CI3 Parte prática. Duração: 2h00m

Em um supermercado, os clientes realizam suas compras com o auxílio de cestos com rodas, onde depositam os seus produtos para transportá-los até ao caixa, onde realizam o pagamento. Como os cestos têm uma base muito pequena, os itens neles depositados acabam for ficar empilhados. Os clientes podem utilizar mais do que um cesto durante as suas compras, tendo de transportá-los todos até ao caixa onde realizará o pagamento.

A classe **Item** guarda informação sobre os produtos seleccionados, que serão depositados nos cestos para serem transportados até ao caixa. Os itens têm uma designação de *produto* (maçã, arroz, lixivia) e um *tipo* (fruta, grão, limpeza, higiene) para além de *preço*, *volume* e *peso*. Todos estes atributos são membros-dado públicos.

Os objetos da classe **Cesto** contêm uma pilha com os itens neles depositados. Poderão também ser considerados como membros-dado volume e volume máximo, peso e peso máximo.

A informação sobre um **Cliente** inclui o nome e a idade. Cada cliente poderá utilizar vários cestos durante as suas compras.

A classe Supermercado é caracterizada por duas filas, uma normal e outra prioritária, representando os caixas onde os clientes poderão realizar o pagamento das suas compras. A fila prioritária só admite clientes com idade igual ou superior a 65 anos que, entretanto, poderão utilizar quaisquer das duas filas, optando sempre pela que tiver menos clientes. Os outros clientes, com idade inferior a 65 anos, só poderão utilizar a fila normal.

As classes **Item**, **Cesto**, **Cliente** e **Supermercado** estão parcialmente definidas a seguir.

```
class Item {
public:
  int preco;
  int volume;
  int peso;
  string produto;
  string tipo;
  Item(int umPreco, int umVolume,
int umPeso, string umProduto, string
umTipo);
};


class Cesto {
  int max_volume, volume;
  int max_peso, peso;
  stack<Item> itens;
public:
  Cesto();
  Cesto(vector<Item> unsItens);
  stack<Item> getItens() const;
  Item& topItem();
  void popItem();
  void pushItem(const Item& umItem);
  bool empty() const;

  int novoItem(const Item& umItem);
};
```

```
class Cliente {
  list<Cesto> cestos;
  string nome;
  int idade;
public:
  Cliente();
  Cliente(string umNome, int umaIdade);
  string getNome() const;
  int getIdade() const;
  int addCesto(Cesto& umCesto);
  int numCestos() const;
  list<Cesto> getCestos() const;
  int numeroItens() const;
  int valorItens() const;
  int trocarItem(Item& novoItem);
  void organizarCestos();
  vector<string> contarItensPorTipo();
  int novoItem(const Item& umItem);
};


class Supermercado {
  queue<Cliente> filaNormal;
  queue<Cliente> filaPrioritaria;
public:
  Supermercado();
  int tamanhoFilaNorma() const;
  int tamanhoFilaPrioritaria() const;
  int novoCliente(Cliente& umCliente);
  Cliente& sairDaFila(string
umNomeDeCliente);
};
```

a) [**2.5 valores**] Implemente na classe **Cliente** os membros-função:

int numeroItens() const

int valorItens() const

O método *numeroItens()* retorna o número total de itens em todos os cestos do cliente, enquanto o método *valorItens()* retorna o valor total correspondente à soma dos preços de todos os itens nos cestos. Caso não existam ainda itens nos cestos, ou o cliente ainda não tenha um cesto, os métodos retornarão 0 (zero).

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

int trocarItem(Item& novoItem)

Esta função substitui, nos cestos do cliente, todos os itens com mesma designação de `produto`, que tenham preço superior ao preço do `novoItem`. Esta função retorna o número de itens substituídos.

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

void organizarCestos()

Para facilitar o transporte dos cestos, o cliente utiliza *organizarCestos()* para colocar os itens mais pesados no fundo do cesto, e os mais leves mais acima, na pilha. Os itens são mantidos nos seus cestos originais, apenas a sua ordem é alterada, dos mais pesados, mais ao fundo, aos mais leves, mais ao topo. A ordem dos cestos na lista `cestos` também deve ser mantida.

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

vector<string> contarItensPorTipo()

Este membro-função conta o número total de itens de cada `tipo` em todos os cestos do cliente. Os tipos de itens podem ser de higiene, limpeza, grãos, fruta, etc. A função retorna um vector de strings com tuplas "<*tipo*> <*quantidade*>", por exemplo "limpeza 4", a indicar que existem 4 itens de limpeza nos cestos do cliente.

e) [**3 valores**] Implemente nas classes **Cesto** e **Cliente** o membro-função:

int novoItem(Item& umItem)

Na classe **Cesto**, este método inclui `umItem` na pilha, caso haja capacidade para tal, em termos de volume e de peso; o volume e o peso do novo item, somados ao volume e peso acumulados no cesto, respectivamente, não poderão ultrapassar `max_volume` e `max_peso` do cesto. Este membro-função, na classe Cesto, retornará o número de itens do cesto, depois da inserção, caso seja bem sucedida, ou 0 (zero), caso não se possa adicionar `umItem` no cesto. Na classe **Cliente**, este método tenta inserir `umItem` em um dos cestos do cliente, pela invocação de *novoItem* em cada cesto. Caso não se consiga inserir o novo item num dos cestos do cliente, o cliente fará uso de um novo cesto vazio, onde depositará o novo item. Em Cliente, esta função retorna o número total de cestos do cliente.

f) [**2.5 valores**] Implemente na classe **Supermercado** o membro-função:

int novoCliente(Cliente& umCliente)

Esta função insere `umCliente` numa das filas de espera para pagamento (`filaNormal`, `filaPrioritaria`). Caso `umCliente` tenha idade igual ou superior a 65 anos, escolherá uma das filias, normal ou prioritária, a depender do menor número de cliente em espera. Os clientes com idade inferior a 65 anos, só poderão escolher a fila normal. A função retorna o número total de clientes em espera, em ambas as filas.

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

Cliente sairDaFila(string umNomeDeCliente)

Esta função simula a saída de um cliente com `umNomeDeCliente` da fila do caixa, prioritária ou normal, antes de realizar o seu pagamento, por exemplo, porque esqueceu de apanhar um item. A função procura na `filaNormal` e na `filaPrioritaria` o cliente de `umNomeDeCliente`. A função devolve o cliente, caso o cliente esteja em espera, em uma das filas, ou lança uma exceção **ClienteInexistente**, caso o cliente não esteja em qualquer das filas. A classe de exceção deverá ser implementada; observe o ficheiro de teste para identificar a estrutura apropriada à esta classe de exceção.