Exercício de Avaliação II

Programação Orientada a Objetos

Data de entrega: 8 Abril 2019

Sistemas Informáticos – MIEBIOM/MIEF
Departamento de Engenharia Informática

• U C •

Regras de submissão do trabalho:

- Este trabalho deve ser realizado por um **grupo de 2 alunos** e conta para a avaliação;
- Deve submeter um ficheiro **.zip** com o **código** (ficheiros de extensão .java) desenvolvido para **este exercício** em https://inforestudante.uc.pt. **Não use outro formato de compressão.**
- O nome do ficheiro **.zip** deve seguir o formato: **nome1-nome2.zip;**
- O aluno que submeter o trabalho tem **de associar o seu colega** de trabalho durante o processo de submissão;
- Após submeter o trabalho em inforestudante, tem de registar o **esforço despendido na realização do trabalho por cada aluno em horas gastas em aula e em horas extra-aula.** Deve contar o tempo desde o início do semestre, caso esteja a submeter o exercício I; caso contrário conta apenas o tempo desde a última submissão de exercício de avaliação**.** Para o efeito preencha o formulário disponível em:
  https://goo.gl/forms/tEjK3vUSt0iU3AC33

Descrição

A empresa *CoimbraAirLines* pretende um sistema que lhe permita gerir o seu negócio, que consiste em gerir uma frota de aviões, um conjunto de viagens, e vendas de viagens a clientes. Assim, é necessário construir um sistema que permita que clientes possam, por exemplo, pesquisar e comprar viagens, e que permita que o administrador possa desempenhar tarefas típicas de gestão da frota e viagens (e.g., adicionar um novo avião ou adicionar uma nova viagem). Neste exercício irá usar os conhecimentos que adquiriu sobre Programação Orientada a Objetos (OOP), para criar um programa que permita simular o funcionamento deste tipo de sistema. Note que o problema pressupõe algumas simplificações da realidade para que seja de desenvolvimento mais simples (e.g., a aplicação não permite múltiplos utilizadores em simultâneo, não é necessário usar datas em viagens ou segmentos de viagem). Segue-se uma breve descrição das funcionalidades a implementar.

Cliente:

– Registar utilizador. Requer um *username* e *password* e um saldo fictício que o cliente usará em compras de viagens.
– *Login* / *Logout*.
– Listar todas as viagens.
– Ver os detalhes de uma viagem.
– Comprar uma viagem. Se o saldo não for suficiente a compra deve ser impedida.
– Listar toda a informação das suas viagens anteriores.
– Pesquisar viagens especificando: i) cidade de origem; ii) cidade de destino; iii) ambas.

Administrador:

– Login usando *username* e *password* (estes dados estão definidos no código) / Logout.
– Fazer a gestão de todo o sistema, permitindo operações sobre 3 tipos de itens:

Avião:
– Adicionar avião.
– Remover avião, caso não esteja a ser usado noutras partes do sistema, por ex. associado a viagens (ver item *viagem*).
– Atualizar informação de avião.
– Listar aviões.

Segmento de viagem:
– Adicionar um novo segmento de viagem composto por cidade de origem, cidade de destino e um dos aviões da frota. Neste sistema, um avião pode estar associado a, no máximo, 3 segmentos.
– Remover um segmento. Pode ser removido desde que não esteja atualmente a ser usado no sistema, por ex. na composição de uma viagem (ver item *viagem*), ou numa compra efetuada pelo cliente.
– Alterar segmento. Neste caso, a alteração possível é a troca de avião. O sistema deve manter informação histórica dos aviões que já estiveram associados a um certo segmento.
– Visualizar todos os segmentos.

Viagem (composta por 1 ou mais segmentos):
– Adicionar uma viagem. Para o efeito especifica-se a cidade de origem e destino e o preço de venda a cliente. O sistema deverá então listar todos os conjuntos de segmentos possíveis para as cidades envolvidas, permitindo:

i) Escolha manual do conjunto de segmentos que devem compor a viagem (a partir de uma listagem de segmentos).
ii) Escolha automática de um dos percursos mais curtos (em termos de número de segmentos).

– Remover viagem (sem remoção de segmentos associados). Pode ocorrer se a viagem não estiver a ser usada numa outra parte do sistema (e.g., não foi comprada por nenhum cliente).
– Remover viagem e respetivos segmentos. Semelhante à opção anterior, mas considera também que, neste caso, os segmentos são apenas removidos se não estiverem a ser usados no sistema (e.g., para compor outras viagens).
– Alterar viagem, deve permitir a escolha manual mencionada na alínea *i)* referente à adição de viagem. Só deve ser possível efetuar a alteração, se a viagem não estiver a ser usada noutra parte do sistema (i.e., não foi ainda comprada por clientes).
– Visualizar todas as viagens.
– Ver estatísticas: valor total das vendas, valor total das vendas para uma certa cidade de origem ou para uma certa cidade de destino, valor médio gasto em compras por utilizador (considerando as compras de todos os utilizadores do sistema), e um outro valor que considere interessante ver num sistema deste tipo.

Notas adicionais

- Extra: Para tornar o trabalho mais realista, pode usar o seguinte código para gravar a sua base de dados para disco (e carregar objetos para memória)[1]. Isto é, sempre que for feita alguma alteração aos dados, pode, para simplificar, gravar toda a base de dados em disco de imediato. A(s) respetivas classes envolvidas têm de ser marcadas com *implements Serializable*, para que possam ser escritas em disco.

```
MySimpleDatabase db = new MySimpleDatabase();
FileOutputStream fos = new FileOutputStream("db.tmp");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(db);
oos.close();
```

  Pode ignorar eventuais erros de leitura/escrita adicionando uma claúsula *throws* ao método que lê/escreve os dados em disco (verifique como fazê-lo com o professor, ou siga a sugestão respetiva do Eclipse).
- Antes de iniciar a implementação deve primeiro modelar o problema utilizando os conceitos OOP, identificando preliminarmente as classes que necessita e suas relações.
- Neste trabalho deve usar uma **lista**, sempre que necessitar de representar algum conjunto de objetos.
- É importante que a configuração da aplicação, caso exista, seja de fácil alteração não estando desnecessariamente dispersa em vários pontos do código.
- Não é necessário fazer validação de tipos de dados, ou formato. Mas deve garantir que a informação inserida/manipulada é válida quando ao seu conteúdo e o programa **cumpre as regras de funcionamento típico de um sistema deste tipo** (por exemplo, impedir o registo de um *username* vazio).

[1] Documentação disponível em
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/ObjectOutputStream.html e
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/ObjectInputStream.html