Programação Orientada pelos Objectos

1º Teste (22/Abril/2016)

MIEI 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.

No grupo I e II terá que implementar parcialmente algumas das classes necessárias à construção de um programa para fazer a gestão de cartões de pagamento. O cartão `BasicCard` representa um cartão que apenas permite operações de débito, enquanto o cartão `GoldCard` representa um cartão que permite operações de débito e de crédito (sendo o crédito ilimitado). **Note que apenas é permitido acrescentar métodos privados às classes** (não pode alterar/acrescentar variáveis ou métodos não privados).

Grupo I

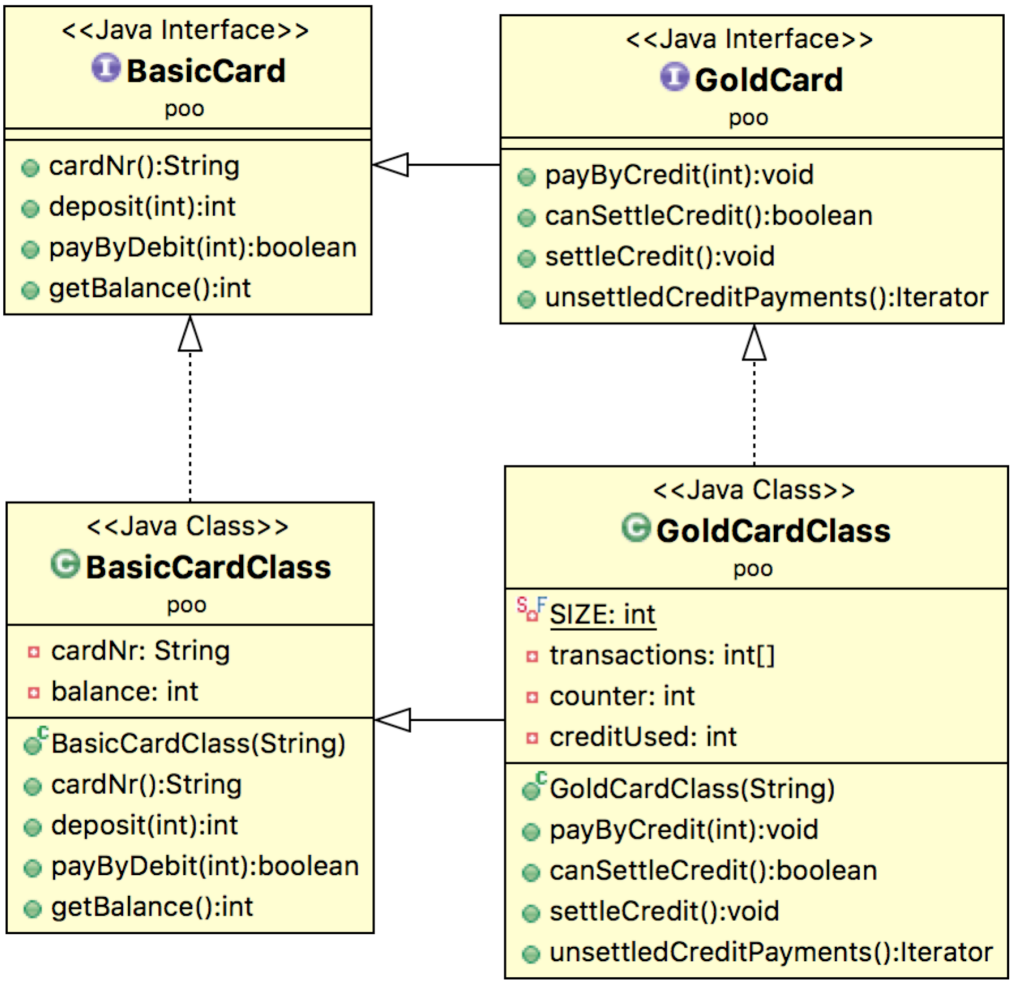

Figura 1 – Cartões de crédito e débito.

A interface `BasicCard` representa um cartão de débito (Fig. 1). Esta interface tem os seguintes métodos:

- `cardNr()` devolve o número do cartão;
- `deposit(int amount)` deposita a quantia `amount` no cartão e devolve o novo saldo do cartão;
- `payByDebit(int amount)` tenta fazer um pagamento da quantia `amount` no cartão: faz o pagamento caso o saldo seja suficiente e devolve `true`; caso o saldo seja insuficiente não faz nada e devolve `false`;
- `getBalance()` devolve o saldo do cartão.

A classe `BasicCardClass` implementa a interface `BasicCard`. Nesta classe temos um construtor `BasicCardClass` que recebe o número de cartão. Os métodos desta classe têm o comportamento já descrito na explicação dada sobre a interface `BasicCard`.

A interface `GoldCard` representa um cartão de crédito (Fig. 1). Esta interface estende a interface `BasicCard` com os seguintes métodos:

- `payByCredit(int amount)` faz o pagamento a crédito do `amount` no cartão, ou seja, o saldo do cartão não é alterado mas o pagamento a crédito é registado no cartão. Assume-se que este tipo de cartão não tem limite de crédito, por isso esta operação tem sempre sucesso;
- `unsettledCreditPayments()` devolve um iterador para os pagamentos a crédito ainda não saldados;
- `canSettleCredit()` devolve `true` caso o saldo do cartão seja suficiente para saldar os pagamentos a crédito registados, caso contrário devolve `false`;
- `settleCredit()` salda todos os pagamento a crédito debitando essa quantia do saldo do cartão e eliminando todos os pagamentos a crédito registados. Este método só é executado se a pré-condição `canSettleCredit()` for verdadeira.

A classe `BasicCardClass` é estendida pela classe `GoldCardClass` que representa cartões que permitem tanto pagamentos a débito como a crédito. Neste tipo de cartões os pagamentos a crédito não alteram o saldo mas ficam registados no cartão e podem ser saldados em qualquer momento através da operação `settleCredit`. Tal como o construtor da sua super-classe, o construtor da classe `GoldCardClass` recebe também o número de cartão.
A constante `SIZE` define a dimensão inicial do vector `transactions`.
Implemente os seguintes métodos das classes `BasicCardClass` e `GoldCardClass`.

a) O construtor `BasicCardClass(String cardNr)`.
b) O construtor `GoldCardClass(String cardNr)`.
c) O método `boolean payDebit(int amount)` da classe `BasicCardClass`.
d) O método `void settleCredit()` da classe `GoldCardClass`. Tenha em atenção que a variável `balance` da classe `BasicCardClass` é privada.

Grupo II

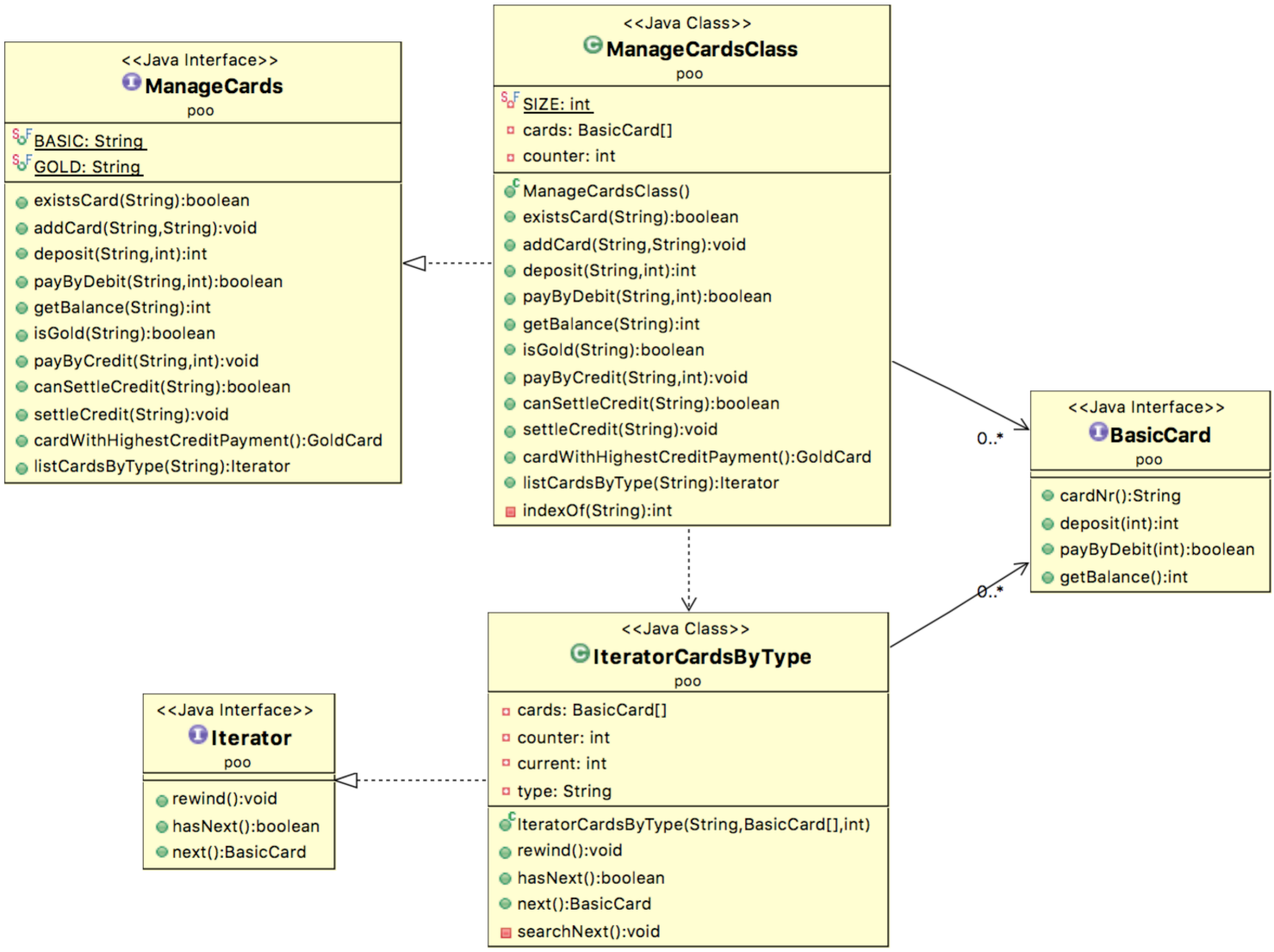

Figura 2 – Aplicação para gestão de cartões.

A interface `ManageCards` representa uma aplicação para a gestão de cartões de pagamentos (Fig. 2). As constantes `BASIC` e `GOLD` definem os dois tipos de cartões.

- `existsCard(String cardNr)` devolve `true` se o número de cartão `cardNr` já existir na aplicação e `false` caso contrário;
- `addCard(String cardNr, String type)` adiciona um novo cartão de tipo `type` (cujo valor é `BASIC` ou `GOLD`). Este método só é executado se a pré-condição `existsCard(cardNr)` for falsa;
- Os métodos `deposit`, `payByDebit`, `getBalance`, `payByCredit`, `canSettleCredit`, `settleCredit` são similares aos métodos das interfaces `BasicCard` e `GoldCard`, sendo que recebem como argumento adicional o número de cartão (`cardNr`). Estas operações só serão executadas se a pré-condição `existsCard(cardNr)` for verdadeira;
- `isGold(String cardNr)` devolve `true` caso o cartão com o número `cardNr` seja um cartão do tipo `GoldCard` e `false` caso contrário. Esta operação só será executada se a pré-condição `existsCard(cardNr)` for verdadeira;
- `cardWithHighestCreditPayment()` devolve o cartão que tem a compra a crédito de maior valor (por saldar). Caso não existam cartões de crédito ou não existam pagamentos a crédito por saldar em nenhum cartão, o método devolve `null`. Em caso de empate deve devolver o cartão mais antigo;
- `listCardsByType(String type)` devolve um iterador que percorre os cartões do tipo `type`.

Implemente os seguintes métodos da classe `ManagerCardsClass`:

a) O método `void addCard(String cardNr, String type).`

b) O método `boolean isGold(String cardNr).`

c) O método `void settleCredit(String cardNr).`

d) O método `GoldCard cardWithHighestCreditPayment().`

e) O método `Iterator listCardsByType(String type).`

Grupo III

Pretende-se implementar um programa que faça a gestão de pacientes, corpo clínico e consultas de uma clínica médica. Esta aplicação deve permitir:

- Definir utilizadores de diferentes tipos: médico e paciente;
- Adicionar uma consulta de rotina de um médico, a um paciente, numa data;
- Adicionar uma consulta de urgência de um médico, a um paciente, numa data;
- Listar todas as consultas de um dado tipo (rotina ou urgência) realizadas por um dado utilizador (médico ou paciente);
- Listar todos os utilizadores de um dado tipo;
- Listar todas as consultas agendadas num dado período de tempo. Por exemplo, deve ser possível listar as consultas de 22 de Abril de 2016 a 24 de Abril de 2016.

Cada utilizador tem um nome, morada e número de contribuinte. Para o corpo clínico é mantida a especialidade. Para os pacientes é mantida a idade e número da segurança social.
As consultas têm uma data e um relatório. As consultas de urgência têm uma classificação (muito urgente, urgente, pouco urgente, não urgente) e o tempo de espera em minutos. As consultas de rotina mantêm a periodicidade (e.g. mensal, trimestral). Assuma que as datas são representadas por Strings.

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