Programação Orientada pelos Objectos
2º Teste (Duração 2h)
MIEI 2017/2018

Instruções:

- Antes de começar a resolver, **leia o enunciado do princípio até ao fim**.
  - As interfaces e classes do grupo de 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.
- **Responda a grupos diferentes em folhas diferentes.**

Introdução aos problemas para os grupos I, II, III e IV:

Nestes 4 grupos vamos implementar parcialmente algumas das classes necessárias à construção de uma aplicação que permita gerir a venda de fotos e álbuns de fotos (banco de fotografias). No primeiro grupo, faremos a implementação de uma classe que representa base de dados com uma lista. No segundo grupo, faremos a implementação de uma classe que representa a base de dados com outras estruturas de dados do Java. Quer no primeiro, quer no segundo grupo, usamos as classes e interfaces especificadas na primeira parte enunciado. No terceiro grupo é pedida a implementação de um método adicional. No quarto grupo, realizamos alguns testes unitários e praticamos o uso de asserções.

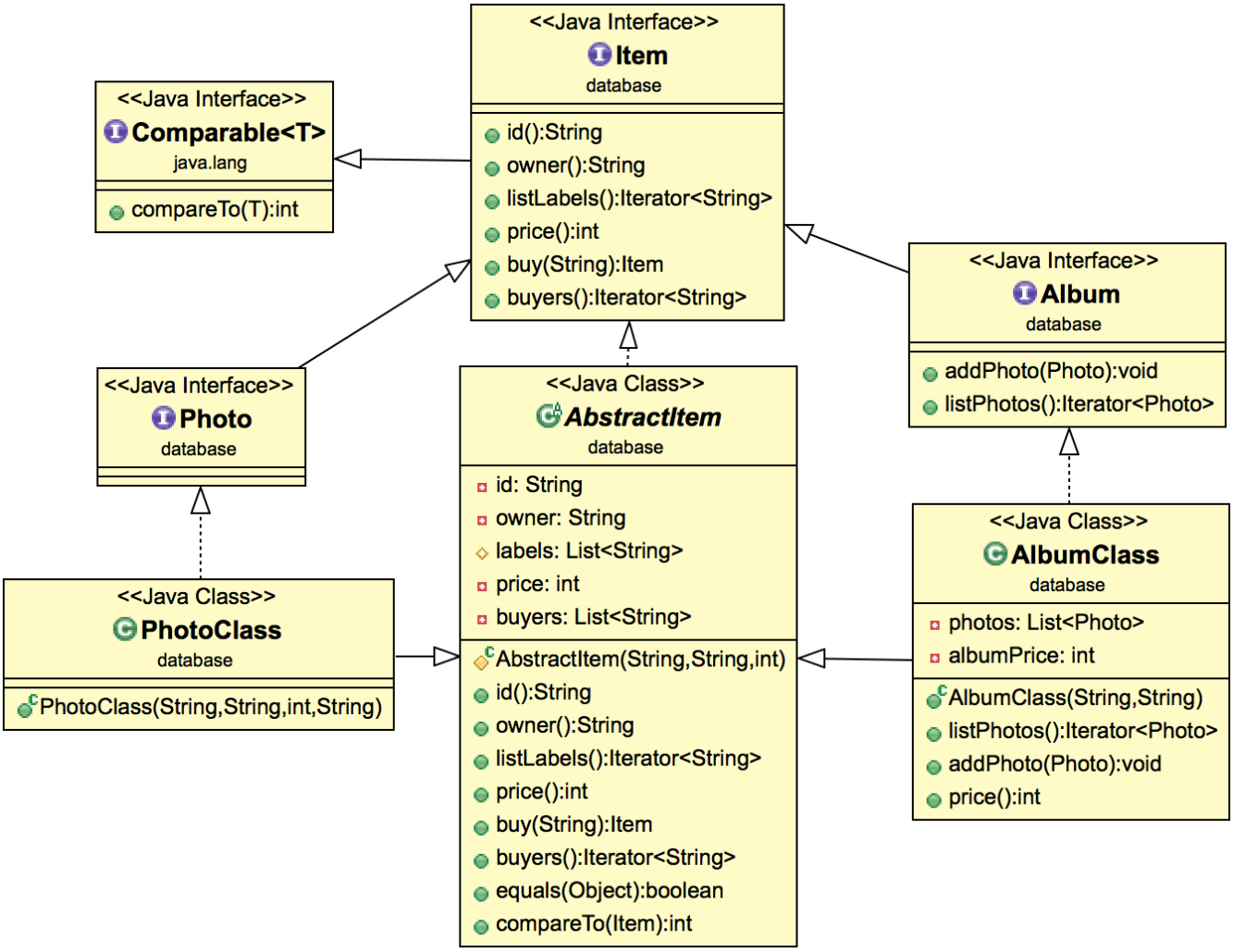

Figura 1: Álbuns e fotos.

A interface `Item` representa uma foto ou um álbum de fotos vendidas por um banco de fotografias. Os objectos da classe `AbstractItem` mantêm informação sobre uma foto ou um álbum de fotos. Na interface `Item`:

- `id` devolve uma `String` com o identificador do item (identificador único).
- `owner` devolve o utilizador detentor dos direitos de autor da foto ou álbum.
- `listLabels` devolve um iterador para as palavras chave da foto ou álbum.
- `price` devolve o preço da foto ou álbum.
- `buy` regista o comprador recebido como argumento e devolve foto ou álbum comprados.
- `listBuyers` devolve um iterador dos compradores da foto ou álbum.
- `compareTo` comparação dos itens por ordem alfabética de identificador.

A classe abstracta `AbstractItem` implementa a interface `Item`. O construtor desta classe recebe o identificador, o utilizador detentor dos direitos de autor e o preço. Os métodos implementados nesta classe obedecem à especificação já apresentada durante a descrição da interface.

A classe abstracta é especializada em duas classes concretas que representam os dois tipos de itens que são vendidos no banco de fotos: fotos individuais ou um álbum com várias fotos.

A interface `Photo`, que representa uma foto individual, é uma interface etiqueta.

Esta interface é implementa pela classe `PhotoClass`. O construtor recebe os argumentos do construtor da superclasse abstracta, para além da palavra chave que representa a foto.

A interface `Album`, que representa um álbum de fotos, estende a interface `Item` com dois métodos adicionais:

- `addPhoto` adiciona a foto recebida como argumento ao álbum, incrementa o preço do álbum com o preço da foto e adiciona a palavra chave da foto à lista de palavras chaves do álbum (caso seja uma nova palavra chave). Esta operação pode lançar as excepções: `PhotoExistsException` no caso foto já existir álbum e `DifferentOwnerException` no caso do utilizador detentor dos direitos de autor da foto for diferente do dono do álbum.
- `listPhotos` devolve um iterador para as fotos do álbum.

Esta interface é implementa pela classe `AlbumClass`. O construtor recebe o identificador e o utilizador detentor dos direitos de autor do álbum. Esta classe redefine o método `price` para devolver a soma dos preços de todas as fotos do álbum.

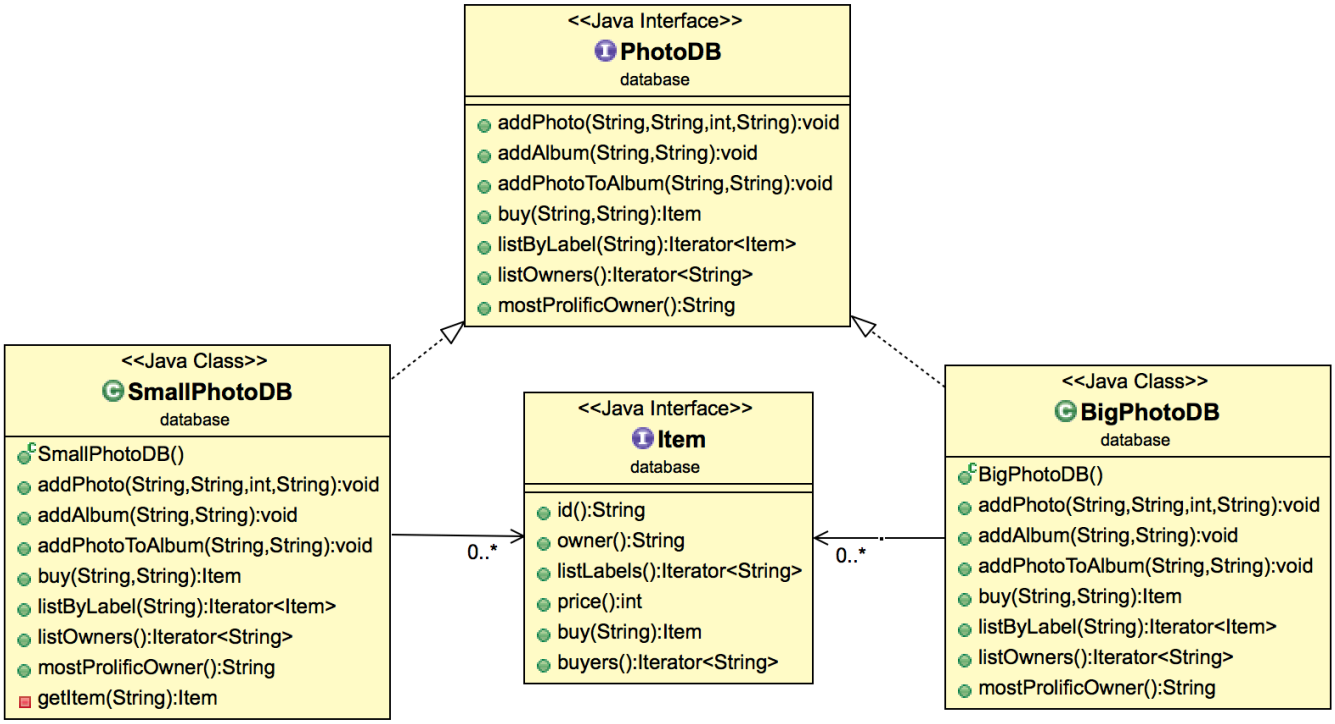

Figura 2: Colecção de fotos e álbuns.

A Fig. 2 apresenta a interface e as classes necessárias à resolução dos grupos I e II. A interface `PhotoDB` representa a colecção de fotos e álbuns e tem os seguintes métodos:

- `addPhoto` adiciona uma nova foto, recebendo, por esta ordem, o identificador, o utilizador detentor dos direitos de autor, o preço e a palavra chave. A operação lança a excepção `ItemExistsException` se já existir um item, foto ou álbum, com o identificador dado.
- `addAlbum` adiciona um novo álbum, recebendo, por esta ordem, o identificador e o utilizador detentor dos direitos de autor. A operação lança a excepção `ItemExistsException` se já existir um item, foto ou álbum, com o identificador dado.
- `addPhotoToAlbum` adiciona uma foto a um álbum, recebendo, por esta ordem, o identificador do álbum e identificador da foto. Esta operação pode lançar as excepções: `InexistentItemException` se os identificadores não existirem ou não corresponderem a uma foto e a um álbum, respectivamente; `PhotoExistsException` no caso da foto já existir no álbum; `DifferentOwnerException` no caso do detentor dos direitos de autor da foto for diferente do dono do álbum.
- `buy` recebe como argumento o identificador da foto ou álbum a comprar e regista o comprador recebido como argumento, devolvendo a foto ou álbum comprados. A operação lança a excepção `InexistentItemException` se não existir um item, foto ou álbum, com o identificador dado.
- `listByLabel` recebe como argumento uma palavra chave e devolve um iterador para todos os itens que tenham essa palavra chave, ordenados pela ordem natural. Caso não existem itens com a palavra chave dada o método deve devolver a excepção `InexistentItemException`.
- `listOwners` devolve um iterador para todos os utilizadores detentor dos direitos de autor de alguma foto ou álbum, ordenados por ordem alfabética. Caso não existam itens registados o método deve devolver a excepção `EmptyDatabaseException`.
- `mostProlificOwner` devolve um dos utilizadores detentor dos direitos de autor do maior número de fotos e álbuns. Caso não existam itens registados o método deve devolver a excepção `EmptyDatabaseException`.

Grupo I – Colecção pequena (`SmallPhotoDB`)

A classe `SmallPhotoDB` destina-se à gestão de uma base de dados de pequenas dimensões (na ordem das dezenas) pelo que vamos usar uma lista denominada `itens` para guardar a colecção de fotos e álbuns. Tendo em conta que neste grupo não pode adicionar variáveis de instância à classe, e que tem apenas acesso à variável `itens`, implemente os seguintes métodos:

a) O construtor de modo a que, ao ser criada a lista, esteja vazia. Apresente também a declaração da variável `itens` (coloque a declaração acima do construtor).

b)
```
void addAlbum(String id, String owner) throws ItemExistsException;
```

c)
```
void addPhotoToAlbum(String idAlbum, String idPhoto)
            throws InexistentItemException, PhotoExistsException,
                   DifferentOwnerException;
```

d)
```
Item buy(String id, String buyer) throws InexistentItemException;
```

e)
```
Iterator<String> listOwners() throws EmptyDatabaseException;
```

Note a classe `SmallPhotoDB` tem o método privado `getItem` que recebe como argumento um identificador e devolve o item com esse identificador ou `null` caso não exista nenhum item com esse identificador

Grupo II – Colecção grande (`BigPhotoDB`)

Pretende-se implementar uma classe para a gestão de uma base de dados de grandes dimensões (na ordem dos milhares). A implementação da classe deve ter em conta a eficiência das **pesquisas pelo identificador de um item**, a eficiência de **das listagens** `listByLabel` e `listOwners` e a eficiência do **método** `mostProlificOwner`.
Na alínea a) deve definir as variáveis de instância e nas seguintes alíneas deve implementar alguns dos métodos da classe `BigPhotoDB`:

a) Defina as variáveis de instância e escolha as estruturas de dados que achar mais adequadas de acordo com os requisitos de eficiência mencionados acima.

b) Defina o construtor de modo a que não existam itens na base de dados.

c)
```
void addPhoto(String id, String owner, int price, String label)
      throws ItemExistsException;
```

d)
```
void addPhotoToAlbum(String idAlbum, String idPhoto) throws
      InexistentItemException, PhotoExistsException, DifferentOwnerException;
```

e)
```
Iterator<Item> listByLabel(String label) throws InexistentItemException;
```

f)
```
String mostProlificOwner() throws EmptyDatabaseException;
```

Grupo III

Considere que existe um método estático adicional na interface `PhotosBD`:

```
static boolean haveCommonLabels(Iterator<Album> albuns, int num).
```

Este método recebe como argumento um iterador de álbuns e um inteiro positivo e devolve `true` caso todos os álbuns tenham pelo menos `num` palavras chave em comum entre eles. Por exemplo, se `num` for 3 deveram existir 3 palavras chave em comum entre todos os álbuns do iterador recebido como parâmetro. Implemente este método.

Grupo IV – Testes unitários e asserções

a) Implemente uma operação de teste ao método `addPhoto` da classe `AlbumClass` apresentado na introdução do teste, usando o JUnit. O seu teste deve verificar que, para além da foto ser inserida no álbum, o preço e as palavras chave são actualizados correctamente. Para cada um dos casos de utilização verifique que obtém o resultado esperado.

b) Considere o seguinte método:

```
1    public int foo(int y) {
2          if ( y > 5 ) return y+2;
3          else return y*3;
4    }
5
6    public void function (int z){
7        if ( z < 7 )
8            assert z > 4 : foo(z);
9
10       switch ( z )
11       {
12           case 4: System.out.println("Four ");
13           case 5: System.out.println("Five ");
14           default: assert z < 10 : "Other ";
15       }
16
17       if ( z < 10 )
18           assert z > 4 : ++z;
19       System.out.println(z);
20   }
```

i. Assumindo que o mecanismo de asserções está activo, diga qual o output produzido na consola pelo método `function` quando chamado com cada um dos seguintes valores: -4, 0, 4, 5.

ii. Indique as linhas onde considera que existe um uso inapropriado de asserções. Justifique resumidamente a sua resposta.