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 07 – Herança e Polimorfismo II

**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.
3. A **ordem de declaração** de atributos e métodos fornecidos **não deve ser alterada**. Caso contrário, poderá haver redução automática da nota.

Considere as classes **Produto**, **Item** e **Pedido**, implementadas nas aulas anteriores. Elas são fornecidas para esta aula.

### Exercício 01

Altere a **Pedido**. Seus métodos públicos são apresentados a seguir:

```
Pedido(int quantidadeMaxima);
~Pedido();

bool adicionar(Produto *produto);
bool adicionar(Produto *produto, int quantidade);

Item** getItens();
int getQuantidadeItens();

double calculaPrecoTotal();
void imprimir();
```

- Na classe **Pedido** fornecida, implemente o método `bool adicionar(Produto *produto)`, que tem o mesmo funcionamento do método `bool adicionar(Produto *produto, int quantidade)` considerando a quantidade como 1.

Perceba que o método adicionar foi **sobrecarregado.**

- Implemente o método `void imprimir()`. Para um **Pedido** com n itens, deve-se imprimir o seguinte:
    - Na primeira linha, imprime-se a quantidade de itens como abaixo, pulando uma linha ao final:

`Pedido com <n> item(ns)`

    - Em seguida, chame o método `imprimir` de cada item no pedido.

Considere, por exemplo, um pedido com os seguintes itens: *Caneca* com preço de 10.50 reais e quantidade 3; e *Bola* com preço de 120.90 reais e quantidade 5. O método `imprimir()` para esse pedido deve gerar a seguinte saída:

```
Pedido com 2 item(ns)
3 unidade(s) de Caneca - 10.50 reais cada
5 unidade(s) de Bola - 120.90 reais cada
```

- Implemente a função `teste1()` conforme os passos a seguir:

1. Crie um pedido de tamanho 2;
2. Crie o produto *Bala de goma* de preço 3.50 reais;
3. Adicione *Bala de goma* ao pedido criado sem passar a quantidade;
4. Crie o produto *Chocolate* de preço 5.20 reais;
5. Adicione *Chocolate* ao pedido criado com quantidade 4;
6. Imprima o pedido;
7. Delete o pedido e os produtos criados.

**Exercício 02**
Altere a classe **ProdutoComDesconto** fornecida, que é uma classe filha de **Produto**. Seus métodos públicos são apresentados a seguir:

```
ProdutoComDesconto(string nome, double preco);
ProdutoComDesconto(string nome, double preco, double desconto);
~ProdutoComDesconto();

double getDesconto();
void setDesconto(double desconto);
double getPreco();
```

O construtor que não possui o parâmetro `desconto` é semelhante ao que possui, sendo que o primeiro considera o desconto como 0;

- Redefina o método `getPreco` da classe mãe para que o preço retornado por este método aplique o desconto. Modifique o que for necessário na classe **Produto** para que haja ligação dinâmica. Lembre-se também de alterar o destrutor apropriadamente, pelo mesmo motivo. Caso contrário, o destrutor de `ProdutoComDesconto` não seria chamado.
    - Caso necessário, altere o modo de visibilidade dos atributos de **Produto** para que assim estejam acessíveis pela classe **ProdutoComDesconto**.
- Implemente a função `teste2()` conforme os passos a seguir:

1. Crie um pedido de tamanho 2;
2. Crie o produto *Linguica* de preço 20.90 reais;
3. Adicione *Linguica* ao pedido criado criado com quantidade 2;
4. Crie o produto com desconto *Picanha* de preço 70.49 reais e desconto de 0.1 (10%), armazenando-o em um ponteiro do tipo Produto;
5. Adicione *Picanha* ao pedido sem passar a quantidade;
6. Imprima o pedido;
7. Delete o pedido e os produtos criados.

Perceba o comportamento diferente do método `getPreco` dependendo se o objeto é **Produto** ou **ProdutoComDesconto**. Isso é a consequência da ligação dinâmica!

**Exercício 03**

Implemente o método `ProdutoComDesconto** getProdutosComDesconto(int &quantidade)` em **Pedido**, que deve retornar um vetor composto pelos produtos com desconto que existem no pedido, caso existam. O método também retorna a quantidade de objetos `ProdutoComDesconto` que existem no vetor, uma vez que `quantidade` é passado por referência (consulte a Aula 2 para lembrar como usar a passagem por referência). Não se esqueça de inicializá-la em `0` dentro do método.

Se não existirem produtos com desconto no pedido, retorne NULL (e `quantidade` deve ser retornado como `0`). Crie o vetor de produtos com desconto dentro do método (ou seja, cada vez que o método for chamado, crie um novo vetor dinamicamente). Para simplificar, você pode alocar um vetor de `quantidadeMaxima`). Além disso, note que Os métodos públicos de **Pedido** devem ser (não altere os demais métodos):

```
Pedido(int quantidadeMaxima);
~Pedido();

bool adicionar(Produto *produto);
bool adicionar(Produto *produto, int quantidade);

Item** getItens();
int getQuantidadeItens();
ProdutoComDesconto** getProdutosComDesconto (int& quantidade);

double calculaPrecoTotal();
void imprimir();
```

- Implemente a função `teste3()` conforme os passos a seguir:

1. Crie um pedido de tamanho 2;
2. Chame o método `getProdutosComDesconto` para obter a quantidade de produtos com desconto do pedido;
3. Imprima a quantidade de produtos com desconto do pedido (pule uma linha no final);

4. Crie o **Produto** *Refrigerante* de preço 7.80 reais e adicione-o ao pedido com quantidade 2;
5. Repita os passos 2 e 3;
6. Crie o **ProdutoComDesconto** *Pizza* de preço 40.99 e adicione-o ao pedido com quantidade 2;
7. Repita os passos 2 e 3;
8. Delete o pedido e os produtos criados.

## Testes do Judge

### Exercício 1

- `Pedido` Teste `adicionar` com quantidade 1 com Pedido vazio;
- `Pedido` Teste `adicionar` com quantidade 1 com Pedido cheio;
- `Pedido` Teste `imprimir`;
- Teste da função `teste1`.

### Exercício 2

- `ProdutoComDesconto` Teste `getPreco` sem desconto;
- `ProdutoComDesconto` Teste `getPreco` com desconto menor que 1;
- `ProdutoComDesconto` Teste `getPreco` com desconto de 1;
- Teste da função `teste2`.

### Exercício 3

- `Pedido` Teste `getProdutosComDesconto` com 5 itens e nenhum produto com desconto;
- `Pedido` Teste `getProdutosComDesconto` com 5 itens e 3 produtos com desconto;
- `Pedido` Teste `getProdutosComDesconto` com 5 itens e 5 produtos com desconto;
- Teste da função `teste3`.