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 4 – Encapsulamento

Atenção

- Código inicial a ser usado na resolução dos exercícios encontra-se **disponível no e-Disciplinas**.
- Submeta um arquivo comprimido (faça um ".zip" – **não pode ser ".rar"**) colocando apenas os arquivos ".cpp" e ".h". Não crie pastas no "zip".
- Ao enviar para o Judge mantenha o `#define MAXIMO_DE_PRODUTOS 10` com o valor 10, por motivos de correção.

Utilize o código fornecido no e-Disciplinas para implementar as classes **`Produto`** e **`Pedido`**, aplicando os conceitos de encapsulamento vistos em aula.

Exercício 1
No arquivo teste.cpp é fornecida a definição e a implementação da classe `Produto`. Separe-a em dois arquivos, "**Produto.h**" e "**Produto.cpp**". O arquivo .h deve conter apenas a definição. O arquivo .cpp deve conter apenas a implementação. Use adequadamente as diretivas de compilação.

Implemente os métodos `getNome`, `setNome`, `getPreco` e `setPreco` apropriadamente (retorne o valor do respectivo atributo no get e defina o valor do respectivo atributo usando o parâmetro do set). Defina a visibilidade dos atributos e dos métodos de modo que os atributos sejam acessíveis **apenas** pelos métodos da classe e os métodos sejam acessíveis externamente.

No arquivo teste.cpp deixe apenas as funções `teste1` e `teste2`. Como o main está em um outro arquivo que é ignorado pelo juiz (pois se chama main.cpp), não é necessário mais comentar o `main`!

Complete o código fornecido, para tanto considere os seguintes passos na função `teste1`:

1. Crie um `Produto` de nome *Escova de dentes* e preço 5.49 (representando 5.49 reais);
2. Imprima *Escova de dentes*.

Exercício 2
Implemente a classe `Pedido`, fornecida. Note que essa classe utiliza a classe `Produto`, do Exercício 1. Novamente, separe a classe em dois arquivos, "**Pedido.h**" e "**Pedido.cpp**", usando adequadamente as diretivas de compilação. Defina corretamente a **visibilidade** de seus métodos e atributos.

A implementação da classe `Pedido` deve atender aos seguintes requisitos:

- O método `getPrecoTotal` deve retornar o somatório dos preços de cada `Produto` no `Pedido`**.** Caso não existam produtos adicionados ao pedido, esse método deve retornar 0.
    - **Dica:** Use o método getPreco da classe `Produto`.
- O método `adicionar` deve adicionar, se possível, o objeto do tipo `Produto` passado como parâmetro ao vetor **`produtos`** (crie este atributo). Caso o vetor já esteja completamente preenchido **ou o objeto já tenha sido adicionado** (compare o objeto com `==`), o método não modifica o vetor e retorna **`false`**. Caso seja bem-sucedido, deve retornar **`true`**. Utilize a constante `MAXIMO_DE_PRODUTOS` como o número máximo de produtos que o pedido comporta. Controle a quantidade de objetos adicionados com um atributo `quantidade`, ou seja, caso exista apenas um produto no pedido, a `quantidade` será igual a 1; caso existam dois produtos, a `quantidade` será igual a 2 (**use esse atributo para saber em qual posição do vetor deve ser adicionado o produto**).
- O método `imprimir` deve mostrar na tela as informações do pedido, seguindo o formato (pulando uma linha no final):

  ```
  Pedido com <quantidade> produtos - <preço total> reais no total
  ```

  Além disso, imprima as informações de todos os produtos que foram adicionados ao pedido (usando o método `imprimir` de Produto). Por exemplo:

  ```
  Pedido com 2 produtos – 17 reais no total
  Copos descartaveis – 10 reais
  Guarana 2L – 7 reais
  ```

- Implemente a função `teste2`, executando apenas os seguintes passos:

    1. Crie um `Produto` de nome *Agua* e de preço 4.90 reais;
    2. Crie outro `Produto` de nome *Desodorante* e de preço 15.59 reais;
    3. Crie um `Pedido`;
    4. Adicione *Agua* e *Desodorante* ao `Pedido`;
    5. Imprima *o* `Pedido`.

  Dica: a classe `Produto` já possui um método `imprimir` – utilize-o para facilitar a implementação do método imprimir de `Pedido`!

Lembre-se de utilizar a diretiva *#ifndef* nos arquivos de cabeçalhos (“**.h**”) para evitar problemas de conflitos de definição de classes causados por múltiplas inclusões de um mesmo cabeçalho.

Dicas importantes:

- Os nomes, os atributos, os métodos, e as respectivas assinaturas das classes dadas **devem seguir exatamente o especificado** para fins de correção automática.
- Note que não é mais necessário comentar a função `main`. Ela agora se encontra num arquivo separado “**main.cpp**”, fornecido, que pode ser enviado ao Judge, já que o sistema injeta a `main` de teste e subescreve a submetida.

Testes do Judge
Exercício 1

- Teste dos *setters* e *getters* de `Produto`;
- Teste do método `imprimir` de `Produto`;
- Teste da função `teste1`.

Exercício 2

- Teste `adicionar` com vetor vazio;
- Teste `adicionar` com vetor parcialmente preenchido;
- Teste `adicionar` com vetor cheio;
- Teste `adicionar` filme repetido;
- Teste `getPrecoTotal` com vetor vazio;
- Teste `getPrecoTotal` com vetor parcialmente preenchido;
- Teste `getPrecoTotal` com vetor cheio;
- Teste `imprimir` sem produtos;
- Teste `imprimir` com 1 produto;
- Teste `imprimir` com vários produtos;
- Teste da função `teste2`.