Projeto 4

Filas

Unidade Curricular de
Laboratório de Programação

2021/2022

Objetivos

- Utilizar a estrutura de dados Fila com a filosofia *First in, First out* (FIFO) estudada em AED;
- Rever a interface `List` e a classe `LinkedList` do pacote `java.util`.

Antes de Começar

Criar um projeto java e importar os ficheiros

- Fazer *download* do ficheiro `alunosProjeto4.zip` na página de LabP e descomprimir o ficheiro *zip* para uma pasta;
  No Eclipse:
- Criar um novo projeto com o nome `Projeto4`: **File > New > Java Project**
- De seguida selecionar o projeto criado e importar os ficheiros obtidos anteriormente: **Import -> General -> File System** e selecionar a pasta resultante de descomprimir o ficheiro `alunosProjeto4.zip`.
- Configurar o `Build Path` do projeto Java para incluir, na `classPath`, a biblioteca `JUnit5` (veja como o fazer na página 14 do tutorial sobre o Eclipse IDE).
- Deverá passar a ter um projeto chamado `Projeto4` contendo:
  - Na pasta `src`, os ficheiros `RunAirport.java`, `DstCode.java`, `TestsProject4.java`;
- Na raiz do projeto, os ficheiros de dados de entrada `bookings.txt` e `airportFlights.txt`, a utilizar pela classe `RunAirport` disponibilizada para testar a vossa solução para este enunciado.

O ficheiro `alunosProjeto4.zip` também inclui o ficheiro `std_output.txt` que contém o *output* esperado pela execução do método `main` da classe `RunAirport`.

Enunciado

Num aeroporto, geralmente operam diferentes companhias aéreas, cada uma oferecendo diferentes voos diários. Cada voo tem um destino e um número de lugares disponíveis para os passageiros.

As agências de viagens pretendem reservar um certo número de lugares de última hora em alguns voos para os oferecer sucessivamente aos seus próprios clientes. Por isso, as agências têm um acordo com uma empresa no aeroporto que permite o acesso a um mecanismo privilegiado de reservas dos voos operados pelas diferentes companhias que operam no aeroporto em qualquer dia.

A empresa do aeroporto basicamente cuida, diariamente, de reencaminhar os pedidos de reserva das agências para as companhias aéreas que estão operando voos naquele dia. Cada companhia aérea pode consultar os pedidos e responder de acordo com a disponibilidade que tem nos voos que opera. O aeroporto trata os pedidos de reserva das diferentes agências no modo *First Come First Served* (FCFS), também conhecido como *First in First out*.

Pede-se aos alunos de LabP que desenvolvam um programa que permita à empresa aeroportuária, com base na descrição dos pedidos de reserva das agências num ficheiro de texto, perguntar automaticamente às companhias aéreas que operam num determinado dia, se estão ou não aptas a responder com a disponibilidade em seus próprios voos para esses pedidos de reserva das agências.

O que implementar (classes e métodos)

Será necessário implementar pelo menos as seguintes **classes**:

- A classe `FlightBooking`, cujas instâncias representam pedidos de reserva de voo. Cada pedido tem um código do aeroporto destino (ou seja, um valor do enumerado definido em `DstCode.java`), um possível horário de partida (no formato hh:mm), o número de lugares a serem reservados e o estado atual da reserva (CONFIRMED ou NOT_CONFIRMED, criem um simples enumerado por isso).

- A classe `Flight`, cujas instâncias representam voos. Cada voo tem um código do aeroporto de destino, um horário de partida (no formato hh:mm) e o número de lugares disponíveis.

- A classe `AirlineCompany`, cujas instâncias representam companhias aéreas. Cada companhia deve ter, entre outras informações que julgar necessárias, um nome que a identifique, uma lista de objetos `Flight` que serão definidos na fase de inicialização e uma fila de pedidos de reserva do tipo `FlightBooking`.

- A classe `Airport` que representa um aeroporto. Esta classe deve ter um nome de aeroporto, uma lista de companhias aéreas que operam no aeroporto, que será definido na fase de inicialização, uma fila de todos os pedidos de reserva para o dia atual e uma fila de pedidos de reserva que o aeroporto consegue alocar corretamente graças às companhias aéreas e aos voos que estão operando atualmente.

NOTAS:

i. Para representar os horários de partida no formato indicato (hh:mm) sugere-se utlizar a classe `LocalTime` do pacote `java.time`.

ii. As filas de reservas nas classes `AirlineCompany` e `Airport` devem ser implementadas usando a classe `LinkedList` do pacote `java.util`. As listas de companhias aéreas na classe `Airport` e de voos na classe `AirlineCompany` devem ser implementados usando a classe `ArrayList` do pacote `java.util`.

iii. Para cada uma destas quatro classes deverá implementar pelo menos um construtor, métodos "getters" que permitam revelar o valor dos atributos quando necessários, um método `toString()` e ainda os métodos abaixo descritos. Verifique no ficheiro de testes *JUnit* as assinaturas dos construtores das classes e dos restantes métodos que deverá implementar em cada uma delas.

Será necessário implementar pelo menos os seguintes **métodos**:

A classe `FlightBooking` deve ter pelo menos os seguintes métodos:

- `void confirm()` que modifica o estado da reserva para confirmada.

A classe `Flight` deve ter pelo menos os seguintes métodos:

- `boolean bookSeats(int numOfSeats)` que reserva e, portanto, diminui o número de lugares disponíveis num voo. Este método retorna *false* caso não seja possível efetuar a reserva por já não haver um número suficiente de lugares disponíveis.
- `boolean verifySchedule(FlightBooking currentBooking)` que verifica se o pedido de reserva passado como parâmetro pode ser atendido para este voo. O método deve devolver o valor *true* se as três condições a seguir forem válidas: (i) o voo e a reserva referem-se ao mesmo destino, (ii) a hora de partida do voo é posterior ou igual à hora de partida do pedido, (iii) o voo dispõe de lugares livres suficientes para responder a este pedido.

A classe `AirlineCompany` deve ter pelo menos os seguintes métodos:

- `void addFlight(DstCode destination, LocalTime time, int seats)` que adiciona um voo à companhia; os parâmetros representam o código do aeroporto destino (do tipo `DstCode`), o horário de partida e o número de lugares.
- `void loadBookings (List<FlightBooking> dayBookings)` que recebe a fila de pedidos de reserva do aeroporto e faz as reservas que puder atender com seus voos. A única modificação do parâmetro que este método deve fazer é marcar um pedido como `CONFIRMED` quando consegue colocá-lo num dos seus voos. Cada pedido marcado também é copiado e armazenado na fila de reservas da companhia.

A classe `Airport` deve ter pelo menos os seguintes métodos:

- `void loadFlights(String filename) throws FileNotFoundException` que lê informações sobre os voos que partem de um aeroporto a partir de um novo ficheiro de entrada. O ficheiro *airportFlights.txt* ilustra o formato dessas informações: numa linha tem o nome de uma companhia que tem voos a partir desse aeroporto e o número desses voos diários. Se for *k* esse número, nas *k* linhas seguintes, uma por linha, as informações sobre cada um desses voos: destino, hora, lugares.

- `void loadBooking(String rawBooking)` que cria um pedido de reserva `FlightBooking` a partir dos dados no parâmetro `rawBooking` e o adiciona à fila de pedidos do aeroporto. O parâmetro tem a mesma estrutura que uma linha do ficheiro *bookings.txt* e indica, separados por um espaço em branco, o código do aeroporto de destino, a hora de partida pretendida e o número de lugares a reservar.
- `void processBookings()` que pede a todas as companhias aéreas que operam no aeroporto que atendam aos pedidos de reserva presentes na fila do aeroporto. Após ter solicitado a uma companhia e antes de passar para a próxima, este método deve chamar o método `sortBookingQueues` para retirar desta fila as reservas que já foram atendidas pela última companhia.
- `void sortBookingQueues()` que reorganiza as reservas na fila, retirando as reservas já atendidas e colocando-as noutra fila que contenha as reservas que já foram alocadas com sucesso.

NOTA: Por uma questão de eficiência, aceita-se que os métodos que retornam valores de atributos que são listas retornem uma referência para o atributo e não uma sua cópia. Nesse caso, deverá portanto garantir que, nas classes clientes, esses valores não são alterados. Os testes JUnit que incidem sobre esses métodos obedecem a essa restrição.

A classe `RunAirport`

É dada uma classe `RunAirport`, já implementada, que devem usar para testar a vossa implementação das classes pedidas. Esta classe lê uma lista de reservas a partir de um ficheiro de entrada `bookings.txt` e, depois de ter criado um aeroporto onde operam as companhias e os voos listados no ficheiro `airportFlights.txt`, tenta responder aos pedidos de acordo com a disponibilidade de voos presentes no aeroporto. Deve verificar nesta classe quais os parâmetros que deve considerar para os construtores das várias classes pedidas acima.

No final da sua execução, esta classe também imprime, na consola, um relatório com os resultados da sua tentativa de alocar as reservas presentes no ficheiro de entrada. Pode verificar se as suas classes implementam o comportamento esperado comparando o *output* da execução da classe `RunAirport` com o *output* fornecido no ficheiro `std_output.txt`.

O que entregar

Deve criar o ficheiro `P4fcxxxxx.zip`, onde `xxxxx` é o seu número de aluno, contendo os ficheiros `Flight.java`, `FlightBooking.java`, `AirlineCompany.java`, e `Airport.java` e submetê-lo no Moodle.

Antes de entregar, certifique-se da correção da formatação das classes e inclua comentários *javadoc* para todos os métodos. Inclua também uma linha contendo a *tag* @author onde deve indicar o seu nome e número de aluno.