FCT/UNL Mestrado Integrado em Engenharia Informática

Programação Orientada pelos Objectos, 2018/2019

Teste 1 **27 de Abril, 2019**

Instruções:

- Antes de começar a resolver, leia o enunciado do princípio até ao fim.
  - As interfaces e classes dos grupos I e II têm mais métodos do que os que deverá implementar na resolução deste teste. Em cada grupo, tenha o cuidado de ver com muita atenção quais os métodos que deve implementar, para não desperdiçar o seu tempo a implementar métodos que não lhe são pedidos.
  - Disponibilizamos a descrição sumária de todos os métodos, incluindo os que não tem de implementar, para que os possa usar na sua resolução.
- Pode usar caneta ou lápis.
- Não é permitido consultar quaisquer elementos para além deste enunciado.
- Por favor, responda a grupos diferentes em folhas separadas. Verifique que todas as suas folhas de resposta estão identificadas.
- **Este teste tem a duração máxima de 120 minutos.**

Nos grupos I e II terá que **implementar parcialmente** algumas classes necessárias à construção de um programa que implementa a gestão de pacotes de canais de um operador de televisão por cabo. Vamos considerar que alguns dos canais oferecidos nos pacotes de televisão são *Premium*. Um canal funciona como uma colecção de programas que ocorrem em determinadas datas. Cada canal é caracterizado pelo seu nome, categoria, país de origem e uma colecção de programas transmitidos por esse canal. Um canal premium é um canal semelhante aos restantes, mas com um valor mensal associado a ser pago para a ele ter acesso e uma forma diferente de determinar a sua categoria. Cada programa (*Show*) é caracterizado por um nome, uma colecção de datas de exibição (note que um mesmo programa pode ser exibido mais que uma vez, incluindo no mesmo dia, pelo que podem existir repetições na colecção de datas em que um programa é exibido num determinado canal.

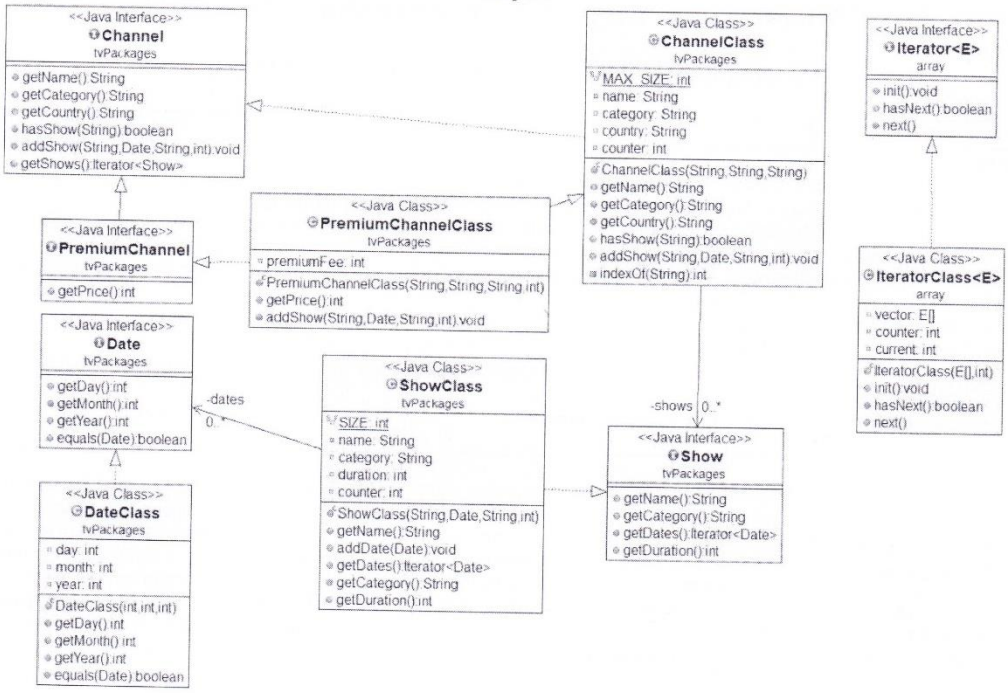
Note que apenas é permitido acrescentar métodos privados às classes `ChannelClass`, `PremiumChannelClass`, e `ShowClass` (no Grupo I), `AbstractPackageClass`, `BasicPackageClass`, `GoldPackageClass` e `PlatinumPackageClass` (no grupo II). **Não pode alterar ou acrescentar variáveis de instância ou métodos não privados a estas classes, nem modificar as restantes classes e interfaces do enunciado). Assuma que todas as colecções usadas neste enunciado são preenchidas por ordem de inserção na colecção.**

No grupo III terá de desenhar um diagrama de classes e interfaces, dada uma especificação em língua natural de um domínio para uma aplicação a desenvolver em Java. Note que, **no grupo III, não deve implementar nada**.

Finalmente, deixamos uma chamada de atenção adicional para a notação usada nos diagramas de classe dos grupos I e II, que pode não ser tão perceptível a preto e branco:

- Os atributos públicos são anotados com cículos. Quase todos os métodos, com excepção de `indexOf()` em `ChannelClass` e `resize()`, em `ArrayClass` (ambos privados) e do construtor da classe `AbstractPackageClass` (protegido) são públicos.
- Os atributos protegidos são anotados com um losango. Apenas temos o construtor da classe `AbstractPackageClass` como protegido.
- Os atributos privados são anotados com um quadrado. Todas as constantes e variáveis de instância neste enunciado são privadas. `indexOf()` em `ChannelClass` e `resize()`, em `ArrayClass` são os únicos métodos privados.
- Além disso, note que optámos pela notação em que variáveis que representam referências para elementos de uma outra classe aparecem denotadas por associações. Por exemplo, na figura do grupo 1, `shows` é uma variável de instância privada definida na classe `ChannelClass`. O modificador de cardinalidade `0..*` indica que se trata de uma colecção (por exemplo, um vector). O mesmo raciocínio se aplica a `dates`, na classe `ShowClass` (novamente, um vector). Na figura do grupo 2, a variável `channels` é uma variável de instância privada de `AbstractPackageClass`. O modificador de cardinalidade `0..1` indica que esta variável guarda, no máximo, uma referência para um `Array` (pode guardar `0`, se tiver como valor `null`). A declaração destas 3 variáveis é apresentada detalhadamente em cada exercício.

Grupo I



Considere a interface `Channel`. As operações definidas nesta interface são as seguintes:

- `getName()` – devolve o nome do canal.
- `getCategory()` – devolve a categoria dominante de programas no canal.
- `getCountry()` – devolve o país de origem do canal.
- `hasShow(String name)` – devolve `true` se o programa com o nome e categoria passados como argumento existe neste canal, ou `false`, caso contrário.
- `addShow(String showName, Date date, String category, int duration)` adiciona um novo programa no canal.
- `getShows()` – devolve um iterador de programas exibidos no canal.

A classe `ChannelClass` implementa a interface `Channel`. Nesta classe, existe uma constante (`MAX_SIZE`) com a dimensão, por omissão, do vector `shows`. Além desta constante, existem 5 variáveis de instância, todas privadas: `name` (nome do canal), `category` (categoria do canal), `country` (país de origem do canal), `shows` (declarado como um vector de programas privado - `private Show[] shows`) e `counter` (o contador de programas guardados no vector `shows`). Os métodos implementados nesta classe, além dos especificados na respectiva interface, são:

- O construtor `ChannelClass(String name, String country, String defaultCategory)`, que inicializa um canal com um nome, país de origem e categoria por omissão, além do vector vazio e o contador a zero.
- O método privado `indexOf(String name)` que devolve o índice em que o programa de nome `name` ocorre no vector `shows`, ou `-1`, se o programa não existir no vector `shows`.

A interface `PremiumChannel` especializa `Channel`, acrescentando um novo método:

- `getPrice()` – devolve o valor da mensalidade deste canal premium.

A classe `PremiumChannelClass` especializa `ChannelClass` e implementa a interface `PremiumChannel`. Os métodos novos ou redefinidos nesta classe são:

- O construtor `PremiumChannelClass(String name, String country, String defaultCategory, int monthlyFee)`, que inicializa o canal como um canal "normal", mas agora com um preço mensal (`monthlyFee`).
- O método de consulta `getPrice()`, especificado na interface `PremiumChannel`.
- `addShow(String showName, Date date, String category, int duration)` adiciona um novo programa no canal. Se necessário, actualiza a categoria dominante no canal. A categoria dominante num canal é aquela que ocupa mais tempo na programação. Por exemplo, se a categoria "notícias" ocupar 100 horas, a categoria de "documentários ocupar 80 horas e a categoria de "ficção" ocupar 40 horas, o canal é considerado de "notícias"; se houver empate entre várias categorias, a categoria que atingiu o topo das categorias há mais tempo é considerada a dominante.

A interface `Show` representa um programa de televisão, que pode ser apresentado várias vezes no mesmo canal. Para todos os efeitos práticos, considera-se que cada programa apenas é transmitido num canal (mesmo se for o mesmo episódio de uma série de ficção, é considerado um programa distinto se passar noutro canal, ou o mesmo programa se passar no mesmo canal. Por exemplo, se a série "A teoria do big bang" for transmitida por mais que um canal, um dado episódio é considerado um programa diferente em cada canal em que for exibido. Assim, as datas de exibição coleccionadas por um determinado programa apenas entram em conta com as datas desse programa nesse canal. A interface `Show` tem os seguintes métodos:

- `getName()` – devolve o nome do programa.
- `getCategory()` – devolve a categoria do programa.
- `getDuration()` – devolve a duração do programa.
- `getDates()` – devolve um iterador para as várias datas de exibição de um determinado programa. Note que o mesmo programa pode ser exibido mais que uma vez num dia. Por exemplo, há séries muito populares cujos episódios repetem em horários distintos, no mesmo dia. As datas de exibição são apresentadas por ordem de inserção na colecção de datas.
- `addDate(Date date)` – adiciona uma nova data de exibição para o programa.

A classe `ShowClass` implementa a interface `Show`. Define uma constante privada de nome `MAX_SIZE`, com o tamanho, por omissão, do vector de Datas em que um programa repete. Define ainda várias variáveis de instância privadas: `name` (nome do programa), `category` (categoria do programa), `duration` (duração do programa), `dates` (um vector de `Date` – `private Date[] dates`) e `counter` (um contador de datas inseridas no vector `dates`). Além dos métodos especificados na interface `Show`, `ShowClass` define ainda:

- Um construtor `ShowClass(String name, Date date, String category, int duration)`

A interface `Date` e a classe `DateClass` são aqui apresentadas apenas para completude, armazenando e permitindo comparar datas. A comparação de datas deve ser feita, se necessário, recorrendo ao método `equals(Date other)` que compara o objecto com a data passada como argumento e devolve `true` se corresponderem à mesma data, ou `false` caso contrário.

A interface `Iterator<E>` e a classe `IteratorClass<E>` são as apresentadas nas aulas teóricas. O construtor de `IteratorClass<E>` recebe um vector de elementos de um determinado tipo (e.g. `Date`) e um inteiro com o número de elementos nesse vector. `hasNext()` devolve `true` se existir mais algum elemento a visitar, ou `false`, caso contrário. `next()` devolve um elemento da colecção, do tipo que tiver sido escolhido no momento da instanciação (e.g. `Date`).

Sabendo tudo isto, <u>implemente apenas os seguintes métodos</u>:

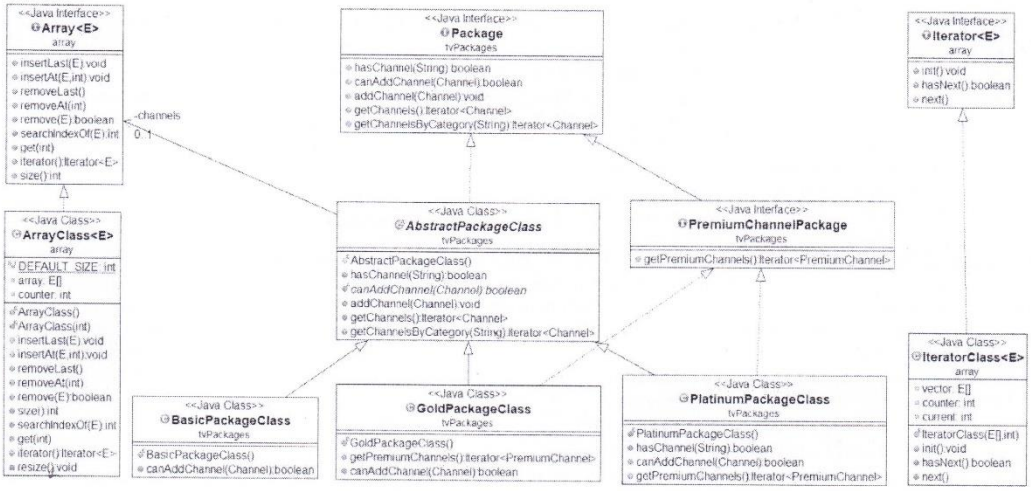
a) O construtor da classe `PremiumChannelClass(String name, String country, String defaultCategory)`

b) `void addShow(Show show)` da classe `ChannelClass`.

c) `void addShow(Show show)` da classe `PremiumChannelClass`. Sugestão: repare que no final deste método, ou se mantém a classificação do canal premium que já estava, ou a introdução de um novo programa faz com que a classificação do canal mude (para a categoria do programa que acabou de adicionar). Se achar conveniente, pode criar um ou mais métodos auxiliares, para simplificar a implementação deste método `addShow`, respeitando as restrições indicadas na introdução do enunciado. Não pode criar variáveis de instância adicionais.

d) `Iterator<Date> dates()` da classe `ShowClass`.

Grupo II



Considere a interface `Package`. As operações oferecidas nesta interface são as seguintes:

- `hasChannel(String channelName)` – devolve `true` se o pacote incluir um canal com o nome dado, `false` caso contrário.
- `canAddChannel(Channel channel)` – decide se determinado canal pode, ou não, ser adicionado a um pacote de canais.
- `addChannel(Channel channel)` – adiciona um canal à lista de canais do pacote. Esta operação tem sempre sucesso, mas assume como pré-condição que a resposta ao método `canAddChannel(Channel channel)` seja `true`.
- `getChannels()` – devolve um iterador para todos os canais do pacote.
- `getChannelsByCategory(String category)` – devolve um iterador para todos os canais de determinada categoria (e.g. "`noticias`").

A interface `PremiumChannelPackage` especializa a interface `Package` adicionando o método:

- `getPremiumChannels()` – devolve um iterador para todos os canais Premium oferecidos pelo pacote.

A classe `AbstractPackageClass` conta apenas com uma variável de instância

- `private Array<Channel> channels` – esta variável contém, por ordem de inserção, os canais que pertencem a cada pacote. No construtor, ela deve ser inicializada com uma colecção vazia. Esta lista pode incluir canais *normais* e *premium* (todos misturados na mesma lista, portanto).

A classe implementa todas as operações da interface `Package`, com excepção de:

- `canAddChannel(Channel channel)` – as regras para adicionar pacotes variam consoante o tipo de pacote, sendo a sua implementação delegada para as classes concretas, por esse motivo. Repare que é através deste método que é possível implementar as regras de negócio relativas a que canais de cada tipo podem ser acrescentados a um pacote.

A classe `BasicPackageClass` especializa `AbstractPackageClass`. Num pacote básico, apenas é possível adicionar canais que ainda não façam parte do pacote e sejam "normais" (ou seja, que não sejam canais Premium). O construtor não acrescenta nada ao de `AbstractPackageClass`.

A classe `GoldPackageClass` especializa `AbstractPackageClass` e implementa `PremiumChannelPackage`. Num pacote gold apenas é possível adicionar canais *normais* e, no máximo um canal *premium*. Assim, se já existir algum canal *premium*, não é possível acrescentar um novo canal *premium*. O iterador devolvido por `getPremiumChannels` terá, neste caso, no máximo um elemento para iterar.

A classe `PlatinumPackageClass` especializa `AbstractPackageClass` e implementa `PremiumChannelPackage`. Num pacote platina, é sempre possível adicionar novos canais, desde que ainda não façam parte do pacote, independentemente de serem normais ou *premium*. O iterador devolvido por `getPremiumChannels` devolverá, neste caso, todos os canais *premium* incluídos neste pacote.

As interfaces `Array<E>` e `Iterator<E>` e as classes que as implementam são as estudadas nas aulas teóricas.

Sabendo tudo isto, implemente apenas os seguintes métodos:

a) `boolean canAddChannel(Channel channel)` da classe `BasicPackageClass`.

b) `boolean canAddChannel(Channel channel)` da classe ~~`PremiumPackageClass`~~ Gold.

c) `Iterator<Channel> getChannelsByCategory(String category)` da classe `AbstractPackageClass`.

d) `Iterator<Channel> getPremiumChannels()` da classe `PlatinumPackageClass`.

e) Suponha que no seu programa principal tem de imprimir os canais *premium* de um determinado pacote de canais. A assinatura do método é a seguinte:

```
private static void printPremiumChannels(Package package)
```

O seu método `printPremiumChannels` recebe um pacote `package` e deve imprimir:
- `"Nao tem canais premium"`, se o pacote em questão não tiver nenhum canal *premium*.
- Um canal por linha, com o nome e o custo do canal, se existirem canais *premium* no pacote.
- No final, deve imprimir ainda o custo total dos canais *premium*, ou 0, se não houver nenhum canal *premium*.

Grupo III

Pretende-se desenvolver um sistema de gestão de clientes para um operador hoteleiro, como se descreve de seguida.

O operador tem uma cadeia de unidades de hotelaria. Cada unidade de hotelaria é caracterizada por um nome, uma localização, uma classificação (entre 1 e 5 estrelas), e um conjunto de habitações para aluguer. Existem vários tipos de unidades hoteleiras, tais como *apart-hotel* (uma unidade hoteleira constituída exclusivamente por *apartamentos*, que pode ter, ou não, *piscinas*), *hotel* (uma unidade hoteleira constituída exclusivamente por *quartos* que pode ter, ou não, *piscinas*, um *spa* e um *restaurante*), *resort* (que tem, simultâneamente, *apartamentos*, *quartos*, uma ou mais *piscinas*, um ou mais *restaurantes*, um *spa* e um *serviço de organização de festas temáticas*.

Actualmente, as habitações consideradas podem ser apenas de dois tipos: *apartamento*, ou *quarto*. No entanto, é possível que no futuro a cadeia venha a considerar outros tipos de habitações (e.g. bungallows, veleiros onde se possa pernoitar, etc.). Por agora, são mesmo apenas os *apartamentos* e *quartos*.

Existem quatro tipos de aluguer que podem ser feitos. Apenas *estadia*, *estadia com pequeno almoço*, *estadia em regime de meia pensão*, e *estadia em regime de pensão completa*. No caso do apart-hotel, por ser uma unidade hoteleira sem restaurante, apenas a estadia simples é possível. Para os restantes tipos de unidade hoteleira, todos os tipos de pensão são aceitáveis.

O sistema deve permitir: adicionar uma unidade hoteleira (dada a referência para um objecto que representa uma unidade hoteleira); remover uma unidade hoteleira, dado o seu nome; reservar uma habitação dado o seu tipo (apartamento, ou quarto), a unidade hoteleira e as datas de entrada e saída e o tipo de estadia (*simples*, com *pequeno almoço*, *meia pensão*, ou *pensão completa*); listar as habitações livres de uma unidade hoteleira numa determinada data; listar as unidades hoteleiras com *spa*.

Apresente a sua proposta de modelação para o programa, através de um diagrama de classes e interfaces, tendo em atenção que deve incluir na sua resposta:

- A interface de topo com a qual o programa principal irá interagir. Esta interface deve ser completamente especificada. Sugestão: apresente a lista de operações desta interface à parte, e não no diagrama, para gerir mais facilmente o espaço na sua resposta.
- As variáveis de instância da classe que implementa a interface de topo.
- Para as restantes componentes do diagrama, isto é, para as interfaces e classes que não a interface de topo, omita a indicação das operações e das variáveis de instância.

Nota 1: Não é necessário implementar nenhuma das operações.

Nota 2: Apresente de forma distinta classes e interfaces, identifique caso existam classes abstractas e indique a visibilidade das variáveis de instância.

Nota 3: Para efeitos de clareza da apresentação represente as relações com uma seta etiquetada, por exemplo $A \xrightarrow{extends} B$ significa A *extends* B (analogamente para *implements*, eventualmente com linha tracejada mas sempre etiquetada com *implements*).