Programação Orientada pelos Objectos

2º Teste (Duração 2h)

LEI 2015/2016

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 e III:

Nestes 3 grupos vamos implementar parcialmente algumas das classes necessárias à construção de um serviço Web que permita anunciar e vender produtos online (OLX). No primeiro grupo, faremos a implementação de uma classe que representa a colecção de utilizadores com uma lista. No segundo grupo, faremos a implementação de uma classe que representa a colecção de utilizadores com outras colecções do Java. Quer no primeiro, quer no segundo grupo, usamos as classes e interfaces especificadas na primeira parte enunciado. No terceiro grupo, realizamos alguns testes unitários e praticamos o uso de asserções.

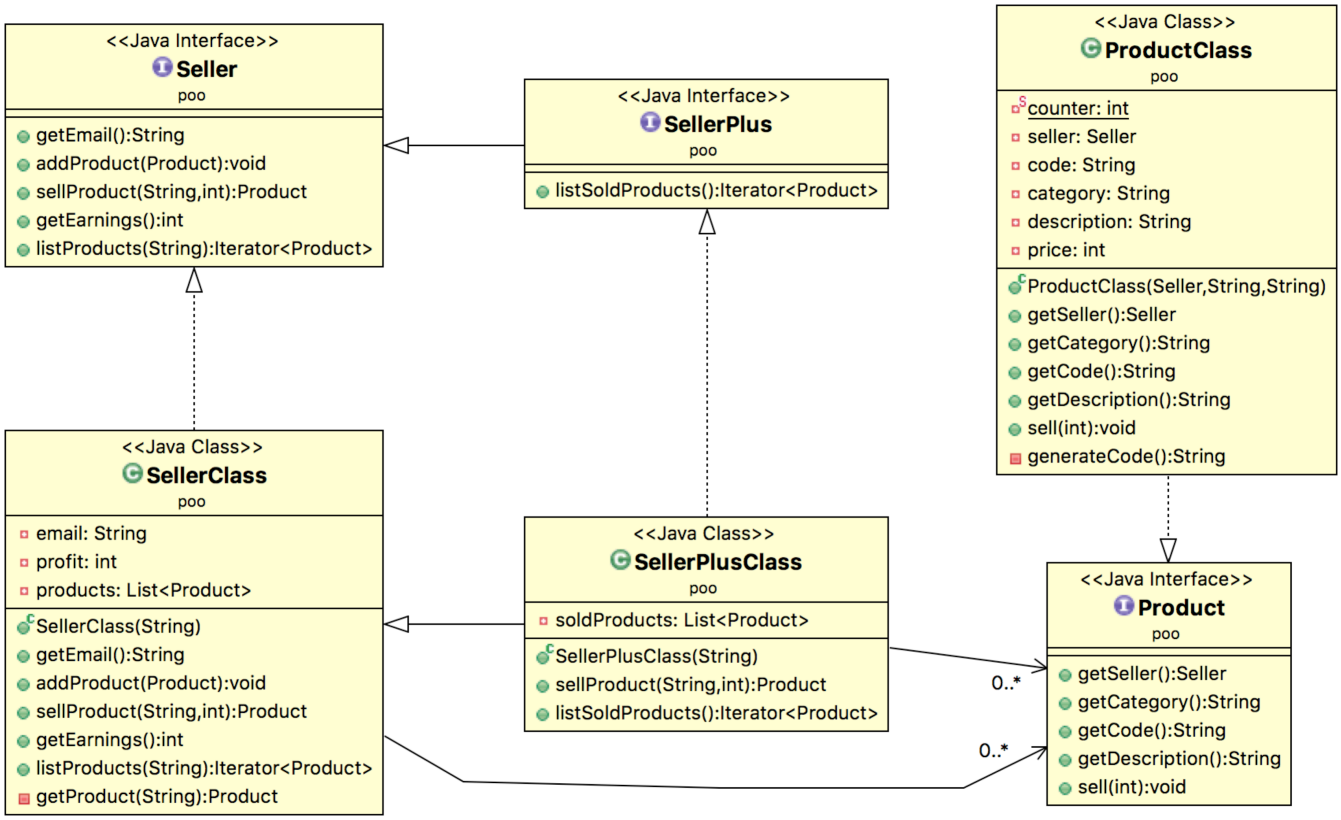

Figura 1: Utilizador registado

A interface `Seller` representa um utilizador registado que pode colocar anúncios de produtos. Note que não pode haver dois utilizadores diferentes com o mesmo email. Os objectos desta classe mantêm uma colecção de todos os produtos à venda do utilizador. Na interface `Seller`:

- `getEmail` devolve uma `String` com o email do utilizador (identificador único).
- `addProduct` regista um novo produto a vender.
- `sellProduct` retira o produto da colecção de produtos à venda, acumula o valor da venda ao total das vendas (recebe o código do produto e preço pelo qual o produto foi vendido) e devolve esse produto. Note que os valores de vendas são sempre números inteiros. Este método lança a excepção `ProductDoesNotExistException` se o código do produto não existir na colecção de produtos à venda do utilizador.
- `getEarnings` devolve a quantia ganha com as vendas dos produtos.
- `listProducts` devolve um iterador dos produtos à venda com a categoria dada, a iteração é feita pela ordem inversa de inserção.

A classe `SellerClass` implementa a interface `Seller`. O construtor desta classe recebe o email do utilizador. Os métodos implementados nesta classe obedecem à especificação já apresentada durante a descrição da interface. Para além dos métodos da interface a classe inclui um método privado:

- `getProduct` recebe o código do produto e devolve o produto à venda com esse código.

A classe `SellerClass` é especializada na classe `SellerPlusClass`, que representa um utilizador *premium* que pagou a adesão ao site. A vantagem destes utilizadores é que na listagem dos produtos à

venda os seus produtos são listados em primeiro lugar. Os objectos desta classe mantêm uma lista de todos os produtos já vendidos. Nesta classe as operações têm um comportamento similar às operações da super-classe `SellerClass`. A única diferença está no seguinte método:

- `sellProduct` retira o produto da colecção de produtos à venda, acumula o valor da venda ao total das vendas (recebe o código do produto e preço pelo qual o produto foi vendido) e insere o produto vendido na colecção de produtos já vendidos.

A classe `SellerPlusClass` implementa também um método adicional da interface `SellerPlus`:

- `listSoldProducts` devolve um iterador dos produtos já vendidos do utilizador com a categoria dada, a iteração é feita pela ordem inversa de venda (o mais recente à frente).

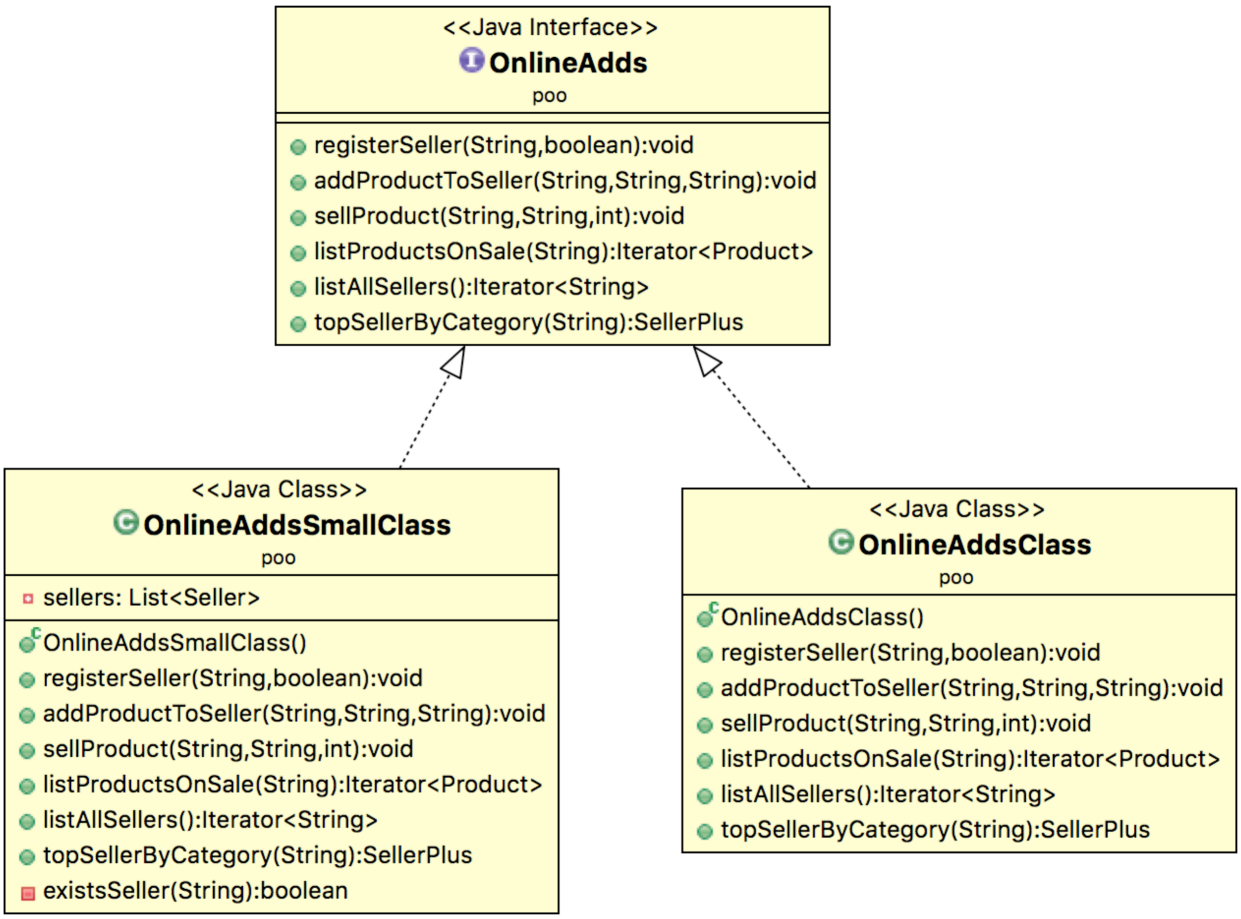

Figura 2: A interface OnlineAdds

A Fig. 2 apresenta a interface e as classes necessárias à resolução dos grupos I e II. A interface `OnlineAdds` representa a colecção de utilizadores de todos os utilizadores.

- O método `registerSeller` adiciona um novo utilizador e recebe o email e um booleano `isPlus` que indica se o utilizador é *premium*. A operação lança a excepção `SellerAlreadyExistsException` se já existir um utilizador com o email dado.
- O método `addProductToSeller` adiciona um novo produto à colecção de produtos à venda do utilizador, recebendo, por esta ordem, o email do utilizador, a categoria e a descrição do produto. Esta operação pode lançar a excepção `SellerDoesNotExistException` no caso do utilizador não existir.
- O método `sellProduct` vende um produto de um utilizador, recebendo, por esta ordem, o email do utilizador, o código e o preço de venda do produto. Esta operação pode lançar as excepções: `SellerDoesNotExistException` no caso do utilizador não existir,

`ProductDoesNotExistException` no caso do produto não existir na colecção de produtos à venda do utilizador.

- A operação `listProductsOnSale` devolve um iterador para todos os produtos à venda de uma dada categoria. Os produtos devem ser ordenados pelo tipo de utilizador a que pertencem e depois por ordem alfabética. Em particular, os produtos dos utilizadores *premium* devem aparecer à frente dos produtos dos restantes utilizadores. Caso não existam produtos dessa categoria, o método deve devolver `null`.
- O método `listAllSellers` devolve um iterador para os emails de todos os utilizadores. Este iterador percorre os utilizadores por ordem alfabética.
- O método `topSellerByCategory` devolve o utilizador *premium* com mais produtos vendidos de uma dada categoria. Esta operação lança a excepção `CategoryDoesNotExistException` no caso da categoria não existir e `NoTopSellerException` no caso de não existirem utilizadores *premium* com produtos vendidos dessa categoria.

Grupo I – Colecção pequena (`OnlineAddsSmallClass`)

A classe `OnlineAddsSmallClass` destina-se a situações com poucos utilizadores e produtos à venda, pelo que vamos usar uma lista denominada `sellers` para guardar a informação sobre os utilizadores. Neste grupo implemente os seguintes métodos:

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

b) O modificador `void registerSeller(String email, boolean isPlus).`

c) O modificador `public void addProductToSeller(String email, String category, String description).`

d) O selector `Iterator<String> listAllSellers().`

e) Implemente o método `compare(Product p1, Product p2)` da classe `ProductsComparator` que estende a classe `Comparator<Product>`. Considere que `p1` é menor que `p2` se `p1` é um produto de um utilizador *premium* e `p2` um produto de um utilizador "simples". No caso dos produtos serem de utilizadores do mesmo tipo, deve ser usada a ordem alfabética.

Grupo II – Colecção grande (`OnlineAddsClass`)

Pretende-se implementar uma classe para a gestão de um número elevado de utilizadores e produtos à venda. A implementação da classe deve ter em conta a eficiência das **pesquisas por email de utilizador**, a eficiência de **todas as listagens** e a eficiência do **método** `topSellerByCategory`. Note que mesmo que não tenha implementado o método `compare` da classe `ProductsComparator`, pode assumir que esta classe está implementada. Implemente as seguintes operações da classe `OnlineAddsClass`:

a) O construtor de modo a que, ao ser criado não existam utilizadores. Apresente também a declaração das variáveis que achar necessárias (pode colocar a declaração acima do construtor).

b) O modificador `void registerSeller(String email, boolean isPlus).`

c) O modificador `void addProductToSeller(String email, String category, String description).`

d) O modificador `void sellProduct(String email, String code, int price).`

e) O selector `Iterator<Product> listProductsOnSale(String category).`

f) O selector `SellerPlus topSellerByCategory(String category).`

Grupo III – Testes unitários e asserções

a) Implemente uma operação de teste ao método `compare` da classe `ProductsComparator`, usando o JUnit. O seu teste deve testar os vários cenários possíveis de utilização do método `compare`. Defina os objectos `Product`, `Seller` e `SellerPlus` que achar necessários. Invoque sucessivamente o método `compare` e verifique que obtém o resultado esperado.

b) Considere o seguinte método:

```
void f(int x) {
  switch (x) {
    case 1: System.out.println("Positive"); break;
    case -1: System.out.println("Negative"); break;
    default: System.out.println("Outro"); assert x == 0 : x;
  }

}
```

Assumindo que o mecanismo de asserções está activo, diga qual o output produzido na consola pelo método `f` quando chamado com cada um dos seguintes valores: -1, 0, 1 e 2.

Algumas das interfaces abaixo reproduzidas poderão ser úteis na resolução deste teste: