2. Comandos comuns aos utilizadores Advisor e Author

1. **Login** (comando `LOGIN`)**.** É fornecido ao sistema o *username* e a senha. A operação falha se: (1) o *username* não existir; (2) o utilizador já tiver uma sessão aberta; (3) já existir outro utilizador com uma sessão aberta; (4) a senha não corresponder ao *username* dado.
2. **Logout** (comando `LOGOUT`)**.** Esta operação não requer argumentos. É dada como terminada a sessão associada ao utilizador que estava *logged in*. A operação falha se: (1) não existir um utilizador *logged in*.
3. **Fazer uma *review* de um hotel** (comando `REVIEW HOTEL`)**.** São fornecidos a cidade, o nome do hotel, a classificação geral, um comentário, a classificação relativa à qualidade do serviço e classificação relativa à localização. **Caso já exista um review do utilizador *logged in* a esse hotel, a nova review deve substituir a review antiga.** A operação falha se: (1) não existir um utilizador *logged in*; (2) não existir o hotel.
4. **Fazer uma *review* de um restaurante** (comando `REVIEW RESTAURANT`)**.** São fornecidos a cidade, o nome do restaurante, a classificação geral, um comentário, a classificação relativa à qualidade da comida e classificação relativa ao ambiente. **Caso já exista um review do utilizador *logged in* a esse restaurante, a nova review deve substituir a review antiga.** A operação falha se: (1) não existir um utilizador *logged in*; (2) não existir o hotel.

2.3. Comandos apenas para os utilizadores Author

1. **Adicionar hotel** (comando `ADD HOTEL`)**.** São fornecidos a cidade, o nome do hotel, a descrição, o endereço, a classe (1 a 5 estrelas) e o preço médio por noite. A operação falha se: (1) não existir um utilizador de tipo *Author logged in* no sistema; (2) já existir uma actividade com o nome e a cidade dados.
2. **Adicionar restaurante** (comando `ADD RESTAURANT`)**.** São fornecidos a cidade, o nome do restaurante, a descrição, o endereço, o tipo de cozinha e tipo de preço (de $ a $$$$). A operação falha se: (1) não existir um utilizador de tipo *Author logged in* no sistema; (2) já existir uma actividade com o nome e a cidade dados.

2.4. Exemplo de interacção com a aplicação

A aplicação desenvolvida tem que garantir o modelo de interacção ilustrado no exemplo seguinte (o carácter ↵ representa uma mudança de linha):

```
REGISTER AUTHOR jb91↵
Jose Bitola↵
Rua de Alcantara, 35, Lisboa↵
jb91@gmail.com↵
> User was registered: author1.
↵
HOTELS ALL Lisboa↵
> No registered hotels in Lisboa.
↵
LOGIN jb91 author1↵
```

> Welcome jb91.↵
↵
ADD HOTEL Lisboa Ritz↵
The best hotel in Lisbon↵
Rua Rodrigo da Fonseca↵
5↵
600↵
> Hotel was registered.↵
↵
ADD HOTEL Lisboa Fenix↵
Central location↵
Rua Joaquim Antonio de Aguiar↵
3↵
80↵
> Hotel was registered.↵
↵
ADD RESTAURANT Lisboa Alfaia↵
The best restaurant in Lisbon↵
Rua Diario de Noticias↵
Portuguese↵
$$$↵
> Restaurant was registered.↵
↵
ADD RESTAURANT Lisboa GatoPardo↵
Good food at good prices ↵
Rua de Sao Vicente↵
Mediterranean↵
$↵
> Restaurant was registered.↵
↵
LOGOUT↵
> Goodbye jb91.↵
↵
REGISTER ADVISOR cpcurto↵
Camila Pe Curto↵
Avenida da Liberdade, 2, Cascais↵
cpc@gmail.com↵
> User was registered: advisor1.
↵
LOGIN cpcurto advisor1↵
> Welcome cpcurto.↵
↵
REVIEW RESTAURANT Lisboa Alfaia↵
Good↵
Good portuguese food↵
Excellent↵
Good↵
> Review was registered.↵
↵
REVIEW HOTEL Lisboa Ritz↵
Excellent↵
Luxury hotel↵
Excellent↵
Good↵
> Review was registered.↵
↵
REVIEW RESTAURANT Lisboa GatoPardo↵
Good↵
Cosy restaurant↵

```
Good↵
Excellent↵
> Review was registered.↵
↵
REVIEW HOTEL Lisboa Fenix↵
Good↵
Really good value↵
Good↵
Good↵
> Review was registered.↵
↵
LOGOUT↵
> Goodbye cpcurto.↵
↵
HOTELS ALL Lisboa↵
> All hotels in Lisboa:↵
> Ranked 1 (1 reviews): Ritz | 5 | 600↵
> Ranked 2 (1 reviews): Fenix | 3 | 80↵
↵
RESTAURANTS ALL Lisboa↵
> All restaurants in Lisboa:↵
> Ranked 1 (1 reviews): Alfaia | Portuguese | $$$↵
> Ranked 2 (1 reviews): GatoPardo | Mediterranean | $↵
↵
LIKE REVIEW Lisboa Fenix cpcurto↵
> Number of likes for review Fenix by cpcurto: 1.↵
↵
REVIEWS HOTEL Lisboa Fenix↵
> Reviews for hotel Fenix (by jb91):↵
> Rua Joaquim Antonio de Aguiar↵
> Central location↵
> Overall review:↵
> Excellent: 0↵
> Good: 1↵
> Average: 0↵
> Poor: 0↵
> Terrible: 0↵
> Service review: Good↵
> Location review: Good↵
> Reviewer cpcurto (1 likes): Really good value↵
↵
EXIT↵
> Exiting...↵
↵
```

3. Desenvolvimento

A sua aplicação deve tirar o melhor partido possível da matéria leccionada. Em particular, fazê-la **o mais extensível que possível**. Deve ser construída de modo a **minimizar as alterações necessárias**, caso se pretendam acrescentar, mais tarde, **novos tipos de actividades**.

Comece por desenvolver a interface principal da sua aplicação, identificando claramente quais os comandos que a sua aplicação deve suportar, bem como quais as entradas e saídas, não esquecendo as pré-condições. Depois, identifique as entidades de que vai necessitar para implementar este sistema. Identifique e especifique cuidadosamente as **interfaces** (Fase 1 e 2) e **classes** (Fase 2) de que necessita. Deve documentar o seu desenvolvimento quer através de um diagrama de classes e interfaces, quer através de documentação adequada no seu código.

Construa o esqueleto da classe Main que trata da entrada e saída dos dados e da interacção com a aplicação. É normal que no princípio o seu programa ainda não faça tudo. Lembre-se da **regra da versão estável**: não tente fazer tudo de uma só vez. Vá fazendo, testando, e avançando por incrementos, à medida que as funcionalidades vão sendo desenvolvidas. Se necessário, crie pequenos programas de teste auxiliares. Desenvolva também as operações de forma incremental.

4. Submissão ao Mooshak

Para submeter o seu programa ao Mooshak registe-se no concurso **POO1213-TP3** e siga as instruções apresentadas no site Moodle da disciplina.

Note que para cada comando será apresentada apenas uma única mensagem de saída. Ou seja, as condições que causam falhas num comando devem ser verificadas pela ordem descrita no enunciado e assim que uma dessas condições se verificar não é necessário verificar as condições seguintes.

4.1. Sintaxe dos comandos

4.1.1. Sintaxe dos comandos comuns a todos utilizadores (inclusive utilizadores não registados)

1. **Seriar hotéis:**

   ```
   HOTELS ALL <city>↵
   ```

   a. Mensagem de erro se não existirem hotéis dessa cidade registados no sistema:

   ```
   > No registered hotels in <city>.↵
   ↵
   ```

   b. Mensagem de sucesso:

   ```
   > All hotels in <city>:↵
   > Ranked 1 (<num1> reviews): <name1> | <class1> | <price1>↵
   > Ranked 2 (<num2> reviews): <name2> | <class2> | <price2>↵
   ...
   > Ranked i (<numi> reviews): <namei> | <classi> | <pricei>↵
   ↵
   ```

2. **Seriar hotéis por classificação** (`<stars>` é um valor inteiro de 1 a 5)**:**

   ```
   HOTELS STARS <city> <stars>↵
   ```

   a. Mensagem de erro se não existirem hotéis que satisfaçam o critério de classificação:

   ```
   > No registered hotels in <city> with <stars> stars.↵
   ↵
   ```

   b. Mensagem de sucesso:

   ```
   > All hotels in <city> with <stars> stars:↵
   > Ranked 1 (<num1> reviews): <name1> | <class1> | <price1>↵
   > Ranked 2 (<num2> reviews): <name2> | <class2> | <price2>↵
   ...
   > Ranked i (<numi> reviews): <namei> | <classi> | <pricei>↵
   ↵
   ```

3. **Seriar hotéis por preço:**

   ```
   HOTELS PRICE <city> <price>↵
   ```

   a. Mensagem de erro se não existirem hotéis que satisfaçam o critério de preço:

   ```
   > No registered hotels in <city> with price by night bellow <price>.↵
   ↵
   ```

   b. Mensagem de sucesso:

   ```
   > All hotels in <city> bellow <price>:↵
   > Ranked 1 (<num1> reviews): <name1> | <class1> | <price1>↵
   > Ranked 2 (<num2> reviews): <name2> | <class2> | <price2>↵
   ```

```
...
> Ranked i (<numi> reviews): <namei> | <classi> | <pricei>↵
↵
```

4. **Seriar restaurantes:**

   `RESTAURANTS ALL <city>↵`

   a. Mensagem de erro se não existirem restaurantes dessa cidade registados no sistema:

   ```
   > No registered restaurants in <city>.↵
   ↵
   ```

   b. Mensagem de sucesso:

   ```
   > All restaurants in <city>:↵
   > Ranked 1 (<num1> reviews): <name1> | <cuisine1> | <cost1>↵
   > Ranked 2 (<num2> reviews): <name2> | <cuisine2> | <cost2>↵
   ...
   > Ranked i (<numi> reviews): <namei> | <cuisinei> | <costi>↵
   ↵
   ```

5. **Seriar restaurantes por tipo de cozinha:**

   `RESTAURANTS CUISINE <city> <cuisine>↵`

   a. Mensagem de erro se não existirem restaurantes que satisfaçam o critério de tipo de cozinha:

   ```
   > No registered restaurants in <city> with cuisine <cuisine>.↵
   ↵
   ```

   b. Mensagem de sucesso:

   ```
   > All restaurants in <city> with <cuisine> cuisine:↵
   > Ranked 1 (<num1> reviews): <name1> | <cuisine1> | <cost1>↵
   > Ranked 2 (<num2> reviews): <name2> | <cuisine1> | <cost1>↵
   ...
   > Ranked i (<numi> reviews): <namei> | <cuisine1> | <cost1>↵
   ↵
   ```

6. **Seriar restaurantes por preço:**

   `RESTAURANTS COST <city> <cost>↵`

   a. Mensagem de erro se não existirem restaurantes que satisfaçam o critério de preço:

   ```
   > No registered restaurants in <city> within price range <cost>.↵
   ↵
   ```

   b. Mensagem de sucesso:

   ```
   > All restaurants in <city> within price range <cost>:↵
   > Ranked 1 (<num1> reviews): <name1> | <cuisine1> | <cost1>↵
   > Ranked 2 (<num2> reviews): <name2> | <cuisine2> | <cost2>↵
   ...
   > Ranked i (<numi> reviews): <namei> | <cuisinei> | <costi>↵
   ↵
   ```

7. **Consultar *reviews* de um hotel:**

   As *reviews* devem ser seriadas pelo número de *likes*, em caso de empate é usada a ordem alfabética do username do autor da *review*.

   `REVIEWS HOTEL <city> <hotel>↵`

   a. Mensagem de erro se o hotel não existir:

   ```
   > Hotel does not exist.↵
   ↵
   ```

   b. Mensagem de sucesso:

   ```
   > Reviews for hotel <hotel> (by <author>):↵
   > <address>↵
   > <description>↵
   ```

```
> Overall review:
> Excellent: <num>
> Good: <num>
> Average: <num>
> Poor: <num>
> Terrible: <num>
> Service review: <average grade>
> Location review: <average grade>
> Reviewer <username1> (<num1> likes): <comment1>↵
> Reviewer <username2> (<num2> likes): <comment2>↵
> ...↵
> Reviewer <usernamei> (<numi> likes): <commenti>↵
↵
```

No caso de não haver reviews o output segue a sintaxe apresentada na alínea b), ou seja:

```
> Reviews for hotel <hotel> (by <author>):↵
> <address>↵
> <description>↵
> Overall review:
> Excellent: 0
> Good: 0
> Average: 0
> Poor: 0
> Terrible: 0
> Service review: Terrible
> Location review: Terrible
↵
```

8. **Consultar *reviews* de um restaurante:**

   As *reviews* devem ser seriadas pelo número de *likes*, em caso de empate é usada a ordem alfabética do username do autor da *review*.

   ```
   REVIEWS RESTAURANT <city> <restaurant>↵
   ```

   a. Mensagem de erro se o hotel não existir:

   ```
   > Restaurant does not exist.↵
   ↵
   ```

   b. Mensagem de sucesso:

   ```
   > Reviews for restaurant <restaurant> (by <author>):↵
   > <address>↵
   > <description>↵
   > Overall review:
   > Excellent: <num>
   > Good: <num>
   > Average: <num>
   > Poor: <num>
   > Terrible: <num>
   > Food review: <average grade>
   > Atmosphere review: <average grade>
   > Reviewer <username1> (<num1> likes): <comment1>↵
   > Reviewer <username2> (<num2> likes): <comment2>↵
   > ...↵
   > Reviewer <usernamei> (<numi> likes): <commenti>↵
   ↵
   ```

   No caso de não haver reviews o output segue a sintaxe apresentada na alínea b), ou seja:

```
> Reviews for restaurant <restaurant> (by <author>):↵
> <address>↵
> <description>↵
> Overall review:
> Excellent: 0
> Good: 0
> Average: 0
> Poor: 0
> Terrible: 0
> Food review: Terrible
> Atmosphere review: Terrible
↵
```

9. Aprovar uma *review*

```
LIKE REVIEW <city> <name> <username>↵
```

a. Mensagem de erro se a actividade não existir:

```
> Activity does not exist.↵
↵
```

b. Mensagem de erro se o utilizador não existir:

```
> User does not exist.↵
↵
```

c. Mensagem de erro se o utilizador não fez uma *review* a essa actividade:

```
> User has not reviewed activity <name>.
↵
```

d. Mensagem de sucesso:

```
> Number of likes for review <name> by <username>: <num>.↵
↵
```

10. Registar utilizador Author

Considere que a password devolvida por este comando é a `String` "author1" para o primeiro utilizador *Author* registado, "author2" para o segundo utilizador *Author* registado e assim sucessivamente.

```
REGISTER AUTHOR <username>↵
<name>↵
<address>
<email>↵
```

a. Mensagem de erro se existir um utilizador *logged in*:

```
> There is a user logged in.↵
↵
```

b. Mensagem de erro se já existir um utilizador com o login ~~ou email~~ dado:

```
> Username already registered.↵
↵
```

c. Mensagem de sucesso:

```
> User was registered: <password>.↵
↵
```

11. Registar utilizador Advisor

Tal como no comando anterior, considere que a password devolvida por este comando é a `String` "advisor1" para o primeiro utilizador *Advisor* registado, "advisor2" para o segundo utilizador *Advisor* registado e assim sucessivamente.

```
REGISTER ADVISOR <username>↵
<name>↵
<address>
```

```
<email>↵
```

a. Mensagem de erro se existir um utilizador *logged in*:

```
> There is a user logged in.↵
↵
```

b. Mensagem de erro se já existir um utilizador com o login ~~ou email~~ dado:

```
> Username already registered.↵
↵
```

c. Mensagem de sucesso:

```
> User was registered: <password>.↵
↵
```

12. Sair da aplicação

```
EXIT↵
```

a. Mensagem de sucesso:

```
> Exiting...↵
↵
```

4.1.2. Comandos comuns aos utilizadores Advisor e Author

1. Login:

```
LOGIN <username> <password>↵
```

a. Mensagem de erro se não existir ao utilizador dado:

```
> User does not exist.
↵
```

b. Mensagem de erro se o utilizador já tiver uma sessão aberta:

```
> User is already logged in.↵
↵
```

c. Mensagem de erro se já existir um utilizador com sessão aberta:

```
> Another user is logged in.↵
↵
```

d. Mensagem de erro se a senha não corresponder ao utilizador dado:

```
> Wrong password.↵
↵
```

e. Mensagem de sucesso:

```
> Welcome <username>.
↵
```

2. Logout:

```
LOGOUT↵
```

a. Mensagem de erro se não existir ao utilizador com a sessão aberta:

```
> No user is logged in.
↵
```

b. Mensagem de sucesso:

```
> Goodbye <username>.
↵
```

3. Adicionar uma *review* de um hotel

```
REVIEW HOTEL <city> <name>↵
<grade>↵
<comment>↵
<service>↵
<local>↵
```

a. Mensagem de erro se não existir um utilizador *logged in*:

```
> Login to add a review.↵
↵
```

b. Mensagem de erro se o hotel não existir:

```
> Hotel does not exist.↵
↵
```

c. Mensagem de sucesso:

```
> Review was registered.↵
↵
```

4. **Adicionar uma *review* de um restaurante**

```
REVIEW RESTAURANT <city> <name>↵
<grade>↵
<comment>↵
<food>↵
<ambience>↵
```

a. Mensagem de erro se não existir um utilizador *logged in*:

```
> Login to add a review.↵
↵
```

b. Mensagem de erro se o **restaurante** não existir:

```
> Restaurant does not exist.↵
↵
```

c. Mensagem de sucesso:

```
> Review was registered.↵
↵
```

4.1.3. Comandos apenas para os utilizadores Author

1. **Adicionar hotel:**

```
ADD HOTEL <city> <name>↵
<description>↵
<address>↵
<class>↵
<price>↵
```

a. Mensagem de erro se não existir um utilizador de tipo *Author logged in*:

```
> Only Author users may add hotels.↵
↵
```

b. Mensagem de erro se já existir uma actividade com o nome e cidade dados:

```
> Activity already exists.
↵
```

c. Mensagem de sucesso:

```
> Hotel was registered.
↵
```

2. **Adicionar restaurante:**

```
ADD RESTAURANT <city> <name>↵
<description>↵
<address>↵
<food>↵
<cost>↵
```

a. Mensagem de erro se não existir um utilizador de tipo *Author logged in*:

```
> Only Author users may add restaurants.↵
↵
```

b. Mensagem de erro se já existir uma actividade com o nome e cidade dados:

```
> Activity already exists.
↵
```

c. Mensagem de sucesso:

```
> Restaurant was registered.
↵
```

4.2. Testes

Os testes do Mooshak verificam de forma incremental a implementação dos vários comandos.

4.2.1. Testes onde não são testadas as condições onde os comandos podem falhar

- **Ficheiro de teste**: `01_in_base_register.txt` (10 pontos)
  Comandos testados: `REGISTER AUTHOR, REGISTER ADVISOR, EXIT`
- **Ficheiros de teste**: `02_in_base_login.txt` (10 pontos)
  Comandos testados: todos os comandos do teste 1 e os comandos `LOGIN, LOGOUT`
- **Ficheiros de teste**: `03_in_base_add.txt` (10 pontos)
  Comandos testados: todos os comandos do teste 2 e os comandos `ADD HOTEL, ADD RESTAURANT, RESTAURANT ALL, HOTEL ALL`
- **Ficheiro de teste**: `04_in_base_list.txt` (10 pontos)
  Comandos testados: todos os comandos do teste 3 e os comandos `HOTELS PRICE, HOTELS STARS, RESTAURANTS CUISINE, RESTAURANTS COST`
- **Ficheiros de teste**: `05_in_review.txt` (10 pontos)
  Comandos testados: todos os comandos do teste 3 e os comandos `REVIEW HOTEL, REVIEW RESTAURANT`
- **Ficheiros de teste**: `06_in_reviews.txt` (10 pontos)
  Comandos testados: todos os comandos do teste 5 e os comandos `REVIEWS HOTEL, REVIEWS RESTAURANT`
- **Ficheiros de teste**: `07_in_like.txt` (5 pontos)
  Comandos testados: todos os comandos do teste 6 e o comando `LIKE REVIEW`

4.2.1. Testes onde são testadas as condições onde os comandos podem falhar

- **Ficheiro de teste**: `08_in_pre_register.txt` (7 pontos)
  Comandos testados: `REGISTER AUTHOR, REGISTER ADVISOR`
- **Ficheiros de teste**: `09_in_pre_login.txt` (10 pontos)
  Comandos testados: `LOGIN, LOGOUT`
- **Ficheiros de teste**: `10_in_pre_add.txt` (6 pontos)
  Comandos testados: `ADD HOTEL, ADD RESTAURANT`
- **Ficheiros de teste**: `11_in_pre_list.txt` (3 pontos)
  Comandos testados: `HOTELS ALL, HOTELS PRICE, HOTELS STARS, RESTAURANTS ALL, RESTAURANTS CUISINE, RESTAURANTS COST`
- **Ficheiros de teste**: `12_in_pre_review.txt` (3 pontos)
  Comandos testados: `REVIEW HOTEL, REVIEW RESTAURANT`
- **Ficheiros de teste**: `13_in_pre_reviews.txt` (3 pontos)
  Comandos testados: `REVIEWS HOTEL, REVIEWS RESTAURANT`
- **Ficheiros de teste**: `14_in_pre_like.txt` (3 pontos)
  Comandos testados: `LIKE REVIEW`