TÉCNICO LISBOA

Projecto de Programação com Objectos
13 de Novembro de 2017

Esclarecimento de dúvidas:

- Consultar sempre o corpo docente atempadamente: presencialmente ou através do endereço **po-tagus@disciplinas.ist.utl.pt**.
- Não utilizar fontes de informação não oficialmente associadas ao corpo docente (podem colocar em causa a aprovação à disciplina).
- Não são aceites justificações para violações destes conselhos: quaisquer consequências nefastas são da responsabilidade do aluno.

Requisitos para desenvolvimento, material de apoio e actualizações do enunciado (ver informação completa na secção **[Projecto]** no Fénix):

- O material de apoio é de uso obrigatório e não pode ser alterado.
- Verificar atempadamente (mínimo de 48 horas antes do final de cada prazo) os requisitos exigidos pelo processo de avaliação.

Processo de avaliação (ver informação completa nas secções **[Projecto]** e **[Método de Avaliação]** no Fénix):

- Datas: **2017/10/18 15:00** (UML); **2017/11/20 23:59** (intercalar); **2017/12/11 23:59** (final); **2017/12/18** (teste prático).
- Os diagramas de classe UML são entregues exclusivamente em papel (impressos ou manuscritos) na portaria do Tagus. Diagramas ilegíveis ou sem identificação completa serão sumariamente ignorados. É obrigatório a identificação do grupo e turno de laboratório (dia, hora e sala).
- **Apenas se consideram para avaliação os projetos submetidos no Fénix.** As classes criadas de acordo com as especificações fornecidas devem ser empacotadas num arquivo de nome `proj.jar` (que deverá apenas conter os ficheiros `.java` do código realizado guardados nos *packages* correctos). O ficheiro `proj.jar` deve ser entregue para avaliação através da ligação presente no Fénix. É possível realizar múltiplas entregas do projecto até à data limite, mas apenas será avaliada a última entrega.
- Não serão consideradas quaisquer alterações aos ficheiros de apoio disponibilizados: eventuais entregas dessas alterações serão automaticamente substituídas durante a avaliação da funcionalidade do código entregue.
- Trabalhos não presentes no Fénix no final do prazo têm classificação 0 (zero) (não são aceites outras formas de entrega).
- A avaliação do projecto pressupõe o compromisso de honra de que o trabalho foi realizado pelos alunos correspondentes ao grupo de avaliação. **Fraudes na execução do projecto terão como resultado a exclusão dos alunos implicados do processo de avaliação**.

O objectivo do projecto é criar uma aplicação que gere os itinerários de passageiros que usam comboios. Os itinerários são constituídos por um ou mais segmentos, realizados por comboios que cumprem serviços pré-definidos.

A secção 1 apresenta as entidades do domínio da aplicação a desenvolver. As funcionalidades da aplicação a desenvolver são descritas nas secções 2, 3 e 4. A secção 5 descreve as qualidades que a solução deve oferecer. Neste texto, o tipo **negrito** indica um literal (i.e., é exactamente como apresentado); o símbolo ␣ indica um espaço; e o tipo *itálico* indica uma parte variável (i.e., uma descrição).

1 Entidades do Domínio

Nesta secção descrevem-se as várias entidades que vão ser manipuladas no contexto da aplicação a desenvolver. Existem três conceitos básicos na aplicação a desenvolver: serviços, passageiros e itinerários

1.1 Estrutura de um Serviço

Um serviço é um percurso realizado por um comboio ao longo de várias estações. A cada estação de um serviço está associado o momento de partida do comboio. O serviço tem ainda um custo total de utilização. Assume-se que todos os serviços se realizam em todos os dias. Além das informações anteriores, o serviço tem ainda um identificador numérico único (definido na criação do serviço).

Exemplo do serviço `850 Valença >> Porto - Campanhã`:

```
Serviço␣#850␣@␣10.95
07:36␣Valença
07:47␣Vila␣Nova␣de␣Cerveira
07:56␣Caminha
08:03␣Âncora␣Praia
08:16␣Viana␣do␣Castelo
08:20␣Areia-Darque
08:30␣Barroselas
08:37␣Tamel
08:53␣Barcelos
```

```
09:05␣Nine
09:13␣Famalicão
09:21␣Trofa
09:35␣Ermesinde
09:45␣Porto␣-␣Campanhã
```

Cada segmento de um serviço tem um custo base, que depende da distância entre estações. A distância entre estações corresponde ao tempo gasto na viagem entre essas estações. Assim, considerando que a totalidade do custo do serviço 850 (no exemplo acima) é de €10.95 (para 129 minutos), o custo do segmento `Viana do Castelo >> Nine` (para 49 minutos) é de $10.95 * 49/129$.

Um cliente pode ainda ter direito a um desconto no valor a pagar. O valor do desconto depende da categoria do cliente (ver secção §1.2). A adição de novas categorias deve ter impacto reduzido no código da aplicação.

Considera-se que apenas existem as estações que aparecem em, pelo menos, um serviço.

1.2 Estrutura de um Passageiro

Um passageiro tem um identificador único (número inteiro atribuído automaticamente, de forma sequencial) e um nome (cadeia de caracteres). O identificador único é atribuído automaticamente no momento do registo do passageiro, sendo atribuído o identificador 0 (zero) ao primeiro passageiro.

Os passageiros são categorizados como normais, frequentes ou especiais. Um passageiro é considerado frequente enquanto o valor dos últimos 10 itinerários for superior a €250. Volta a ser normal se esta condição deixar de se verificar. Um cliente é considerado especial enquanto tiver gasto mais de €2500 nos últimos 10 itinerários. Deixa de ser especial se esta condição deixar de se verificar. Se se verificarem simultaneamente as condições para ser um cliente frequente e especial, o cliente é considerado especial. Aplicam-se as seguintes condições para atribuição de descontos em itinerários: passageiros normais, 0%; passageiros frequentes, 15%; passageiros especiais, 50%.

1.3 Estrutura de um Itinerário

Um itinerário corresponde a uma viagem realizada num dado dia, com base num conjunto de segmentos lógicos, correspondentes a partes de serviços, i.e., partidas (ou chegadas) agendadas para cada estação. É possível saber, através do serviço associado, qual o percurso a realizar nesse segmento. Um itinerário não passa duas vezes na mesma estação nem pode ter mais de um segmento de um serviço já utilizado.

O custo base do itinerário é a soma dos custos parciais (sem arredondamentos) de cada serviço que o constitui, correspondentes aos segmentos realizados nesses serviços. Dependendo da categoria do cliente, podem ser realizados descontos. Os itinerários são identificados por um número de ordem no contexto de cada passageiro, começando esta numeração no valor 1. Não é possível remover itinerários já registados.

Exemplo de um 3º itinerário `Évora >> Loulé` de um dado passageiro:

```
Itinerário␣3␣para␣2017-10-18␣-␣Évora␣>>␣Loulé␣@␣25.84
Serviço␣#690␣@␣6.94
07:06␣Évora
07:17␣Casa␣Branca
07:31␣Vendas␣Novas
07:53␣Pinhal␣Novo
Serviço␣#180␣@␣28.28
09:06␣Pinhal␣Novo
10:54␣Tunes
11:01␣Albufeira␣-␣Ferreiras
11:13␣Loulé
```

Os passageiros são creditados com o valor monetário de cada itinerário comprado e com o tempo decorrido nesses itinerários (medido entre os pontos extremos).

2 Funcionalidade da Aplicação

A aplicação mantem informação sobre os serviços existentes, regista e gere passageiros e permite criar novos itinerários para eles. Possui ainda a capacidade de preservar o seu estado (não é possível manter várias versões do estado da aplicação em simultâneo).

Note-se que não é necessário concretizar de raiz a aplicação. Já é fornecido algum código, nomeadamente, o esqueleto das classes necessárias para concretizar os menus e respectivos comandos a utilizar na aplicação a desenvolver, a classe `mmt.app.App`, que representa o ponto de entrada da aplicação a desenvolver, as classes do core da aplicação `mmt.core.TicketOffice` e `mmt.core.TrainCompany` e um conjunto de excepções a utilizar durante o desenvolvimento da aplicação. As classes

`mmt.core.TicketOffice` e `mmt.core.TrainCompany` estão parcialmente concretizadas e devem ser adaptadas onde cada grupo achar necessário. A classe `mmt.code.TicketOffice` deverá representar a interface geral do domínio da aplicação. Por esta razão, os comandos a desenvolver na camada de interface com o utilizador devem utilizar exclusivamente esta classe para aceder às funcionalidades suportadas pelas classes do domínio da aplicação (a serem desenvolvidas por cada grupo na package `mmt.core`). Desta forma será possível concretizar toda a interacção com o utilizador completamente independente das entidades do domínio.

2.1 Consultas de Serviços

A base de dados de serviços da aplicação é carregada no início da aplicação. Não é possível adicionar ou remover serviços durante a execução da aplicação. É possível fazer várias consultas sobre os serviços existentes (ver secção §3.2).

2.2 Manipulação de Passageiros

A aplicação permite realizar várias operações sobre passageiros. É possível obter a lista completa de passageiros conhecidos, assim como informação detalhada de um dado passageiro. É ainda possível registar novos passageiros e alterar o nome dos passageiros.

2.3 Manipulação de Itinerários

A aplicação permite obter informações sobre todos os itinerários já comprados ou, em pormenor, sobre os itinerários de passageiros específicos. Permite ainda criar novos itinerários.

2.4 Serialização

É possível reiniciar (ou seja, mantendo a informação sobre serviços, eliminar a informação sobre passageiros e itinerários), guardar e recuperar o estado actual da aplicação, preservando toda a informação sobre serviços, passageiros e itinerários.

3 Interação com o Utilizador

Descreve-se nesta secção a **funcionalidade máxima** da interface com o utilizador. Em geral, os comandos pedem toda a informação antes de proceder à sua validação (excepto onde indicado). Todos os menus têm automaticamente a opção **Sair** (fecha o menu).

As operações de pedido e apresentação de informação ao utilizador **devem** realizar-se através dos objectos *form* e *display*, respectivamente, presentes em cada comando. As mensagens são produzidas pelos métodos das bibliotecas de suporte (**po-uuilib** e **mmt-app**). **Não** podem ser definidas novas mensagens. Potenciais omissões devem ser esclarecidas com o corpo docente antes de qualquer concretização.

Não deve haver código de interacção com o utilizador no núcleo da aplicação (*mmt.core*). Desta forma, será possível reutilizar o código do núcleo da aplicação (onde é concretizado o domínio da aplicação) com outras concretizações da interface com o utilizador.

As excepções usadas no código de interacção com o utilizador para descrever situações de erro, excepto se indicado, são subclasses de `pt.tecnico.po.ui.DialogException` e devem ser lançadas pelos comandos (sendo depois tratadas automaticamente pela classe já existente `pt.tecnico.po.ui.Menu`). Outras excepções não devem substituir as fornecidas nos casos descritos. Note-se que o programa principal e os comandos e menus, a seguir descritos, já estão parcialmente concretizados nas classes dos packages `mmt.app`, `mmt.app.main`, `mmt.app.service`, `mmt.app.passenger` e `mmt.app.itinerary`. Estas classes são de uso obrigatório e estão disponíveis na secção *Projecto* da página da cadeira.

3.1 Menu Principal

As acções deste menu permitem gerir a salvaguarda do estado da aplicação e abrir submenus. A lista completa é a seguinte: **Reiniciar**, **Abrir**, **Guardar** , **Consulta de Serviços**, **Manipulação de Passageiros** e **Manipulação de Itinerários**. Inicialmente, a aplicação apenas tem informação sobre os serviços que foram carregados no arranque. As etiquetas das opções deste menu estão definidas na classe `mmt.app.main.Label`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `mmt.app.main.Message`.

Os comandos que vão concretizar as funcionalidades deste menu já estão parcialmente concretizados nas várias classes da package `mmt.app.main`, respectivamente: `DoReset`, `DoOpen`, `DoSave`, `DoOpenServicesMenu`, `DoOpenPassengersMenu` e `DoOpenItinerariesMenu`.

3.1.1 Salvaguarda do estado actual

O conteúdo da aplicação (inclui todos os serviços, passageiros e itinerários actualmente em memória) pode ser guardado para posterior recuperação (via serialização Java: `java.io.Serializable`). Na leitura e escrita do estado da aplicação, devem ser tratadas as excepções associadas. A funcionalidade é a seguinte:

Reiniciar – Reinicia a aplicação: destrói toda a informação sobre passageiros e itinerários; mantém toda a informação sobre serviços.

Abrir – Carrega os dados de uma sessão anterior a partir de um ficheiro (ficando associado à aplicação). A informação a carregar compreende serviços, passageiros e itinerários. A mensagem (cadeia de caracteres) a utilizar para pedir o nome do ficheiro a abrir é a devolvida pelo método `openFile()`. Caso o ficheiro não exista é apresentada a mensagem `fileNotFound()`.

Guardar – Guarda o estado actual da aplicação (serviços, passageiros e itinerários) no ficheiro associado. Se não existir associação, pede-se o nome do ficheiro a utilizar, ficando a ele associado. Esta interacção realiza-se através do método `newSaveAs()`.

As opções `Reiniciar` e `Abrir` substituem a informação na aplicação nas condições indicadas acima. A opção `Sair` **nunca** guarda o estado da aplicação, mesmo que existam alterações.

3.1.2 Gestão e consulta de dados da aplicação

As opções relacionadas com a gestão e consulta de dados da aplicação são as seguintes:

Consulta de Serviços – Abre o menu de consulta de serviços (se existirem serviços registados).

Gestão de Passageiros – Abre o menu de gestão de passageiros (se existirem passageiros registados).

Gestão de Itinerários – Abre o menu de gestão de itinerários (se existirem itinerários registados).

3.2 Menu de Consulta de Serviços

Este menu permite efectuar consultas sobre a base de dados de serviços. A lista completa é a seguinte: **Mostrar todos os serviços**, **Mostrar serviço com um dado número**, **Mostrar serviços com origem numa estação dada**, **Mostrar serviços com término numa estação dada**.

As etiquetas das opções deste menu estão definidas na classe `mmt.app.service.Label`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `mmt.app.service.Message`.

Sempre que for pedido o identificador de um serviço (utilizando a mensagem devolvida por `requestServiceId()`) e o serviço não existir, deve ser lançada a excepção `mmt.app.NoSuchServiceException`. Sempre que for pedido o nome de uma estação (mensagem devolvida por `requestStationName()`) e a estação não existir, deve ser lançada a excepção `mmt.app.NoSuchStationException`.

Os comandos deste menu já estão parcialmente concretizados nas classes da package `mmt.app.service`: `DoShowAllServices`, `DoShowServiceByNumber`, `DoShowServicesDepartingFromStation` e `DoShowServicesArrivingAtStation`.

3.2.1 Mostrar todos os serviços

Apresenta informações sobre todos os serviços conhecidos. A lista apresentada é ordenada pelo identificador do serviço e o formato é o descrito na secção §3.2.2.

3.2.2 Mostrar serviço com um dado número

Apresenta informações sobre um serviço. Depois de pedir o identificador do serviço (mensagem devolvida por `requestServiceId()`), apresenta a informação sobre o serviço, tal como indicado no formato seguinte, para um serviço com N paragens (HH;MM representa o tempo em horas e minutos). O preço deve ser apresentado com duas casas decimais:

```
Serviço␣#número-do-serviço␣@␣preço
HH:MM␣nome-de-estação-1
...
HH:MM␣nome-de-estação-N
```

Por exemplo, o serviço `180: Porto >> Faro` deve ser apresentado da seguinte forma:

```
Serviço #180 @ 51.50
05:47 Porto - Campanhã
05:52 Vila Nova de Gaia-Devesas
06:21 Aveiro
06:46 Coimbra-B
08:23 Lisboa - Oriente
08:31 Lisboa - Entrecampos
09:06 Pinhal Novo
10:54 Tunes
11:01 Albufeira - Ferreiras
11:13 Loulé
11:23 Faro
```

3.2.3 Mostrar serviços com origem numa estação dada

Apresenta os serviços com origem na estação indicada como resposta a `requestStationName()`, ordenado por hora de partida do serviço. O formato de apresentação dos serviços encontrados é o descrito na secção §3.2.2.

3.2.4 Mostrar serviços com término numa estação dada

Apresenta os serviços com término na estação indicada como resposta a `requestStationName()`, ordenado por hora de chegada do serviço. O formato de apresentação dos serviços encontrados é o descrito na secção §3.2.2.

3.3 Menu de Gestão de Passageiros

Este menu permite efectuar operações sobre um passageiro. A lista completa é a seguinte: **Mostrar passageiros**, **Mostrar passageiro**, **Registar passageiro**, **Alterar nome de passageiro**, **Suspender passageiro** e **Activar passageiro**. As etiquetas das opções deste menu estão definidas na classe `mmt.app.passenger.Label`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `mmt.app.passenger.Message`. Os comandos que concretizam estas operações já estão concretizados nas classes da package `mmt.app.passenger`: `DoShowPassengerById`, `DoRegisterPassenger` e `DoChangePassengerName`.

Considere ainda que sempre que for pedido o identificador de um passageiro deve ser utilizada a mensagem devolvida por `requestPassengerId()`. Se o passageiro não existir deve ser lançada a excepção `mmt.app.NoSuchPassengerException`.

3.3.1 Mostrar passageiros

Apresenta informações sobre todos os passageiros registados na aplicação. A lista é ordenada pelo identificador do passageiro e o formato de apresentação para cada passageiro é o descrito na secção §3.3.2.

3.3.2 Mostrar passageiro

Apresenta informações sobre um passageiro. Depois de pedir o identificador do passageiro (`requestPassengerId()`), apresenta o nome, o tipo de passageiro (NORMAL, FREQUENTE, ESPECIAL), o número de itinerários adquiridos, o valor pago por esses itinerários (o valor pago deve ser apresentado com duas casas decimais) e o o tempo acumulado (formato HH:MM) correspondente aos itinerários adquiridos. O formato a utilizar para apresentar um passageiro é o seguinte:

```
id|nome|categoria|número-de-itinerários|valor-pago|tempo-acumulado
```

Por exemplo, o passageiro *John Doe* seria apresentado da seguinte forma:

```
1234|John Doe|NORMAL|123|9876.54|123:45
```

3.3.3 Registar passageiro

Permite registar um passageiro no sistema. É pedido o nome do passageiro (`requestPassengerName()`). O novo passageiro pertence à categoria *NORMAL*.

3.3.4 Alterar nome de passageiro

Permite alterar o nome do passageiro. Para tal, primeiro pede-se o identificador do passageiro cujo nome deve ser alterado (`requestPassengerId()`) e de seguida o novo nome (`requestPassengerName()`).

3.4 Menu de Gestão de Itinerários

Este menu permite efectuar operações sobre itinerários. A lista completa é a seguinte: **Mostrar todos os itinerários**, **Mostrar itinerários associados a um passageiro**, **Registar itinerário para um passageiro**. As etiquetas das opções deste menu estão definidas na classe `mmt.app.itinerary.Label`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `mmt.app.itinerary.Message`. Os comandos que concretizam as operações deste menu já estão parcialmente implementados nas classes da package `mmt.app.passenger`, respectivamente: `DoShowAllItineraries`, `DoShowPassengerItineraries`, `DoRegisterItinerary`, `DoShowTotalValue` e `DoShowServiceHistograms`.

3.4.1 Mostrar todos os itinerários

Apresenta todos os itinerários registados para todos os passageiros. O formato de apresentação é como para os serviços, mas apenas se apresenta o segmento viajado e o respectivo custo. Os passageiros são apresentados por ordem crescente do seu número de identificação. A apresentação dos vários itinerários de um passageiro segue o formato descrito na secção §3.4.2.

Note-se que a apresentação de itinerários apenas tem significado quando existem itinerários para mostrar, i.e., não se devem apresentar passageiros que não têm itinerários.

3.4.2 Mostrar itinerários associados a um passageiro

Apresenta os itinerários do passageiro cujo identificador é pedido através de `requestPassengerId()`. A lista é ordenada pela data dos itinerários. Se o passageiro não tiver itinerários, deve ser apresentada a mensagem `noItineraries()`.

O formato de apresentação é análogo ao dos serviços, iniciado com uma linha com o identificador do passageiro, à qual se segue a lista de itinerários, cada um precedido da linha com o número de ordem do itinerário (no contexto de cada passageiro), da data correspondente e do preço do itinerário:

```
␣==␣Passageiro␣identificador-do-passageiro:␣nome-do-passageiro␣==
␣Itinerário␣número-de-ordem-do-itinerário␣para␣data␣@␣preço-total-do-itinerário
```

Note-se que o preço apresentado para cada serviço do itinerário é o da fracção do tempo do serviço nesse itinerário mas considerando apenas duas casas decimais. O preço do itinerário é a soma (sem arredondamentos) dos preços dos segmentos que o formam. Este valor é depois apresentado também com duas casas decimais.

Exemplo de apresentação de itinerários para um passageiro (com o identificador 2):

```
==␣Passageiro␣2:␣José␣Simão␣==

Itinerário␣1␣para␣2017-10-18␣@␣20.03
Serviço␣#692␣@␣6.37
09:06␣Évora
09:17␣Casa␣Branca
09:31␣Vendas␣Novas
09:53␣Pinhal␣Novo
Serviço␣#570␣@␣13.66
10:48␣Pinhal␣Novo
11:19␣Grândola
11:33␣Ermidas-Sado
11:55␣Funcheira
12:24␣Santa␣Clara-Sabóia
12:52␣Messines-Alte
13:02␣Tunes

Itinerário␣2␣para␣2017-10-25␣@␣22.57
Serviço␣#184␣@␣16.41
15:35␣Tunes
17:23␣Pinhal␣Novo
Serviço␣#596␣@␣6.17
17:48␣Pinhal␣Novo
18:10␣Vendas␣Novas
18:24␣Casa␣Branca
18:35␣Évora
```

3.4.3 Registar itinerário para um passageiro

Esta opção permite registar um novo itinerário para um dado passageiro. Para tal, deve pedir o identificador do passageiro (`requestPassengerId()`), o nome da estação de partida (`requestDepartureStationName()`), o nome da estação de destino (`requestArrivalStationName()`), a data da partida, no formato YYYY-MM-DD (`requestDepartureDate()`), e a hora mínima para a partida, no formato HH:MM (`requestDepartureTime()`). Se a data não estiver no formato correcto, o comando que concretiza esta opção deverá lançar a excepção `mmt.app.BadDateException`, não sendo realizada nenhuma acção. Se a hora não estiver no formato correcto, deve ser lançada a excepção `mmt.app.BadTimeException`, não sendo realizada nenhuma acção. Se não for possível fazer uma reserva, i.e., o número de candidatos é nulo, então não é realizada qualquer acção new apresentada uma mensagem.

Em caso de sucesso, como resultado, o sistema apresenta uma lista com todos os candidatos encontrados, ordenados hierarquicamente pelos seguintes critérios: hora de partida, hora de chegada e tempo de viagem. Se forem encontradas múltiplas alternativas com o mesmo serviço de partida, apenas se apresenta a que chega mais cedo (excepto se existir uma ligação directa nesse serviço, mesmo que mais lenta). De seguida, o sistema pede o número da alternativa (`requestItineraryChoice()`) a guardar (1 para a primeira alternativa, 2 para a segunda, etc.) no registo dos itinerários adquiridos pelo passageiro, actualizando-se o estado correspondente. Se a escolha for 0 (zero), não é realizada nenhuma acção e os itinerários candidatos são descartados. Se a escolha corresponder a um número inválido, é lançada a excepção `mmt.app.NoSuchItineraryException`, não sendo realizada nenhuma acção.

O formato de apresentação das escolhas é como definido para os itinerários, precedido da linha com o número da escolha (N representa a escolha), mas sem a linha relativa ao passageiro.
Exemplo de apresentação da hipótese 3 de itinerário: `Évora >> Silves`:

```
Itinerário␣3␣para␣2017-11-21␣@␣20.38
Serviço␣#694␣@␣6.65
16:57␣Évora
17:08␣Casa␣Branca
17:22␣Vendas␣Novas
17:29␣São␣João␣das␣Craveiras
17:33␣Pegoes
17:38␣Fernando␣Pó
17:43␣Poceirão
17:51␣Pinhal␣Novo
Serviço␣#574␣@␣13.73
18:18␣Pinhal␣Novo
18:46␣Grândola
19:01␣Ermidas-Sado
19:29␣Funcheira
19:57␣Santa␣Clara-Sabóia
20:24␣Messines-Alte
20:34␣Tunes
```

4 Leitura de Dados a Partir de Ficheiros Textuais

Além das opções de manipulação de ficheiros descritas na secção §3.1.1, é possível iniciar a aplicação com um ficheiro de texto especificado pela propriedade Java com o nome `import`. Este ficheiro contém a descrição de serviços, passageiros e itinerários a carregar no estado inicial da aplicação.

4.0.4 Exemplo de ficheiro a importar

O formato do ficheiro a importar é como exemplificado a seguir. A notação "..."significa repetição de formato.

```
SERVICE|180|51.5|05:47|Porto␣-␣Campanhã|...|11:23|Faro
SERVICE|694|12.2|16:57|Évora|17:08|Casa␣Branca|...|18:36|Lisboa␣-␣Oriente
SERVICE|420|11.45|08:36|Valença|09:09|Viana␣do␣Castelo|09:50|Nine|10:18|Porto␣-␣Campanhã
SERVICE|5500|11.35|15:48|Elvas|...|18:25|Entroncamento
PASSENGER|Obi-Wan
PASSENGER|Yoda
ITINERARY|0|2017-10-18|690/Évora/Pinhal␣Novo|180/Pinhal␣Novo/Tunes|5904/Tunes/Silves
```

O primeiro campo de um itinerário é o identificador único do passageiro que adquiriu o itinerário em causa. Pode assumir que não existem entradas mal-formadas nestes ficheiros.

5 Considerações sobre Flexibilidade e Eficiência

Devem ser possíveis extensões ou alterações de funcionalidade com impacto mínimo no código já produzido para a aplicação. O objectivo é aumentar a flexibilidade da aplicação relativamente ao suporte de novas funções sem que isso implique alterações no código existente do domínio da aplicação. Os seguintes casos têm interesse especial:

- A aplicação deve suportar a adição de novas categorias de passageiro sem alterações no código já realizado.
- A solução deve suportar a adição de novos procuras de serviço com um custo baixo em termos de código a realizar para suportar a nova procura.

6 Execução dos Programas e Testes Automáticos

Usando os ficheiros `test.import`, `test.in` e `test.out`, é possível verificar automaticamente o resultado correcto do programa. Note-se que pode ser necessária a definição apropriada da variável de ambiente `CLASSPATH` (ou da opção equivalente `-cp` do comando `java`), para localizar as classes do programa, incluindo a que contém o método correspondente ao ponto de entrada da aplicação (`mmt.app.App.main`. As propriedades são tratadas automaticamente pelo código de apoio.

```
java -Dimport=test.import -Din=test.in -Dout=test.outhyp mmt.app.App
```

Assumindo que aqueles ficheiros estão no directório onde é dado o comando de execução, o programa produz o ficheiro de saída `test.outhyp`. Em caso de sucesso, os ficheiros das saídas esperada (`test.out`) e obtida (`test.outhyp`) devem ser iguais. A comparação pode ser feita com o comando:

```
diff -b test.out test.outhyp
```

Este comando não deve produzir qualquer resultado quando os ficheiros são iguais. Note-se, contudo, que este teste não garante o correcto funcionamento do código desenvolvido, apenas verifica alguns aspectos da sua funcionalidade.

7 Notas de Concretização

Tal como indicado neste documento, algumas classes fornecidas como material de apoio, são de uso obrigatório e não podem ser alteradas. Outras dessas classes são de uso obrigatório e têm de ser alteradas.

A serialização Java usa as classes da package `java.io`, em particular, a interface `java.io.Serializable` e as classes de leitura `java.io.ObjectInputStream` e escrita `java.io.ObjectOutputStream` (entre outras). A representação e manipulação de datas e tempos deve ser realizada através das classes da package `java.time`, em particular, através das classes `java.time.LocalDate` e `java.time.LocalTime`. Diferenças entre tempos são representadas pela classe `java.time.Duration`.