TESTES E EXAMES

POO

Nome:_____ **Nº:**_____ **Curso:**_____

Teste de Programação Orientada aos Objectos **(A)**

MiEI e LCC - DI/UMinho

21/05/2021
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 as seguintes definições das classes Jogador e Equipa:

```
public abstract class Jogador {
        private String numero;
        private String nome;
        private List<Integer> golos; // lista com os golos em cada jogo
        //...
        public double mediaGolos() {
            //...
        }
        //...
}

public class Equipa {
        private String clube;
        private Map<String,Jogador> jogadores;
        //...
}
```

clube inf. sobre jogador (que estão na classe Jogador)

Sabendo que o método `mediaGolos(String numJogador)` da classe `Equipa` (que calcula a média de golos do jogador indicado) não deverá produzir um valor para a média caso o número de jogador indicado não exista, indique qual das seguintes implementações do método considera correcta:

◯
```
public double mediaGolos(String num) {
        double m = 0;
        Jogador jog = this.jogadores.get(num);
        if (jog != null) {
                m = jog.mediaGolos();
        }
        return m;
}
```

→ se o jogador não existir produz média 0 e não queremos

→ vai buscar o jogador

→ acede ao método que está na classe jogador

◯
```
public double mediaGolos(String num) throws JogadorNaoExisteException {
        return this.jogadores.get(num).mediaGolos();
}
```

→ se o jogador não existir, vamos obter NullPointerException

Nome:_____ **Nº:**_____ **Curso:**_____

●
```
public double mediaGolos(String num) throws JogadorNaoExisteException {
        Jogador jog = this.jogadores.get(num);
        if (jog == null) {
              throw new JogadorNaoExisteException(num);
        }
        return jog.mediaGolos();
   }
```

○
```
public double mediaGolos(String num) {
        Jogador jog = this.jogadores.get(num);
        if (jog == null) {
              throw new JogadorNaoExisteException(num);
        }
        return jog.mediaGolos();
   }
```

Não temos nenhuma exceção declarada na assinatura

○ Nenhuma das implementações é válida pois o método `mediaGolos()` na classe `Jogador` não é abstracto. → classes abstratas podem implementar métodos.

2. Considere que a classe Convocatoria foi declarada do seguinte modo:

```
public abstract class Convocatoria {
       private String codJogo;
       private LocalDateTime data;
       private List<Jogador> convocados;
    ...
}
```

Prestando atenção ao encapsulamento (em particular às noções de composição e agregação), indique quais dos seguintes pares de métodos estão correctamente implementados (**atenção**: indique **todos** os pares que considera correctos):

●
```
public void setConvocados(List<Jogador> conv) {
        this.convocados = conv;
   }

   public List<Jogador> getConvocados() {
        return this.convocados;
   }
```

Trabalha diretamente com referências nos getters e nos setters.

○
```
public void setConvocados(List<Jogador> conv) {
        this.convocados = new ArrayList(conv);
   }

   public List<Jogador> getConvocados() {
        return this.convocados.stream()
                              .map(Jogador::clone)
                              .collect(Collectors.toList());
   }
```

○
```
public void setConvocados(List<Jogador> conv) {
        this.convocados = conv.stream()
                              .map(Jogador::clone)
                              .collect(Collectors.toList());
   }

   public List<Jogador> getConvocados() {
        return new ArrayList(this.convocados);
   }
```

Nome:_____ **Nº:**_____ **Curso:**_____

●
```
public void setConvocados(List<Jogador> conv) {
        this.convocados = conv.stream()
                               .map(Jogador::clone)
                               .collect(Collectors.toList());
    }

    public List<Jogador> getConvocados() {
        List<Jogador> conv = new ArrayList();

        for(Jogador j: this.convocados) {
            conv.add(j.clone());
        }
        return conv;
    }
```

Faz clone nos getters e setters

○
```
public void setConvocados(List<Jogador> conv) {
        this.convocados = new ArrayList(conv);
    }

    public List<Jogador> getConvocados() {
        return this.convocados.stream().collect(Collectors.toList());
    }
```

3. Considere as seguintes definições:

```
public interface I {
    public int miA();
    public int miB();
}

public class A implements I {
    ...
    public A() { .. }
    public int m1() { ... }
    public int m2() { ... }
    public int miA() { ... }
}

public class B extends A implements I {
    ...
    public B() { ... }
    public int miB() { ... }
}
```

→ A tem que implementar todos os métodos da interface I

→ A é superclasse portanto B herda de A.

Qual das seguintes afirmações é válida:

○ A definição da interface I está errada pois os seus métodos têm que ser abstractos. F

○ A classe A não está correcta pois não pode definir o método miA() da interface I. F

○ A classe B não está correcta pois não define a implementação do método miA(). F

● A seguinte expressão é válida: `I i = new B();`

4. Relembre a classe DriveIt desenvolvida nas aulas, em que se armazenam Veículos (indexados pela matrícula) na seguinte estrutura:

→ marca

```
private Map<String, Veiculo> viaturas;
```

↳ matrícula

Selecione o método que correctamente devolve os veículos de uma dada marca, ordenados alfabeticamente por matrícula:

Nome:_____ **Nº:**_____ **Curso:**_____

○
```
public Iterator<Veiculo> veiculosDaMarca(String marca) {
    Iterator<Veiculo> r = this.viaturas.values().iterator();
    while(r.hasNext()) {
        Veiculo v = r.next();
        if (!v.getMarca().equals(marca)) r.remove();
    }
    r.sort((v1, v2) -> v1.getMatricula().compareTo(v2.getMatricula()));
    return r;
}
```
→ vai buscar a marca
→ remove os que não são daquela marca
→ Não admite método sort

○
```
public List<Veiculo> veiculosDaMarca(String marca){
    Comparator<Veiculo> comp =
        (v1, v2) -> v1.getMatricula().compareTo(v2.getMatricula()));
    return this.viaturas.stream()
                .map(Veiculo::clone)
                .filter(v -> !v.getMarca().equals(marca))
                .sorted(comp)
                .collect(Collectors.toList());
}
```
→ Map não tem método stream
X

●
```
public Set<Veiculo> veiculosDaMarca(String marca){
    TreeSet<Veiculo> r = new TreeSet<>(
              (v1, v2) -> v1.getMatricula().compareTo(v2.getMatricula()));
    for (Veiculo v : this.viaturas.values()) {
        if (v.getMarca().equals(marca)) r.add(v.clone());
    }
    return r;
}
```

○
```
public Set<Veiculo> veiculosDaMarca(String marca){
    List<Veiculo> r = new List<>();
    for (Map.Entry<String, Veiculo> e : this.viaturas.entrySet()) {
        Veiculo v = e.getValue();
        if (v.getMarca().equals(marca)) r.add(v.clone());
        r.sort((v1, v2) -> v1.getMatricula().compareTo(v2.getMatricula()));
    }
    return r;
}
```
→ tínhamos que retornar um Set<...>
List não extende Set

5. Considere a seguinte estrutura de dados usada para representar uma colecção de videos numa plataforma de streaming. A cada autor está associada uma colecção com todos os seus vídeos, que por sua vez é indexada por um código único do vídeo.

```
public Map<String, Map<String, Video>> videos;
```
Nome autor

Considerando tudo o que aprendeu sobre o tratamento de erros, selecione o método que mais correctamente implementa a obtenção de um vídeo, dado o nome do utilizador e o código do vídeo:

○
```
public Video getVideo(String user, String codVideo) {
    return this.videos.get(user).get(codVideo).clone();
}
```
→ Não trata de erro nenhum

○
```
public Video getVideo(String user, String codVideo) {
    if (!this.videos.containsKey(user))
          throw new Exception("User " + user + " Inexistente");
    if (!this.videos.get(user).containsKey(codVideo))
```
→ Não está declarado na assinatura

Nome:_____ **Nº:**_____**Curso:**_____

```
            throw new Exception("Video " + codVideo + " Inexistente");
        return this.videos.get(user).get(codVideo).clone();
    }
```

○
```
    public Video getVideo(String user, String codVideo) {
        Video v;
        try {
            v = this.videos.get(user).get(codVideo).clone();

        } catch (Exception e) {
            v = null;
        }
        return v;
    }
```

●
```
    public Video getVideo(String user, String codVideo)
                            throws UserInexistenteException,
                                   VideoInexistenteException {
        if (!this.videos.containsKey(user))
                throw new UserInexistenteException("User "+user+" Inexistente");
        if (!this.videos.get(user).containsKey(codVideo))
            throw new VideoInexistenteException("Video "+codVideo+" Inexistente");
        return this.videos.get(user).get(codVideo).clone();
    }
```

Nome:_____ **Nº:**_____**Curso:**_____

Parte II - 12.5 valores

Considere o exercício dos Smart Devices que foi resolvido numa das aulas práticas. De acordo com o exercício existem actualmente dois tipos de `SmartDevice`, as colunas de som (as `SmartSpeaker`) e as lâmpadas (as `SmartBulb`), com as definições que se apresentam:

```
public class SmartDevice {

    private String id;
    private boolean on;
    private double consumoPorHora;
    private LocalDateTime inicio;

    ...
    public SmartDevice( String id, double consumoPorHora) {
       this.id = id;
       this.on = false;
       this.consumoPorHora = consumoPorHora;
    }


    // devolve o consumo desde o inicio
    public double totalConsumo() {...}
    ...

    //liga o device. Se for a primeira vez inicializa o tempo de inicio
    public void turnOn() {
        this.on = true;
        if (this.inicio == null)
           this.inicio = LocalDateTime.now();
    }
}


public class SmartBulb extends SmartDevice {
    public static final int WARM = 2;
    public static final int NEUTRAL = 1;
    public static final int COLD = 0;
    private int tone;

    public SmartBulb(String id, int tone, double consumoPorHora) {
       super(id, consumoPorHora);
       this.tone = tone;
    }

    ...
    public void setTone(int t) {
        if (t>WARM) this.tone = WARM;
        else if (t<COLD) this.tone = COLD;
        else this.tone = t;
    }

    public int getTone() {
        return this.tone;
    }
}

public class SmartSpeaker extends SmartDevice {
    public static final int MAX = 20; //volume maximo da coluna
```

```
    private int volume;
    private String channel;

    public SmartSpeaker(String id, String channel, double consumoPorHora) {
       super(id, consumoPorHora);
       this.channel = channel;
       this.volume = 10;
    }
   ...
   ...
}
```

Considere que se pretende implementar uma classe `CasaInteligente` que guarda a informação dos dispositivos existentes na casa e regista também para cada divisão da casa (identificadas por Strings como "Sala Jantar", "Quarto", "Escritório", etc.) os dispositivos que nelas se encontram.

Resolva os seguintes exercícios:

Nome:_____ **Nº:**_____ **Curso:**_____

6. Efectue a declaração das variáveis de instância de `CasaInteligente` e codifique o construtor que recebe uma coleção de `SmartDevice` e que assume que estamos numa estratégia de composição, `public CasaInteligente(Collection<SmartDevice> devices)`.

usar clone

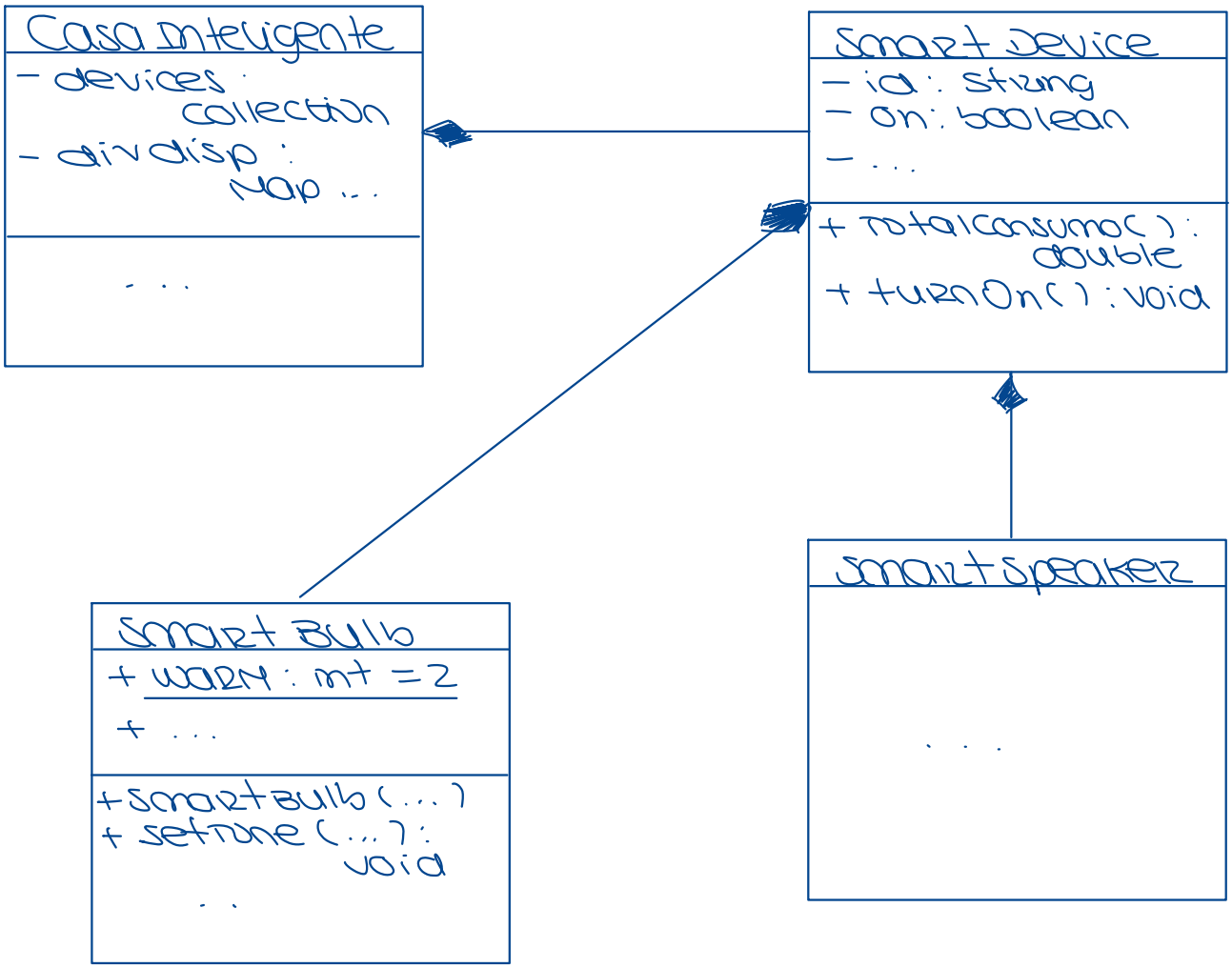
Resposta:

```
public class CasaInteligente {
    private HashMap<String, Collection<SmartDevice>> divDisp;
    private Collection<SmartDevice> devices;

    public CasaInteligente(Collection<SmartDevice> devices) {
        this.devices = devices.stream().
                       map(SmartDevice::clone).
                       toList();
    }
}
```

Nome:_____ **Nº:**_____**Curso:**_____

7. Desenhe o Diagrama de Classes da solução.

   Resposta:



Nome:_____ **Nº:**_____ **Curso:**_____

8. Codifique o método `public void remove(String id) throws...`, que remove completamente do sistema o dispositivo cujo identificador é passado por parâmetro (codifique também a excepção).

   Resposta:

```
public void remove(String id) throws DispNaoExiste{

if (!this.devices.contains(id))
    throw new DispNaoExiste(id);
SmartDevice d = this.device.stream().
                filter(s -> s.id().toString()
                .equals(id));
this.device.remove(d);
this.divdisp.values().forEach(c ->
                c.remove(d));
}
```

Nome:_____ **Nº:**_____ **Curso:**_____

9. Codifique o método `public Iterator<SmartDevice> devicesPorConsumoCrescente()`, que devolve um iterador com ordenação crescente por consumo (deve codificar e utilizar a ordem natural dos `SmartDevice`).

   Resposta:

```
public Iterator<SmartDevice> devicesCres(){
    return this.devices.stream()
                       .map(SD::clone)
                       sorted(Comparator.
                       comparingDouble(SD::total_
                       consumo)).iterator();
}
```

10. Forneça a implementação para o método `public String divisaoMaisEconomica()`, que determina a divisão da casa que apresenta o menor consumo. Se duas divisões apresentarem o mesmo consumo então deverá ser devolvida a divisão cuja designação tem o maior valor alfabético.

    Resposta:

Nome:_____ **Nº:**_____**Curso:**_____

11. Considere que se pretende acrescentar um novo tipo de lâmpada que permita regular a intensidade da sua luz. A `SmartBulbDimmable` quando é ligada pela primeira vez fica com a intensidade da luz a 50% e gasta também metade do consumo anunciado. Crie esta classe, identificando as variáveis de instância necessárias, o construtor parametrizado e todos os métodos herdados que necessitam de ser reescritos.

    Resposta:

Nome:_____ **Nº:**_____**Curso:**_____

12. Relembre a matéria das aulas teóricas e forneça uma implementação para o método da classe `CasaInteligente` que permita fazer alterações ao estado interno das `SmartBulbDimmable`. Esse método deve ter a assinatura `public void alteraInfo(Consumer<SmartBulbDimmable> bd)`. Forneça também a implementação para o `Consumer<SmartBulbDimmable>` que altera a luminosidade de uma `SmartBulbDimmable` para 25% do seu valor actual.

    Resposta:

Nome:_____ **Nº:**_____**Curso:**_____

13. Codifique o método `public boolean apenasNumaDivisao()`, que dá true se não existir nenhum `SmartDevice` registado em mais do que uma divisão da casa.

    Resposta:

Nome:_____ **Nº:**_____**Curso:**_____

14. Codifique o método que grava num ficheiro de objectos, cujo nome é fornecido no parâmetro, todas os `SmartSpeaker` existentes na casa.

    ```
    public boolean gravaEmFichObjectos(String fich) throws FileNotFoundException, IOException
    ```

    Resposta:

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

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

MiEI e LCC - DI/UMinho

14/06/2021
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();
}
```

→ Não usa clone

→ Quando o clube não existe obtemos NullPointerException

○
```
public Equipa getEquipa(String idClube, String idEquipa)
    throws ClubeNaoExisteException, EquipaNaoExisteException{
    return this.clubes.get(idClube).get(idEquipa).clone();
}
```
✗

○
```
public Equipa getEquipa(String idClube, String idEquipa)
    throws ClubeNaoExisteException, EquipaNaoExisteException{
    return this.clubes.values().stream()
           .filter(eq -> eq.getNome().equals(idClube))
           .findFirst().get()
           .getEquipas().values().stream()
           .filter(e-> e.getId().equals(idEquipa))
           .findFirst().get().clone();
}
```
✗

●
```
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;
}
```

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{
    if (this.clubes.containsKey(idClube)){
        return this.clubes.get(idClube).getEquipas()
                .values().stream()
                .filter( e->e.getEscalao().equals(escalao))
                .collect(Collectors.toList());
    }else{
        throw new ClubeNaoExisteException();
    }
    return new ArrayList();
}
```

○
```
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(idEquipa)){
            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<>();
    if (this.clubes.containsKey(idClube)){
        for(Equipa e: this.clubes.get(idClube).getEquipas().values()){
            if(e.getEscalao().equals(escalao)){
                res.add(e.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);
                }
            }
        }
    return res;
}
```

→ não trata do erro

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<Boolean> getEEstatus1(List<Empregado> l) {
    return l.stream().filter(e -> e instanceof Aluno).map(a -> a.epocaEspecial())
                     .collect(Collectors.toList());
}

public List<Boolean> getEEstatus2(List<Aluno> l) {
    return l.stream().filter(a -> a instanceof Empregado).map(e -> e.epocaEspecial())
                     .collect(Collectors.toList());
}

public List<Boolean> getEEstatus3(List<Empregado> l) {
    return l.stream().map(e -> (Aluno) e).map(a -> a.epocaEspecial())
                     .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());

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

para cada afirmação assinale, **caso exista**, a opção que a torna verdadeira (se nenhuma opção for válida, não assinale nada):

a) A expressão ● `getEEstatus1(lemp);` | ○ `getEEstatus2(lal);` | ○ `getEEstatus3(lemp);` gera um erro de compilação.

b) A expressão ○ `getEEstatus1(lemp);` | ● `getEEstatus2(lal);` | ○ `getEEstatus3(lemp);` gera a lista `[true,true]`.

c) A expressão ○ `getEEstatus1(lemp);` | ○ `getEEstatus2(lal);` | ● `getEEstatus3(lemp);` gera um erro de execução.

d) A expressão ○ `getEEstatus1(lemp);` | ○ `getEEstatus2(lal);` | ○ `getEEstatus3(lemp);` gera a lista `[true,false,true,false]`.

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 sizeb - sizea;
        else return (int) (vb - va);
    };
    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 as turmas sem alunos são consideradas com média 0.0.

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

○ O método está incorretamente implementado, pois não se podem construir streams sobre o resultado do método `entrySet`.

○ 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 atletas.clone().stream().collect(Collectors.toSet());
}

public void setAtletas(Set<Pessoa> s) {
    atletas = s.stream().map(Pessoa::clone).collect(Collectors.toSet());
}
```

Assinale a afirmação verdadeira:

- ◯ 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.
- ● Os métodos não estão estão correctamente implementados, não sendo consistentes no tratamento do encapsulamento.
- ◯ 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.

Parte II - 12.5 valores

Considere que se pretende ter um sistema que implemente uma serviço de disponibilização de podcasts. Um podcast possui um identificador (um nome) e tem associada uma lista de episódios que foram disponibilizados.

A entidade episódio de um podcast foi definida da seguinte forma:

```
public class Episodio {
    private String nome;
    private double duracao;
    private int classificacao; //dada pelos seus ouvintes (valor de 0..100)
    private List<String> conteudo; //corresponde ao texto que e' dito
                                   //quando se reproduz o episodio
    private int numeroVezesTocada; //qts vezes e' que o podcast foi ouvido
    private LocalDateTime ultimaVez; //regista quando foi reproduzido
                                     //ultima vez

   ...
   ...
}
```

Considere também que o sistema completo a desenvolver `SpotifyPOO` guarda, além dos podcasts existentes e dos episódios destes, informação relativa aos utilizadores 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 podcasts que tem subscritos.

Resolva os seguintes exercícios:

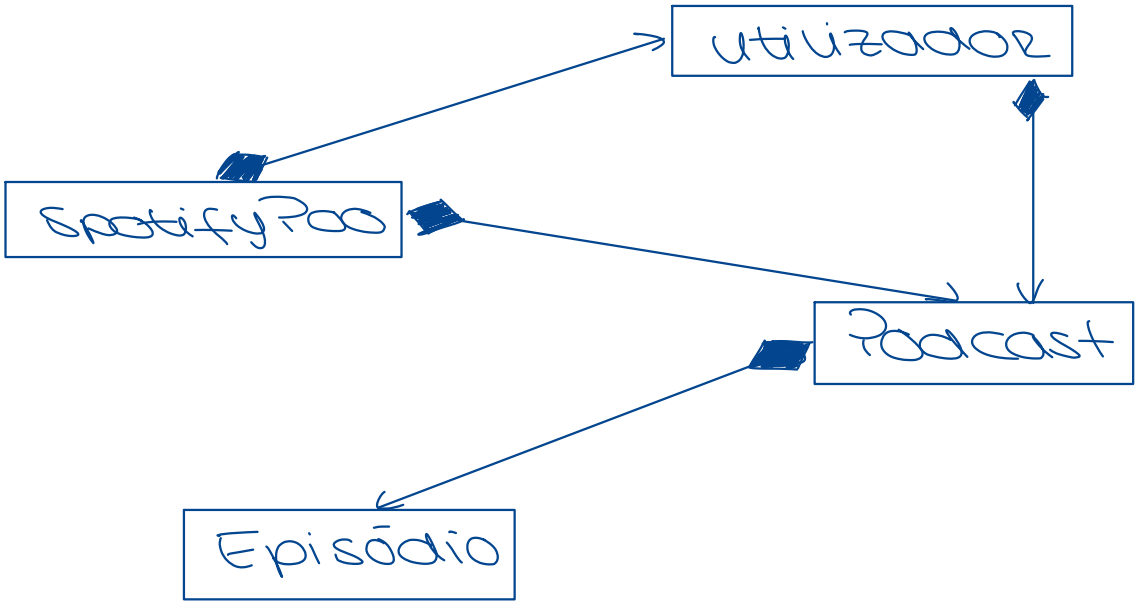
Nome:_____ **Nº:**_____ **Curso:**_____ (**A**)

6. Efectue a declaração das classes `Podcast`, `Utilizador` e `SpotifyPOO`, identificando apenas as variáveis existentes e codificando o método `public List<Episodio> getEpisodios(String nomePodcast)`, da classe `SpotifyPOO`, que dado um identificador de podcast devolve, numa lógica de composição, uma lista com os episódios disponíveis para esse podcast.

```
public class Podcast {
    private String nome;
    private List<Episodio> ep;
}

public SpotifyPoo {
    private HashMap<String, Podcast> pods;

    private static class Utilizador {
        private String email;
        private String nome;
        private Set<Podcast> pod;
    }

    private HashMap<String, Utilizador> users;

    public List<Episodio> setEp (String nomeP) {
        Podcast p = pods.get(nomeP);
        if (p == null)
            return new ArrayList<Episodio>();
        return p.getEp().stream().map(Episódio::
                                      clone).toList();
    }
```

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

   Resposta:



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

   Resposta:

```
public void remove(String nomeP) throws
                                PodNaoExiste,
                                PodSubscrito {
    Podcast p = pods.get(nomeP);
    if (p == null)
        throw new PodNaoExiste;
    if(users.values()
            .stream()
            .map(Utilizador::getPodcasts)
            .flatMap(Collection::stream)
            .anyMatch(n -> n.equals(nomeP)))
        throw new PodSubscrito;
    pods.remove(nomeP);
}
```

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

9. Codifique o método `public Episodio getEpisodioMaisLongo(String u)`, da classe `SpotifyPOO`, que para o utilizador passado por parâmetro, devolve o episódio mais longo de entre os podcasts que esse utilizador tem subscritos.

```
public Episodio getEpisodioMaisLongo(String u){
    return users.get(u)
                .getPodcasts()
                .stream()
                .flatMap(pName -> pods.
                                  .get(pName)
                                  .ep.stream())
                .max(Comparator.
                     .comparingDouble
                     (Episodio::getDuracao))
                .get();
}
```

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

10. Desenvolva o método `public Map<Integer,List<Episodio>> episodiosPorClassf()`, da classe `SpotifyPOO`, que associa a cada valor de classificação a lista dos episódios, de todos os podcasts, com essa mesma classificação.

```
public Map<Integer,List<Episodio>>
                    episodiosPorClassf() {
Map<Integer,List<Episodio>> m = new
                    HashMap<>();

pods.values()
    .stream()

    (...)
```

11. Considere agora que a classe `Episodio` deverá implementar a interface `Playable`, definida como

```
public interface Playable {
  public void play();
}
```

Tendo em consideração que existirá um objecto chamado `System.media`, que tem o mesmo comportamento do `System.out` e que transforma em som o conteúdo em texto do episódio, altere a classe `Episodio` de modo a que implemente `Playable`.

12. Considere agora que se criaram novos tipos de conteúdo que passam pela disponibilização de episódios com som e vídeo. Pretende criar-se o `EpisodioVideo`, que para além do audio também possui uma lista de `Byte` que representa o conteúdo visual. Codifique a classe `EpisodioVideo`, apresentando a sua declaração e variáveis, o construtor parametrizado e a codificação do método `play`. Por simplificação assuma que, para reproduzir estes conteúdos, pode primeiro tratar do vídeo e depois do som, e que o `System.media` também sabe reproduzir vídeo.

13. Considere que é possível efectuar a reprodução de um podcast por parte de um Utilizador, através do método `public void playEpisodio(String idPodCast, String nomeEpisodio) throws AlreadyPlayingException` da classe `Utilizador`. A excepção é lançada quando esse utilizador já está no momento a reproduzir um episódio. Considere que se pretende criar agora a noção de `UtilizadorPremium`, que é um utilizador que, enquanto reproduz um episódio, possui a capacidade de colocar os outros episódos que pretende reproduzir numa lista de espera.

    Codifique a classe `UtilizadorPremium` com as suas variáveis de instância e a implementação do método `playEpisodio`.

14. Codifique o método `public void gravaInfoEpisodiosParaTocarMaisTarde(String fich)`, que grava em ficheiro de texto os episódios dos `UtilizadorPremium` que estão na fila de espera para serem reproduzidos. A informação deve ficar guardada com o formato

```
Nome Utilizador
Id do Episodio - duracao
Id do Episodio - duracao
...
...
Nome Utilizador
Id do Episodio - duracao
...
```

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

+1/1/60+

Nome: .....

Curso: .....

Número:

☐0	☐0	☐0	☐0	☐0
☐1	☐1	☐1	☐1	☐1
☐2	☐2	☐2	☐2	☐2
☐3	☐3	☐3	☐3	☐3
☐4	☐4	☐4	☐4	☐4
☐5	☐5	☐5	☐5	☐5
☐6	☐6	☐6	☐6	☐6
☐7	☐7	☐7	☐7	☐7
☐8	☐8	☐8	☐8	☐8
☐9	☐9	☐9	☐9	☐9

Programação Orientada aos Objetos

Teste

LEI/LCC, Universidade do Minho

20 de Maio, 2022 – Duração: 2h

Instruções: *Não se esqueça de preencher o nome, curso e número. Indique o número à direita, assinalando um dígito por coluna.*

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

Parte 1 - 7.5 valores

Parte 2 - 12.5 valores

Considere as seguintes definições de classes de uma aplicação que implementa uma loja de livros digitais. A aplicação da `LivrosDigitais` possui a informação dos utilizadores que nela estão registados e para cada utilizador é guardada a informação respeitante à colecção de livros que adquiriu. A informação dos livros indica as páginas lidas e por ler e consequentemente em qualquer altura sabe-se sempre qual é o sítio do livro que se está a ler.

Considere os seguintes excertos de código:

```
public class Livro implements Comparable<Livro>, Serializable {
  public String codISBN;         //código ISBN do livro
  private String nomeLivro;
  private String autor;
  private String editora;
  private List<Pagina> pagLidas; // páginas já lidas
  private List<Pagina> pagPorLer; //páginas ainda por ler.
                                  //o primeiro elemento é a página a ser lida no momento
  ....

  /* método que devolve a página com o número indicado */
  public Pagina devolvePag(int numPag) throws PagInexistenteException {
   Pagina res = null;
    int numLidas = this.pagLidas.size(); //número de páginas lidas
    int porLer = this.pagPorLer.size();

    if (numPag > numLidas+porLer)
      throw new PagInexistenteException(numLidas);
    if (numPag <= numLidas )
       res = this.pagLidas.get(numPag -1);
    else
       res = this.pagPorLer.get(numPag-numLidas -1);

    return res.clone();
  }

}

public class Pagina implements Comparable<Pagina>, Serializable {
private List<String> texto;

public Pagina() {
  this.texto = new ArrayList<>();
}
...
/* método que devolve uma formatação do texto */
public String reproduzPagina() {...}
}
```

```
public class PaginaComAudio extends Pagina implements Comparable<PaginaComAudio>,
                                                       Serializable {
   private String narrador;
   private List<Byte> som;
   ...

   public PaginaComAudio(List<String> texto, String narrador, List<Byte> audio) {
        super(texto);
    this.narrador = narrador;
    this.audio = new ArrayList<>(audio);
   }
   /* método que devolve uma formatação do texto e audio */
    public String reproduzPagina() {...}

}


public class Utilizador implements Serializable {

  private String numUser;
  private String nomeUser;
  private LocalDate dataAdesao;  // data de adesão do utilizador à aplicação
  ...


}
```

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

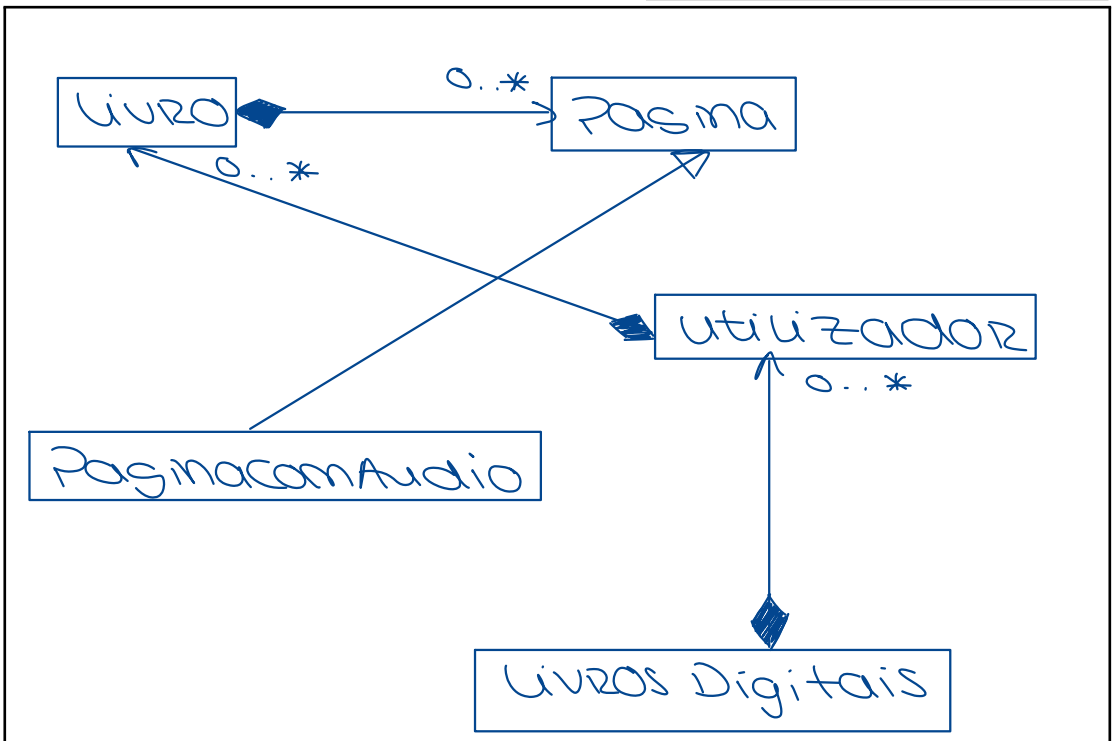
Questão 1

Efectue a declaração das variáveis de instância de `LivrosDigitais` e complete a declaração das variáveis de instância de `Utilizador`. Codifique o construtor parametrizado de `Utilizador` que recebe uma série de instâncias de `Livro` e que assume que estamos numa estratégia de composição, `public Utilizador(String numUser, String nomeUser, Iterator<Livro> livros)`.

☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1

```
public class LivrosDigitais {
   private Set<Utilizador> users;
}

public class Utilizador {
   private String numUser;
   private String nomeUser;
   private LocalDate dataAdesao;
   private Set<Livros> colecao;

   public Utilizador (String num,
         String nome, LocalDate d,
         Iterator<Livro> l) {
      this.numUser = num;
      this.nomeUser = nome;
      this.colecao = new TreeSet<>();
      while (livros.hasNext()) {
         this.colecao.add(livro.next().
                                clone());
      {
```

Questão 2 Desenhe o Diagrama de Classes. ☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1



Questão 3 Codifique o método `public void avancaPags(String codISBN, int n) throws...`, da classe `Utilizador`, que avança `n` páginas na leitura desse livro.

☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1

```
public void avancaPags(String codISBN,
  int n) throws PagInexistente{
    Livro livro = null;
    for(Livro l: this.colecao){
      if(l.getCodISBN().equals(codISBN)){
        livro = l.clone();
        while(n>=0){
          Pagina p = l.getPagPorLer().
                                    get(0);
          livro.getPagLidas().add(p);
          livro.getPagPorLer().remove(0);
        }   n--;
      }
}
    if(livro == null){
    {  throw new PagInexistente();
}
```

Questão 4 Codifique o método `public Livro livroMaisLido()`, que determina o livro mais lido. Em caso de existir mais do que um livro candidato deverá ser devolvido aquele que seja alfabeticamente maior. O livro mais lido é aquele que registar mais páginas lidas.

☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1

```
public Livro livroMaisLido(){
    Map<Livro, Integer> map = new HashMap<>();

    for(Utilizador user : this.users){
        for(Livro l : user.getColecao()){
            if(!map.contains(l)){
                map.put(l, 0);
            {
            int n = l.getPagLidas().size() +
                                    map.get(l);
            map.replace(l, n);
        }
    }
    Map.Entry<Livro, Integer> max = null;
    for(Map.Entry<Livro, Integer> entry: map.
                                    entrySet()){
        if(max == null || entry.getValue() >
                                    max.getValue()){
            max = entry;
        {
        else if(entry.getValue() == max.
                                    getValue()){
            if(entry.getKey().getNomeLivro()
                .compareTo(max.getKey()
                .getNomeLivro()){
                max = entry;
            {
        }
    }
    return max.getKey();
}
```

Questão 5

Codifique o método `public Map<String,List<Livro» livrosPorEditora()`, da classe `LivrosDigitais`, que para cada nome de editora associa a lista dos livros dessa mesma editora.

☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1

```
public Map <String, List <Livro >> LivrosPorEditora(){
    Map < String, List < Livro>> edit = new HashMap
                                        <>();
    for(Utilizador user : this. users ){
        for(Livro L : user getColecao( ){
            String editora = L. getEditora( );
            if(! edit. contains (editora)){
                edit put (editora, new ArrayList<>( ),
            }
            edit. get (editora) . add (L. clone( ));
        }
    }
    return edit;
}
```

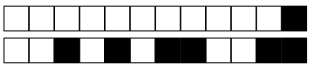
Questão 6

Considere que pretendemos adicionar aos livros um novo tipo de páginas que possuam texto, audio e vídeo. Temos, no entanto, uma implementação já existente da classe `PaginaMultimedia`, com a seguinte declaração:

```
public class PaginaMultimedia {
  private List<String> texto;
  private List<Byte> audio;
  private List<Byte> video;

  public PaginaMultimedia(List<String> texto, List<Byte> audio, List<Byte> video) {
    this.texto = texto;
    this.audio = audio;
    this.video = video;
  }

/**
 * método que devolve uma formatação do texto, audio e vídeo.
 * Está devidamente implementado.
 */
public String fazPagina() {
...
}

}
```

Esta é uma classe já antiga e não podemos alterar o código dela mas queremos aproveitar o comportamento que ela apresenta e especialmente o resultado do método `fazPagina` que devolve uma representação de uma página em que está colocado o texto, o audio e o vídeo. Queremos evitar ter que desenvolver uma classe de raíz e implementar novamente o método que faz a reprodução de uma página com texto, audio e vídeo.

Diga como é que podemos compatibilizar esta classe com o resto das classes existentes, mostre o que é preciso alterar ou criar, e codifique o método `public List<String> reproduzLivros()`, da classe `Utilizador`, que fornece a reprodução de todos os livros existentes.



☐0 ☐.2 ☐.4 ☐.5 ☐.6 ☐.8 ☐1

Para o seu exame, use preferencialmente documentos compilados do auto-multiple-choice.