Programação Orientada pelos Objectos

1º Teste (2/Maio/2017)

MIEI 2016/2017

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 um pedómetro para ser instalado numa aplicação de telemóvel. O pedómetro `Pedometer` representa um pedómetro simples que apenas tem suporte para operações básicas, enquanto o pedómetro `PedometerPlus` tem funcionalidades extra. **Note que apenas é permitido acrescentar métodos privados às classes** (não pode alterar/acrescentar variáveis ou métodos não privados).

Grupo I

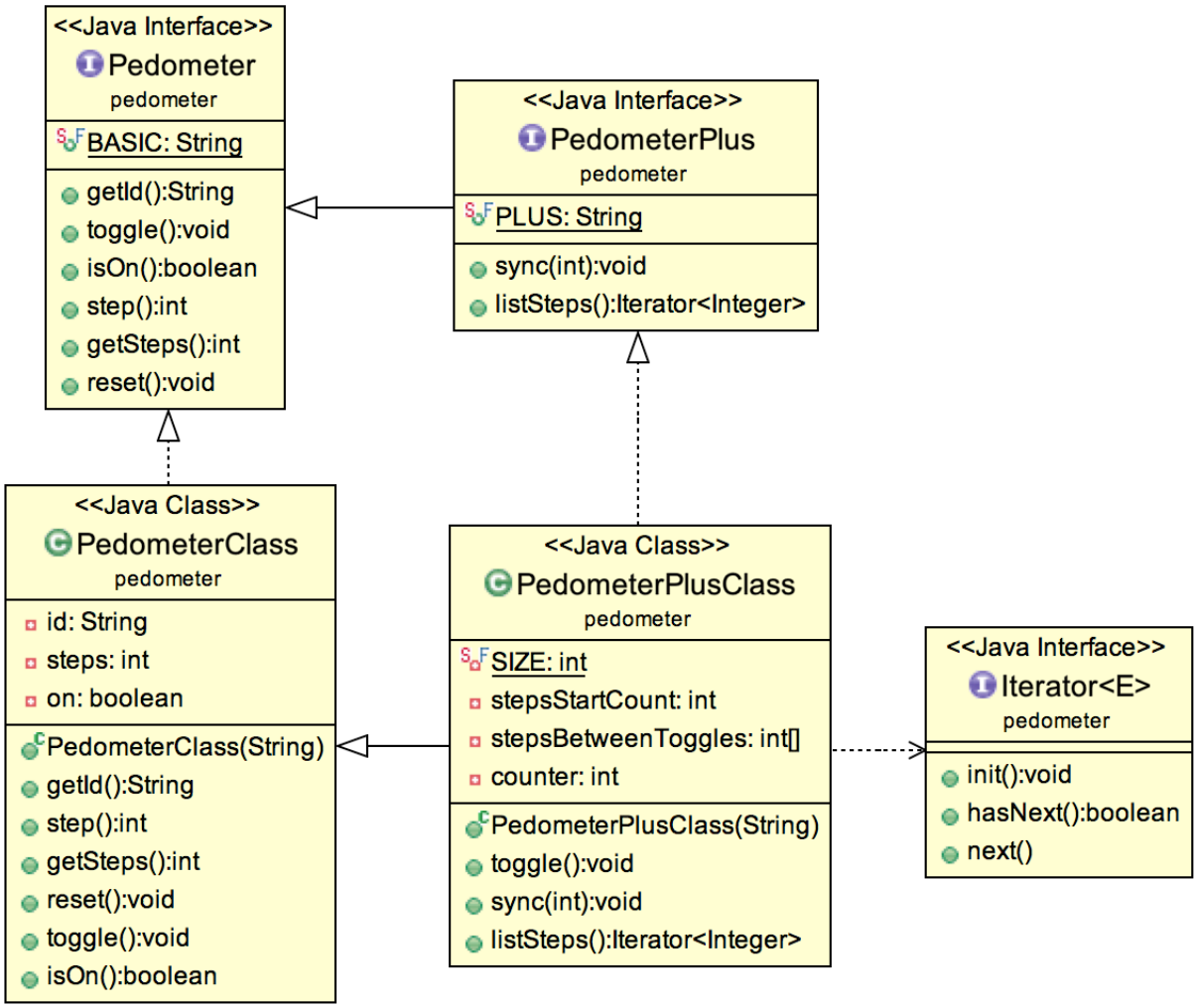

Figura 1 – Pedómetro simples e multifunções.

Note que pedómetro tem dois estados, activo ou suspenso, sendo que os passos do utilizador só são registados quando o pedómetro está activo.

A interface `Pedometer` representa um pedómetro (Fig. 1). Esta interface tem os seguintes métodos:

- `getId()` devolve o identificador do pedómetro;
- `toggle()` prossegue com a contagem, ou suspende a contagem (funcionando alternadamente);
- `isOn()` devolve `true` se a contagem estiver activa ou `false` se a contagem estiver suspensa;
- `step()` regista um passo do utilizador caso a contagem esteja activada ou, caso contrário, não altera a contagem. Em qualquer dos casos devolve o valor actual de passos;
- `reset()` coloca a contagem a zero e suspende a contagem;
- `getSteps()` devolve o valor de passos registado no pedómetro.

A classe `PedometerClass` implementa a interface `Pedometer`. Nesta classe temos um construtor que recebe o identificador e inicializa o pedómetro a zero e com contagem suspensa. Os métodos desta classe têm o comportamento já descrito na explicação dada sobre a interface `Pedometer`.

A interface `PedometerPlus` representa um pedómetro com funcionalidades extra (Fig. 1). Uma das funcionalidades é a possibilidade de sincronização com outros dispositivos. A outra funcionalidade é a capacidade de registar o número de passos entre contagens. Esta interface estende a interface `Pedometer` com os seguintes métodos:

- `sync(int steps)` faz a sincronização com outro dispositivo registando o número de passos recebido como argumento. O registo dos passos deve ser efectuado mesmo que a contagem esteja suspensa;
- `listSteps()` devolve um iterador para o número de passos entre contagens.

Para a interacção seguinte o método `listSteps()` iria permitir iterar sobre os valores: 2; 3; 1.

```
PedometerPlus plus = new PedometerPlusClass("alice");
plus.toggle(); // contagem iniciada
plus.step();
plus.step();
plus.toggle(); // contagem suspensa
plus.toggle(); // contagem iniciada
plus.step();
plus.step();
plus.step();
plus.toggle(); // contagem suspensa
plus.toggle(); // contagem iniciada
plus.toggle(); // contagem suspensa
plus.toggle(); // contagem iniciada
plus.step();
plus.toggle(); // contagem suspensa
```

Figura 2 – Exemplo de interacção.

A classe `PedometerClass` é estendida pela classe `PedometerPlusClass` que representa o pedómetro com duas funcionalidades adicionais. Tal como o construtor da sua super-classe, o construtor da classe `PedometerPlusClass` recebe também o identificador do pedómetro.

A constante `SIZE` define a dimensão inicial do vector `stepsBetweenToggles`.
Implemente os seguintes métodos das classes `PedometerClass` e `PedometerPlusClass`.

a) O construtor `PedometerPlusClass(String id)`.

b) O método `int step()` da classe `PedometerClass`.

c) O método `void sync(int steps)` da classe `PedometerPlusClass`.

d) O método `void toggle()` redefinido na classe `PedometerPlusClass`.

Tenha em atenção que as variáveis `on` e `steps` da classe `PedometerClass` são privadas.

Grupo II

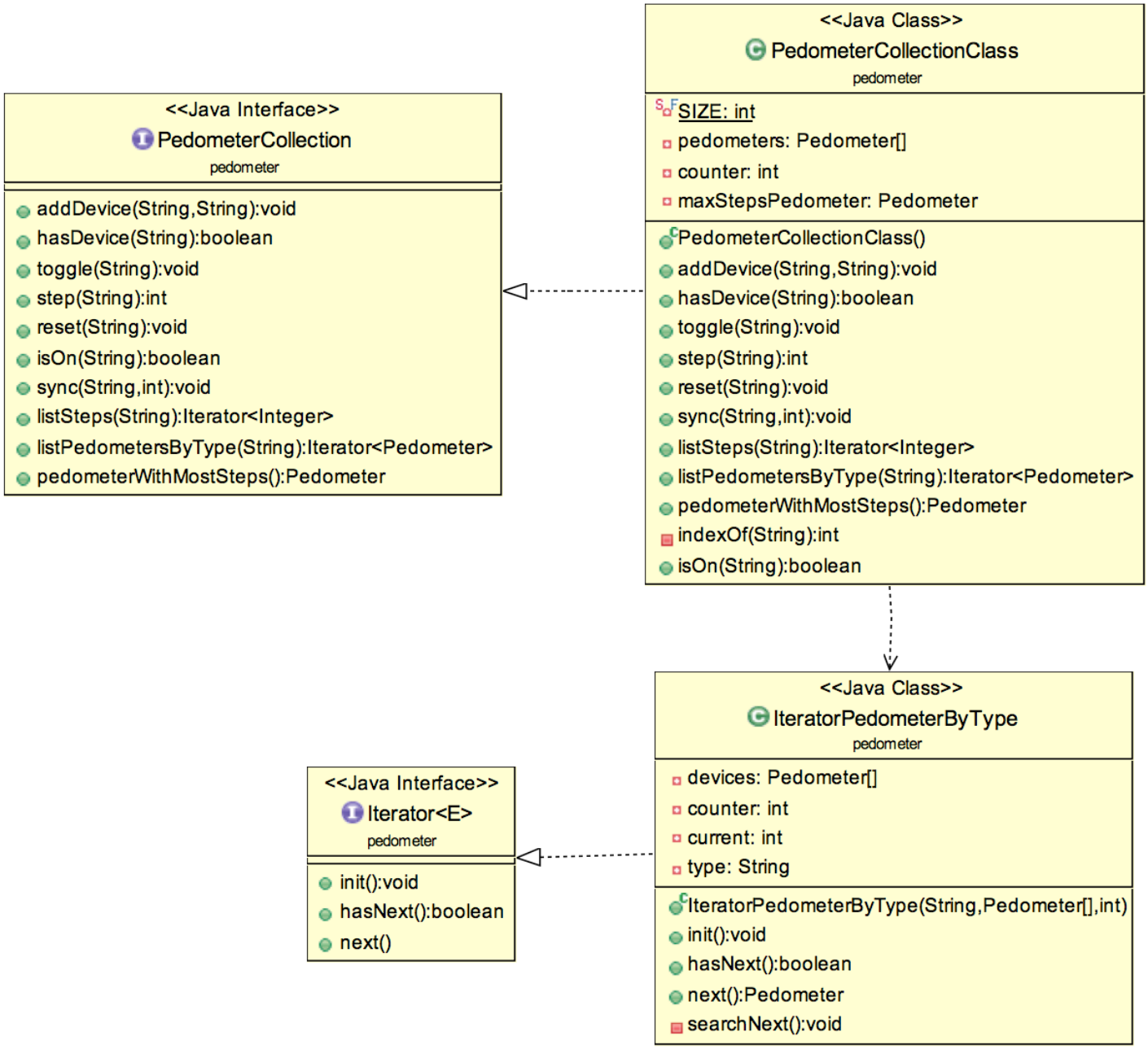

Figura 3– Aplicação para gestão dos dispositivos.

A interface `PedometerCollection` representa uma aplicação para a gestão de dispositivos (Fig. 3). As constantes `BASIC` e `PLUS` estão definas no diagrama da Fig. 2 nas respectivas interfaces.

- `addDevice(String id, String type)` adiciona um novo pedómetro de tipo `type` (cujo valor é `BASIC` ou `PLUS`). Este método só é executado se a pré-condição `hasDevice(id)` for falsa;
- `hasDevice(String id)` devolve `true` se existir um pedómetro com o identificador `id` na aplicação e `false` caso contrário;
- Os métodos `toggle`, `step`, `reset`, `isOn`, `sync` e `listSteps` são similares aos métodos das interfaces `Pedometer` e `PedometerPlus`, sendo que recebem como argumento adicional o identificador do pedómetro (`id`). Estas operações só serão executadas se a pré-condição `hasDevice(id)` for verdadeira;
- `listPedometerByType(String type)` devolve um iterador que percorre os pedómetros do tipo `type`;

- `pedometerWithMostSteps()` devolve o pedómetro que acumulou mais passos. Caso não existam dispositivos o método devolve `null`. Em caso de empate deve devolver o pedómetro que atingiu o número máximo de passos primeiro.

Note que a classe `PedometerCollectionClass` tem a variável `maxStepsPedometer` que mantém o pedómetro com mais passos registados. Se só existir um pedómetro deve ser esse, independentemente do número de passos. Deve considerar a actualização desta variável nos métodos que achar adequados.

Implemente os seguintes métodos da classe `PedometerCollectionClass`:

a) O método `void addDevice(String id, String type).`

b) O método `void toggle(String id).`

c) O método `void sync(String id, int steps)`

d) O método `Iterator<Pedometer> listPedometersByType(String type).`

Grupo III

Considere um método adicional da interface `PedometerCollection`:

```
Iterator<PedometerPlus> listPedometersIncreasingSteps(int num).
```

Este método devolve um iterador para os pedómetros (do tipo `PedometerPlus`) que têm pelo menos uma sequência crescente de passos entre contagens de dimensão igual ou superior a `num`. Por exemplo, o objecto `plus` da Fig. 2 tem uma sequência crescente de passos entre contagens de dimensão 2. O número de passos da primeira contagem é 2, sendo que o número de passos na segunda contagem aumenta para 3. Na terceira contagem o número de passos diminui para 1. Desta forma há uma sequência crescente de dimensão 2.

Implemente este método.

Grupo IV

Pretende-se implementar uma aplicação para a gestão de utentes de uma cadeia de ginásios, como se descreve de seguida.
A cadeia de ginásios tem vários ginásios onde se oferecem aulas colectivas e aulas individuais com um treinador dedicado. Algumas aulas colectivas requerem marcação prévia, realizada online pelos utentes, por serem muito concorridas. Todas as aulas com um treinador pessoal requerem marcação.
Existem três tipos de utentes do ginásio: regular, *premium* e *globe-trotter*.
Um utente regular tem acesso a um número ilimitado de aulas, num determinado ginásio. Caso pretenda participar numa aula com marcação, tem de a marcar antecipadamente. Pode também marcar aulas individuais, sendo que nesse caso o custo dessas aulas será reflectido na mensalidade do utente.
Um utente *premium* tem acesso a um número ilimitado de aulas, num determinado ginásio. Caso pretenda participar numa aula com marcação, tem de a marcar antecipadamente, usufruindo de vagas especialmente dedicadas a utentes *premium*. Pode também marcar aulas individuais, sem custos adicionais na sua mensalidade.
Um utente *globe-trotter* tem exactamente as mesmas regalias que um utente *premium*, mas a possibilidade de participar em aulas em qualquer dos ginásios da cadeia. Naturalmente, tem um preço mais elevado na mensalidade e, ao marcar uma aula, pessoal ou colectiva, tem de indicar em qual dos ginásios a pretende ter.
Deve ser possível registar novos clientes, criar novos ginásios da cadeia, criar aulas colectivas indicando o ginásio onde ocorrem, a capacidade e a data/hora de início. Assuma que a data e hora são representadas por Strings.
Além disso, deve ser possível o cliente inscrever-se numa determinada aula colectiva a decorrer numa determinada data/hora num determinado ginásio. Quanto às aulas individuais, são criadas a pedido do cliente, com uma determinada data/hora, num determinado ginásio.
Deve ser possível listar os ginásios. Deve ser possível listar os clientes com direito de acesso a um dado ginásio (incluindo os *globe-trotters*).
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.

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).