Sumário
- Herança

Documentação complementar Java:
- Inheritance

Nota: Gere o JavaDoc para o(s) projeto(s) utilizado(s) na resolução desta ficha de trabalho.

Parte 1

Desenvolva uma API (*Application Programming Interface*) que permita armazenar informação relativa a um conjunto de veículos disponíveis para venda num *stand* de automóveis. O *stand* possui vários tipos de veículos como ligeiro, motociclo e pesado.
De um modo geral pode afirmar-se que um veículo possui as seguintes características:

- Identificador interno do *stand* (`id`). Este deverá ser único e auto-incrementar sempre que se adicionar um novo veículo
- Número de chassis (`vin`)
- Marca (`brand`)
- Modelo (`model`)
- Data de fabrico (`manufacturingDate`)
- Origem (`origin`), que poderá tomar os valores nacional (`National`) ou importado (`Imported`)
- Números de quilómetros (`kms`)
- Condição (`condition`), que poderá ser novo (`new`) ou usado (`used`)
- Preço (`price`), que deverá ser um número inteiro

Em função do tipo específico de veículo poderão ser incluídos os seguintes atributos mais específicos

- Automóvel
  - Número de ocupantes (`occupantsNumber`)
  - Número de portas (`doorsNumber`). Por omissão um automóvel terá 3 portas
- Motociclo
  - Cilindrada (`cubicCapacity`)
  - Diâmetro de rodas (`wheelSize`)
- Pesado
  - Comprimento (`length`)
  - Carga útil (`load`)
  - Tipologia (`truckType`). Poderá tomar os valores `Truck` ou `TIR`
  - Atrelado (`trailer`). Pode não existir

Entre particularidades de cada tipo de veículo, foi implementada uma política que determina o preço de um veículo de acordo com regras especificas para cada tipo:

- Automóvel
  - Se novo deve ter o preço definido originalmente
  - Se usado deve ter um desconto de 30% sobre o preço original
- Motociclo
  - Tem sempre o preço definido originalmente
- Pesado
  - Se tiver atrelado e a condição for nova o preço deve ter um desconto de 5%
  - Se não tiver atrelado e for novo deve ter o preço original
  - Em todos os outros casos o preço final deve sofrer uma redução de 15%

Devido ao volume de vendas e por questões estratégicas de comércio, o *stand* sentiu necessidade de vender atrelados (`trailer`) especificando-os com a seguinte estrutura:

- Número de eixos (`axesNumber`). Por omissão o número de eixos é 2
- Capacidade (`load`)

Um atrelado pode ser sub-dividido em:

- Reboque (`towHaul`)
  - Tipologia (`trailerType`). Poderá ser do tipo basculante (`tipper`), aberto (`open`), fechado (`closed`), frigorífico (`fridge`)
- Semi-reboque (`semiTrailer`)
  - Número de pneus sobressalentes (`spareTyreNumber`)

Na resolução dos exercícios propostos considere que deve:

- Garantir o encapsulamento de todas as classes criadas
- Criar os métodos de acesso necessários para as classes criadas
- Criar, num *package* específico, as enumerações necessárias para suportar o problema apresentado
- Criar métodos específicos para manipulação de coleções, ou seja, não deverá ser permitido o acesso direto às variáveis que representam coleções, devendo existir métodos para adicionar, remover, editar e listar elementos.

Antes de resolver os exercícios, deve estruturar as classes de acordo com as relações entre elas e os packages a que pertencem. Atente à Figura 1 que representa um diagrama de classes. Este diagrama está propositadamente incompleto.

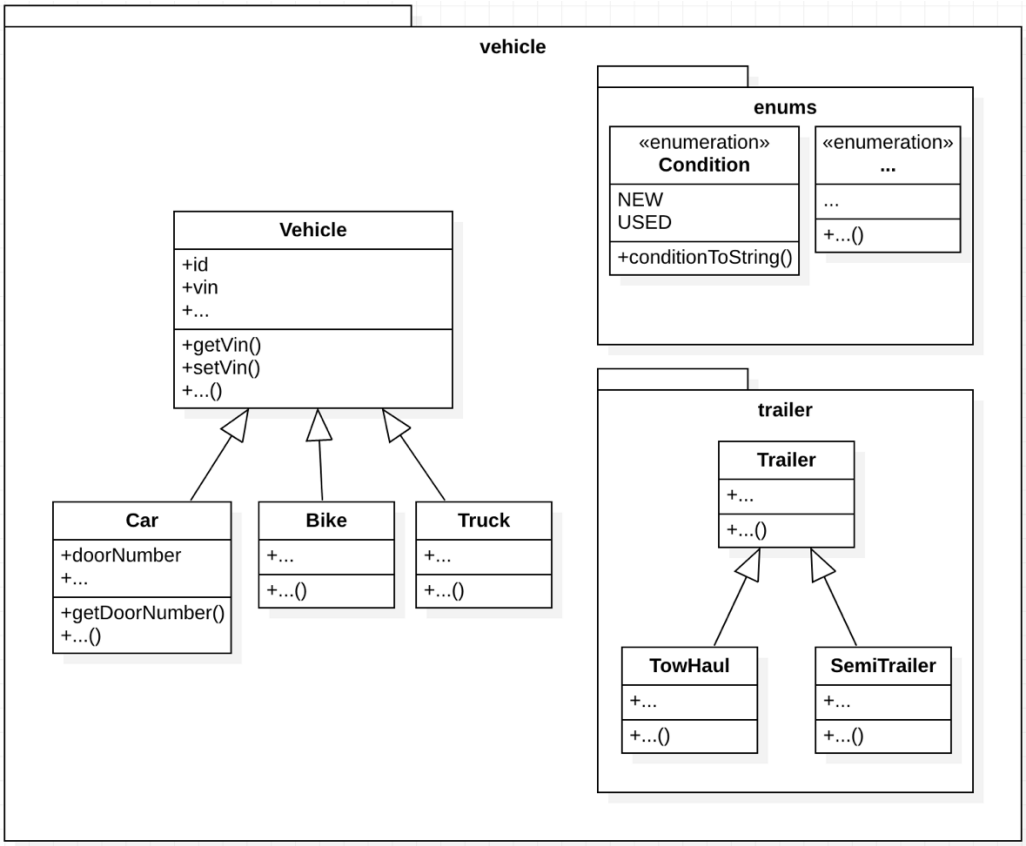

Figura 1 - Diagrama (parcial) de classes

Exercício 1

1.1) Complete o diagrama presente na Figura 1 substituindo os atributos, tipos e valores exemplo de acordo com o enuciado do problema.
1.2) Crie um projeto com o nome `pp_fp08` e, no package `pp_fp08.stand`, implemente o código Java para representar a estrutura descrita anteriormente.
1.3) Crie a classe `VehicleDemo` de forma a testar as classes implementadas. Inicialize alguns elementos relativos aos diferentes tipos de veículos.
1.4) De modo a ser possível obter o preço correto de cada veículo, deve reescrever o método `getPrice` em cada classe de forma a cumprir os objetivos do enunciado. Note que poderá ter de implementar o método em várias subclasses.
1.5) De modo a ser possível imprimir a informação armazenada num objeto criado reescreva o método `toString` herdado da classe `Objet` do Java. Este método é responsável por devolver uma `String` com todos os dados. Note que poderá ter de implementar o método nas várias subclasses.

Exercício 2

2.1) Crie uma classe `VehicleManagement` que permita armazenar um conjunto de veículos que estão disponíveis para venda. Implemente os métodos necessários para adicionar, remover e listar os diversos veículos.
2.2) A adição de novos veículos não deverá permitir a repetição do número de chassis (`vin`).
2.3) Altere o método de adicionar de modo a não limitar o número de veículos disponíveis.
2.4) Implemente um método que permita verificar se já existe um veículo. Como deverá realizar a comparação de dois objetos?
2.5) Crie um método que permita contar quantos veículos se encontram disponíveis para venda e por tipologia. O método deverá devolver uma `String`.
2.6) Implemente um método para imprimir toda a informação armazenada no gestor de veículos.