Nome:_____ **Nº:**_____ **Curso:**_____ (**E**)

Exame de Programação Orientada aos Objectos **(E)**

MiEI e LCC - DI/UMinho

28/06/2021
Duração: **2h**

Leia o teste com muita atenção antes de começar
Assuma que gets e sets estão disponíveis, salvo se forem explicitamente solicitados.
Na Parte I não existem erros sintácticos propositados.

Parte I - 7.5 valores

1. Considere que lhe pediram para fazer uma aplicação para a gestão do campeonato de formação de hóquei em patins - CHP. O CHP é constituído por clubes, que podem ter várias equipas inscritas nos diversos escalões e cada equipa tem um capitão de equipa e atletas.

   Considerando as definições na Figura 1, qual seria a implementação correcta, numa estratégia de composição de objetos, para o método,

   ```
   public Equipa getEquipa(String idClube, String idEquipa)
          throws ClubeNaoExisteException, EquipaNaoExisteException
   ```

   que enviado a uma instância de CHP devolve a Equipa correspondente (caso esta exista – ver próxima folha):

Figura 1. Gestão de Campeonatos de Hóquei em patins

```
public class Equipa {
    private String id;
    private String escalao;
    private Pessoa capitao;
    private Set<Pessoa> atletas;
...
}
public class Clube {
    private String nome;
    private Map<String,Equipa> equipas;
...
}

public class CHP {
    private Map<String, Clube> clubes;
...
}
```

◯
```
public Equipa getEquipa(String idClube, String idEquipa)
    throws ClubeNaoExisteException, EquipaNaoExisteException {
    for(String c : this.clubes.keySet()){
        if(c.equals(idClube)){
            for(Equipa e: c.getEquipas().values()){
                if (e.getId().equals(idEquipa)){
                    return e;
                }
            }
        }
    }
    throw new EquipaNaoExisteException();
}
```

◯
```
public Equipa getEquipa(String idClube, String idEquipa)
    throws ClubeNaoExisteException, EquipaNaoExisteException{
    for(Clube c : this.clubes.entrySet()){
        if(c.getNome().equals(idClube)){
            for(Equipa e: c.getEquipas().values()){
                if (e.getId().equals(idEquipa)){
                    return e.clone();
                }
            }
        }else{
            throw new ClubeNaoExisteException();
        }
    }
    throw new EquipaNaoExisteException();
}
```

◯
```
public Equipa getEquipa(String idClube, String idEquipa)
    throws ClubeNaoExisteException, EquipaNaoExisteException{
    Equipa res = null;
    Clube c = this.clubes.get(idClube);
    if (c!= null){
        Map<String,Equipa > equipas = c.getEquipas();
        if (equipas.containsKey(idEquipa)){
            res = equipas.get(idEquipa).clone() ;
        }
    }else{
        throw new ClubeNaoExisteException()
    }
    if (res == null){
        throw new EquipaNaoExisteException();
    }
    return res;
}
```

◯
```
public Equipa getEquipa(String idClube, String idEquipa)
    throws ClubeNaoExisteException, EquipaNaoExisteException{
    for(EntrySet<String, Clube> c : this.clubes.entrySet()){
        if(c.getValue().getNome().equals(idClube)){
            for(Equipa e: c.getEquipas().values()){
                if (e.getId().equals(idEquipa)){ return e; }
            }
        }
    }
    throw new EquipaNaoExisteException();
}
```

Nome:_____ **Nº:**_____ **Curso:**_____ (**E**)

2. Considere que lhe pediram para fazer uma aplicação para a gestão do campeonato de formação de hóquei em patins - CHP. O CHP é constituída por equipas, que podem ter várias equipas inscritas nos diversos escalões e cada equipa tem um capitão de equipa e atletas.

   Considerendo as definições na Figura 1, qual seria a implementação correcta, numa estratégia de composição de objetos, para o método

   ```
   List<Equipa> getEquipas(String idClube, String escalao)
       throws ClubeNaoExisteException
   ```

   que enviado a uma instância de CHP devolve a Lista de Equipas de um clube que são de um determinado escalão

   ◯
   ```
   public List<Equipa> getEquipas(String idClube, String escalao) throws ClubeNaoExisteException {
       boolean clubeExiste = false;
       List<Equipa> res = new List<>();
       for(Equipa e: this.equipas.values()){
           if (e.getNome().equals(idClube)){
               clubeExiste = true;
               for (Equipa b : e.getEquipas().values()){
                   if(b.getEscalao().equals(escalao)){
                       res.add(b);
                   }
               }
       }
       if(!clubeExiste){
           throw new ClubeNaoExisteException();
       }
       return res;
   }
   ```

   ◯
   ```
   public List<Equipa> getEquipas(String idClube, String escalao) throws ClubeNaoExisteException {
       List<Equipa> res = new List<>();
       for(Equipa e: this.equipas.values()){
           if (e.getNome().equals(idClube)){
               for (Equipa b : e.getEquipas().values()){
                   if(b.getEscalao().equals(escalao)){
                       res.add(b.clone());
                   }
               }
           }else{
               throw new ClubeNaoExisteException();
           }
       }
       return res;
   }
   ```

   ◯
   ```
   public List<Equipa> getEquipas(String idClube, String escalao) throws ClubeNaoExisteException {
       List<Equipa> res = new ArrayList<>();
       for( Map.Entry<String, Clube> c : this.clubes.entrySet())
           if(c.equals(idClube)){
               for(Equipa e: c.getValue().getEquipas().values()){
                   if(e.getEscalao().equals(escalao)){
                       res.add(e.clone());
                   }
               }
           }
       return res;
   }
   ```

◯
```
public List<Equipa> getEquipas(String idClube, String escalao) throws ClubeNaoExisteException {
    if (this.clubes.containsKey(idClube)){
        return this.clubes.get(idClube).getEquipas().values()
            .stream()
            .filter(e -> e.getEscalao().equals(escalao))
            .map(Equipa::clone)
            .collect(Collectors.toList());
    }else{
        throw new ClubeNaoExisteException();
    }
}
```

3. Considere as seguintes definições:

```
public interface Empregado {
    public String getEmpregador();
}

public class Aluno {
    ...
    public Aluno() { ... }
    public boolean epocaEspecial() { return false; }
}

public class AlunoTE extends Aluno implements Empregado {
    ...
    public AlunoTE() { ... }
    public boolean epocaEspecial() { return true; }
    public String getEmpregador() { return "Externo"; }
}

public class Funcionario implements Empregado {
    ...
    public Funcionario() { ... }
    public String getEmpregador() { return "UMinho"; }
}
```

Considere ainda que estão disponíveis as seguintes definições:

```
public List<String> getEstatus1(List<Empregado> l) {
    return l.stream().filter(e -> e instanceof Aluno).map(a -> a.getEmpregador())
                     .collect(Collectors.toList());
}

public List<String> getEstatus2(List<Aluno> l) {
    return l.stream().filter(a -> a instanceof Empregado).map(e -> e.getEmpregador())
                     .collect(Collectors.toList());
}

public List<String> getEstatus3(List<Empregado> l) {
    return l.stream().map(e -> (Aluno) e).map(a -> a.getEmpregador())
                     .collect(Collectors.toList());
}
```

Sabendo que irão ser utilizadas as seguintes listas:

```
List<Empregado> lemp = new ArrayList<>();
lemp.add(new Funcionario());
lemp.add(new AlunoTE());
lemp.add(new Funcionario());
```

```
lemp.add(new AlunoTE());
lemp.add(new Funcionario());
lemp.add(new AlunoTE());

List<Aluno> lal = new ArrayList<>();
lal.add(new AlunoTE());
lal.add(new Aluno());
lal.add(new AlunoTE());
lal.add(new Aluno());
lal.add(new AlunoTE());
```

para cada afirmação, assinale a opção que a torna verdadeira:

a) A expressão ◯ `getEstatus1(lemp);` | ◯ `getEstatus2(lal);` | ◯ `getEstatus3(lemp);` | ◯ `Nenhuma das anteriores` gera um erro de compilação.

b) A expressão ◯ `getEstatus1(lemp);` | ◯ `getEstatus2(lal);` | ◯ `getEstatus3(lemp);` | ◯ `Nenhuma das anteriores` gera a lista `["UMinho","UMinho","UMinho"]`.

c) A expressão ◯ `getEstatus1(lemp);` | ◯ `getEstatus2(lal);` | ◯ `getEstatus3(lemp);` | ◯ `Nenhuma das anteriores` gera um erro de execução.

d) A expressão ◯ `getEstatus1(lemp);` | ◯ `getEstatus2(lal);` | ◯ `getEstatus3(lemp);` | ◯ `Nenhuma das anteriores` gera a lista `["UMinho","Externo","UMinho","Externo","UMinho","Externo"]`.

4. Considere o seguinte tipo de dados para representar as turmas de um curso. Cada turma, indexada pelo seu nome, possui um conjunto de alunos, indexados pelo seu número:

```
private Map<String, Map<Integer, Aluno>> turmas;
```

Considere o seguinte método, que irá indicar a turma com maior média de notas, considerando apenas os alunos com nota média maior do que 10. Caso várias turmas tenham a mesma média, deve-se selecionar a que tiver o maior número de alunos (independentemente da nota). Assuma que o método `getMedia` da classe `Aluno` existe e calcula a média de um aluno.

```
public String melhorTurma() {
    Comparator<Map.Entry<String, Map<Integer, Aluno>>> comp = (a, b) -> {
        double va = a.getValue().values().stream().filter(al -> al.getMedia() > 10.0)
                .mapToDouble(Aluno::getMedia).average().orElse(0.0);
        double vb = b.getValue().values().stream().filter(al -> al.getMedia() > 10.0)
                .mapToDouble(Aluno::getMedia).average().orElse(0.0);
        int sizea = a.getValue().size();
        int sizeb = b.getValue().size();
        if (va==vb) return sizea - sizeb;
        else return (int) (va - vb);
    };
    return turmas.entrySet().stream().sorted(comp)
                .map(e -> e.getKey()).findFirst().orElse("N/A");
}
```

Selecione a alínea correta:

◯ O método está corretamente implementado, mas na última linha usa-se um `orElse` desnecessário; bastava terminar a linha com `findFirst`.

◯ O método está corretamente implementado, mas poderá causar um erro de execução se houver 0 turmas registadas na variável `turmas`.

◯ O método está corretamente implementado, porém numa estratégia de composição seria necessário acrescentar invocações apropriadas ao método `clone`.

- ◯ O método está incorretamente implementado, a ordem imposta pelo `Comparator` é inversa à que deveria ser usada para obter resultados corretos.
- ◯ O método está incorretamente implementado pois, como se pretende ter uma ordenação com dois critérios, deve-se usar dois `Comparators`.
- ◯ O método está incorretamente implementado visto que é necessário usar o método `compareTo` para produzir o resultado do `Comparator`.

5. Considere o código da Figura 1. Considere ainda que os métodos `Set<Pessoa> getAtletas()` e `setAtletas(Set<Pessoa> s)`, da classe `Equipa`, foram implementados do seguinte modo:

```
public Set<Pessoa> getAtletas() {
    return new HashSet<>(atletas);
}

public void setAtletas(Set<Pessoa> s) {
    atletas = s.clone();
}
```

Assinale a afirmação verdadeira:

- ◯ Os métodos não estão estão correctamente implementados, não sendo consistentes no tratamento do encapsulamento.
- ◯ Os métodos estão correctamente implementados e respeitam o encapsulamento, se a relação entre `Equipa` e `Pessoa` for de agregação.
- ◯ Os métodos estão correctamente implementados e respeitam o encapsulamento, se a relação entre `Equipa` e `Pessoa` for de composição.
- ◯ Não é possível dizer, apenas a partir da sua implementação, se estes métodos estão, ou não, correctamente implementados, no que respeita à noção de encapsulamento.

Nome:_____ **Nº:**_____ **Curso:**_____ (**E**)

Parte II - 12.5 valores

Considere que se pretende ter um sistema que permita a disponibilização de canais de vídeos curtos (uma espécie de TikTok) que são criados por um determinado *influencer*. Um canal possui um identificador (um nome), o nome do seu criador e tem associado uma lista de vídeos que vão sendo disponibilizados.

A entidade vídeo foi definida da seguinte forma:

```
public class Video {
    private String nome;
    private double duracao;
    private int classificacao; //dada pelos seus subscritores (valor de 0..100)
    private List<Byte> conteudo; //corresponde ao conteudo armazenado sob
                                 //a forma de lista de bytes
    private int numeroVezesRep; //qts vezes e' que video foi visto
    private LocalDateTime ultimaVez; //regista quando foi reproduzido
                                     //pela ultima vez

   ...
   ...
}
```

Considere também que o sistema completo a desenvolver `TikTokPOO` guarda, além dos canais existentes e dos vídeos destes, informação relativa aos utilizadores (subscritores) do sistema. Para cada utilizador guarda-se o seu identificador (que neste sistema é a String do seu email), o seu nome e a informação dos canais que tem subscritos.

Resolva os seguintes exercícios:

Nome:_____ **Nº:**_____ **Curso:**_____ (**E**)

6. Para a classe `Video` codifique o seu construtor de cópia e o método `public boolean equals(Object o)`.
   Resposta:

7. Faça as alterações necessárias na classe `Video` e codifique o método que implementa a ordem natural de `Video`, em que se ordena os elementos por ordem crescente do número de vezes que o vídeo foi reproduzido e em caso de igualdade nesse parâmetro de comparação deve ordenar de forma a garantir que aparecem primeiro os vídeos que foram tocados mais recentemente. O método `isBefore` da classe `LocalDateTime` aceita um `LocalDateTime` como parâmetro e devolve um boolean como resultado.

Nome:_____ **Nº:**_____ **Curso:**_____ (**E**)

8. Efectue a declaração das classes `Canal`, `Utilizador` e `TikTokPOO`, identificando apenas as variáveis existentes e codificando o método `public List<Video> getVideos(String nomeCanal)`, da classe `TikTokPOO`, que dado um identificador de canal devolve, numa lógica de composição, uma lista com os vídeos disponíveis para esse mesmo canal.

Nome:_____ **Nº:**_____ **Curso:**_____ (**E**)

9. Desenhe o Diagrama de Classes da solução `TikTokPOO`. Considere que não necessita de colocar os métodos `get` e `set`.

   Resposta:

10. Codifique o método `public void remove(String nomeC) throws...`, da classe `TikTokPOO`, que remove do sistema o canal identificado. Esta remoção não poderá ser possível se o canal não existir registado no sistema ou se o mesmo canal tiver utilizadores que actualmente o estejam a subscrever. Indique na assinatura do método as excepções de que necessitar (não necessita de as codificar).

    Resposta:

Nome:_____ **Nº:**_____ **Curso:**_____ (**E**)

11. Desenvolva o método `public Map<Integer,List<Video>> videosPorClassf()`, da classe `TikTokPOO`, que associa a cada valor de classificação a lista dos vídeos, de todos os canais, com essa mesma classificação. A ordenação da lista, para uma mesma classificação, deverá ser por ordem crescente de duração do vídeo.

12. Considere agora que a classe `Video` deverá implementar a interface `Instagrammable`, definida como

```
public interface Instagrammable {
  public void playInInstagram();
}
```

Tendo em consideração que existirá um objecto chamado `System.instagram`, que tem o mesmo comportamento do `System.out` e que transforma em output do Instagram o conteúdo em bytes do vídeo, altere a classe `Video` de modo a que implemente `Instagrammable`.

13. Considere agora que se criaram novos tipos de conteúdo que passam pela disponibilização de vídeos com conteúdo vídeo e háptico (ao reproduzir o conteúdo o dispositivo vibra e apresenta force feedback). Pretende-se criar-se o `HapticVideo`, que para além do vídeo também possui uma lista de `Byte` que representa as instruções de movimento do dispositivo. Codifique a classe `HapticVideo`, apresentando a sua declaração e variáveis, o construtor parametrizado e a codificação do método `playInInstagram`. Por simplificação, assuma que, para reproduzir estes conteúdos, pode primeiro tratar do vídeo e depois da camada de comandos de movimento hápticos, e que o `System.instagram` também sabe reproduzir movimento físico do dispositivo.

14. Codifique o método `public void gravaInfoVideos(String fich)`, que grava em ficheiro de objectos os vídeos `Instagrammable` que existem no `TikTokPOO`.

    Tenha em atenção as possíveis excepções resultantes do uso de ficheiros.