Teste de Programação Orientada aos Objectos

MiEI e LCC
DI/UMinho

13/06/2019
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 PARA O EFEITO E MARCADAS COM O TÍTULO *Resolução*.

PARTE I - 4 VALORES

1. Considere que se pretende desenvolver um programa para manipular polinómios. Em teste anterior foi sugerido o seguinte interface:

```
public interface Poly {
  public void   addMonomio(int grau, double coeficiente);
  public double calcula(double x);
  public String toString();
}
```

ao qual as diferentes implementações de polinómios devem obedecer.

Considere agora que se pretende implementar uma solução em que o polinómio é representado por um *Map* em que a chave é o grau (tipo `Integer`) e o valor o coeficiente (`Double`).

Desenvolva a classe `PolyAsMap` que define esta implementação: apresente a(s) variável(is) de instância, o construtor por omissão `PolyAsMap()` e os métodos definidos no interface `Poly`. Tenha ainda em atenção que o método `toString` deve mostrar os polinómios apresentando os monómios por ordem decrescente. Por exemplo, o resultado do seguinte fragmento Java:

```
Poly p = new PolyAsMap();
p.addMonomio(0,5);
p.addMonomio(5,-10.0);
p.addMonomio(2,-4.0);
System.out.println(p.toString());
```

deverá ser a `String` `-10.0x^5 -4.0x^2+5.0x^0`.

PARTE II - 6 VALORES

Considere que se criou um programa para um ginásio em que os clientes tem acesso ao registo das actividades efectuadas no seu plano de treinos. Como entidades principais o programa considera os tipos `Actividade`, `Cliente` e `Exercício`.

```
public abstract class Actividade implements Serializable {
  private String designacao;
  private double caloriasPorUnidadeTreino; // calorias
                                           // por unidade de treino

  public abstract double caloriasGastas(); //o consumo de
            // calorias depende de cada actividade específica
  ...
}

public class Corrida extends Actividade implements ComDistancia {
  private double kmsPercorridos;
  private double elevacao;
  private double velocidade;
  ...
}

public class Eliptica extends Actividade implements ComDistancia {
  private double kmsPercorridos;
  private double nivelEsforco;
  private double minutos;
  ...
}

public class Abdominais extends Actividade {
  private int numeroRepeticoes;
  private String tipoExercicio;
  ...
}

public class Cliente implements Serializable {
  private String nome;
  private String codCliente;
  private Map<LocalDate,List<Exercicio>> meusExercicios;
  ...
}

public class Exercicio implements Serializable {
  private Actividade actividade;
  private String professor;
  private String codExercicio;
  ...
}

public interface ComDistancia {
  public double getKmsPercorridos();
}

public class GinasioPOO implements Serializable {
```

```
    private Map<String, Cliente> clientes;
    ...
}
```

O cálculo de consumo de calorias numa corrida corresponde à multiplicação dos kms percorridos pelas calorias por unidade de treino. Por cada metro de elevação acrescenta-se mais 25% de uma unidade de exercício. Na elíptica o consumo calórico é a multiplicação da distância percorrida pelas calorias de uma unidade de exercício. Por cada minuto de exercício acrescenta-se mais 20% até ao nível de esforço 4 e acima desse nível o acréscimo é de 5% por cada nível. Nos abdominais o consumo de calorias é directamente proporcional ao número de repetições.

2. Codifique os seguintes métodos:

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

   (b) `public double valorTotalCaloriasGastas(String codCliente) throws ...`, que determina o valor total das calorias gastas pelo cliente indicado no parâmetro. O método deve prever a situação do cliente não existir.

   (c) `public double totalKmsCliente(String codCliente, LocalDate dataExercicio) throws ClienteNaoExiste, ExercicioInexistente`

   da classe `GinasioPOO`. Este método devolve o número de kms que o cliente percorreu nas diversas actividades dos exercícios efectuados na data passada como parâmetro. Codifique uma das classes de excepção.

   (d) `public boolean existeProfessor(String prof)`, que determina se o professor passado como parâmetro alguma vez trabalhou no ginásio.

Parte III - 5 valores

Considere ainda que em relação à pergunta anterior são solicitadas as respostas às questões seguintes:

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

4. Codifique o método `public Map<String, List<Exercicio>> exerciciosPorProf()` da classe `Cliente`, que dá a lista de exercícios de cada treinador para um cliente.

5. Codifique o método `public String professorMaisExigente()`, da classe `GinasioPOO`, que determina o professor que mais calorias fez os alunos gastar.

Parte IV - 5 valores

6. Considere que se está a construir a classe `GrowingArrayOfActividade`. A classe deve internamente gerir um array de `Actividade`, garantindo que este tem sempre espaço disponível para adicionar novos objectos.

   Considere ainda que já foram escritas as seguintes declarações:

```
public class GrowingArrayOfActividade implements Serializable,
          Comparable {
        // variáveis de instância
        private Actividade[] lista;
        private int tamanho;
}
```

   Sabendo que o `GrowingArrayOfActividade` obedece às conhecidas regras do encapsulamento, codifique agora:

   (a) `public Actividade get(int indice)` , que devolve a `Actividade` que está na posição indicada (lança uma excepção se a posição for inválida).

   (b) `public void add(Actividade a)`, que adiciona uma `Actividade` à lista, garantindo que existe espaço disponível.

   (c) `public static GrowingArrayOfActividade leGrowingArrayOfActividade(String fich) throws ..` que lê um `GrowingArrayOfActividade` do ficheiro de objectos indicado como parâmetro.