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 08 – Classes Abstratas, Herança Múltipla e Métodos e Atributos Estáticos

Atenção

- Código inicial a ser usado na resolução dos exercícios encontra-se disponível no e-Disciplinas.
- 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.
- Submeta um arquivo comprimido (faça um ".zip" – **não pode ser ".rar"**) colocando apenas os arquivos ".cpp" e ".h". Não crie pastas no "zip".

AVISO: para evitar problemas de compilação no Judge, envie os exercícios à medida que implementá-los!

Exercício 1

No código fornecido é entregue a classe `Produto`, a qual você deve tornar abstrata. Faça com que o método `imprimir()` seja abstrato.

Implemente a classe `ProdutoNormal` que será uma subclasse concreta de `Produto` e terá a implementação do `imprimir()`. Note que a única funcionalidade, além do construtor e do destrutor, que `ProdutoNormal` deve implementar é o método `imprimir()` (que deve ser a mesma fornecida em Produto). Não esqueça de realizar as alterações necessárias, como alterar as visibilidades de alguns atributos ou tornar alguns métodos virtuais. O destrutor de ProdutoNormal deve imprimir uma mensagem de destruição padrão no seguinte formato (pulando uma linha no final):

ProdutoNormal <NomeDoProduto> destruido

Implemente a classe concreta `Combo`, que também é filha de `Produto` e deve ser capaz de armazenar diversos produtos normais em um vetor. Para isso, seu construtor recebe uma quantidade máxima, que deve ser usada para alocar o vetor de ponteiros para os objetos do tipo `ProdutoNormal`. Os métodos públicos que devem ser implementados para essa classe estão listados abaixo, mas é permitido criar métodos e atributos privados caso necessário.

Note que o construtor não recebe o valor de preço, pois ainda não há produtos normais adicionados. Porém, lembre-se de que a classe possui um atributo preço (herdado de `Produto`) que deve ser inicializado apropriadamente e atualizado de forma a conter o preço total de todos os produtos normais adicionados.

```
    Combo(string nome, int quantidadeMaxima);
    virtual ~Combo();

    bool adicionar(ProdutoNormal *pn);
    ProdutoNormal **getProdutosNormais();
    int getQuantidadeDeProdutosNormais();

    void imprimir();
```

No destrutor, delete todos os atributos alocados dinamicamente. Não destrua os produtos. Ele deve imprimir uma mensagem de destruição padrão no seguinte formato:

Combo <NomeDoCombo> destruido

O método `adicionar()` recebe um ponteiro para um `ProdutoNormal` e deve adicioná-lo ao vetor se houver espaço. Caso tenha conseguido adicionar, retorne **true**, caso contrário, retorne **false**.

O método `getProdutosNormais()` deve retornar o vetor de ponteiros para `ProdutoNormal` e o método `getQuantidadeDeProdutosNormais()` deve retornar a quantidade de produtos normais adicionados.

O método `imprimir()` deve mostrar o nome, a preço total e o número de produtos normais de `Combo` e, em seguida, imprimir cada um dos produtos normais adicionados:

```
<nome> - <preco> reais - <quantidadeDeProdutosNormais> produtos normais
      \t1: <nome> - <preco> reais
      \t2: <nome> - <preco> reais
      ...
      \t<quantidadeProdutosNormais>: <nome> - <preco> reais
```

Por exemplo:
Sushi Executivo - 50 reais - 3 produtos normais
1: Temaki Atum - 20 reais
2: Shimeji - 15 reais
3: Sashimi - 15 reais

Dica: use o método imprimir desenvolvido para cada `ProdutoNormal`.

Nota: o caractere '\t' é usado para indentar a impressão dos produtos normais. Por exemplo:

```
cout << "\t1: Temaki Atum";
```

Imprime:

```
        1: Temaki Atum
```

Além disso, implemente a função `teste1()` seguindo os passos descritos a seguir.

1. Crie um ProdutoNormal chamado "Temaki Atum" com o valor de 20 reais
2. Crie um ProdutoNormal chamado "Shimeji" com o valor de 15 reais
3. Crie um ProdutoNormal chamado "Sashimi" com o valor de 15 reais
4. Crie um Combo chamado "Sushi Executivo" e 3 de quantidade máxima.
5. Adicione os produtos ao Combo na mesma ordem com que foram criados!
6. Imprima Combo
7. Delete todos os objetos alocados dinamicamente.

Exercício 2

Modifique a classe `Produto` adicionando um *id* inteiro e sequencial a todos os Produtos independentemente de seu tipo (`ProdutoNormal` ou `Combo`). Para isso, crie (em `Produto`) um novo atributo id e seu getter, aqui chamado de `getId()`. O controle dos *ids* atribuídos a cada novo objeto (de `ProdutoNormal` ou `Combo`) será feito por meio de um atributo estático e de seu getter, o `getProximoId()`.

Lembre-se que na criação de cada `Produto` deve-se incrementar o valor da variável estática que é retornada por `getProximoId()`.

Dessa forma, os novos métodos públicos que devem ser adicionados a classe `Produto` são os seguintes:

```
static int getProximoId();
int getId();
```

Observação: o id do primeiro `Produto` criado deve ser 0.

Implemente a função `teste2()` considerando os seguintes passos:

1. Refaça os passos de 1 a 5 do teste 1.
2. Imprima o valor de getId() de cada `Produto` (`ProdutoNormal` e `Combo`), seguindo a ordem de criação, pulando uma linha após cada impressão.

3. Delete os objetos alocados dinamicamente.

Perceba que os ids são incrementados sequencialmente à medida que objetos que herdam de `Produto` são criados e que dois objetos diferentes nunca podem ter o mesmo id, caso contrário, seu propósito de ser um identificador único para cada `Produto` não faria sentido.

Testes do Judge
Exercício 1

- `Produto` é classe abstrata;
- `ProdutoNormal` e `Combo` herdam de `Produto`;
- Getters herdados de `Produto`;
- Métodos `adicionar`, `getProdutosNormais` e `getQuantidadeDeProdutosNormais` de Combo
- Teste de `imprimir` de ProdutoNormal
- Teste de `imprimir` de Combo
- Função `teste1`.
- Destrutor de Produto Normal
- Destrutor de Combo

Exercício 2

- `getProximoId` retorna zero antes de criar qualquer objeto
- `getProximoId` chamado duas vezes seguidas
- `getProximoId` é incrementado após criar um objeto
- `getProximoId` é incrementado após criar diversos objetos
- `getId` após criar um objeto
- `getId` após criar diversos objetos
- Função `teste2`.