CI1 Parte prática. Duração: 1h30m

Pretende-se implementar um sistema de gestão e apoio de compras à distância através do telefone (um *call center*) para uma cadeia nacional de retalho de eletrodomésticos que permita agilizar o processo de encomenda à distância para o Natal 2020, fornecendo um meio alternativo à realização de compras *online*.

Para tal, foi implementado um sistema de apoio ao cliente (classe **CallCenter***)* que permite que o cliente obtenha informações sobre um dado produto e faça a sua encomenda de uma forma simplificada através do telefone (classe **OrderCall**).

Todas as chamadas (classe ***Call***) estão associadas a um código (*codCall*) e ao número de identificação fiscal do cliente (*nifCode*). As chamadas para a realização de encomenda (**OrderCall**) são ainda caracterizadas pelo número da encomenda (*numOrder*) e pelo código postal (*zipCode*) do cliente.

A classe **CallCenterAgent** identifica os trabalhadores do *call center*, e inclui um identificador (*idAgent*), o nome do trabalhador (*nameAgent*) e a fila de chamadas que este está encarregue de atender (*myWaitingCalls*).

A classe **CallCenter** identifica o departamento de apoio ao cliente do retalhista e inclui informação sobre os trabalhadores do *call center* (lista *agents*) e informação sobre as chamadas atendidas para realização de encomendas (vetor *attendedOrderCalls*). Por solicitação do departamento de marketing, as chamadas para a realização de encomendas estão agrupadas por código postal e dispostas por ordem da última ocorrência. Assim, os elementos do vetor *attendedOrderCalls* são pilhas de chamadas relativas a encomendas, sendo que cada pilha inclui encomendas efetuadas por clientes com o mesmo código postal.

As classes ***CallCenter***, ***Call***, ***OrderCall*** e ***CallCenterAgent*** estão parcialmente definidas a seguir.

```
class Call {
    string codCall;
    unsigned nifCode;
public:
    Call(string codeCall, unsigned nifCod);
    virtual ~Call();
    string getZipCode() const;
    //---
};
```

```
class OrderCall: public Call {
    string orderNum;
    string zipCode;
public:
    OrderCall(string codeCall, unsigned
nifCod, string orderN, string zCode);

    //---
};
```

```
class CallCenterAgent {
    unsigned idAgent;
    string nameAgent;
    queue<Call *> myWaitingCalls;
public:
    CallCenterAgent(string nm, unsigned id);
    //---
};
```

```
class CallCenter {
    list<CallCenterAgent> agents;
    vector<stack<OrderCall *> >
                    attendedOrderCalls;
public:
    CallCenter();

    //---
};
```

a) [**2,5 valores**] Implemente na classe **CallCenter** o membro-função

*queue<Call *> removeAgent (string name)*

Esta função remove da lista `agents` o trabalhador de nome `name`, retornando a fila de chamadas que estavam a cargo desse agente (`myWaitingCalls`). Se não existir um trabalhador com nome `name`, a função deve lançar a exceção `NoAgentException`. Esta exceção já está implementada

b) [**2,5 valores**] Implemente na classe **CallCenter** o membro-função:

void sortAgents()

que ordena a lista de agentes (`agents`) por ordem decrescente de ocupação (número de chamadas a aguardar serem atendidas). Em caso de empate, a ordenação é realizada por ordem alfabética do nome do agente (`nameAgent`). Em caso de empate, a ordenação é realizada por ordem crescente do código do agente (`idAgent`).

c) [**3 valores**] Implemente na classe **CallCenter** o membro-função

void addAgent (string name, unsigned idAgent)

Esta função adiciona no final da lista `agents` o trabalhador de nome `name` e código `idAgent`. Para tentar reduzir a fila de chamadas associadas aos restantes trabalhadores, algumas das chamadas serão realocadas a este novo trabalhador. Assim, deve ser movida para a fila de chamadas `myWaitingCalls` deste novo trabalhador, a primeira chamada de cada uma das filas de chamadas dos restantes trabalhadores, se estas filas tiverem 2 ou mais elementos.

d) [**3 valores**] Implemente na classe **CallCenter** o membro-função:

*bool putInLessBusyAgent (Call *newCall)*

Esta função aceita a chamada `newCall` e adiciona-a à fila do agente menos ocupado (com menor número de chamadas em espera). No caso de empate, adiciona ao agente que aparece primeiro na lista.

Se não existir nenhum agente na lista, a função retorna `false`. Caso contrário, retorna `true`.

Nota: Na classe **CallCenterAgent** já está implementado o membro-função `void addCall(Call *c)` que adiciona a chamada `c` no final da lista de chamadas do agente.

e) [**3 valores**] Implemente na classe **CallCenterAgent** o membro-função

void moveToFront (string codCall1)

que coloca a chamada de código `codCall1` no início da fila de chamadas em espera. As restantes chamadas na fila mantêm a sua posição relativa. Se não existir nenhuma chamada de código `codCall1` na fila `myWaitingCalls`, a fila mantém-se inalterada.

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

*void addOrder(OrderCall *call1)*

que adiciona ao vetor `attendedOrderCalls` a chamada `call1`. Se já existir uma pilha com chamadas relativas a encomendas de código postal igual ao de `call1`, a `call1` deve ser adicionada a essa pilha. Senão, será criada uma nova pilha com `call1` e esta pilha é adicionada no final do vetor `attendedOrderCalls`.

g) [**3 valores**] Implemente na classe **CallCenter** o membro-função

unsigned processOrderZip(string zip1, unsigned n)

que processa `n` encomendas com o código postal `zip1`. Processar uma encomenda significa removê-la da pilha respetiva no vetor `attendedOrderCalls`. Se a pilha se tornar agora vazia, deve ser eliminada do vetor. A função retorna o número de encomendas processadas (note que o número de encomendas com código postal igual a `zip1` pode ser inferior a `n`).