Programação Orientada pelos Objectos

Exame de Recurso (Duração 2h30m)

LEI 2013/2014

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.

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

Nestes 3 grupos vamos implementar algumas das classes necessárias à construção de programas para a gestão de um cartão de descontos de um supermercado. No primeiro grupo, faremos a implementação completa de algumas classes (Fig. 1). No segundo grupo, faremos a implementação parcial de uma classe que representa a colecção dos cartões de desconto (Fig. 2). No terceiro grupo, realizamos alguns testes unitários e praticamos o tratamento de excepções.

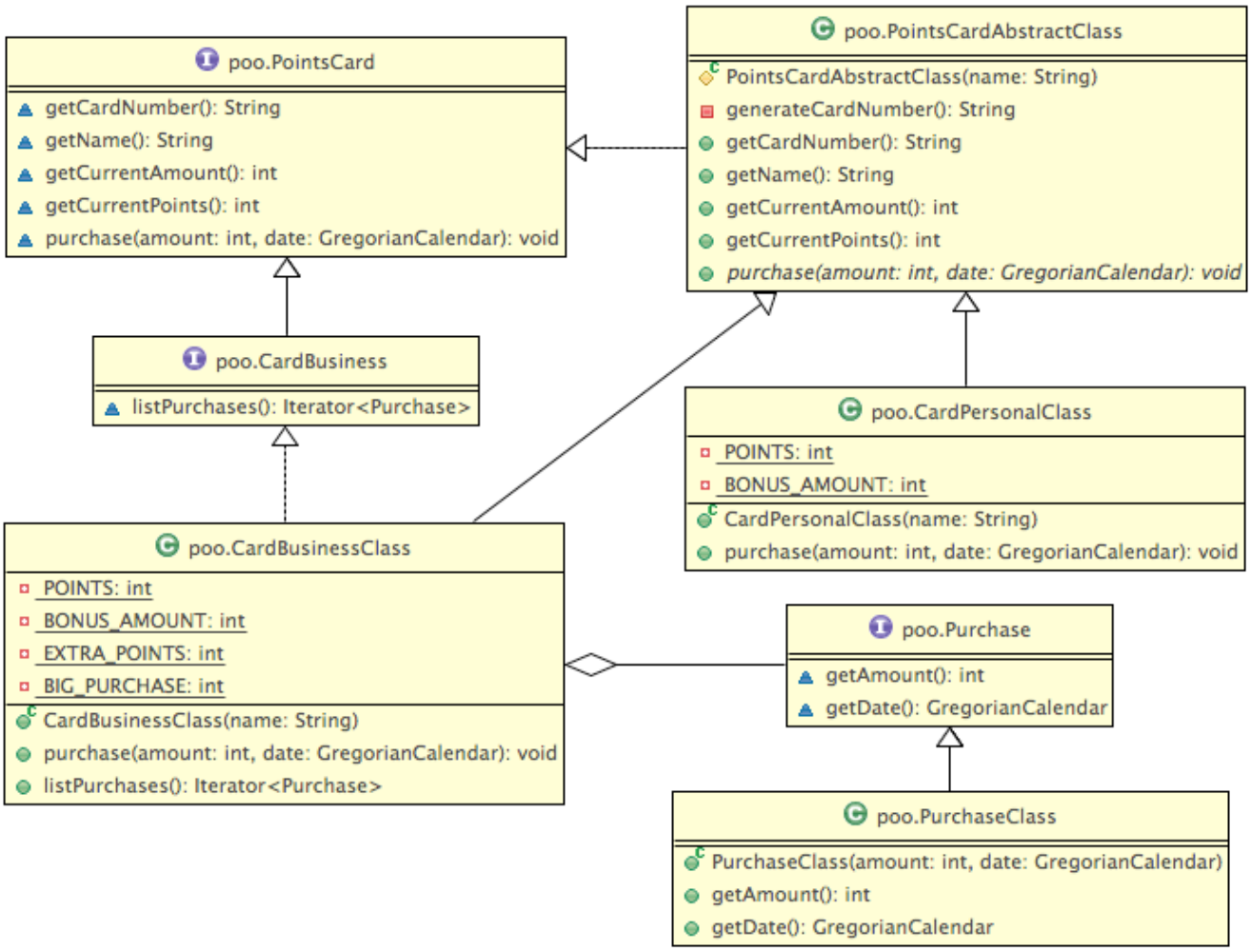

Figura 1: Cartão de descontos.

A interface `PointsCard` representa o cartão de desconto. O cartão de descontos funciona da seguinte forma, cada euro gasto no supermercado é convertido num ponto no cartão. Depois de colectar um número pré-determinado de pontos esses pontos são revertidos num desconto em euros, que poderá ser usado no pagamento de uma ou várias compras. Para simplificar, assuma que todas as quantias são valores em euros. Na interface `PointsCard` o método `getCardNumber` devolve uma `String` com o identificador único do cartão. O método `getName` devolve o nome do titular do cartão. O método `getCurrentAmount` devolve a quantia acumulada no cartão. O método `getCurrentPoints` devolve o número de pontos actualmente no cartão. O método `purchase` regista uma compra no cartão, aumentando o número de pontos acumulados e caso o cartão tenha atingido um número de pontos pré-definidos esses pontos são convertidos num desconto monetário.

A classe abstracta `PointsCardAbstractClass` implementa a interface `PointsCard`. Na classe `PointsCardAbstractClass` temos um construtor protegido `PointsCardAbstractClass`. Este construtor recebe o nome do titular. Os métodos implementados nesta classe abstracta obedecem à especificação já apresentada durante a descrição da interface `PointsCard`.

A classe abstracta `PointsCardAbstractClass` é especializada em duas classes concretas, `CardPersonalClass` e `CardBusinessClass`. Em que `CardPersonalClass` representa um cartão em nome pessoal, enquanto que `CardBusinessClass` representa um cartão em nome de uma empresa. Uma das diferenças entre estes cartões é a periodicidade entre a conversão dos pontos em desconto.

Um cliente individual tem que colectar 500 pontos para obter 5 euros de desconto. Enquanto um cliente empresarial tem que colectar 100 pontos para obter 1 euro de desconto. Estes valores são representadas pelas constantes `POINTS` e `BONUS_AMOUNT` nas classes `CardPersonalClass` e `CardBusinessClass`. Para além disso, os clientes empresariais recebem 50 pontos extra (constante `EXTRA_POINTS`) para compras superiores a 200 euros (constante `BIG_PURCHASE`).

Outra diferença entre as classes `CardPersonalClass` e `CardBusinessClass`, é que a classe `CardBusinessClass` mantém uma lista de todas as compras efectuadas. A classe `CardBusinessClass` implementa o método adicional `listPurchases` (da interface `CardBusiness`) o qual permite iterar sobre todas as compras feita por um cliente empresarial, ordenadas pela ordem de inserção (a mais recente primeiro). Note que não lhe vai ser pedido para implementar a classe `PurchaseClass`.

Grupo I – PointsCardAbstractClass, CardPersonalClass e CardBusinessClass

Neste grupo tem que apresentar a **implementação completa** das seguintes classes:

a) Implemente a classe abstracta `PointsCardAbstractClass`, excepto o método privado `generateCardNumber()`.
b) Implemente a classe `CardPersonalClass`.
c) Implemente a classe `CardBusinessClass`.

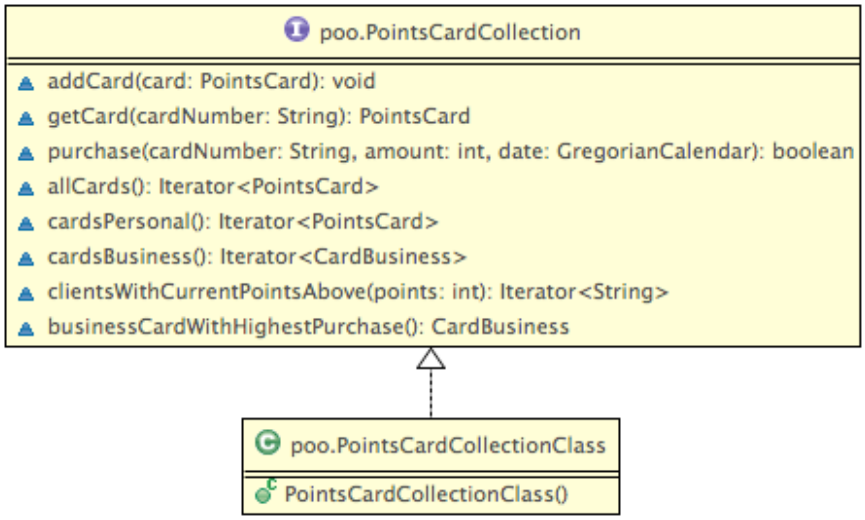

Figura 2: A interface PointsCardCollection e a sua implementação.

A Fig. 2 apresenta a interface e a classe necessária à resolução do Grupo II. A interface `PointsCardCollection` representa uma colecção de cartões de desconto. O método `addCard` adiciona um novo cartão, o qual é recebido como argumento. Se o número de cartão já existir na colecção este método não faz nada. O método `getCard` devolve o cartão identificada por `cardNumber`, ou `null` caso o número de cartão não exista. O método `purchase` regista uma nova compra com a quantia `amount`, na data `date`, no cartão identificado por `cardNumber`, devolvendo `true` caso o cartão exista e `false` caso contrário. O método `allCards` devolve um iterador para todos os cartões de clientes individuais, por ordem alfabética do número de cartão. O método `cardsPersonal` devolve um iterador para todos os cartões de clientes individuais, ordenado por ordem decrescente da quantia acumulada no cartão. O método `cardsBusiness` devolve um iterador para os cartões empresariais, também ordenado por ordem decrescente da quantia acumulada no cartão. O método `cardsWithPointsAbove` devolve um iterador para os nomes associados a cartões com mais pontos do que o número de pontos `points` recebido como argumento. Este iterador deve permitir visitar a colecção por ordem alfabética do nome associado ao cartão. O método `businessCardWithHighestPurchase` devolve o objecto `CardBusiness` com a compra de maior valor entre todos os cartões empresariais. Em caso de empate não é relevante qual dos cartões é devolvido.

Grupo II – PointsCardCollectionClass

Assuma que existem milhares de aderentes aos cartões de desconto e que se pretende optimizar as pesquisas por número de cartão, a eficiência das listagens e a eficiência do método `businessCardWithHighestPurchase`. Nesta classe, implemente apenas:

a) O construtor de modo a que no início não existam cartões registados.
b) O modificador `void addCard(PointsCard).`
c) O modificador `boolean purchase(String, amount, GregorianCalendar).`
d) O selector `Iterator<CardBusiness> cardsBusiness()`. Assuma que está disponível uma classe `CardByAmountComparator` (que implementa a interface `Comparator<PointsCard>`), que permite estabelecer uma relação de ordem parcial entre os cartões com base na quantia acumulada no cartão.
e) O selector `Iterator<String> clientsWithCurrentPointsAbove(int).`
f) O selector `CardBusiness businessCardWithHighestPurchase().`

Grupo III – Testes unitários e excepções

a) Pretende-se alterar o método `purchase(String, int, GregorianCalendar)` da classe `PointsCardsCollectionClass` para uma abordagem baseada em excepções verificadas. Esboce uma implementação de `purchase(String, int, GregorianCalendar)` baseada na excepção verificada `CardNumberNotFoundException`. Para esse efeito, apresente a implementação completa de `CardNumberNotFoundException` e o esboço da nova implementação do método `purchase (String, int, GregorianCalendar)`, incluindo o novo cabeçalho do método.

b) Implemente uma operação de teste ao comando `purchase(int)`, da classe `CardBusinessClass`, usando o JUnit. O seu teste deve construir dois cartões de desconto, e efectuar compras sucessivas com um dos cartões, verificando que a partir da altura esperada os pontos de desconto são convertidos num desconto monetário (em euros), enquanto que nada acontece ao outro cartão. A operação de teste deve permitir verificar que no início ambos os cartões de desconto não tem quaisquer pontos ou descontos, e que os pontos vão sendo correctamente atribuídos, e que a atribuição dos descontos acontece no momento certo para um dos cartões, sem que isso afecte o outro.

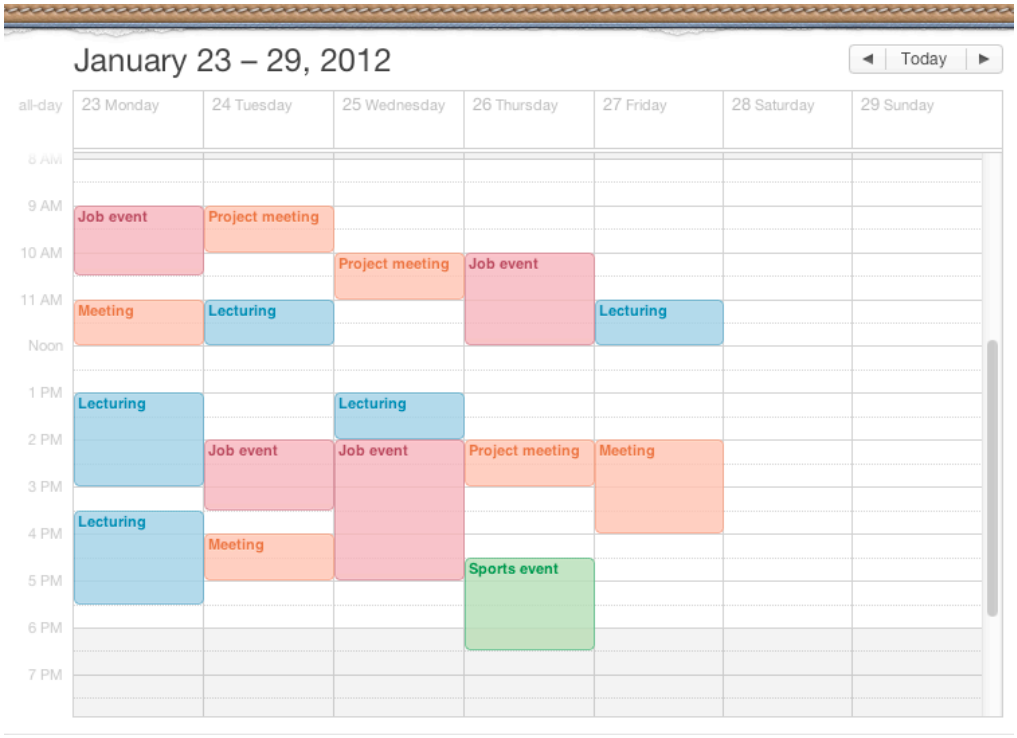

Figura 3: Agenda pessoal de eventos.

Grupo IV – Modelação

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

- A agenda permite definir diferentes categorias de eventos. Por exemplo, uma categoria de eventos para casa, outra para eventos da escola, do emprego, etc.
- A visualização e/ou impressão das várias categorias pode ser feita de modo seletivo, isto é, uma ou mais categorias em simultâneo e, além disso, com um determinado horizonte temporal associado. Esse horizonte tem um enquadramento em termos de ano, mês, semana e dia.
- Um evento, para um determinado dia, tem uma descrição textual e uma delimitação temporal.

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).

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