Programação Orientada pelos Objectos

1º Teste (02/Maio/2015)

MIEI 2014/2015

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.

No grupo I e II terá que implementar parcialmente algumas das classes necessárias à construção de um programa para manter uma aplicação para imprimir documentos online (WebPrinting). **Note que apenas é permitido acrescentar métodos privados às classes** (não pode alterar/acrescentar variáveis ou métodos não privados).

Grupo I

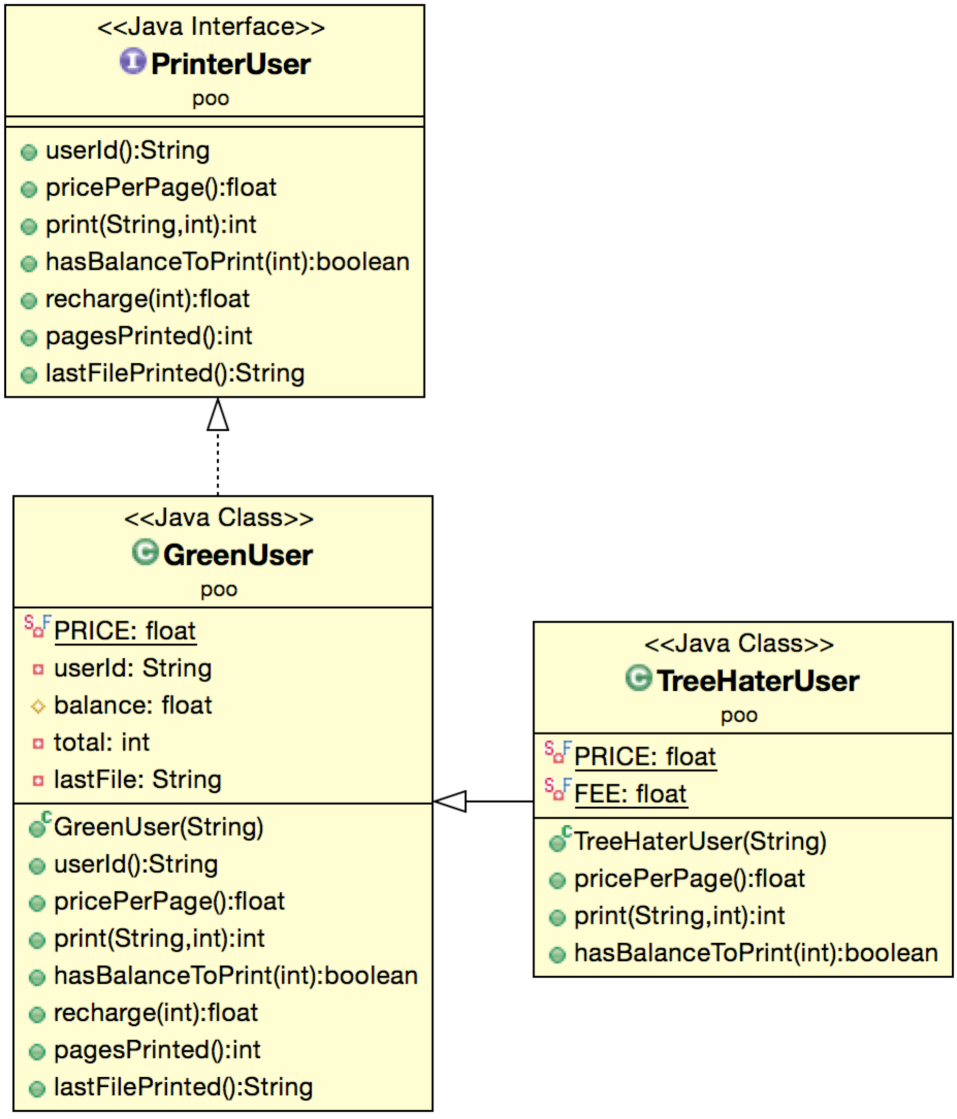

Figura 1 – Utilizador da aplicação WebPrinting

A interface `PrinterUser` representa um utilizador da aplicação para imprimir documentos online (Fig. 1). Esta interface tem os seguintes métodos:

- `userId()` devolve o identificador do utilizador;
- `pricePerPage()` devolve o preço de imprimir uma página;
- `print(String filename, int pages)` devolve o número de páginas impressas do documento `filename`;
- `hasBalanceToPrint(int pages)` devolve `true` se o saldo permite imprimir o número de páginas `pages` e `false` caso contrário;
- `recharge(int amount)` devolve o saldo após o carregamento com `amount` euros;
- `pagesPrinted()` devolve o número total de páginas impressas pelo utilizador;
- `lastFilePrinted()` devolve o nome do último ficheiro impresso pelo utilizador.

A classe `GreenUser` implementa a interface `PrinterUser`. Nesta classe temos um construtor `GreenUser` que recebe o identificador do utilizador. Os métodos desta classe têm o comportamento já descrito na explicação dada sobre a interface `PrinterUser`. No método `print` o saldo do utilizador pode não permitir a impressão de todas as páginas do documento e nesse caso são impressas o número de páginas permitidas pelo saldo do utilizador.

A classe `GreenUser` é estendida pela classe `TreeHaterUser` que representa utilizadores que imprimem documentos com um número elevado de páginas. Estes utilizadores pagam menos por cada página impressa, mas em troca pagam uma taxa fixa por documento impresso e para além disso apenas podem imprimir documentos completos. Ou seja, não é possível imprimir parte de um documento de acordo com o saldo disponível e no caso do saldo ser insuficiente o método `print` deve devolver o valor `0` (indicando que nenhuma página do documento foi impressa).

As constantes `PRICE` e `FEE` definem, respectivamente, o preço de impressão por página e a taxa por documento impresso.

Implemente os seguintes métodos das classes `GreenUser` e `TreeHaterUser`.

a) O construtor `GreenUser (String userId)`.

b) O método `int print(String filename, int pages)` da classe `GreenUser`.

c) O método `int print(String filename, int pages)` da classe `TreeHaterUser`. Tenha em atenção que as variáveis `total` e `lastFile` da classe `GreenUser` são privadas.

d) O método `boolean hasBalanceToPrint(int pages)` da classe `TreeHaterUser`.

Grupo II

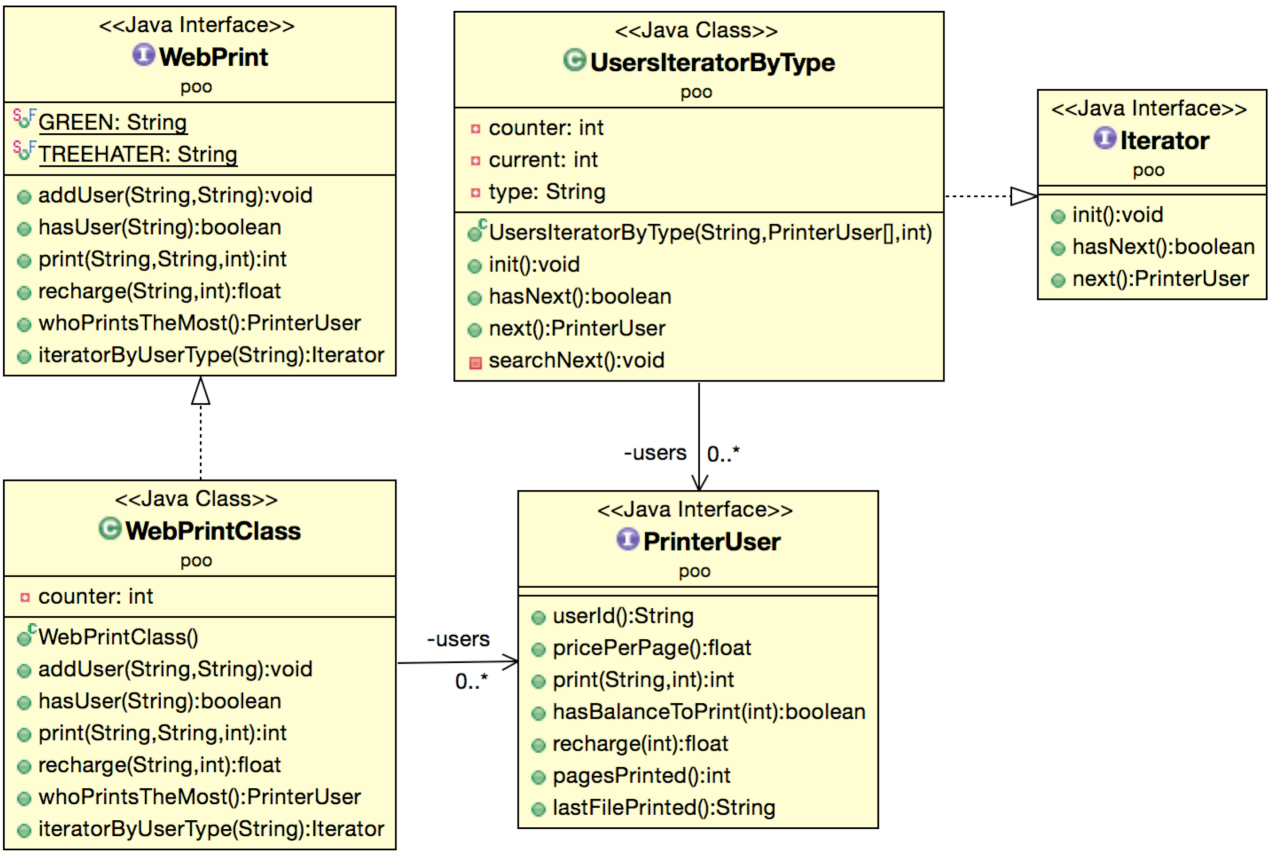

Figura 2 – Aplicação WebPrinting.

A interface `WebPrint` representa uma aplicação para imprimir documentos online (Fig. 2). As constantes `GREEN` e `TREEHATER` definem os dois tipos de utilizadores.

- `addUser(String userId, String type)` recebe o identificador e o tipo do utilizador, e adiciona um novo utilizador à aplicação. Se o vector estiver cheio, o método não faz nada. Este método só é executado se a pré-condição `!hasUser(userId)` for verdadeira;
- `hasUser(String userId)` devolve `true` se o identificador do utilizador já existir na aplicação e `false` caso contrário.
- `print(String userId, String filename, int pages)` devolve o número de páginas impressas do documento `filename` do utilizador `userId`. Este método só é executado se a pré-condição `hasUser(userId)` for verdadeira;
- `recharge(String userId, int amount)` devolve o saldo do utilizador `userId` após o carregamento com `amount` euros. Este método só é executado se a pré-condição `hasUser(userId)` for verdadeira;
- `whoPrintsTheMost()` devolve o utilizador que imprimiu mais páginas na aplicação WebPrinting. Caso nenhum utilizador tenha imprimido documentos o método deve devolver `null` e em caso de empate deve devolver o utilizador mais antigo;
- `IteratorByUserType(String type)` devolve um iterador que percorre os dois tipos de utilizadores.

Implemente os seguintes métodos da classe `WebPrintClass`:

a) O método `void addUser(String userId, String type)`.
b) O método `recharge(String userId, int amount)`.
c) O método `PrintUser whoPrintsTheMost()`.
d) O método `Iterator iteratorByUserType(String type)`.

Grupo III

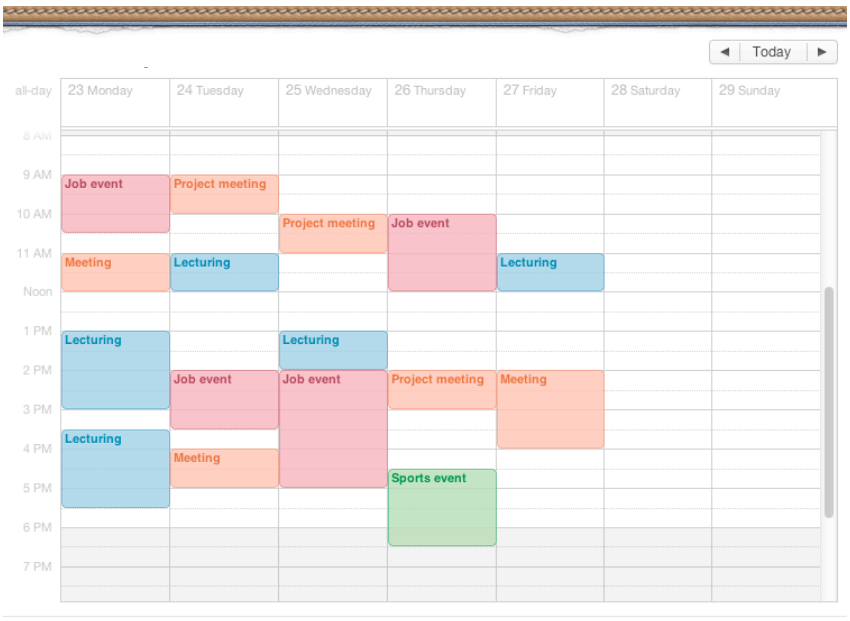

Figura 3 – Agenda de eventos.

Pretende-se implementar um programa que faça a gestão de uma agenda pessoal de eventos. A Fig. 3 ilustra o tipo de agenda que se pretende, a qual deve permitir:

- Definir eventos de diferentes categorias. Considere que existem quatro tipos de eventos: familiares, profissionais, desportivos e de voluntariado.
- Adicionar participantes a cada evento;
- Adicionar tarefas aos eventos profissionais. Considere que apenas é possível adicionar tarefas a este tipo de eventos;
- Listar de forma selectiva eventos de várias categorias, isto é, uma ou mais categorias em simultâneo e, além disso, com um determinado intervalo temporal. Por exemplo, deve ser possível visualizar todos os eventos profissionais e de voluntariado entre 04-Maio-2015 e 12-Maio-2015.
- Listar as tarefas de um dado evento profissional;
- Listar os participantes de um evento.

Um evento tem uma descrição textual, uma data, uma delimitação temporal e é de uma dada categoria. Cada participante tem um nome e um endereço de email. As tarefas têm uma descrição e uma prioridade (valor de 1 a 5, sendo que 1 indica a tarefa mais prioritária).

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 – *PersonalCalendar* – com a qual o programa principal irá interagir. Esta interface deve ser completamente especificada.
- As variáveis de instância da classe que implementa a interface *PersonalCalendar*.
- Para as restantes componentes do diagrama, isto é, para as interfaces e classes que não a interface *PersonalCalendar*, 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).