Teste de Programação Orientada aos Objectos

MiEI e LCC
DI/UMinho

23/05/2019
Duração: **2h**

Leia o teste com muita atenção antes de começar.
Assuma que gets e sets estão disponíveis, salvo se forem explicitamente solicitados.

RESPONDA A CADA PARTE NAS FOLHAS FORNECIDAS.

PARTE I - 4 VALORES

1. Considere que se pretende desenvolver um programa para manipular polinómios (do tipo `Poly`). Uma vez que existem várias formas de representar polinómios, todas as possíveis implementações têm de disponibilizar pelo menos os seguintes métodos:

```
public void  addMonomio(double coef, int grau);
public double calcula(double x);
public Poly derivada();
```

Considere que se pretende implementar uma solução em que o polinómio é representado por uma lista de coeficientes e a posição na lista dá o expoente correspondente. Por exemplo, o polinómio $2.2x^3 + 1.3x - 4.7$ é representado pela lista $[-4.7, 1.3, 0, 2.2]$.

Apresente as definições necessárias para ter o tipo `Poly` e a classe `PolyAsList` (uma implementação de um polinómio baseado numa lista). Apresente também a(s) variável(is) de instância necessárias e apenas os métodos referidos acima. Note que a derivada do polinómio $2.2x^3 + 1.3x - 4.7$ é $6.6x^2 + 1.3$.

PARTE II - 6 VALORES

Considere que se criou um programa para registar alugueres de imóveis (à semelhança do que fazem algumas plataformas acessíveis por apps). Como entidades principais o programa considera os tipos `Imovel`, `Cliente` e `Aluguer`.

```
public abstract class Imovel implements Serializable {
  private String codImovel;
  private String morada;
  private String nifProprietario;
  private double area;
  private double precoBase;
  public abstract double precoDia();
  ...
}

public class Apartamento extends Imovel {
  private String andar;
  private double factorQualidade;
  ...
}

public class Moradia extends Imovel {
  private double areaPrivativa;
  private double areaExterior;
  ...
}

public class Bungalow extends Imovel {
  private double factorQualidade;
  private double espessuraParedes;
  ...
}

public class Cliente implements Serializable {
  private String nome;
  private String codCliente;
  private List<Aluguer> meusAlugueres;
  ...
}

public class Aluguer implements Serializable {
  private String codCliente;
  private String codImovel;
  private LocalDate dataInicio;
  private LocalDate dataFim;
}

 public class POOAirBnB implements Serializable {
   private Map<String, Imovel> imoveis;
   private Map<String, Cliente> clientes;
   ...
 }

}
```

O custo por dia de cada um dos diferentes tipos de imóveis está relacionado com o preço base definido para a entidade imóvel. Sabe-se que o preço por dia de um apartamento é função (multiplicativa) do factor de qualidade em relação ao preço base ($precoBase * factorQualidade$), de uma moradia é afectado por um rácio de 30% sobre a área privativa e 70% sobre a área exterior ($precoBase * (0,3 * areaPrivativa + 0,7 * areaExterior)$) e que o preço por dia de um bungalow é também calculado com base em iguais parcelas de factor de qualidade geral e da espessura das paredes ($precoBase * (0,5 * factorQualidade + 0,5 * espessuraParedes)$).

2. Codifique os seguintes métodos:

   (a) `public double precoDia()`, nas classes em que tal seja necessário.

   (b) `public Imovel getImovel(String codImovel) throws ImovelNaoExistente`, da classe `POOAirBnB`. Codifique a classe de excepção.

   (c) `public double valorTotalAluguerCliente(String codCliente)`, que determina o valor total dos alugueres pagos pelo cliente indicado no parâmetro. O método deve prever a situação do cliente não existir.

   (d) `public Map<String, Set<String>> clientesPorImovel()`, que devolve um map onde se associa a cada imóvel o conjunto dos clientes que alguma vez o alugaram.

PARTE III - 5 VALORES

3. Apresente as alterações necessárias à classe `Imóvel` para que a ordem natural das instâncias da classe seja a ordenação por preço.

4. Sabendo que a classe `POOAirBnB` possui o seguinte método (já implementado) para ordenar os seus imóveis:

```
private List<Imovel> ordenaImovel (Comparator<Imovel> c) { ... }
```

Apresente as alterações a efectuar para que `POOAirBnB` implemente a interface `OrdenaImoveis`:

```
public interface OrdenaImoveis {
        /**
         * Imoveis ordenados por tipo (Apartamento, Bungalow, Moradia)
         * e, dentro do tipo, por ordem crescente de área
         */
        public List<Imovel> imoveisPorTipoArea();
}
```

5. Pretende-se agora ter persistência de informação na classe `POOAirBnB`. Escreva os métodos necessários para ler e gravar instâncias dessa classe em ficheiro de objectos. O nome do ficheiro deverá ser passado como parâmetro.

6. Relembre o exercício de uma das fichas práticas em que se pedia para criar uma classe para representar grafos dirigidos. Para tal, foi decidido utilizar uma lista de adjacência que associa, a cada vértice (representado como sendo uma *String*), os vértices que podem ser visitados a partir dele. Foi já definida a seguinte estrutura base:

```
import java.util.Set;
import java.util.Map;
import java.util.HashMap;

public class Grafo {
    // variáveis de instância
    private Map<String, Set<String>> adj;
}
```

Complete a classe definindo:

(a) `void addArco(String vOrig, String vDest)`, método que adiciona um arco ao grafo. Note que todos os vértices do grafo devem ter uma entrada na lista de adjacência (que eventualmente poderá ser vazia).

(b) `int size()`, método que calcula o tamanho do grafo (o tamanho de um grafo com n vértices e m arcos é $n + m$).

(c) `boolean haCaminho(String vOrig, String vDest)`, método que determina se existe um caminho entre os dois vértices passados como parâmetro. Tenha em consideração que poderão existir ciclos no grafo.