Parte 2 - 12.5 valores

Considere que se pretende criar uma empresa de aluguer de automóveis, que é assegurada pela classe `UberPOO`. Para o efeito a empresa possui uma frota de carros, sendo que neste momento existem dois tipos de carros disponíveis, os carros eléctricos (instâncias de `CarroElectrico`) e os carros a combustão (instâncias de `CarroCombustao`).

A `UberPOO` possui também um nome comercial, pelo qual é conhecida, e guarda também a informação dos seus clientes.

Para os carros a combustão, é guardada a informação relativa ao tamanho (em litros) do seu depósito e ao consumo (em litros) aos 100km. O preço por litro do combustível é também um valor que tem de ser guardado, sabendo-se que todos os carros a combustão utilizam o mesmo tipo de combustível. Para os carros eléctricos é guardada a dimensão da bateria (em Kwh) e o consumo em Kwh aos 100kms. O preço por kw é igual para todos os carros eléctricos.

O valor a pagar por dia pelo aluguer de um carro é função do seu consumo aos 100kms e do número de kms percorridos por dia. A classe `Carro` declara um método `public abstract double valorPorDia()`, que devolve esse valor.

Os utilizadores, instâncias de `Utilizador` possuem um número de cliente, que deve ser *sequencial e atribuído automaticamente*, o seu nome, morada e a informação dos seus registos de aluguer de carros.

Os registos de aluguer de carro por parte de um utilizador, instâncias de `Registo`, possuem a informação sobre o carro alugado, sobre a data de início e a data de fim do aluguer e sobre o número de kms contratados. Por simplificação assume-se que o número de kms diários é a divisão dos kms contratados pelo número de dias.

Considere os seguintes excertos de código:

```
public class UberPOO implements Serializable {
  private Map<String, Carro> carros;
  // outras variáveis de instância...
}


public class Registo implements Comparable<Registo>, Serializable {

  // outras variáveis
  // ...


  public double valorAPagar() {...} // determina o valor a pagar pelo
                                    // aluguer do carro tendo em conta
                                    // o seu consumo e o tempo
}


public abstract class Carro implements Serializable {
  private String matricula;

  ...
  public abstract double valorPorDia(); // determina o valor do custo
                                        // por dia do carro

}
```

Considere que a estratégia de associação entre `UberPOO` e os seus carros e utilizadores é de **composição**, mas tal já não é necessário na relação entre o `Registo` e o `Carro` a que se refere.

Assuma, para as perguntas seguintes, que os métodos usuais (get, set, equals, clone, hashcode, ...) estão disponíveis, a menos que sejam solicitados, e responda às questões:

Questão 6

1) Efectue a declaração das variáveis de instância de `UberPOO`, `Utilizador` e `Registo` e 2) **justifique brevemente** a escolha das estruturas de dados que faz. 3) Codifique também o método construtor `public UberPOO(Collection<Carro> carros)`, da classe `UberPOO`, que cria uma instância com os carros fornecidos por parâmetro.

☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1 *Reservado aos docentes*

Questão 7

Codifique o método `public void adicionaRegisto(LocalDate inicio, LocalDate fim, String matricula, int idUtilizador) throws...`, da classe `UberPOO`, que adiciona um registo de aluguer de um carro. Identifique todas as situações necessárias para que o estado do objecto fique coerente. Codifique também o construtor da classe `Registo` que utilizar. Se pretender poderá utilizar os métodos `isBefore` e `isAfter` da classe `LocalDate`.

☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1 *Reservado aos docentes*

Questão 8

Codifique o método `public Utilizador utilizadorMaisGastador()`, da classe `UberPOO`, que devolve o utilizador que tiver gasto mais dinheiro em aluguer de carros. Esse valor deverá ser obtido através da ordem natural dos utilizadores, que ordena por ordem decrescente de valor gasto os utilizadores e, caso exista mais do que um utilizador nesta circunstância, dá o utilizador que tenha efectuado menos registos de aluguer.

Codifique todas as alterações que terá de efectuar nas diferentes classes para que seja possível responder a este método.

☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1 *Reservado aos docentes*

Questão 9

Um carro híbrido é um carro que tem duas motorizações, um motor a combustão (com um depósito de combustível) e um motor eléctrico (e respectiva bateria). Do ponto de vista conceptual é um conceito que herda as definições dos carros a combustão e dos carros eléctricos. Crie a classe `CarroHibrido` e codifique o método da classe `UberPOO`, `public List<CarroHibrido> getHibridos()` que devolve os carros híbridos existentes na frota ordenados de forma crescente pelo tamanho da sua bateria.

☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1 *Reservado aos docentes*

Questão 10

Considere agora que existe uma outra classe `MasterUber` que comprou o negócio das empresas de frotas de carros de aluguer. Essa empresa funciona como um franchise (exemplo: uma `UberPOO` em Braga, outra em Viana, outra em Vila Real, etc.) e deixa cada uma das instâncias de `UberPOO` funcionar com o seu negócio de forma normal, mas existe uma alteração importante ao modelo de negócio. Os carros não pertencem às empresas `UberPOO` mas são adquiridos centralmente pela `MasterUber`.

1) Crie a classe `MasterUber`, identificando as suas variáveis de instância,

2) justifique quais as alterações que teria de fazer na classe `UberPOO` para que possa funcionar neste novo modelo e

3) codifique o método `public void forneceCarro(Carro c, String empresaLocal)` que permite passar para a `UberPOO` empresaLocal o carro fornecido como parâmetro (efectue todas as operações necessárias para se perceber como é que este método funcionaria).

☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1 *Reservado aos docentes*