CI3 Parte prática. Duração: 2h00m

O Sr. Silva, dono do restaurante Dishes&Dishes, é fanático por coleções de pratos. Para além disso, estima muito os seus serviços de pratos, pelo que gosta de saber minuciosamente onde estão os pratos do restaurante.

Cada prato (**Dish**) tem um tipo – raso (*Plate*), sobremesa (*Dessert*) ou sopa (*Bowl*) – e é de uma coleção (Spring, Asian, White, Fernando Pessoa, Viana, …). Ao longo do dia os pratos vão mudando de localização. Os pratos limpos (*clean*) podem estar empilhados, por tipo e coleção, em aparadores (móveis da sala). Quando uma mesa é posta, pratos rasos da mesma coleção são lá colocados, e ao longo da refeição são também colocados pratos de sopa e de sobremesa (sempre da mesma coleção). Sempre que os pratos sujos (*dirty*) são levantados da mesa, são colocados numa fila (tapete rolante) que os direciona para a cozinha para serem lavados. Há uma fila única para todos os pratos sujos, de todos os tipos e coleções. Após a lavagem, os pratos molhados são colocados a secar (*drying*) na vertical, por uma qualquer ordem, o que pode ser modelado por uma lista.

A classe **Restaurant** contém as diversas estruturas de dados correspondentes aos locais onde podem estar os pratos. Uma mesa (**Table**) contém um número de lugares e um vector de lugares, representados como pilhas de pratos (em cada lugar pode haver, por exemplo, um prato de sopa em cima de um prato raso).

As classes **Restaurant**, **Table** e **Dish** estão parcialmente definidas a seguir.

```
class Restaurant {
        vector<Table> tables;
        vector< stack<Dish*> > clean;
        queue<Dish*> dirty;
        list<Dish*> drying;
public:
        Restaurant();
        const Table&
        stack<Dish*>& getCleanDishStack(
             string collection, TypeOfDish type);

        void addDishes(unsigned int n,
             string collection, TypeOfDish type);
        Dish* washDish();
        void clearTable(
                   vector<Table>::size_type idx);
        void storeDryDishes(string collection,
                            TypeOfDish type);
        void setupTable(
                    vector<Table>::size_type idx,
                    string collection);
        list<Dish*> pickupAndGroupDryDishes();
        int storeGroupedDishes(
                     list<Dish*> grouped_dishes);

        class NotEnoughDishesException {
        public:
                NotEnoughDishesException() {}
        };

        class TableNotReadyException {
        public:
                TableNotReadyException() {}
        };
};
```

```
class Table {
        vector< stack<Dish*> > places;
public:
        Table(unsigned int n);
        unsigned int size() const;
        bool empty() const;
        void putOn(vector<Dish*> dishes);
        vector<Dish*> clear();

        class WrongNumberOfDishesException {
        public:
                WrongNumberOfDishesException() {}
        };

        class WrongCollectionException {
        public:
                WrongCollectionException() {}
        };

};


enum TypeOfDish {Plate, Dessert, Bowl};

class Dish {
        const string collection;
        const TypeOfDish type;
public:
        Dish(string c, TypeOfDish t);
        string getCollection() const;
        TypeOfDish getType() const;
        bool operator ==(const Dish& d) const;
};
```

a) [**2.5 valores**] O Sr. Silva compra regularmente novos pratos, não só quando estes se partem mas também para inovar com novas coleções. Implemente na classe ***Restaurant*** o membro-função

```
void Restaurant::addDishes(unsigned int n, string collection, TypeOfDish type)
```

que adiciona ***n*** pratos da coleção ***collection*** e do tipo ***type*** à pilha de pratos limpos respetiva. Note que o membro-função ***Restaurant::getCleanDishStack(…)*** permite obter a pilha de pratos pretendida, criando uma nova se necessário.

b) [**2.5 valores**] Lavar um prato significa retirá-lo da fila de pratos sujos e colocá-lo a secar, após lavagem, na lista de pratos molhados. Implemente na classe ***Restaurant*** o membro-função

```
Dish* Restaurant::washDish()
```

que executa estas operações (o Sr. Silva só está preocupado com a localização dos pratos, pelo que pode ignorar a utilização de água e detergente…). A função deve retornar o prato que foi lavado.

c) [**3 valores**] Quando os clientes terminam a refeição, torna-se necessário levantar os pratos da mesa e encaminhá-los para a zona de lavagem. Implemente na classe ***Table*** o membro-função

```
vector<Dish*> Table::clear()
```

que levanta a mesa, retirando todos os pratos de todos os lugares e retornando um vetor com eles. Implemente também na classe ***Restaurant*** o membro-função

```
void Restaurant::clearTable(vector<Table>::size_type idx)
```

que levanta a mesa de índice ***idx***, se existir, colocando os seus pratos na fila de pratos sujos.

d) [**3 valores**] Quando ficam secos, os pratos são colocados novamente nas pilhas de pratos limpos. Nas horas de maior afluência, e para aproveitar de forma rápida a viagem de regresso da cozinha à sala, o Sr. Silva retira da lista de pratos que estão a secar apenas aqueles que são de uma determinada coleção e tipo, colocando-os na pilha respetiva. Implemente na classe ***Restaurant*** o membro-função

```
void Restaurant::storeDryDishes(string collection, TypeOfDish type)
```

que executa estas operações, isto é, que retira da lista de pratos a secar cada prato da coleção ***collection*** e do tipo ***type***, colocando-os na pilha de pratos limpos respetiva.

e) [**3 valores**] O Sr. Silva gosta de variar na composição das mesas, escolhendo de entre as coleções disponíveis. Implemente na classe ***Restaurant*** o membro-função

```
void Restaurant::setupTable(vector<Table>::size_type idx, string collection)
```

que prepara a mesa de índice ***idx***, se existir, nela colocando pratos rasos limpos da coleção ***collection***. Esta função deve lançar uma exceção ***TableNotReadyException*** no caso de a mesa não estar vazia. Deve ainda lançar uma exceção ***NotEnoughDishesException*** no caso de não haver pratos rasos limpos suficientes para colocar em todos os lugares da mesa. Note que estas classes já se encontram definidas dentro da classe ***Restaurant***. Note ainda que o método ***Table::putOn(…)***, já implementado, permite distribuir um vetor de pratos pelos lugares da mesa.

f) [**3 valores**] No final do dia, todos os pratos que estiveram a secar têm que ser guardados nas pilhas de pratos limpos. Para agilizar este processo, o Sr. Silva opta sempre por recolher os pratos secos para um carrinho, agrupando-os por coleção e tipo. Implemente na classe ***Restaurant*** o membro-função

```
list<Dish*> Restaurant::pickupAndGroupDryDishes()
```

que retira todos os pratos da lista de pratos a secar e retorna uma lista com esses pratos agrupados. Note que estes grupos podem ter qualquer ordem (isto é, pode haver alternância de coleções ou tipos, embora todos os pratos da mesma coleção+tipo tenham que ficar seguidos).

g) [**3 valores**] Depois de empurrar o carrinho até à sala, o Sr. Silva coloca os pratos nos seus devidos lugares (pilhas de pratos limpos). Implemente na classe ***Restaurant*** o membro-função

```
int Restaurant::storeGroupedDishes(list<Dish*> dishes)
```

que coloca os pratos que estão lista de pratos agrupados ***dishes*** nas suas pilhas respetivas. A função deve retornar o número de pilhas que foram alteradas. Note que deve tirar partido do facto de os pratos da mesma coleção+tipo estarem seguidos.