Projeto 3

Pilhas

Unidade Curricular de
Laboratório de Programação

2021/2022

Objetivos

- Utilizar a estrutura de dados Pilha;
- Utilizar o interface `Stack` e a classe `ArrayStack` desenvolvidos na disciplina de AED.

Antes de Começar

Criar um projeto java e importar os ficheiros

- Fazer *download* do ficheiro `alunosProjeto3.zip` na página de LabP e descomprimir o ficheiro *zip* para uma pasta;
- No Eclipse:
  - Criar um novo projeto com o nome `Projeto3`: **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 `alunosProjeto3.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 `Projeto3` contendo:
    - Na pasta `src`, os ficheiros `RunBookings.java`, `BookingStatus.java`, `TypeOfVehicle.java`, `TestsProject3.java`, `Stack.java` e `ArrayStack.java`;
    - Na raiz do projeto, os vários ficheiros de dados (`reserva1.txt`, `reserva2.txt`, `reserva3.txt`).

    O ficheiro `alunosProjeto3.zip` também inclui o ficheiro *`std_output.txt`* que contém o *output* esperado pela execução do método `main` da classe `RunBookings.java`.

Enunciado

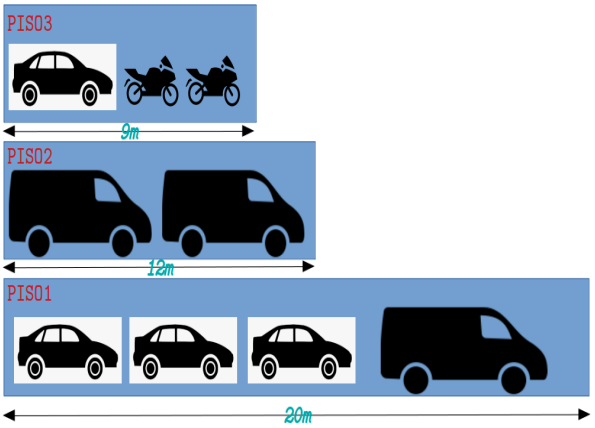
A empresa *ParkAndGo* pretende automatizar o processo de gestão do seu parque de estacionamento num edifício no centro da cidade de Lisboa. O objetivo da automatização do processo será (i) evitar movimentos múltiplos de veículos estacionados durante o dia devido à existência de zonas de estacionamento do tipo LIFO (Last In First Out) nos pisos do edifício, (ii) evitar a aceitação de pedidos de estacionamento sem ter espaço suficiente para estacionar um veículo.

Pede-se aos alunos de LabP que desenvolvam um programa que permita à empresa, a partir da descrição das reservas de estacionamento de veículos num dia, organizar

automaticamente os lugares disponíveis no parque e identificar antecipadamente possíveis problemas na organização dos veículos a receber.

Considere o estacionamento ao lado com três pisos, onde cada piso tem o comprimento máximo em metros indicado pela seta abaixo:

- piso1: car, car, car, van
- piso2: van, van
- piso3: car, moto, moto



uma possível descrição de reserva que criou o estado de estacionamento atual na imagem seria a seguinte:

```
2355xf-car (11:05,4.7), 7880lm-car (12:05,4.8), 7710rm-car
(13:25,5.0), 2143ad-van (11:05,6.2), 3421re-van (22:05,5.6),
3348xh-car (12:15,4.5), 9345wq-van (14:35,4.9), 3367ui-moto
(13:45,2.2), 2115aq-moto (14:35,2.2), 2245aq-van (14:35,6.5)
```

Onde cada elemento inclua o número de matrícula e uma descrição do tipo de veículo, a indicação da hora prevista de entrada do veículo no parque de estacionamento e a dimensão do veículo. A partir de uma descrição deste tipo, o vosso programa deve produzir uma série de instruções que servirão para saber se há lugares suficientes para todos os veículos no estacionamento e como estacioná-los nos diferentes andares para que possam entrar no estacionamento nos horários indicados sem ter que mover outros veículos. O vosso programa deve começar inserindo veículos em baixo (piso1) e, quando não houver mais vagas ou quando houver incompatibilidade de horário, passar para o próximo piso (piso2), e assim sucessivamente.

Para este exemplo, essas instruções seriam:

```
Go pick car 2355xf and park it on floor 1
Go pick car 7880lm and park it on floor 1
Go pick car 7710rm and park it on floor 1
Go pick van 2143ad and park it on floor 2
Go pick van 3421re and park it on floor 2
Go pick car 3348xh and park it on floor 3
Go pick van 9345wq and park it on floor 1
Go pick moto 3367ui and park it on floor 3
Go pick moto 2115aq and park it on floor 3
No space left to park the following vehicles:  2245aq
```

As descrições fornecidas devem obedecer às seguintes regras para serem consideradas reservas válidas:

1. Todos os veículos devem aparecer com o formato completo *matricula-tipo (hh:mm, x.y) ;* a falta de um ou mais dos quatro elementos da descrição, ou seja, *matrícula*, *tipo*, *hh:mm*, *x.y*, representa uma descrição mal formatada;
2. Cada número de matrícula apenas pode aparecer uma vez numa lista;
3. O tipo de veículo deve ser um tipo permitido (*car*, *van*, *moto*);
4. As horas têm que ter o formato *hh:mm* onde *hh* deve representar um valor entre 0 e 23 e *mm* entre 0 e 59 (tenha em atenção que os números menores que 10 são representados com um zero à esquerda);
5. O comprimento *x.y* de cada veículo deve ser de pelo menos 2 metros.

Objetivos

No contexto de um parque de estacionamento com três pisos, pretendemos que seja possível realizar as seguintes várias ações a partir de uma descrição de reserva de estacionamento:

(i) Saber se a descrição de uma reserva está correta, ou seja, se está respeitando as regras especificadas acima. Exemplos de descrições incorretas:

- `2355xf-car (11:05,3.6), 7880lm-car (4.0), 2143ad-van (22:05,5.9)` — o veículo `7880lm-car` não mostra a hora de entrada *(violação regra1).* Pode-se supor que a descrição de cada elemento estará sempre bem formatada e a violação da regra1 só acontece quando um dos 4 componentes da descrição não é encontrado;
- `2355xf-car (11:05,3.6), 7880lm-car (11:45,4.3), 2143ad-van (22:05,5.9), 7880lm-car (11:45,3.7)` — a matrícula `7880lm` aparece duas vezes na lista *(violação regra2)*;
- `2355xf-car (11:05,3.6), 7880lm-bike (11:45,4.3), 2143ad van- (22:05,5.9)` — o tipo de veículo *bike* não é permitido *(violação regra3)*;
- `2355xf-car (11,3.6), 7880lm-car (11:45,4.3), 2143ad van- (22:05,5.9)` — a hora de entrada do `2355xf-car` não tem o formato válido (*violação regra4*);
- `2355xf-car (11,3.6), 7880lm-car (11:45,4.3), 2115aq-moto (08:35,1.8)` — o comprimento do veículo `2115aq-moto` é inferior a 2 metros (*violação regra5*).

(ii) Saber se a reserva corresponde a uma operação suportada pelo parque de estacionamento. O número e o tamanho dos veículos especificados encontram espaço suficiente nos andares do estacionamento. Se estiver, estacionar os veículos e imprimir uma descrição das operações conforme acima (página 2).

(iii) Uma vez validada a reserva e estacionados os veículos correspondentes, proceder ao levantamento de um veículo. Isso só é possível se o veículo estiver estacionado como o último no seu piso.

O que fazer

Comecemos por raciocinar sobre os tipos de dados necessários para desenvolver o programa pedido.
Temos *Veículo*, com características que os definem e os distinguem uns dos outros:

- Um número de matrícula único (no exemplo `2355xf`, `7880lm`, `2115aq`, etc.);
- Um tipo de veículo (*car*, *van*, *moto*);
- Uma hora de entrada no estacionamento (ex., 22:30);
- Um comprimento em metros (ex., 3.6m).

Temos *ParkingLot* com três pisos diferentes utilizados para estacionar veículos. Cada piso tem um comprimento máximo definido na fase de criação do estacionamento. Cada piso poderá conter a qualquer momento um número variável de veículos estacionados; este número dependerá da soma dos comprimentos dos veículos ser menor que o comprimento máximo do piso. Não há espaço reservado para as categorias de veículos nem uma hierarquia dos mesmos. Os lugares de estacionamento podem ser distribuídos de forma *ad-hoc*, conforme os veículos forem chegando.

Temos também o conceito de *reserva*. Mas, na verdade, o que nos interessa é:

- saber se uma descrição de reserva é válida, ou seja, se está bem construída;
- se a descrição for válida, obter a sucessão de instruções para estacionar os veículos.

Ou seja, nada nos faz sentir a necessidade de ter objetos que representem reservas.

Então, faz sentido criarmos as duas classes seguintes:

- `Vehicle`, cujas instâncias representam veículos. Deve definir:
  - atributos que representem as características indicadas anteriormente;
  - um construtor que inicialize os atributos de um novo artigo com os valores dados nos parâmetros;
  - métodos que permitam revelar os valores dos atributos;
- `ParkingLot`, cujas instâncias representam parques de estacionamento. Deve definir:
  - atributos que representem os três pisos indicados anteriormente;
  - um construtor que receba como parâmetros os comprimentos máximos dos três pisos do estacionamento;

  Para estar de acordo com o código da classe `RunBookings` por nós fornecida, que é cliente de `ParkingLot`, esta classe deve oferecer também os seguintes métodos públicos:

  - `BookingStatus validBooking(String description)`, que verifica se `description` é uma reserva válida. Este método deve verificar as regras (pag. 2) na ordem apresentada acima, começando pela regra 1, seguindo com a regra número 2 e assim sucessivamente até à regra número 5. O método pode retornar o primeiro erro encontrado nesta validação e como resultado terá um dos seguintes valores:
    - `VALID_BOOKING` – descrição válida segundo as regras já descritas anteriormente;
    - `BAD_FORMAT` – a descrição de algum veículo está incompleta (regra 1).
    - `DUPLICATE_VEHICLE` – a descrição contem números de matrícula repetidos (regra2).

        - `UNKNOWN_VEHICLE_TYPE` - a descrição contém um tipo de veículo não permitido (regra3).
        - `INVALID_TIME` – algum veículo na descrição mostra um hora de entrada não formatada corretamente (regra4).
        - `MINIMUM_LENGTH` – algum veículo na descrição não tem o comprimento mínimo exigido (regra5).
    - `String parkVehicles(String description)` que, supondo que `description` é uma descrição válida, retira os veículos que aparecem na descrição e coloca-os nos pisos de estacionamento. Retorna uma *string* contendo a sequência de instruções correspondentes (ver pág. 2);
    - `boolean pickVehicle(String plate_number)` que devolve um veículo estacionado a partir do seu número de matrícula, supondo que o veículo está estacionado em qualquer uma das últimas posições nos pisos do parque de estacionamento;
    - `void free()`, que descarrega totalmente o parque de estacionamento removendo todos os veículos estacionados;
    - `boolean isEmpty()`, que retorna o valor *true* quando o parque de estacionamento está sem veículos.

Os tipos de dados seguintes são fornecidos por nós e devem ser usados na vossa solução:

- `BookingStatus`, enumerado cujos valores representam resultados possíveis na validação de uma descrição de reserva do parque;
- `TypeOfVehicle`, enumerado cujos valores representam tipos de veículos permitidos;
- `Stack`, o interface construído em AED que define o tipo de dados Pilha; deverão usar este tipo de dados na implementação de alguns métodos da classe `ParkingLot`, como por exemplo, os métodos `parkVehicles` e `pickVehicle`.
- `ArrayStack`, uma classe que implementa o interface `Stack`.

Como exemplo de utilização das classes que precisam de criar, vejam o método `main` da classe `RunBookings` fornecida por nós. Devem executá-lo para testarem a vossa solução. Esse método `main`:

- Cria uma instância da classe `ParkingLot`, que representa um parque de estacionamento com três pisos;
- Para cada um dos ficheiros `reserva1.txt`, `reserva2.txt`, `reserva3.txt`:
    - Verifica se a reserva é válida e:
        - se for válida, faz primeiro a reserva correspondente e imprime o *status* do estacionamento;
        - se não for válida, imprime o problema detetado.
    - Por fim esvazia o parque descarregando a reserva.

Antes de Entregar

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

Entrega

Deve criar o ficheiro P3fcxxxxx.zip, onde xxxxx é o seu número de aluno, contendo os dois ficheiros `Vehicle.java` e `ParkingLot.java` e submetê-lo no Moodle.