CI2 Parte prática. Duração: 2h00m

Natal: tempo de amor, paz, amizade, e… presentes, muitos presentes. É essa a sina de quem trabalha num hipermercado, onde as campanhas de descontos nos presentes para a pequenada começam cedo e levam milhares de pessoas a comprar freneticamente mais presentes do que aquilo que poderiam comprar noutras circunstâncias. E depois vêm os embrulhos… E com eles, este enunciado!

O hipermercado Tontinenti investiu num serviço que pretende tornar toda esta experiência de compra e embrulho de presentes menos frustrante. Assim, após pagar os artigos (presentes ou outros), os clientes transportam-nos dentro de sacos (pilhas de artigos). Os clientes podem depositar os artigos que pretendem ver embrulhados numa mesa (um vector), por ordem de chegada, que serão depois processados pela equipa de pós-venda. Os presentes poderão ser entregues, já embrulhados, em casa. Contudo, há clientes que preferem esperar e levar eles próprios (alguns dos) presentes logo para casa.

Na equipa de pós-venda há um funcionário encarregue de priorizar os artigos a embrulhar. Para tal, este funcionário procura manter a parte inicial da mesa ordenada por um critério que dá primazia aos artigos cujos clientes estão à espera deles, por ordem de chegada (os outros serão entregues em casa no dia seguinte). Contudo, como há clientes que compram mesmo muitos presentes, um cliente que compre poucos (por exemplo, um só) não deve ficar demasiado tempo à espera de vez – procura-se assim agradar a todos com um serviço mais expedito.

Após ordenar a mesa, o funcionário vai colocando os artigos num tapete rolante (uma fila de tamanho limitado), que serão depois processados por especialistas em embrulhos. Os presentes a expedir para casa no dia seguinte seguem para um segundo tapete rolante (outra fila); os que correspondem a clientes que estão à espera são entregues de imediato ao cliente.

As classes ***Article***, ***Purchase*** e ***AfterSalesService*** estão parcialmente definidas a seguir. A classe ***Article*** representa um artigo. A classe ***Purchase*** representa a compra de um cliente, agregando os artigos adquiridos numa lista de sacos (modelados como pilhas) de tamanho *BAG_SIZE*. A classe ***AfterSalesService*** representa o serviço pós-venda, contendo uma mesa de presentes que serão processados, uma fila (com tamanho máximo) de presentes a embrulhar e outra fila de presentes que serão entregues em casa.

```
class Article {
        const long client;
        const long barCode;
        bool present;
        bool deliverHome;
        int presentNumber;
        int arrivalNumber;
public:
        Article(long client, long barCode);
        long getClient() const;
        bool getPresent() const;
        void setPresent(bool present);
        bool getDeliverHome() const;
        void setDeliverHome(bool deliverHome);
        int getPresentNumber() const;
        int getArrivalNumber() const;
};

class Purchase {
        const long client;
        list< stack<Article*> > bags;
public:
        static const stack<Article*>::size_type
                BAG_SIZE = 5;
        Purchase(long client);

        //--- TO IMPLEMENT
        Article* createArticle(long barCode,
                bool present, bool deliverHome);
        void putInBag(Article* article);
        vector<Article*> popPresents();
};
```

```
class AfterSalesService {
        vector<Article*> presentsTable;
        const queue<Article*>::size_type
                toWrapQueueSize;
        queue<Article*> toWrap;
        queue<Article*> toDeliver;
public:
        AfterSalesService(int toWrapQueueSize);
        void dropPresentsOnTable(
                vector<Article*> presents);

        //--- TO IMPLEMENT
        vector<Article*> pickPresentsFromTable(
                long client);
        void sortArticles();
        void enqueueArticles();
        Article* wrapNext();
};
```

a) [**2.5 valores**] Implemente na classe ***Purchase*** o membro-função

```
Article* Purchase::createArticle(long barCode, bool present, bool deliverHome)
```

que cria um novo artigo com base nos dados do cliente. O construtor de ***Article*** deve ser invocado passando o membro-dado ***client*** da classe ***Purchase***, juntamente com o código de barras ***barCode***. O argumento ***present*** indica se o artigo é um presente (deve atualizar o membro-dado respetivo na classe ***Article***). O argumento ***deliverHome*** indica se o artigo vai ser entregue em casa (deve atualizar o membro-dado respetivo na classe ***Article***). A função deve retornar o artigo criado.

b) [**3 valores**] Implemente na classe ***Purchase*** o membro-função

```
void Purchase::putInBag(Article* article)
```

que adiciona à lista de sacos de compras do cliente o artigo passado no argumento. Adicionar o artigo significa colocá-lo no último saco (que é uma pilha) da lista ***bags***. Caso esse saco esteja cheio (ver ***Purchase::BAG_SIZE***), deve ser adicionado um novo saco à lista, que conterá o artigo.

c) [**3 valores**] Após a compra, o cliente desloca-se à zona de embrulhos. Implemente na classe ***Purchase*** o membro-função

```
vector<Article*> Purchase::popPresents()
```

que retira dos sacos de compras todos os artigos que são presentes. Os restantes artigos devem voltar a ficar nos sacos onde estavam, pela mesma ordem relativa. Os sacos onde havia presentes deixam portanto de estar cheios. A função deve retornar um vector com os presentes, na ordem em que foram encontrados (percorrendo os sacos do início para o fim do vector ***bags***).

d) [**3 valores**] Os presentes dos clientes são colocados na mesa ***presentsTable*** (ver função ***dropPresentsOnTable***), com vista a serem embrulhados. Contudo, há sempre clientes impacientes que se cansam de esperar pelos embrulhos. Implemente na classe ***AfterSalesService*** o membro-função

```
vector<Article*> AfterSalesService::pickPresentsFromTable(long client)
```

que retira da mesa todos os presentes do cliente ***client***. A função deve retornar esses presentes.

e) [**3 valores**] De modo a oferecer um serviço mais expedito e que agrade a todos, um funcionário gere a ordem dos presentes que estão na mesa. Implemente na classe ***AfterSalesService*** o membro-função

```
void AfterSalesService::sortArticles()
```

que ordena os 10 primeiros artigos que estão na mesa, assim: (i) artigos para entregar em casa (membro-dado ***deliverHome***) devem passar para o fim; (ii) se dois artigos para levar (não para entregar em casa) têm número de presente (relativo ao cliente, ver membro-dado ***presentNumber***) diferindo em mais do que 2 unidades, o artigo com número menor deve passar à frente (por exemplo, o 1º presente de um cliente deve ser embrulhado antes do 4º presente de outro cliente); (iii) em todos os outros casos, a ordem dos artigos na mesa deve ser mantida (membro-dado ***arrivalNumber***).

f) [**3 valores**] Implemente na classe ***AfterSalesService*** o membro-função

```
void AfterSalesService::enqueueArticles()
```

que coloca na fila ***toWrap*** os primeiros artigos existentes na mesa ***presentsTable***, tendo em conta que a fila ***toWrap*** deve ter no máximo ***AfterSalesService::toWrapQueueSize*** elementos.

g) [**2.5 valores**] Implemente na classe ***AfterSalesService*** o membro-função

```
Article* AfterSalesService::wrapNext()
```

que processa (retira) o primeiro presente da fila ***toWrap***. Se for um artigo para entregar em casa (membro-dado ***deliverHome***), o presente deve ser colocado na fila ***toDeliver***. Caso contrário, deve ser retornado pela função (que de outra forma deve retornar ***NULL***).