POO em PHP

[Nome da empresa] | [Endereço da empresa]

Curso alura

Guydo Ventura Gonçalves

2022

Interface gráfica do usuário, Aplicativo, Tabela, PowerPoint

Descrição gerada automaticamente  
Figura 1 - Objeto referência um espaço na memória

Interface gráfica do usuário, Aplicativo, Tabela

Descrição gerada automaticamenteFigura 2 Duas variáveis apontando para o mesmo objeto

# 1) Classe:

- A classe possui o mesmo nome de seu arquivo  
- Cada uma de suas variáveis é chamada de atributo  
- Valores definidos na **classe** serão herdados por todos os seus objetos. Isso é uma má prática, os seus valores devem ser definidos em cada uma de suas variáveis  
- Por segurança é recomendado definirmos o tipo de dados que queremos receber em cada variável diretamente na classe  
- Por padrão as instâncias são passadas por referência!  
- É recomendado que tenhamos apenas **propriedades privadas** e **métodos públicos** em uma classe.  
- Devemos utilizar substantivos para os nomes dos parâmetros de uma classe

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Figura 3 - Definindo classe conta e seus modificadores de acesso

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Figura 4 Criando o método sacar

**Método** = Uma função dentro de uma classe  
- Devemos utilizar no infinitivo para nome de métodos  
- Precisamos indicar ao php qual objeto sofrerá a ação desse método, para isso utilizamos o $this.

**$this** = Variável que já contêm a referência dentro do próprio método. Ou seja, $this se refere a referência atual que chamou esse método.  
**Ex**: Se uma conta chamou o método sacar, dentro desse método o valor de $this vai ser a referência para esse objeto.   
Ou seja quando usamos o comando **$primeitaConta->sacar(150);** o $this atuará como ponteiro para o objeto $primeiraConta, assim indicando ao php onde realizar a ação.

- Não é indicado definir métodos de acesso para todos os atributos de uma classe!  
Ex: Não foi criado um método definirSalto para evitar que ele possua valor negativo. É preciso modifica-lo através de outros métodos como: depositar, sacar e transferir.

**\_\_construct()**- Sempre que instanciarmos uma classe esse método será chamado

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

- O constructo irá receber os parâmetros $cpfTitular e $nomeTitular por parâmetro.  
- O cpfTitular da nossa instância ira receber o $cpfTitular passado por parâmetro na hora de instanciar a classe, tornando essa informação obrigatória.

# 2) Encapsulamento

- Esconder as funcionalidades e funcionamento do nosso código dentro de pequenas unidades (normalmente métodos e funções). Isso possibilita que modificações no sistema possam ser feitas de maneira mais cirúrgicas, sem que uma funcionalidade esteja espalhada por diversas partes do sistema.

**Tipos de modificadores de acesso:**

**1) Public:** Este é o nível de acesso mais permissivo. Ele indica que o método ou atributo da classe é público, ou seja, pode ser acessado em qualquer outro ponto do código e por outras classes.

**2) Private:** Este modificador é o mais restrito. Com ele definimos que somente a própria classe em que um atributo ou método foi declarado pode acessá-lo. Ou seja, nenhuma outra parte do código, nem mesmo as classes filhas, pode acessar esse atributo ou método. **3) Protected:** Esse modificador indica que somente a própria classe e as classes que herdam dela podem acessar o atributo ou método. Dessa forma, ao instanciar a classe os elementos protegidos (protected) não podem ser acessados diretamente, como ocorre com o public.

> Para obter o valor de um atributo protegido da classe pai basta usar $this->atributo;  
> Para alterar o valor de um atributo protegido da classe pai é preciso redeclará-lo na classe filha;  
> Para chamar um método protegido da classe pai é necessário usar parent::metodo().

**4) Static:** está associado com a definição de atributos nas classes, de forma que os objetos da classe consigam compartilhar informações entre si. Ou seja, é um atributo da **classe** que é compartilhado com suas instancias.

Ex:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

public static $numeroDeContas = 0; => Atributo pertencente a classe

Como o método $this acessa a instancia de uma classe por referência é preciso uma forma de acessar diretamente a classe conta, para isso utilizamos o Conta::$numeroDeContas++; => Responsável por somar +1 a variável $numeroDeContas sempre que a classe Conta foir instanciada.

- Para acessarmos uma classe ao invés de escrevermos seu nome podemos usar o termo **self**Ex:   
self::$numeroDeContas++;

Porém, quando chamarmos o método ainda devemos especificar o nome da classe!  
Ex:echo Conta::recuperarNumeroDeContas() . PHP\_EOL;

**Garbage collector**

**Comandos no terminal:**

**php -a**  
- ativa o terminal interativo do php

**require ‘src/Conta.php’;**   
- Os arquivos são chamados baseando-se no caminho do arquivo informado

**$primeiraConta = new Conta();**- Cria um novo objeto da classe Conta  
- A variável primeiraConta não guarda em si os valores de seus atributos. Ela atua como um **endereço** de onde todos esses dados estão salvos  
- Por isso não é possível modificá-la como fazemos com o array

**$primeiraConta->saldo = 200;**- Estamos acessando o atributo saldo do Objeto primeiraConta e atribuindo seu valor.

**$primeiraConta->cpfTitular = ‘123.456.789-10’;**- Mesmo que o anterior, porém atribuindo um valor tipo str

**Var\_dump($primeiraConta);**- Retornar as informações do objeto mostrando sua classe e os tipos e valores de cada um de seus atributos.

**$primeitaConta->sacar(150);**- Chama o método sacar para remover 150 do saldo.

**$primeiraConta->transferir(300, $contaDestino);**- Chama o método transferir

**Vale lembrar:**- Código explicito é sempre melhor que implícito  
- Sempre colocar se o código é público ou não  
- Tipos primitivos são sempre passados por valor  
- Objetos guardam a referência para a instância daquela classe  
- Existe uma linha de pensamento (**Early Return**) que diz que devemos evitar utilizar o **else**!!!