Programação Orientada pelos Objectos

1º Teste (27/Abril/2018)

MIEI 2017/2018

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.

Nos grupos I e II terá que implementar parcialmente algumas classes necessárias à construção de um programa que implementa a gestão de bonificações dos empregados de uma empresa. Vamos considerar que alguns dos empregados têm funções de gestão. **Note que apenas é permitido acrescentar métodos privados às classes** `EmployeeClass` e `ManagerClass` (não pode alterar ou acrescentar variáveis de instância ou métodos não privados a estas classes).

Grupo I

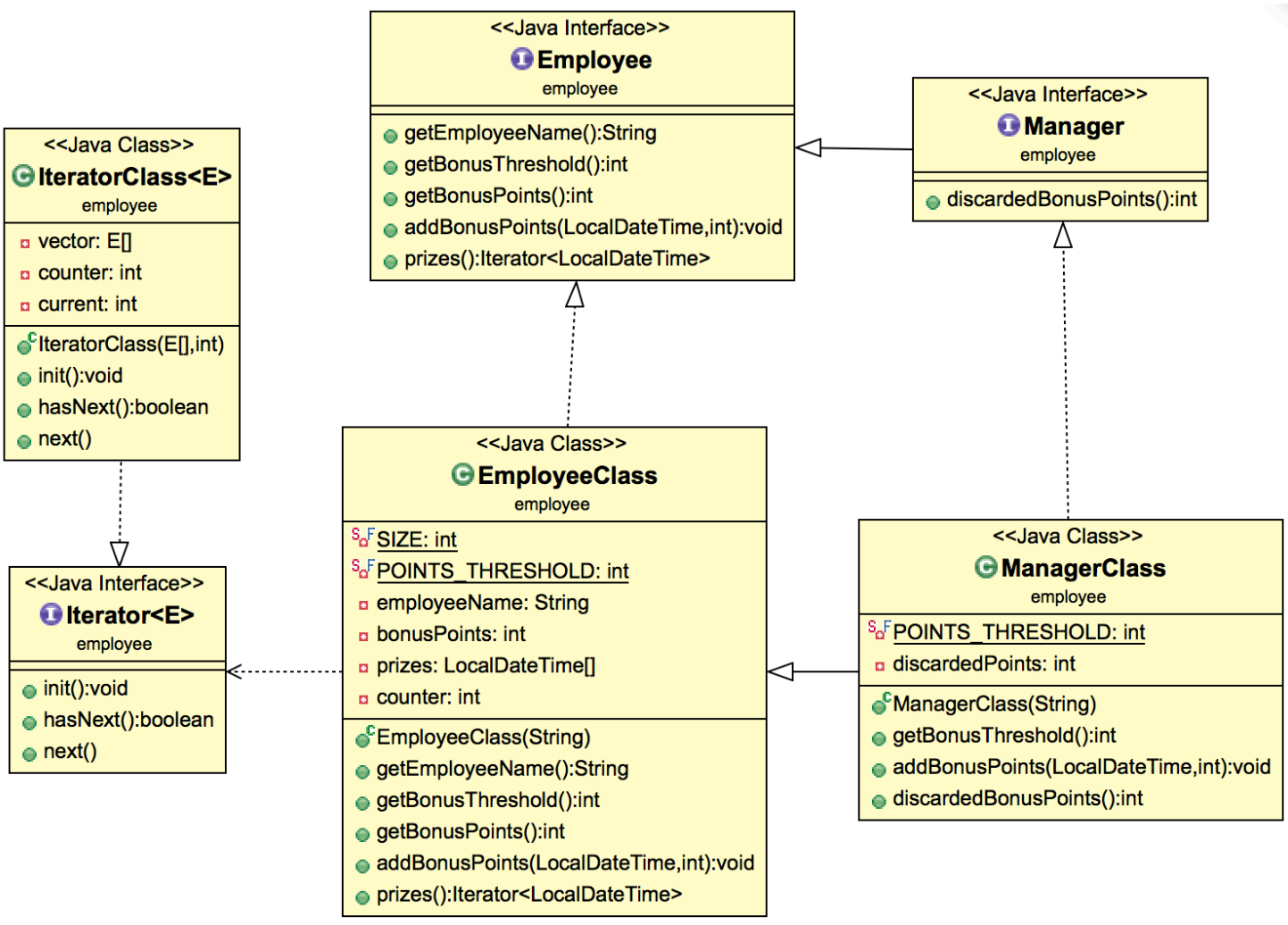

Figura 1 – Bonificações e prémios de um empregado.

As bonificações funcionam com um sistema de pontos: um resultado positivo na avaliação do empregado implica um número de pontos de bónus. De cada vez que o limite de pontos associado é ultrapassado, o empregado recebe um prémio (por exemplo, aumento de salário ou redução de horário de trabalho, etc.) por um determinado período de tempo. No início, os empregados não têm pontos de bonificação.

Por exemplo, se um empregado tiver um total acumulado de 9 pontos de bonificação, e o limiar for 12 pontos, se o empregado receber 4 pontos de bónus, então ser-lhe-á atribuído um prémio por ultrapassar os 12 pontos. Neste caso, é registada a data da atribuição do prémio.

Quando um empregado recebe um prémio, os pontos sobrantes são mantidos. Por exemplo, se um empregado normal atingir 15 pontos e o limiar vigente for 12 pontos, então é registado um prémio e o total de pontos de bonificação acumulados passa a 3. No caso de um *manager*, quando este recebe um prémio, os pontos de bonificação são "limpos". No exemplo anterior, o total de pontos de bonificação acumulados para o *manager* passaria a 0.

Considere-se a Fig. 1. A interface `Employee` representa um empregado e as suas bonificações. A classe `EmployeeClass` implementa a interface `Employee` e é especializada na classe `ManagerClass`.

Apresenta-se de seguida a descrição dos métodos da interface `Employee`:

- `getEmployeeName()` devolve o nome do empregado;
- `getBonusThreshold()` devolve o número de pontos mínimo para obter uma bonificação;
- `getBonusPoint()` devolve o número de pontos de bonificação registados;
- `addBonusPoints(LocalDateTime date, int points)` adiciona pontos de bonificação ao empregado, dados a data da bonificação e o número de pontos de bonificação. O mecanismo de bonificação segue as regras estabelecidas no enunciado, podendo originar a atribuição de prémios (assuma que não é necessário redimensionar o vector prizes).

  Deve assumir que `points <= getBonusThreshold();`
- `prizes()` devolve um iterador para todas as datas onde foram registados prémios.

A interface `Manager` representa um empregado com funções de gestão. A diferença principal entre a classe especialização está no mecanismo de bonificações (já explicado), nos limiares de pontos a partir dos quais existe a atribuição de um prémio, e ainda no facto permitir consultar os pontos "perdidos" sempre que é atribuído um prémio. Esta interface estende a interface `Employee` com o seguinte método:

- `discardedBonusPoints()` devolve o número total de pontos descartados aquando da atribuição de prémios.

A constante `SIZE` define a dimensão inicial do vector `prizes`. Ambas as classes têm uma constante `POINTS_THRESHOLD` que define o número mínimo de pontos para obter um prémio.

Implemente os seguintes métodos das classes `EmployeeClass` e `ManagerClass`.

a) O construtor `ManagerClass(String employeeName)`.

b) O método `void addBonusPoints(LocalDateTime date, int points)` da classe `EmployeeClass`.

c) O método `void addBonusPoints(LocalDateTime date, int points)` da classe `ManagerClass`.

d) O método `Iterator<LocalDateTime> prizes()` da classe `EmployeeClass`.

Tenha em atenção que as variáveis `employeeName, bonusPoints, prizes` e `counter` da classe `EmployeeClass` são privadas.

Grupo II

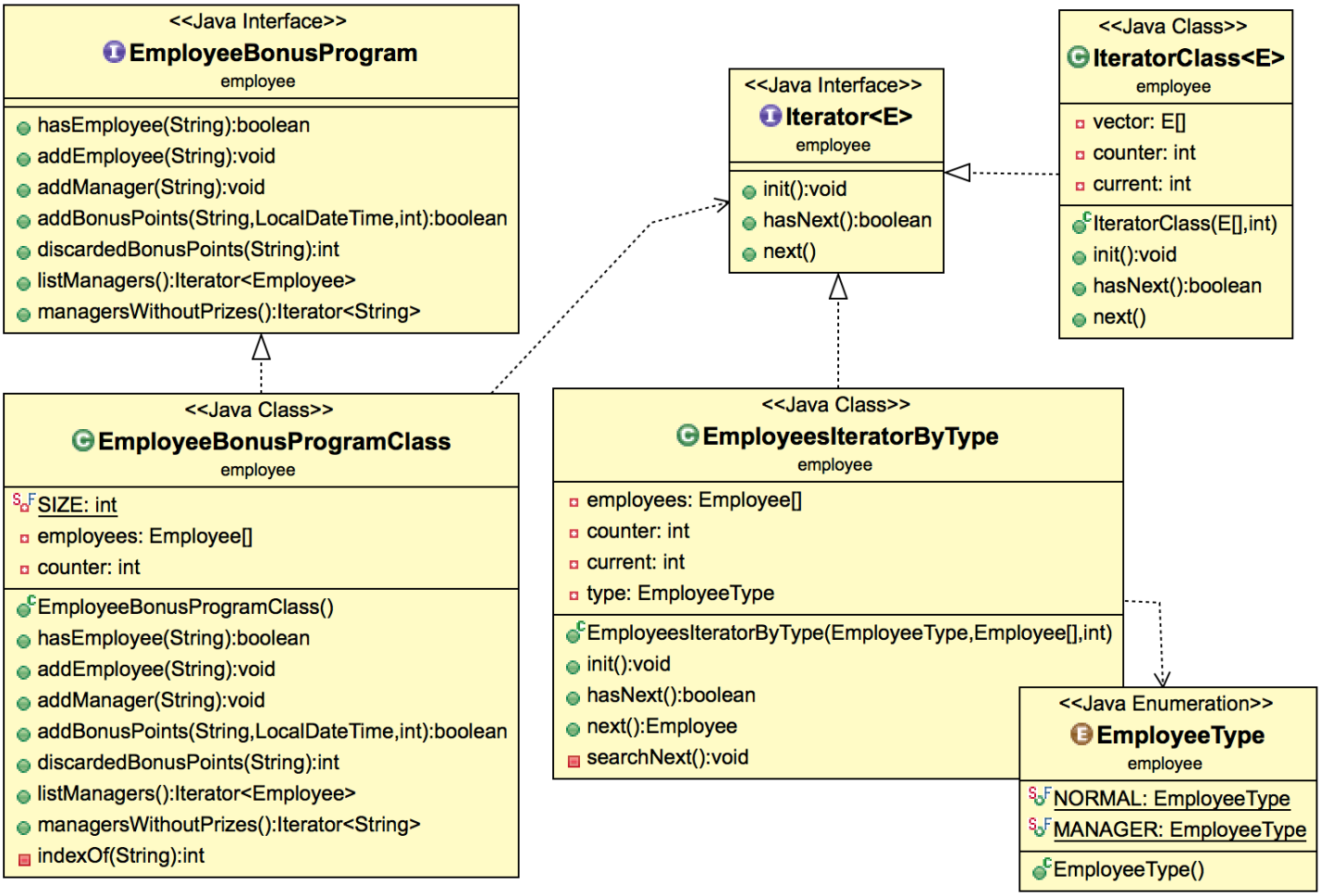

Figura 2 – Aplicação para gestão de bonificações.

A interface `EmployeeBonusProgram` representa uma aplicação para a gestão de bonificações dos empregados de uma empresa (Fig. 2).

- O método `addEmployee(String employeeName)` (respectivamente `addManager(String employeeName)`) adiciona um novo empregado (respectivamente manager). Este método só é executado se a pré-condição `hasEmployee(employeeName)` for falsa;
- `hasEmployee(String employeeName)` devolve `true` se existir um empregado com o nome `employeeName` na aplicação e `false` caso contrário;
- `addBonusPoints` é similar ao método da interface `Employee`, sendo que recebe como argumento adicional o nome do empregado (`employeeName`). Esta operação só será executada se a pré-condição `hasEmployee(employeeName)` for verdadeira;
- `discardedBonusPoints` é similar ao método da interface `Manager`, sendo que recebe como argumento o nome do empregado (`employeeName`). Esta operação só será executada se a pré-condição `hasEmployee(employeeName)` for verdadeira. Caso o empregado não seja manager, o método deve devolver zero.
- `listManagers()` devolve um iterador que percorre todos empregados managers;
- `managersWithoutPrizes()` devolve um iterador para os nomes de todos os managers que não ganharam qualquer prémio. Caso não existam managers nesta situação o método devolve `null`.

A classe `EmployeeBonusProgramClass` implementa o método privado `indexOf` que dado o nome do empregado devolve a sua posição no vector ou -1 caso esse empregado não exista.

O enumerado `EmployeeType` representa os dois tipos de empregados, `NORMAL` e `MANAGER`.

Implemente os seguintes métodos da classe `EmployeeBonusProgramClass`:

a) O método `void addManager(String employeeName).`

b) O método `void addBonusPoints(String employeeName, LocalDateTime date, int points).`

c) O método `int discardedBonusPoints(String employeeName).`

d) O método `Iterator<Employee> listManagers().`

e) O método `Iterator<Employee> managersWithoutPrizes().`

Grupo III

Pretende-se desenvolver um sistema de gestão de clientes para um operador de comunicações, como se descreve de seguida.
O operador de comunicações inclui uma vasta escolha de canais de televisão. Cada canal é identificado pelo nome, uma categoria (música, humor, etc.), o país de origem e se está activo ou desactivado. Alguns destes canais são qualificados como *Premium*. Sendo que estes canais têm um custo mensal associado.
O operador de comunicações oferece três tipos de pacotes distintos com características e mensalidades distintas. O pacote *Basic* inclui apenas canais normais (exclui canais *Premium*). O pacote *Silver* para além dos canais disponíveis no pacote *Basic* permite subscrever um canal *Premium*. O pacote *Gold* permite subscrever um número ilimitado de canais Premium. Os clientes podem ser clientes individuais ou empresarias. Os clientes individuais estão associados a uma localização e podem aderir a um único pacote. Os clientes empresarias podem estar associados a várias localizações e podem aderir a múltiplos pacotes, um por cada localização.
O sistema deve permitir: adicionar clientes, adicionar canais; activar e desactivar canais; associar pacotes a clientes; listar todos os canais de um pacote; listar todos os canais *Premium* de um cliente; calcular o valor da mensalidade de um cliente.

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.
- 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.

<u>Nota 1</u>: Não é necessário implementar nenhuma das operações.

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

<u>Nota 3</u>: 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).