TÉCNICO LISBOA

Projecto de Programação com Objectos

29 de Setembro de 2021

Esclarecimento de dúvidas:

- Consultar sempre o corpo docente atempadamente: presencialmente ou através do endereço **po-tagus@disciplinas.tecnico.ulisboa.pt**.
- Não utilizar fontes de informação não oficialmente associadas ao corpo docente (podem colocar em causa a aprovação à disciplina).
- Não são aceites justificações para violações destes conselhos: quaisquer consequências nefastas são da responsabilidade do aluno.

Requisitos para desenvolvimento, material de apoio e actualizações do enunciado (ver informação completa na secção **[Projecto]** no Fénix):

- O material de apoio é de uso obrigatório e não pode ser alterado.
- Verificar atempadamente (mínimo de 48 horas antes do final de cada prazo) os requisitos exigidos pelo processo de avaliação.

Processo de avaliação (ver informação completa nas secções **[Projecto]** e **[Método de Avaliação]** no Fénix):

- Datas: **2021/10/08 23:59** (UML); **2021/10/29 23:59** (intercalar); **2021/11/12 23:59** (final); **2021/11/16-2021/11/17** (teste prático).
- Não serão consideradas quaisquer alterações aos ficheiros de apoio disponibilizados: eventuais entregas dessas alterações serão automaticamente substituídas durante a avaliação da funcionalidade do código entregue.
- Trabalhos não entregues no Fénix até final do prazo têm classificação 0 (zero) (não são aceites outras formas de entrega).
- A avaliação do projecto pressupõe o compromisso de honra de que o trabalho foi realizado pelos alunos correspondentes ao grupo de avaliação.
- **Fraudes na execução do projecto terão como resultado a exclusão dos alunos implicados do processo de avaliação**.

O objectivo do projecto é desenvolver uma aplicação de gestão do inventário de um entreposto de recursos naturais e seus derivados (formados por um ou mais produtos ou seus derivados). O entreposto concede prémios de fidelização aos bons parceiros (que lhe compram e vendem produtos), baseando-se no volume de negócio (compras e vendas). A funcionalidade da aplicação inclui, entre outras acções, manipular dados de produtos para negociar, registar/manipular dados de parceiros, registar/manipular transacções de compra e venda e outras e fazer pesquisas várias sobre a informação armazenada.

Este documento está organizado da seguine forma. A secção 1 apresenta as entidades do domínio da aplicação a desenvolver. As funcionalidades da aplicação a desenvolver são descritas nas secções 3 e 4. A secção 2 descreve os requisitos de desenho que a solução deve oferecer.

Neste texto, o tipo **negrito** indica um literal (i.e., é exactamente como apresentado); o símbolo ␣ indica um espaço; e o tipo *itálico* indica uma parte variável (i.e., uma descrição).

1 Entidades do Domínio

Nesta secção descrevem-se as várias entidades que vão ser manipuladas no contexto da aplicação a desenvolver.

Existem vários conceitos importantes neste contexto: o entreposto, produtos, parceiros, transacções e tempo. Os produtos e os parceiros possuem uma chave única (cadeia de caracteres, não havendo distinção entre maiúsculas e minúsculas), ou seja, ”aBc”e ”Abc”não representam chaves distintas. As transacções possuem uma chave inteira.

Os conceitos listados não são os únicos possíveis no modelo a realizar por cada grupo e as suas relações (assim como relações com outros conceitos não mencionados) podem depender das escolhas do projecto.

1.1 Produtos

O entreposto mantém os vários lotes de produtos disponíveis para venda no entreposto. Um produto pode ser simples ou derivado. Um produto derivado tem uma receita de fabrico. A receita corresponde à discriminação dos identificadores dos componentes (simples ou derivados) que formam o produto e respectivas quantidades. Não é possível a definição de receitas cujos componentes não sejam previamente conhecidos.

Os produtos, tanto simples, como derivados, são comprados, vendidos, ou fabricados (no caso dos produtos derivados) em lotes. Cada lote tem um fornecedor (o parceiro a quem é comprado o produto), o número de unidades disponíveis do produto no lote e o preço de cada unidade. Podem existir vários lotes do mesmo produto, com o mesmo fornecedor, com preços iguais ou diferentes. O preço de um produto é sempre um número em vírgula flutuante não negativo. O preço dos produtos é definido no acto da compra pelo entreposto. Este processo é uniforme para todos os produtos, excepto onde indicado. Quando um lote é esgotado, é removido do registo de dados.

O entreposto pode realizar operações de agregação e de desagregação. Uma operação de agregação consiste em criar um produto derivado a partir das existências dos seus componentes. O preço dos produtos derivados é calculado com base nos preços dos componentes, de acordo com a receita, tendo um agravamento multiplicativo (número positivo em vírgula flutuante), definido pela receita, relativo ao valor acrescentado associado à combinação. Por exemplo: a água é feita de hidrogénio (2x) e de oxigénio (1x). O seu preço é: $P_{H_2O} = (1 + \alpha)(2P_H + P_O)$ (onde P_x são preços e α é o factor de agravamento, e.g. 0.1, correspondente a um aumento de 10% sobre o valor dos componentes). A reserva de recursos para construção de determinada quantidade de produto derivado contempla a totalidade da construção (e.g., não é possível respeitar um pedido de 10 unidades de água se apenas estiverem disponíveis 8 unidades de oxigénio). Em geral, operações de consumo de existências começam sempre pelas existências de menor preço, consumindo um ou mais lotes, total ou parcialmente, no processo.

É possível desagregar um produto derivado e recuperar os seus componentes. Não há perdas de produtos nesta operação, mas pode haver perda de valor: o valor do produto derivado pode ser superior à soma dos valores dos preços dos seus componentes. O preço dos componentes desagregados é calculado como indicado na secção de desagregação (ver 1.3.3.. O parceiro que pede a operação de desagregação paga o diferencial de valor, caso seja positivo. Esta operação é registada como uma transacção de desagregação (com o valor do diferencial, mesmo que negativo).

1.2 Parceiros

Cada parceiro tem um nome (cadeia de caracteres) e uma morada (cadeia de caracteres). Associada a cada parceiro existe ainda informação relativa às suas transacções. Um parceiro tem ainda um estatuto (e.g., **Elite**, etc- – ver abaixo), o qual tem impacto na sua relação com o entreposto. Dado um parceiro, é possível aceder ao historial das suas transacções (ver a seguir).

1.3 Transacções

As transacções são identificadas por um número (inteiro), atribuído de forma automática pela aplicação. Este identificador começa em 0 (zero), sendo incrementado quando se regista uma nova transacção. A sequência de identificadores é partilhada por todos os tipos de transacções. Todas as transacções têm uma data de pagamento.

O entreposto realiza com os parceiros transacções de compra (do entreposto aos parceiros), venda (do entreposto aos parceiros) e desagregações (pelo entreposto a pedido dos parceiros). As desagregações: podem ser vistas como compras e vendas combinadas, para efeitos de progressão do estatuto dos parceiros. Uma desagregação corresponde a uma venda do produto a desagregar e à compra dos componentes que resultam da desagregação.

As compras são sempre pagas instantaneamente. As vendas podem ser pagas pelos parceiros mais tarde (excepto no contexto de uma desagregação: nesse caso, são pagas imediatamente).

1.3.1 Compras (do entreposto a parceiros)

Uma compra está associada a um parceiro e envolve uma ou mais unidades de um produto fornecido pelo parceiro em causa. O custo unitário do produto é definido pelo parceiro no momento da transacção. Quando se faz uma compra deve considerar-se que a compra é paga imediatamente e que as existências do entreposto são actualizadas considerando os produtos comprados. É mantida informação sobre cada produto comprado, o parceiro que o forneceu e o preço que foi pago e o número de unidades disponíveis.

1.3.2 Vendas (do entreposto a parceiros)

Uma venda está associada a um parceiro e envolve uma ou mais unidades de um único produto. As unidades podem ser provenientes de vários lotes e ter preços diferentes, dependendo dos parceiros que forneceram o produto. Cada venda tem uma data limite de pagamento: primeiro realiza-se a venda e só depois é que se procede ao seu pagamento.

Se o produto a adquirir for derivado e não existir em quantidade suficiente no entreposto, a quantidade em falta pode ser fabricada a partir de outros produtos, de acordo com a receita correspondente ao produto em venda. O preço a pagar é o definido pela reserva dos componentes individuais (com a selecção do preço mais baixo, etc.) e o correspondente factor de valor acrescentado (como indicado em 1.1). Note-se que, na venda de múltiplas unidades de um produto derivado, o processo de construção deve ser repetido, pelo que cada uma das unidades do produto derivado pode ter custo diferente das anteriores. Caso falte algum componente da receita, então a operação de agregação deve ser abortada indicando o primeiro produto (pela ordem da receita) que viola a condição de disponibilidade.

O preço a pagar por um parceiro (numa operação de venda por parte do entreposto) depende ainda do tempo que demora a realizar o pagamento. São considerados os seguintes períodos (N é 5 para produtos simples e 3 para produtos derivados):

- P_1 - Até N dias antes do limite de pagamento ($data\_limite\_de\_pagamento - data\_actual \geq N$).

- P_2 - Até à data limite ($0 \leq data\_limite\_de\_pagamento - data\_actual < N$).
- P_3 - Até N dias depois da data limite ($0 < data\_actual - data\_limite\_de\_pagamento \leq N$).
- P_4 - Após N dias depois da data limite ($data\_actual - data\_limite\_de\_pagamento > N$).

1.3.3 Desagregações de produtos (pelo entreposto a pedido de parceiros)

Uma desagregação é uma transacção associada a um parceiro e envolve um único produto. Um parceiro pode pedir uma desagregação para permitir outras operações sobre os componentes do produto a desagregar. Se o produto a desagregar for simples, a acção não tem qualquer efeito.

As desagregações são operações sobre o estado do entreposto por parte de um parceiro: a desagregação remove uma quantidade de produto derivado das existências do entreposto (como se fosse uma venda) e introduz nas existências do entreposto os seus componentes (como se fosse uma compra ao parceiro envolvido). O preço de inserção é o menor preço nos lotes disponíveis para esse produto. Se não existirem lotes disponíveis, o preço é o maior preço associado a esse produto no histórico de operações do entreposto. O parceiro paga ao entreposto a diferença (positiva) entre os valores inicial (o valor da quantidade de produto não desagregado) e final (o valor das quantidades de componentes obtidas com base na desagregação). Se a diferença for um valor negativo, a desagregação tem um custo de 0 (zero).

1.4 Notificações

Quando o entreposto recebe um novo produto, os parceiros devem ser colocados como entidades interessadas em receber notificações sobre eventos a ele associados. Em qualquer momento, um parceiro pode activar ou desactivar as notificações relativas a um produto. Os eventos a considerar são os seguintes: (i) quando o produto passa de stock 0 (zero) para outro valor (positivo); (ii) quando aparece um lote mais barato de um produto. As notificações são compostas pelo identificador do produto e pela descrição da notificação: **NEW**, para novas existências de produtos, mas não quando se regista um novo produto; e **BARGAIN**, para disponibilidade de preços mais baixos. As notificações são registadas nos parceiros que as recebem.

A entrega de notificações deve ser flexível e prever vários meios de entrega, e.g., correio postal, SMS, email, entre outras. Deve ainda ser possível adicionar novos tipos de entidades interessadas em receber notificações sem que isso implique alterações no código já realizado. O meio de entrega por omissão corresponde a registar a notificação na aplicação.

1.5 Contabilização de Pontos (Parceiros)

As multas e os descontos aplicam-se apenas no pagamento de transacções de venda. Os valores pagos (instantaneamente) nas transacções de desagregação também dão direito à contabilização de pontos (apenas quando é positivo).

Existem três classificações distintas de parceiros: **Normal**, **Selection** e **Elite**. A classificação de um parceiro tem impacto nas multas e descontos a aplicar no pagamento de uma venda.

Quando um parceiro paga uma venda dentro do prazo, acumula um número de pontos correspondente a 10 vezes o valor pago. Não há contabilização de pontos em pagamentos atrasados. A verificação do atraso é realizada quando se realiza o pagamento de uma venda.

Após a contabilização dos pontos, se o número de pontos de um parceiro for superior a 25000 então o parceiro é **Elite**. Se o número de pontos não for superior a 25000 mas for maior do que 2000, então o parceiro é **Selection**. Se um parceiro se atrasa no pagamento da venda, é despromovido: um parceiro **Elite** passa a **Selection** se o pagamento ocorrer com um atraso de pagamento superior a 15 dias (perdendo ainda 75% dos pontos acumulados); um parceiro **Selection** passa a **Normal** se o pagamento ocorrer com um atraso de pagamento superior a 2 dias (perdendo ainda 90% dos pontos acumulados). Se um parceiro **Normal** se atrasa num pagamento, perde os pontos que tem.

As multas e os descontos a aplicar no pagamento de uma venda dependem do estatuto do parceiro e do período em que o pagamento é realizado ($P1, P_2, P_3$ e P_4):

- P_1 - independentemente do estatuto do parceiro, a multa é 0 e o desconto é 10%;
- P_2 - para parceiros com o estatuto **Normal**, a multa e o desconto são ambos 0; para **Selection** a multa é 0 e o desconto é 5% até 2 dias antes da data limite e 0 após; para **Elite** a multa é 0 e o desconto é 10%;
- P_3 - para parceiros com o estatuto **Normal**, a multa é 5% diários e o desconto é 0%; para **Selection** o desconto é 0 e a multa é 0 até um dia depois da data limite e 2% diários após; para **Elite** a multa é 0 e o desconto é 5%;
- P_4 - para parceiros com o estatuto **Normal** a multa é 10% diários e o desconto é 0%; para **Selection** o desconto é 0 e a multa é 5% diários; para **Elite** a multa e o desconto são 0;

1.6 Gestão de Tempo

A data é representada por um número inteiro e tem inicialmente o valor 0 (zero). A data pode começar com outro valor se se recuperar o estado da aplicação a partir de um suporte persistente. Os avanços de data são valores inteiros positivos que representam o número de dias a avançar. Não é possível andar andar para trás no tempo.

2 Requisitos de Desenho

Devem ser possíveis extensões ou alterações de funcionalidade com impacto mínimo no código já produzido para a aplicação. O objectivo é aumentar a flexibilidade da aplicação relativamente ao suporte de novas funções. Assim, deve ser possível:

- Definir novas entidades que desejem ser notificadas da alteração do estado dos produtos;
- Adicionar novos modos de entrega de mensagens (notificações);
- Adicionar novas políticas de recompensa de parceiros;
- Adicionar novas formas de consulta.

Embora na especificação actual não seja possível remover entidades, a inclusão desta funcionalidade deve ser prevista, por forma a minimizar o impacto da sua futura inclusão.

2.1 Funcionalidade da Aplicação

A aplicação permite gerir a informação sobre as entidades do modelo, Possui ainda a capacidade de preservar o seu estado (não é possível manter várias versões do estado da aplicação em simultâneo). Inicialmente, a aplicação apenas tem informação sobre as entidades que foram carregados no arranque da aplicação. É possível saber os saldos do entreposto (diferencial entre vendas e compras). Existe um saldo disponível, correspondente à diferença entre as vendas realmente pagas e as compras, e um saldo contabilístico, correspondente à diferença entre o valor contabilístico das vendas (pagas ou não e considerando descontos/penalizações à data da consulta de saldo) e as compras. Note-se que as desagregações são consideradas combinações de compras e vendas, pelo que também influenciam o saldo do entreposto.

Note-se que não é necessário concretizar a aplicação de raiz . Já é fornecido algum código, nomeadamente, o esqueleto das classes necessárias para concretizar os menus e respectivos comandos a utilizar na aplicação a desenvolver, a classe `ggc.app.App`, que representa o ponto de entrada da aplicação a desenvolver, algumas classes do domínio da aplicação e um conjunto de excepções a utilizar durante o desenvolvimento da aplicação. A interface geral do core já está parcialmente concretizada na classe `ggc.core.WarehouseManager` e outras fornecidas (cujos nomes devem ser mantidos), devendo ser adaptadas onde necessário.

A classe `ggc.core.WarehouseManager` deverá ser finalizada por cada grupo e deverá representar a interface geral do domínio da aplicação desenvolvida. Por esta razão, os comandos a desenvolver na camada de interface com o utilizador devem utilizar exclusivamente esta classe para aceder às funcionalidades suportadas pelas classes do domínio da aplicação (a serem desenvolvidas por cada grupo na package `ggc.core`). Desta forma será possível concretizar toda a interacção com o utilizador completamente independente da concretização das entidades do domínio. As excepções a criar na camada de serviços (ou seja nos comandos) já estão todas definidas no package `ggc.app.exception`. O package `ggc.core.exception` contém algumas excepções a utilizar na camada do domínio. Caso seja necessário podem ser definidas novas excepções neste package para representar situações anómalas que possam ocorrer nas entidades do domínio. Finalmente, será ainda necessário concretizar de raiz algumas classes do domínio da aplicação necessárias para suportar a operação da aplicação.

2.2 Serialização

É possível guardar e recuperar o estado actual da aplicação, preservando toda a informação relevante do domínio da aplicação e que foi descrita na secção 1.

3 Interação com o utilizador

Esta secção descreve a **funcionalidade máxima** da interface com o utilizador. Em geral, os comandos pedem toda a informação antes de proceder à sua validação (excepto onde indicado). Todos os menus têm automaticamente a opção **Sair** (fecha o menu).

As operações de pedido e apresentação de informação ao utilizador **devem** realizar-se através dos objectos *form* e *display*, respectivamente, presentes em cada comando. No caso de um comando utilizar mais do que um formulário para interagir com

o utilizador então será necessário criar pelo menos mais uma instância de `Form` durante a execução do comando em causa. As mensagens são produzidas pelos métodos das bibliotecas de suporte (**po-uilib** e de classes presentes no esqueleto da aplicação. **Não** podem ser definidas novas mensagens. Potenciais omissões devem ser esclarecidas com o corpo docente antes de qualquer concretização.

A apresentação de valores monetários (preços, saldos, etc.) é sempre feita com arredondamento ao inteiro mais próximo.

Por forma a garantir uma independência entre as várias camadas da aplicação não deve haver código de interacção com o utilizador no núcleo da aplicação (*ggc.core*). Desta forma, será possível reutilizar o código do núcleo da aplicação (onde é concretizado o domínio da aplicação) com outras concretizações da interface com o utilizador. Para garantir um melhor desenho da aplicação toda alógica de negócio deve estar concretizada no núcleo da aplicação e não na camada de interação com o utilizador. Assim potencia-se a reutilização de código e o código está concretizado nas entidades a que diz respeito.

As excepções usadas no código de interacção com o utilizador para descrever situações de erro, excepto se indicado, são subclasses de `pt.tecnico.uilib.menus.CommandException` e devem ser lançadas pelos comandos (sendo depois tratadas automaticamente pela classe já existente `pt.tecnico.po.ui.Menu`). Estas excepções já estão definidas no package (fornecido) `ggc.app.exception`. Outras excepções não devem substituir as fornecidas nos casos descritos. É da responsabilidade de cada grupo definir as excepções que achar necessárias no contexto das entidades do domínio da aplicação. As excepções disponíveis em `ggc.app.exception`. apenas devem ser utilizadas no código de interacção com o utilizador.

Note-se que o programa principal e os comandos e menus, a seguir descritos, já estão parcialmente concretizados no package `ggc.app` e nos seus sub-packages (por exemplo, `ggc.app.main`). Estas classes são de uso obrigatório e estão disponíveis no arquivo `ggc-skeleton.jar` disponível na secção *Projecto* da página da cadeira.

A apresentação de listas de entidades do domínio (parceiros, transacções, etc.) faz-se por ordem crescente da respectiva chave: dependendo dos casos, a ordem pode ser numérica ou lexicográfica (UTF-8), não havendo distinção entre maiúsculas e minúsculas.

Sempre que existe uma interacção com o utilizador, seja para pedir dados seja para apresentar dados, é indicado (caso seja necessário) qual é o método da classe `Message` do package em causa que é responsável por devolver a mensagem a apresentar ao utilizador.

3.1 Menu Principal

As acções deste menu permitem gerir a salvaguarda do estado da aplicação, ver e alterar a data actual e abrir submenus. A lista completa é a seguinte: **Abrir**, **Guardar** , **Mostrar data actual**, **Avançar data actual**, **Gestão de Produtos**, **Gestão de Parceiros**, **Gestão de Transacções**, **Consultas** e **Mostrar Saldo Global**. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `ggc.app.main.Message`.

Os comandos que vão concretizar as funcionalidades deste menu já estão parcialmente concretizados nas várias classes do package `ggc.app.main`: `DoOpen`, `DoSave`, `DoDisplayDate`, `DoAdvanceDate`, `DoOpenMenuProducts`, `DoOpenMenuPartners`, `DoOpenMenuTransactions`, `DoOpenMenuLookups` e `DoShowGlobalBalance`.

3.1.1 Salvaguarda do estado actual

Inicialmente, a aplicação está vazia ou tem apenas informação sobre as entidades que foram carregados no arranque via ficheiro textual (ver 4).

O conteúdo da aplicação (que representa o estado relevante actualmente em memória da aplicação) pode ser guardado para posterior recuperação (via serialização Java: `java.io.Serializable`). Na leitura e escrita do estado da aplicação, devem ser tratadas as excepções associadas. A funcionalidade é a seguinte:

Abrir – Carrega os dados de uma sessão anterior a partir de um ficheiro previamente guardado (ficando este ficheiro associado à aplicação, para futuras operações de salvaguarda). Pede-se o nome do ficheiro a abrir (utilizando a cadeia de caracteres devolvida por `openFile()`). Caso ocorra um problema na abertura ou processamento do ficheiro, deve ser lançada a excepção `FileOpenFailedException`. A execução bem sucedida desta opção substitui toda a informação da aplicação.

Guardar – Guarda o estado actual da aplicação no ficheiro associado. Se não existir associação, pede-se o nome do ficheiro a utilizar ao utilizador (utilizando a cadeia de caracteres devolvida por `newSaveAs()`), ficando a aplicação associada a este ficheiro.

Note-se que a opção `Abrir` não suporta a leitura de ficheiros de texto (estes apenas são utilizados na inicialização da aplicação). A opção `Sair` **nunca** guarda o estado da aplicação, mesmo que existam alterações.

3.1.2 Mostrar data actual

A data actual do sistema é apresentada através da mensagem devolvida por `currentDate()`.

3.1.3 Avançar data actual

O número de dias a avançar é pedido utilizando a mensagem devolvida por `requestDaysToAdvance()`. Esta operação deve lançar a excepção `InvalidDateException` se o número de dias a avançar for inválido.

3.1.4 Gestão e consulta de dados da aplicação

A gestão e consulta de dados da aplicação são realizadas através das seguintes opções:

Menu de Gestão de Produtos – Abre o menu de gestão de produtos.

Menu de Gestão de Parceiros – Abre o menu de gestão de parceiros.

Menu de Gestão de Transacções – Abre o menu de gestão de transacções.

Menu de Consultas – Abre o menu de consultas (pesquisas).

3.1.5 Mostrar saldo global

Esta opção apresenta os valores (inteiros) correspondentes aos saldos disponível e contabilístico da empresa. Embora internamente o valor dos saldos esteja representados em vírgula flutuante, a apresentação é arredondada ao inteiro mais próximo. A apresentação dos saldos faz-se utilizando a mensagem devolvida por `currentBalance()`.

3.2 Menu de Gestão de Produtos

Este menu permite efectuar operações sobre a base de dados de produtos oferecidos pela empresa. A lista completa é a seguinte: **Visualizar todos os produtos**, **Visualizar todos os lotes**, **Visualizar os lotes fornecidos por um dado parceiro** e **Visualizar os lotes de um dado produto**. Os comandos deste menu já estão parcialmente concretizados nas classes do package `ggc.app.products`: `DoShowAllProducts`, `DoShowAvailableProducts`, `DoShowBatchesByPartner` e `DoShowProductBatches`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `Message` deste package.

Sempre que no contexto de uma operação for pedido o identificador de um produto (utilizando a mensagem devolvida por `requestProductKey()`) e o produto não existir, a operação deve lançar a excepção `UnknownProductKeyException` (excepto no processo de registo). Sempre que for pedido o identificador de um parceiro (utilizando a mensagem devolvida por `requestPartnerKey()`) e o parceiro não existir, a operação lança a excepção `UnknownPartnerKeyException`. Na ocorrência de excepções (estas ou outras), as operações não devem ter qualquer efeito.

3.2.1 Visualizar todos os produtos

Apresenta todos os produtos disponibilizados pela empresa, um produto por linha. O formato de apresentação de cada tipo de produto é o seguinte:
Produtos simples:

```
idProduto|preço-máximo|stock-actual-total
```

Produtos derivados (é apresentado o produto e a lista de componentes e respectivas quantidades da sua receita):

```
idProduto|preço-máximo|stock-actual-total|componente-1:quantidade-1#...#componente-n:quantidade-n
```

Exemplo:

```
ÁGUA|5000|800|HIDROGÉNIO:2#OXIGÉNIO:1
HIDROGÉNIO|200|5000
OXIGÉNIO|1200|2500
```

Em ambos os casos, o campo `stock-actual-total` pode ser 0 (zero).

3.2.2 Visualizar todos lotes

Apresenta os vários lotes disponíveis para cada produto. O formato de apresentação de cada lote de produto é o seguinte (cada linha indica um lote):

```
idProduto|idParceiro|preço|stock-actual
```

Exemplo:

```
ÁGUA|EPAL|1150.00|80
HIDROGÉNIO|Cryostuffs|110.50|2000
HIDROGÉNIO|Loja␣da␣esquina|190.50|3000
OXIGÉNIO|Cryostuffs|820.50|2500
```

Note-se que podem existir vários lotes para o mesmo produto. Neste caso, a ordenação é, primeiro, pelo identificador do produto e, depois, pelo identificador do parceiro. Se um parceiro fornecer vários lotes do mesmo produto, apresentam-se por ordem crescente de preço e existências.

3.2.3 Visualizar os lotes fornecidos por um dado parceiro

É pedido o identificador do parceiro e são apresentados os lotes por ele fornecidos. A ordenação é como indicada para a listagem de todos os lotes.

3.2.4 Visualizar os lotes de um dado produto

É pedido o identificador do produto e são apresentados os lotes conhecidos para esse produto. A ordenação é como indicada para a listagem de todos os lotes.

3.3 Menu de Gestão de Parceiros

Este menu permite efectuar operações sobre a base de dados de parceiros. A lista completa é a seguinte: **Mostrar parceiro**, **Mostrar todos os parceiros**, **Registar parceiro**, **Activar/desactivar notificações de um produto**, **Mostrar transacções de compra com parceiro** e **Mostrar transacções de venda (e desagregação) com parceiro**. Os comandos deste menu já estão parcialmente concretizados nas classes do package `ggc.app.partners`: `DoShowPartner`, `DoShowAllPartners`, `DoRegisterPartner`, `DoToggleProductNotifications`, `DoShowPartnerAcquisitions` e `DoShowPartnerSales`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `ggc.app.partners.Message`.

Sempre que for pedido o identificador de um parceiro (utilizando a mensagem devolvida por `requestPartnerKey()`) e o parceiro não existir, deve ser lançada a excepção `UnknownPartnerKeyException` (excepto no processo de registo). No processo de registo, caso o identificador indicado já exista, deve ser lançada a excepção `DuplicatePartnerKeyException`.

3.3.1 Mostrar parceiro

É pedido o identificador do parceiro e apresentada a sua informação. Note-se que compras e vendas são ponto de vista do entreposto e não do parceiro. Os valores dos preços são apresentados como inteiros (valores arredondados). O formato de apresentação é o seguinte (o valor das vendas efectuadas – não inclui desagregações – refere-se ao momento da venda; o valor das vendas pagas (não inlui igualmente desagregações) refere-se ao valor realmente pago):

```
id|nome|endereço|estatuto|pontos|valor-compras|valor-vendas-efectuadas|valor-vendas-pagas
```

O estatuto corresponde a `NORMAL`, `SELECTION` ou `ELITE`, conforme o caso. Após esta linha, são apresentadas as notificações do parceiro recebidas através do modo de entrega por omissão, pela ordem em que foram enviadas pela aplicação, utilizando o seguinte formato:

```
tipo-de-notificação|idProduto|preço-do-produto
```

Após esta visualização, considera-se que o parceiro fica sem notificações registadas.

3.3.2 Mostrar parceiros

Apresenta informações sobre todos os parceiros. O formato de apresentação de cada parceiro segue o mesmo formato descrito na secção 3.3.1, mas não se apresentam as notificações dos parceiros.

3.3.3 Registar parceiro

São pedidos o identificador do parceiro, o nome (`requestPartnerName()`) (uma cadeia de caracteres) e o endereço do parceiro (`requestPartnerAddress()`) (cadeia de caracteres) e regista-se o novo parceiro. Quando um parceiro é registado, aceita notificações relativas a todos os produtos. Se já existir um parceiro com o mesmo identificador, deve ser lançada a excepção `DuplicatePartnerKeyException`, não se realizando o registo.

3.3.4 Activar/desactivar notificações de um produto

São pedidos o identificador do parceiro e o identificador do produto (`requestProductKey()`). Se as notificações relativas ao produto estavam activas, passam a estar inactivas, e vice-versa. Se não existir um produto com o identificador indicado, então esta operação deve lançar a excepção `UnknownProductKeyException`.

3.3.5 Mostrar transacções de compra com parceiro

É pedido o identificador do parceiro e apresentadas todas as transacções de compra com ele realizadas. O formato de apresentação é como descrito na secção 3.4.1.

3.3.6 Mostrar transacções de venda (e desagregação) com parceiro

É pedido o identificador do parceiro e apresentadas todas as transacções de venda (incluindo as de desagregação) com ele realizadas. O formato de apresentação é como descrito na secção 3.4.1.

3.4 Menu de Gestão de Transacções

Este menu apresenta as operações relacionadas com transacções. A lista completa é a seguinte: **Visualizar**, **Registar Desagregação**, **Registar Venda**, **Registar Compra** e **receber Pagamento de Venda**. Estes comandos já estão parcialmente concretizaos nas classes do package `ggc.app.transactions`: `DoShowTransaction`, `DoRegisterBreakdownTransaction` `DoRegisterSaleTransaction`, `DoRegisterAcquisitionTransaction`, e `DoReceivePayment`. Todos os métodos correspondentes às mensagens de diálogo utilizadas nas operações deste menu estão definidos na classe `ggc.app.transactions.Message`.

No contexto de cada comando deste menu deve ser tido em conta o seguinte. Sempre que é pedido o identificador de um parceiro (`requestPartnerKey()`) e o parceiro indicado não existir, a operação em causa deve lançar a excepção `UnknownPartnerKeyException`. Sempre que é pedido o identificador de um produto (`requestProductKey()`), a operação deve lançar a excepção `UnknownProductKeyException`, se o produto indicado não existir. Sempre que é pedido o identificador de uma transacção (`requestTransactionKey()`), a excepção `UnknownTransactionKeyException` deve ser lançada se a transacção indicada não existir.

3.4.1 Visualizar

O sistema pede o identificador da transacção a visualizar e de seguida apresenta-a. O formato de apresentação depende do tipo da transacção. O campo *valor-base* presentes nos vários formatos a utilizar corresponde ao valor da transacção sem multas/descontos. O formato para uma venda a um parceiro é o seguinte:

```
VENDA|id|idParceiro|idProduto|quantidade|valor-base|valor-a-pagamento|data-limite|data-pagamento
```

O campo `valor-a-pagamento` corresponde ao valor que foi realmente pago (venda já paga) ou que será pago, considerando possíveis multas/descontos à data da visualização (caso a venda não esteja ainda paga). A data de pagamento (e o separador correspondente) só é apresentada se a venda já tiver sido paga.

Se a transacção corresponder a uma compra a um parceiro, apresenta-se com o seguinte formato:

```
COMPRA|id|idParceiro|idProduto|quantidade|valor-pago|data-pagamento
```

Se a transacção for uma desagregação, então deve ser apresentada, primeiro a "venda"do produto derivado, seguida da "compra"dos componentes obtidos na desagregação. O formato para N componentes é o seguinte:

```
DESAGREGAÇÃO|id|idParceiro|idProduto|quantidade|valor-base|valor-pago|data-pagamento|idC1:q1:v1#...#idCN:qN:vN
```

O campo *valor base* de uma desagregação é o valor real da transacção (diferencial), ou seja corresponde à diferença entre a compra e a venda. O campo *valor-pago* corresponde ao valor realmente pago pelo parceiro e é um valor não negativo, podendo ser 0 caso o valor base seja negativo. Neste formato, os vários campos `idCx, qx, vx` são, respectivamente, o identificador, a quantidade e o valor do componente x. Cada componente é separado do seguinte por **#**.

3.4.2 Registar Desagregação (pelo entreposto a pedido de um parceiro)

Para registar uma desagregação, é pedido o identificador do parceiro que requisita a transacção e o identificador do produto a desagregar e a respectiva quantidade (requestAmount()). Se a quantidade for superior às existências actuais, deve ser lançada a excepção `UnavailableProductException` e não se realiza a desagregação. Se o produto a desagregar não for derivado, esta operação não tem efeito. A actualização dos produtos do entreposto, bem como o respectivo valor, tem lugar logo após o registo da desagregação, ou seja, considera-se que a operação é instantânea. Esta transacção regista o produto de origem e os resultantes, bem como os respectivos valores.

3.4.3 Registar Venda (do entreposto a um parceiro)

Para registar uma venda, é pedido o identificador do parceiro que vai comprar o produto, a data limite para o pagamento (`requestPaymentDeadline()`), o identificador do produto a vender e a respectiva quantidade (`requestAmount()`). Se a quantidade for superior às existências actuais, esta operação aborta, lançando a excepção `UnavailableProductException` e a venda não é realizada.

A determinação das existências começa com o preço mais baixo e vai prosseguindo por todos os parceiros que tenham o produto disponível. O preço a pagar é determinado pelo preço das várias fracções.

Se a venda corresponder a um produto derivado e não existir produto suficiente nos lotes disponíveis, a quantidade em falta pode ser fabricado a partir de componentes de outros lotes. As existências dos componentes têm de ser suficientes para satisfazer a totalidade das necessidades de produto derivado a fabricar. O preço é calculado como descrito acima. A actualização dos produtos do entreposto tem lugar logo após o registo da venda, ou seja, considera-se que a venda é instantânea.

3.4.4 Registar Compra (do entreposto a um parceiro)

Para registar uma compra, primeiro é pedido o identificador do parceiro a quem se realiza a compra, o identificador do produto que se está a comprar, o preço do produto (requestPrice()) e a respectiva quantidade (`requestAmount()`). O sistema mantém depois esta informação: em particular, se vários parceiros fornecerem o mesmo produto, podem fazê-lo com preços diferentes. O entreposto poderá, então, vender um dado produto com diferentes preços.

Se o identificador do produto for desconhecido, então é perguntado ao parceiro se quer introduzir uma receita para um produto derivado (`requestAddRecipe()`). Se a resposta for negativa, trata-se de um produto simples. Caso contrário, é pedido o número de componentes da receita (requestNumberOfComponents()), o valor do agravamento (requestAlpha()) e, em ciclo, os identificadores e quantidades dos vários componentes: os identificadores são pedidos como para qualquer outro identificador de produto; as quantidades são pedidas utilizando a mensagem devolvida por `requestAmount()`. A actualização de existências dos produtos do entreposto tem lugar logo após o registo da compra, ou seja, considera-se que a compra é instantânea. A actualização do saldo do entreposto também é assumida como instantânea, i.e., assume-se que a compra é paga a pronto.

3.4.5 Receber Pagamento de Venda (do entreposto a um parceiro)

Apenas é permitido o pagamento de vendas a parceiros. É pedido o identificador da transacção a pagar. Tentativas de pagamento de transacções que não correspondam a vendas não produzem nenhum resultado. Se a venda já tiver sido paga, não é realizada nenhuma acção.

3.5 Menu de Consultas

Este menu apresenta as operações relacionadas com consultas. A lista completa é a seguinte: `Mostrar produtos com preço abaixo de limite` e `Mostrar facturas pagas por parceiro`. Estas operações já estão parcialmente implementados nas classes do package `ggc.app.lookups`: `DoLookupPaymentsByPartner` e `DoLookupProductsUnderTopPrice`. Todos os métodos correspondentes às mensagens de diálogo para este menu estão definidos na classe `gcc.app.lookups.Message`. A apresentação de resultados é como indicado nos casos já descritos de apresentação das várias entidades.

Sempre que é pedido o identificador de um parceiro (`requestPartnerKey()`), a excepção `UnknownPartnerKeyException` é lançada se o parceiro indicado não existir. Sempre que é pedido o identificador de produto (`requestProductKey()`), é lançada a excepção `UnknownProductKeyException`, se o produto indicado não existir. A apresentação de resultados é como indicado nos casos já descritos de apresentação das várias entidades.

3.5.1 Mostrar lotes de produtos com preço abaixo de limite

Apresenta os lotes de produtos cujo preço é inferior ao indicado pelo utilizador. A operação deve pedir o valor limite pretendido (requestPriceLimit()) e de seguida apresentar todos os lotes de produtos cujo preço é inferior ao preço indicado. É apresentado um lote por linha,

3.5.2 Mostrar transacções pagas por parceiro

Pede-se o identificador do parceiro e apresentam-se as transacções do parceiro que já estão pagas (uma transacção por linha).

4 Leitura de Dados a Partir de Ficheiros Textuais

Além das opções de manipulação de ficheiros descritas na secção §3.1.1, é possível iniciar a aplicação com um ficheiro de texto especificado pela propriedade Java com o nome `import`. Este ficheiro contém a descrição dos parceiros e produtos a carregar no estado inicial da aplicação. Cada linha deste ficheiro textual descreve uma dada entidade a carregar no estado inicial do sistema. Pode assumir que não existem entradas mal-formadas nestes ficheiros. A codificação dos ficheiros a ler é garantidamente `UTF-8`.

As várias entidades têm os formatos descritos abaixo. Assume-se que os campos de cada entidade que correspondem a uma cadeia de caracteres não podem conter o carácter separador | e que o preço é um número. Cada linha tem uma descrição distinta mas segue um dos seguintes formatos:

```
PARTNER|id|nome|endereço
BATCH_S|idProduto|idParceiro|preço|stock-actual
BATCH_M|idProduto|idParceiro|preço|stock-actual|agravamento|componente-1:quantidade-1#...#componente-n:quantidade-n
```

para parceiros, produtos simples e produtos derivados, respectivamente. Sugere-se a utilização do método `String.split` para o processamento preliminar destas linhas. De seguida apresenta-se um exemplo de conteúdo do ficheiro inicial:

```
PARTNER|S1|Toshiba|Tokyo,␣Japan
PARTNER|W2|Pedraria␣Fonseca|Oeiras,␣Portugal
PARTNER|P1|Lages␣do␣Chão|Lisboa,␣Portugal
PARTNER|P3|O'Brian␣Rocks|Köln,␣Germany
PARTNER|R2|Jorge␣Figueiredo|Lisboa,␣Portugal
PARTNER|E4|Filomena␣Figueiredo|Lisboa,␣Portugal
PARTNER|ER|Abdul␣Figueiredo|Casablanca,␣Morocco
PARTNER|O9|Hellen␣Figueiredo|San␣Francisco,␣CA,␣USA
PARTNER|MM|John␣Figueiredo|Wellington,␣New␣Zealand
PARTNER|M1|Rohit␣Figueiredo|New␣Delhi,␣India
BATCH_S|HIDROGÉNIO|S1|200|5000
BATCH_S|OXIGÉNIO|P1|1200|2500
BATCH_M|ÁGUA|P3|800|10000|0.1|HIDROGÉNIO:2#OXIGÉNIO:1
```

Note-se que o programa **nunca** produz ficheiros com este formato.

5 Execução dos Programas e Testes Automáticos

Usando os ficheiros `test.import`, `test.in` e `test.out`, é possível verificar automaticamente o resultado correcto do programa. O primeiro ficheiro representa o ficheiro de texto com a descrição do estado inicial das entidades do sistema, o segundo ficheiro simula um utilizador, contentdo os caracteres a serem fornecdidos ao programa durante a sua execução e o terceiro ficheiro irá conter todos os caracteres escritos pelo programa (via bilioteca *po-uilib*) durante a sua execução. Note-se que pode ser necessária a definição apropriada da variável de ambiente `CLASSPATH` (ou da opção equivalente `-cp` do comando `java`), para localizar as classes do programa, incluindo a que contém o método correspondente ao ponto de entrada da aplicação (`ggc.app.App.main`. As propriedades são tratadas automaticamente pelo código de apoio.

```
java -Dimport=test.import -Din=test.in -Dout=test.outhyp gcc.app.App
```

Assumindo que aqueles ficheiros estão no directório onde é dado o comando de execução, o programa produz o ficheiro de saída `test.outhyp`. Em caso de sucesso, os ficheiros da saída esperada (`test.out`) e obtida (`test.outhyp`) devem ser iguais. A comparação pode ser feita com o comando:

```
diff -b test.out test.outhyp
```

Este comando não deve produzir qualquer resultado quando os ficheiros são iguais. Note-se, contudo, que este teste não garante o correcto funcionamento do código desenvolvido, apenas verifica alguns aspectos da sua funcionalidade.

6 Notas de Concretização

Tal como indicado neste documento, algumas classes fornecidas como material de apoio, são de uso obrigatório e **não** podem ser alteradas: as várias classes `Message`, `Label` e de excepção presentes em diferentes subpackages de `ggc.app`. Outras dessas classes são de uso obrigatório e têm de ser alteradas: os diferentes comandos presentes em subpackages de `ggc.app` e algumas classes do domínio da aplicação já parcialmente concretizadas em `ggc.core`.

A serialização Java usa as classes da package `java.io`, em particular, a interface`java.io.Serializable` e as classes de leitura `java.io.ObjectInputStream` e escrita `java.io.ObjectOutputStream` (entre outras).