ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Departamento de Engenharia de Computação e Sistemas Digitais

PCS 3111 - Laboratório de Programação Orientada a Objetos para Engenharia Elétrica

2023

Aula 11 – Namespaces, Templates e Biblioteca Padrão

Atenção

1. As definições das classes usadas nos exercícios encontram-se **disponíveis no e-Disciplinas**. Use o código fornecido.
2. Os nomes, os atributos, os métodos, e as respectivas assinaturas das classes dadas **devem seguir o especificado** em cada exercício para fins de correção automática.

Considere as classes `Produto`, `Item`, `Combo` e `Pedido` implementadas nas aulas anteriores e que são fornecidas para esta aula.

Exercício 01

Até agora, vínhamos trabalhando com vetores para armazenar coleções em uma classe. Agora, queremos trabalhar com novas estruturas de dados, como os *containers* vistos nesta aula. Para isso, modifique a classe `Combo` fornecida, filha de `Produto`, para que os seus produtos sejam armazenados em um container do tipo `vector`, e não mais em um vetor simples. Os métodos públicos da classe devem ficar como a seguir:

```
Combo(string nome);
virtual ~Combo();

void adicionar(Produto *p);
double getPreco();
vector <Produto*>* getProdutos();

void imprimir();
```

- No construtor, aloque dinamicamente o `vector` que será utilizado para armazenar os ponteiros (note que não é mais recebida a quantidadeMaxima);
- No destrutor, destrua o `vector` de produtos alocado dinamicamente (mas não destrua os produtos);
- Remova o método `getQuantidade` e outros membros desnecessários.

- No método `adicionar`, se o produto que se deseja adicionar já está presente no `vector`, jogue uma exceção do tipo `invalid_argument`. Caso contrário, adicione o produto. Note que não existe mais o problema de *overflow* para a estrutura do `vector`.
- Os métodos `getPreco`, `getProdutos` e `imprimir` devem possuir os mesmos comportamentos fornecidos no código, mas devem ser adaptados para o uso de `vector`;
- Implemente a função `teste1()` conforme os passos a seguir:
    1. Crie um combo de nome *Burger e bebida*;
    2. Crie o produto *X-Bacon* de preço 24.99 reais e adicione-o ao combo;
    3. Crie o produto *Coca-Cola Lata* de preço 4.49 reais e adicione-o ao combo;
    4. Imprima o combo;
    5. Delete o combo e os produtos criados.

Exercício 02

Modifique a classe `Pedido`, de forma semelhante ao que foi feito no exercício 1, para que seus itens sejam armazenados em um container do tipo `list`, e não mais em um vetor simples . Esta classe representa um pedido que pode conter itens com combos ou produtos (a classe `Item` já é fornecida; não altere-a). Seus métodos públicos devem ficar como a seguir:

```
Pedido();
virtual ~Pedido();

void adicionar(Produto* produto, int quantidade);
list<Item*>* getItens();

void imprimir();
```

- No construtor, aloque dinamicamente a `list` de itens do pedido (note que o construtor não recebe mais parâmetros);
- No destrutor, destrua os itens do `list` e depois destrua o `list` de itens alocado dinamicamente. Faça alterações para que a impressão do destrutor funcione da mesma forma da apresentada no código fornecido;

- Remova o método `getQuantidade` e outros membros desnecessários.
- No método `adicionar`, caso o produto passado como argumento já exista em um item no Pedido, jogue uma exceção do tipo `invalid_argument` (a mensagem de erro passada para a exceção não será analisada). Caso contrário, crie um item com o produto e a quantidade passados e adicione-o ao final da lista. Note que não existe mais o problema de *overflow* para a estrutura do `list`.
- O método `getItens` deve retornar o ponteiro para a `list` de itens do pedido;
- O método `imprimir` deve possuir o mesmo comportamento fornecido no código, mas deve ser adaptado para o uso de `list;`
- Implemente a função `teste2()` conforme os passos a seguir:
    1. Crie um pedido;
    2. Repita os passos de 1 a 3 do `teste1`;
    3. Adicione ao pedido o combo criado com quantidade 2;
    4. Crie o produto *Sundae* de preço 7.85 reais;
    5. Adicione ao pedido o *Sundae* com quantidade 2;
    6. Imprima o pedido;
    7. Delete o pedido, os itens, o combo e os produtos criados.

Testes do Judge

Exercício 1

- `Combo` Teste `getProdutos`;
- `Combo` Teste `getPreco`;
- `Combo` Teste `adicionar` produtos distintos;
- `Combo` Teste `adicionar` produto já adicionado;
- `Combo` Teste `imprimir`.
- Teste da função `teste1`.

Exercício 2

- `Pedido` Teste `getItens`;
- `Pedido` Teste `adicionar` produtos distintos;
- `Pedido` Teste `adicionar` produto já adicionado;
- `Pedido` Teste `imprimir`;
- Teste da função `teste2`.