CI1 Parte prática. Duração: 2h00m

Numa garagem estão armazenados um conjunto de veículos. Destes, interessam-nos os carros, dos quais existem a gasolina, elétricos e híbridos, Cada carro (***Car***) tem uma marca (***brand***), modelo (***model***) e preço (***price***). Os carros a gasolina (***FuelCar***) têm um depósito (***tank***) com uma determinada capacidade em litros, bem como um consumo médio expresso em litros por 100 km (***l_100km***). Já um carro elétrico (***ElectricCar***) possui uma bateria (***battery***) de capacidade expressa em kWh, e um consumo médio em kWh por 100 km (***kWh_100km***). Finalmente, um carro híbrido (***HybridCar***) possui as características de um carro a gasolina e de um carro elétrico.

A partir da capacidade do depósito ou da bateria, bem como do consumo médio por 100 km, é possível calcular a autonomia do carro (km que é possível efetuar sem qualquer carregamento de combustível e/ou de bateria). No caso dos híbridos, considera-se, por simplificação, que a sua autonomia é dada pela soma das autonomias do motor a gasolina e do motor elétrico.

A garagem tem uma capacidade máxima (em número de veículos que comporta).

As classes ***Garage***, ***Car***, ***FuelCar***, ***ElectricCar*** e ***HybridCar*** estão parcialmente definidas a seguir.

NÃO PODE acrescentar membros-dado a estas classes, para além dos que já se encontram declarados.

```
template <class Vehicle>
class Garage {
   vector<Vehicle *> vehicles;
   const unsigned int capacity;
public:
   bool addVehicle(Vehicle *vehicle);
   Vehicle* removeVehicle(string brand,
                          string model);
   float avgPrice(string brand) const;
   void sortVehicles();
};

class Car {
   const string brand;
   const string model;
   float price;
public:
   Car(string b, string m);
   string getBrand() const;
   string getModel() const;
   float getPrice() const;
   void setPrice(float price);
   virtual float range() const = 0;
   bool operator == (const Car &car)
                                 const;
   friend ostream & operator<<(
         ostream & os, const Car &car);
};
```

```
class FuelCar : public Car {
   float tank;   // in liters
   float l_100km;
public:
   FuelCar(string brand, string model,
           float price, float tank,
           float l_100km);
};

class ElectricCar : public Car {
   float battery; // in kWh
   float kWh_100km;
public:
   ElectricCar(string brand,
           string model,
           float price, float battery,
           float kWh_100km);
};

class HybridCar : public FuelCar,
           public ElectricCar {
public:
   HybridCar(string brand, string model,
           float price,
           float tank,
           float l_100km,
           float battery,
           float kWh_100km);
};
```

a) [**2.5 valores**] Implemente os seguintes construtores das classes ***FuelCar*** e ***ElectricCar***:

```
FuelCar(std::string brand, std::string model, float price,
        float tank, float l_100km);
ElectricCar(std::string brand, std::string model, float price,
        float battery, float kWh_100km);
```

b) [**2.5 valores**] Implemente na classe ***Car*** o operador ==. Dois carros são iguais se tiverem a mesma marca (***brand***) e modelo (***model***).

```
bool operator == (const Car &car) const;
```

c) [**2.5 valores**] Implemente, nas classes necessárias, o membro-função que calcula a autonomia de um carro (km que pode realizar sem qualquer abastecimento):

```
float range() const
```

Note que esta função já está declarada e encontra-se comentada na classe ***Car***. Considere, de forma simplista, que a autonomia de um carro híbrido corresponde à soma das autonomias das suas componentes de carro a gasolina e de carro elétrico.

d) [**2.5 valores**] Implemente na classe genérica ***Garage*** o membro-função:

```
bool addVehicle(Vehicle *vehicle);
```

Esta função deve adicionar ao vector ***vehicles*** o argumento ***vehicle***, mas apenas se a garagem não estiver cheia (membro dado ***capacity***), e se o veículo a adicionar ainda não existir. Se o veículo for adicionado, a função deve retornar ***true***, senão deve retornar ***false***.

e) [**2.5 valores**] Implemente na classe genérica ***Garage*** o membro-função:

```
void sortVehicles();
```

Esta função ordena os elementos do vector ***vehicles***. Note que, no ficheiro de teste, a classe genérica ***Garage*** é convertida em classe ordinária usando a classe ***Car***. Os carros devem ficar ordenados por ordem crescente de marca (***brand***) e, para uma mesma marca, por ordem decrescente de preço.

f) [**2.5 valores**] Implemente na classe genérica ***Garage*** o membro-função:

```
Vehicle* removeVehicle(std::string brand, std::string model);
```

Esta função remove do vector ***vehicles*** o veículo de marca ***brand*** e modelo ***model***, retornando o elemento removido. No caso de o veículo não existir, a função deve lançar a exceção ***NoSuchVehicleException*** (já fornecida).

g) [**2.5 valores**] Implemente na classe genérica ***Garage*** o membro-função:

```
float avgPrice(std::string brand) const;
```

Esta função calcula e retorna o preço médio dos veículos de marca ***brand***. No caso de não haver veículos dessa marca, a função deve lançar uma exceção ***NoSuchBrandException***, a qual deve conter um membro-dado que é o nome da marca e um membro função de acesso a esse membro-dado, chamado ***getBrand***.

h) [**2.5 valores**] Implemente na classe ***Car*** o operador **<<**:

```
std::ostream & operator<<(std::ostream & os, const Car &car);
```

Esta função deve imprimir em ***os*** o carro ***car***, da forma "marca modelo (preço)".