# POO (MiEI/LCC)

2020/2021

Ficha Prática #06

Hierarquia, herança, polimorfismo, interfaces, excepções e serialização de informação.

# Conteúdo

# 1 Objectivos

Durante três aulas, o projecto DriveIt servirá para abordar vários conceitos leccionados em POO. Do mesmo modo, servirá como auto-avaliação para que os alunos possam fazer um ponto da situação sobre a disciplina.

O objectivo deste projecto é o desenvolvimento, de uma forma incremental, de uma aplicação que combina todos os conceitos leccionados ao longo do semestre. Na primeira fase serão abordados os conceitos de hierarquias de classes, assim como mapeamentos (c.f. API `Map<K,V>` já praticada anteriormente na Ficha 6). A segunda fase abordará os conceitos de Interface, nomeadamente as interfaces `Comparable<T>` e `Comparator<T>`. Na terceira e última fase será abordado o tema de *streams* de dados assim como o tratamento de excepções.

# 2 Projeto DriveIt

Considere que se pretende desenvolver um sistema para a gestão de uma empresa que possui veículos para alugar (classe DriveIt). Pretende-se ter uma classe que agrupe a informação sobre os veículos, sendo que existem veículos de diferentes características que são oferecidos aos clientes. O sistema a desenvolver deverá ser capaz de representar essa informação, e fornecer um conjunto de funcionalidades que permita consultá-la e manipulá-la.

## 2.1 Fase 1

Um **Veículo** é uma classe que representa a informação que todos conhecemos dos veículos. A informação que se guarda para cada instância desta classe é a seguinte:

- um código de identificação (que poderá ser a matrícula)
- marca do veículo
- modelo
- ano de construção
- velocidade média por km
- preço teórico base por km
- classificação do veículo, dada numa escala de 0 a 10 e que é calculada com base na classificação dada pelos clientes depois de terminado o aluguer do carro.

O cálculo do custo real por km é função do número de kms efectuados até ao momento tendo em conta o preço teórico por km e acrescido de 10% (obtido

de forma empírica através de testes efectuados). Este cálculo é feito por cada um dos veículos.

Actualmente a empresa DriveIt tem veículos homogéneos pelo que de momento a informação acima identifica é tudo o que é necessário saber de um veículo (previsivelmente esta situação mudará num futuro próximo).

1. Crie a classe `Veiculo` (se fez pode reutilizar a classe `Carro` da Ficha 3) e a classe `DriveIt`, e implemente nesta os seguintes métodos:

   (a) Verificar a existência de um veículo, dado o seu código.
   ```
   public boolean existeVeiculo(String cod)
   ```

   (b) Devolver a quantidade de veículos existentes na empresa de aluguer.
   ```
   public int quantos()
   ```

   (c) Devolver o número total de veículos de uma dada marca.
   ```
   public int quantos(String marca)
   ```

   (d) Devolver a informação de um veículo, dado o seu código.
   ```
   public Veiculo getVeiculo(String cod)
   ```

   (e) Adicionar um novo veículo.
   ```
   public void adiciona(Veiculo v)
   ```

   (f) Devolver uma lista contendo a cópia de todos os veículos no sistema.
   ```
   public List<Veiculo> getVeiculos()
   ```

   (g) Adicionar a informação de um conjunto de veículos que foram adquiridos e passam agora a fazer parte da empresa.
   ```
   public void adiciona(Set<Veiculo> vs)
   ```

   (h) Registar um aluguer de um veículo e indicar o número de kms que foram feitos por um cliente.
   ```
   public void registarAluguer(String codVeiculo, int
   numKms)
   ```

   (i) Classificar um veículo quando o cliente termina o aluguer.
   ```
   public void classificarVeiculo(String cod, int
   classificacao)
   ```

   (j) Calcula o custo real por km de um veículo de acordo com a regra enunciada anteriormente.
   ```
   public int custoRealKm(String cod)
   ```

2. Crie agora uma nova classe de veículos, o `VeiculoOcasiao` em que o custo por km depende se a empresa está ou não com promoções para este tipo de veículos. Se estiverem em promoção o custo por km destes veículos é 25% inferior ao que deveria ser o custo real.

   Refaça a classe `DriveIt` para que possa passar a gerir além dos veículos normais este novo tipo de veículos.

3. Considere agora que aos tipos de veículos já especificados se acrescentam os veículos *VeiculoPremium*, nos quais existe uma `taxa` de luxo (variável de veículo para veículo), que é um factor multiplicativo do valor do custo por km que é calculado.

   Utilizando as implementações de `Veículo`, `VeículoOcasião` e `VeículoPremium` já desenvolvidas, implemente novamente as seguintes funcionalidades na classe `DriveIt`:

   (a) Verificar a existência de um veículo, dado o seu código.
   ```
   public boolean existeVeiculo(String cod)
   ```

   (b) Devolver a quantidade de veículos existentes na empresa de aluguer.
   ```
   public int quantos()
   ```

   (c) Devolver a quantidade de veículos de um dado tipo (VeículoPremium, VeículoOcasião, etc.).
   ```
   public int quantosT(String tipo)
   ```

   (d) Devolver a informação de um veículo, dado o seu código.
   ```
   public Veiculo getVeiculo(String cod)
   ```

   (e) Adicionar um novo veículo.
   ```
   public void adiciona(Veiculo v)
   ```

   (f) Devolver uma lista contendo a cópia de todos os veículos no sistema.
   ```
   public List<Veiculo> getVeiculos()
   ```

   (g) Devolver uma lista dos veículos ordenados decrescentemente pelo valor do custo real por km
   ```
   public List<Veiculo> veiculosOrdenadosCusto()
   ```

   (h) Devolver o veículo com o custo real mais baixo
   ```
   public Veiculo veiculoMaisBarato()
   ```

   (i) Devolver o veículo com menos kms percorridos
   ```
   public Veiculo veiculoMenosUtilizado()
   ```

   (j) Adicionar a informação de um conjunto de veículos que foram adquiridos e passam agora a fazer parte da empresa.
   ```
   public void adiciona(Set<Veiculo> vs)
   ```

   (k) Registar um aluguer de um veículo e indicar o número de kms que foram feitos por um cliente.
   ```
   public void registarAluguer(String codVeiculo, int
   numKms)
   ```

   (l) Classificar um veículo quando o cliente termina o aluguer.
   ```
   public void classificarVeiculo(String cod, int
   classificacao)
   ```

(m) Calcula o custo real por km de um veículo de acordo com a regra enunciada anteriormente.

```
public int custoRealKm(String cod)
```

(n) colocar a empresa de aluguer em modo de promoção, sendo que só se aplica aos VeiculoOcasiao, ou acabar com a promoção.

```
public void alteraPromocao(boolean estado)
```

Pretende-se agora criar um novo tipo de veículo, os veículos `AutocarroInteligente`. Estes têm um custo por km variável de acordo com a sua ocupação. O custo real por km desses veículos varia linearmente desde 50% do valor calculado (face aos veículos normais), para uma ocupação de 0% até 60% (número de passageiros). A partir de 61% a redução é de 75% do valor calculado.

10. Adicione a classe `AutocarroInteligente` à hierarquia de veículos já definida.

11. Acrescente, na classe `DriveIt`, um método para calcular o valor médio por km, considerando uma ocupação dos autocarros de 85%.

**Auto avaliação 1** Como forma de auto avaliação, propõe-se a implementação da classee `DriveItList`, com os mesmos requisitos que `DriveIt`, mas utilizando como estrutura de dados um `List<Veiculo>`.

**Auto avaliação 2** Propõe-se também a implementação da classe `DriveItSet`, com os mesmos requisitos que `DriveIt`, mas utilizando como estrutura de dados um `TreeSet<Veículo>`.

## 2.2 Fase 2

Considere agora que se decide assumir que só existem veículos dos tipos `VeiculoNormal`, `VeiculoPremium`, `VeiculoOcasiao` e `AutocarroInteligente` e que a classe `Veiculo` deverá ser abstracta e impor às suas subclasses a necessidade de estas responderem (pelo menos) ao método abstracto `public abstract double custoRealKm();`

1. Modifique a classe `Veiculo` para que esta passe a ser abstracta. Verifique se o programa continua a funcionar correctamente.

2. Pretende-se agora poder consultar a informação de forma ordenada por diferentes critérios. Assim, altere a classe `DriveIt` para suportar as seguintes funcionalidades.

   (a) Obter um `Set<Veiculo>`, ordenado de acordo com a ordem natural dos veículos (assuma que a ordem natural dos veículos é a ordem alfabética da sua marca e caso esta seja igual o segundo factor é o modelo)
   `public Set<Veiculo> ordenarVeiculos()`

   (b) Faça o mesmo que na alínea anterior, mas agora para a assinatura:
   `public List<Veiculo> ordenarVeiculos()`

   (c) Obter um `Set<Veículo>`, ordenado de acordo com o comparador fornecido:
   `public Set<Veiculo> ordenarVeiculos(Comparator<Veiculo> c)`. Experimente criar ordenações em função do número de kms percorridos e em função do preço real por km (experimente comparadores por ordem crescente e decrescente).

   (d) Guardar (de forma a que sejam identificáveis por nome) comparadores na classe `DriveIt`, permitindo assim ter disponíveis diferentes critérios de ordenação.

   (e) Obter um iterador de `Veiculo`, ordenado de acordo com o critério indicado:
   `public Iterator<Veiculo> ordenarVeiculo(String criterio)`

## 2.3 Fase 3

Considere que esta empresa de veículos também criou o seu cartão de fidelidade que atribui aos clientes de alguns dos seus veículos. Ao fazer um aluguer nos veículos `VeiculoPremium` e `AutocarroInteligente`, é possível obter pontos de bonificação (X pontos por cada km percorrido, sendo X um valor variável em função de cada veículo).

1. Desenvolva a interface `BonificaKms`, que deverá ser implementada por estes dois tipos de veículos. Esta interface deverá garantir as funcionalidades a seguir descritas.
   - Definir o valor de pontos a atribuir por cada km.
   - Obter o valor de pontos que se está a a atribuir por cada km.
   - Obter o valor de pontos que um determinado veículo já acumulou.
2. Actualize as classes `AutocarroInteligente` e `VeiculoPremium` de modo a que implementem a interface.
3. Acrescente, na classe `DriveIT`, um método para obter a lista dos veículos que atribuem pontos.

   ```
   public List<BonificaKms> daoPontos()
   ```

## 2.4 Fase 4

Deverá ser possível persistir a informação dos veículos, representada em memória por `DriveIt`. Assim, implemente em `DriveIt` funcionalidades descritas a seguir.

1. Exportar a informação de todos os veículos como um ficheiro CSV.
2. Gravar e carregar a instância de `DriveIt` num ficheiro de objeto em disco.

De forma a melhorar a implementação de `DriveIt`, implemente agora excepções para suportar o tratamento dos erros a seguir descritos:

3. Veículo inexistente numa consulta.
4. Adicionar veículo repetido.
5. Definição de valores inválidos (e.x. valores negativos).

Finalmente, para completar a implementação do sistema...

6. Desenvolva a classe `DriveIt` que deverá dar acesso às funcionalidades da classe `DriveIt` através de um menu em modo texto.