


# PROGRAMAÇÃO DE COMPUTADORES



Técnico Integrado em  
**INFORMÁTICA**

Professor Jefferson Chaves  
[jefferson.chaves@ifc-araquari.edu.br](mailto:jefferson.chaves@ifc-araquari.edu.br)

- Dizer o que é e para que serve orientação a objetos;
- Conceituar classes, atributos e comportamentos;
- Entender o significado de variáveis e objetos na memória;


A faint, dark silhouette of an elephant is visible in the background on the left side of the slide.

**PROGRAMAÇÃO DE  
COMPUTADORES**

**ANTES DE COMEÇAR!**

- Estamos prestes a dar um passo a mais em programação;
- O entendimento desse conteúdo ficará **facilitado** se houver o conhecimento do conteúdo de algoritmos;

- O que estudar?
  1. Variáveis;
  2. Estruturas condicionais (if, else);
  3. Estruturas de repetição (for, while);
  4. Vetores (arrays);
  5. Funções;

A faint, dark green silhouette of an elephant is visible in the background on the left side of the slide.

**PROGRAMAÇÃO DE  
COMPUTADORES**

**ORIENTAÇÃO A  
OBJETOS BÁSICA**

- O que são paradigmas?
  - Paradigma procedural;
  - Paradigma orientado a objetos;
- O problema do paradigma procedural;
  - O caso da validação de um CPF;

- Vamos supor que tenhamos que validar os dados de um formulário, em especial, um CPF:
  - Normalmente recebemos os dados de um formulário;
  - E então chamamos uma função para validar o CPF;



---

```
1 $cpf = $_POST['cpf']
```

```
2
```

```
3 valida_dados($cpf);
```

```
4
```

```
5
```

```
6
```

```
7
```


```
8
```

```
9
```

```
10
```

- Alguém te **obriga** a sempre validar esse CPF?
- Considere um sistema com 50 formulários e a necessidade de validar CPF em todos!
- Mais ainda, considere a entrada de novos programadores: **todos deveriam ser avisados;**
- Surgimento de manuais;

- Pense em uma outra validação: **verificar se o cliente tem idade maior que 18 anos;**
- Temos que colocar um **if** espalhado em cada um dos 50 formulários;
- O ideal seria concentrar essa responsabilidade em um só lugar, não correndo o risco de esquecer;
- Melhor ainda, fazer essa validação e outros programadores nem fiquem sabendo!



Quais as  
vantagens  
?

Jefferson de Oliveira Chaves

- **Conexão** forte entre dados e funcionalidades;
- Concentração de **responsabilidades**;
- Organização do código;
- Encapsulamento da lógica de negócios;
- **Polimorfismo** das referências;

A faint, dark silhouette of an elephant is visible in the background on the left side of the slide, partially obscured by the text.

**PROGRAMAÇÃO DE  
COMPUTADORES**


**DEFININDO UM  
CENÁRIO**

- Para as próximas aulas, vamos considerar, principalmente, um sistema para um BANCO;
- Vamos usar a lógica de negócios que conhecemos para programar nosso sistema;
- Um entidade importante para o nosso sistema é a conta;

- Dados da conta
  - Número da conta;
  - Nome do titular da conta;
  - Saldo;
  - Limite;



- Funcionalidades/comportamentos:
  - Sacar;
  - Depositar;
  - Transferir;
- Mas isso é apenas um **PROJETO!**


A faint, dark silhouette of an elephant is visible on the left side of the slide, partially obscured by the text.

**PROGRAMAÇÃO DE  
COMPUTADORES**

**CLASSES E OBJETOS**

- A palavra classe vem da biologia: classes biológicas;
- Definem atributos e comportamentos em comum;
- Os atributos podem variar;
- A forma com que realizam os comportamentos também;

- **Homo Sapiens** define um classe de seres com características em comum;
- Homo Sapiens **é um definição**;
- Para mandar alguém correr, pular, falar, precisamos de uma instância de Homo Sapiens, ou então, um objeto do tipo **Homo Sapiens**;

A black and white portrait of a man with dark hair, wearing a suit and tie. He is looking upwards and to the right with a thoughtful expression, his hand resting on his chin. A large green speech bubble is overlaid on the right side of the image.

Vish!  
Algum  
outro  
exemplo?


Jefferson de Oliveira Chaves

- Uma receita de bolo é um bolo?
  - Uma receita tem sabor? Cheiro?
  - Podemos comer uma receita?
- Em nosso contexto a receita é uma especificação (**classe**)
- Se quisermos ter bolo para o café precisamos **instanciá-lo**, criar um **objeto** do tipo Bolo;

- Podemos criar centenas de bolos com essas classe (a receita);
- Pode ser semelhantes ou até mesmo idênticos, contudo, são **objetos diferentes**;
- Alguma outra analogia?

- Em resumo:
- Um modelo (definição, receita) é uma **classe**;
- Aquilo que for criado a partir de uma **classe** será um **objeto**;



A faint, dark green silhouette of an elephant is visible in the background on the left side of the slide.

**PROGRAMAÇÃO DE  
COMPUTADORES**

**CRIANDO NOSSA  
PRIMEIRA CLASSE**

- Vamos criar nossa primeira classe, a partir da nossa especificação de conta;
- Começaremos com o que uma conta tem;

```
1 class Conta{  
2  
3     public $numero;  
4     public $dono;  
5     public $saldo;  
6     public $limite;  
7  
8     //....  
9 }  
10
```

- Estes são os **atributos** que toda conta (quando instanciada) pode ter;
- Repare que essas variáveis foram criadas dentro do **escopo** da classe;
- Variáveis declaradas no escopo da classe são chamadas de variável do objeto ou **atributos**;



**PROGRAMAÇÃO DE  
COMPUTADORES**

**CRIANDO E USANDO  
UM OBJETO**

- Já temos uma classe que especifica o que toda conta deve ter;
- Para **INSTÂNCIAR (CRIAR, CONSTRUIR)** devemos usar a palavra chave **new**.
- Podemos usar parênteses após o nome da classe – veremos detalhes posteriormente;

```
1 new Conta( );
```

```
2
```

```
3 //mas como acessar o objeto instanciado?
```

```
4
```

```
5 $minha_conta = new Conta( );
```

```
6
```

```
7
```

```
8
```


```
9
```

```
10
```

- Por meio da variável **\$minha\_conta**, podemos acessar o objeto para alterar seu dono, saldo, etc.;
- **É importante** fixar que a seta (->) serve para acessar algo em \$minha\_conta;
- Vamos criar um arquivo para testar nossa conta. **Lembre-se de do require para usar nossa classe;**



```
1 $minha_conta= new Conta( );
2
3 $minha_conta->dono = "Paulo";
4 $minha_conta>saldo = "1000.00";
5
6 echo "Saldo atual".$minha_conta>saldo;
7
8
9
10
```

A faint, dark green silhouette of an elephant is visible in the background on the left side of the slide.

**PROGRAMAÇÃO DE  
COMPUTADORES**

**MÉTODOS**

- Dentro de uma classe também podemos declarar o que uma Conta faz, isso é, seu comportamento;
- Sacar, transferir, depositar
- Comportamento definido dentro da própria classe e não em um local desatrelado;

- Por isso, **funções** de classes são chamadas de **métodos**, pois é a maneira, o método que a classe usa para realizar uma operação;
- Vamos criar um método que realiza o **saque** de um determinado valor;

```
1 class Conta {  
2  
3     public $saldo;  
4     //.. outros atributos  
5  
6     function saca( $valor){  
7         $novo_saldo = $this->saldo - $valor;  
8         $this->saldo = $novo_saldo;  
9     }  
10
```

- Para sacar, é preciso dizer quanto quer sacar;
- Chamamos isso de **parâmetros** e o declaramos dentro dos parênteses do método;
- Parâmetro é uma variável temporária: ela deixará de existir no final da execução do método;

- Para acessar os atributos da classe usaremos a palavra reservada **this**;
- Isso serve para mostrar que não se trata de uma simples variável, mas sim um atributo da classe;
- A conta ainda pode estourar o limite: trataremos disso em breve;

- De forma semelhante ao método saca, temos o método deposita de nossa Classe;



```
1 class Conta {  
2  
3     //... demais atributos e métodos  
4  
5     function deposita( $valor){  
6         $this->saldo += $valor;  
7     }  
8  
9  
10
```

- Observe que não usamos uma variável auxiliar;
- Usamos também a abreviação ( += ) que soma a quantidade ao valor anterior do saldo e guarda (atribui) ao próprio saldo;
- Para pedir ao objeto que execute esse método, usamos a seta (->);
- O termo para isso é **invocação de método**;

```
1 $conta = new Conta( );  
2  
3 $conta->dono = "Paulo";  
4 $conta->saldo = "1000.00";  
5  
6 $minha_conta->saca(200);  
7 $minha_conta->deposita(500);  
8  
9 echo "Saldo atual".$conta->saldo;  
10
```

A faint, dark silhouette of an elephant is visible in the background on the left side of the slide, partially obscured by the text.

**PROGRAMAÇÃO DE  
COMPUTADORES**

**MÉTODOS COM  
RETORNO**


- Um método pode retornar um valor para quem o chamou;
- Nosso método saca poderia, por exemplo, retornar um valor **booleano** para quem o chamou;
- Caso o saque fosse feito retornaria **true**;
- Caso contrário, retornaria **false**;

```
1 class Conta {  
2     // ... outros atributos e métodos  
3  
4     function saca( $valor ){  
5         if($this->saldo >= $valor){  
6             $this->saldo = $this->saldo - $valor;  
7             return true;  
8         }  
9         else { return false; }  
10    }
```

PROGRAMAÇÃO DE  
COMPUTADORES

MÉTODOS COM  
RETORNO

- Vamos aos exemplos de uso!

A faint, dark silhouette of an elephant is visible in the background on the left side of the slide.

**PROGRAMAÇÃO DE  
COMPUTADORES**

**OBJETOS SÃO  
ACESSADOS POR  
REFERÊNCIA**



- Os tipos de dados que conhecidos até agora foram:
  - Inteiro (int);
  - Decimal (float);
  - Texto (string);
  - Booleano (lógico);

- Ao atribuir um valor para uma variável, essa variável armazena o **valor** informado;

```
1 <?php
2
3     $nome = "José";    //$nome armazena o
4     valor José
5
6     $idade = 16;        //$idade armazena o
7     valor 16
8
9     $tem_acesso = true; //$tem_acesso
10    armazena true
```

- Ao declararmos uma variável para associar a um objeto, essa variável não guarda o objeto, mas sim, **uma maneira de acessá-la** (um endereço);
- Essa maneira se chama **referência** ;

- É por esse motivo que, diferente dos tipos primitivos (int, float, string e boolean) precisamos do operador **new** depois da atribuição da variável;

- É **incorreto** dizer que \$conta1 é um objeto
- É **correto** dizer que \$conta1 é uma referência a um objeto do tipo Conta;
- Uma variável **nunca será** um objeto;

1 \$conta1 = new Conta( );

2 \$conta2 = new Conta( );

3

4

5

{representação da memória}

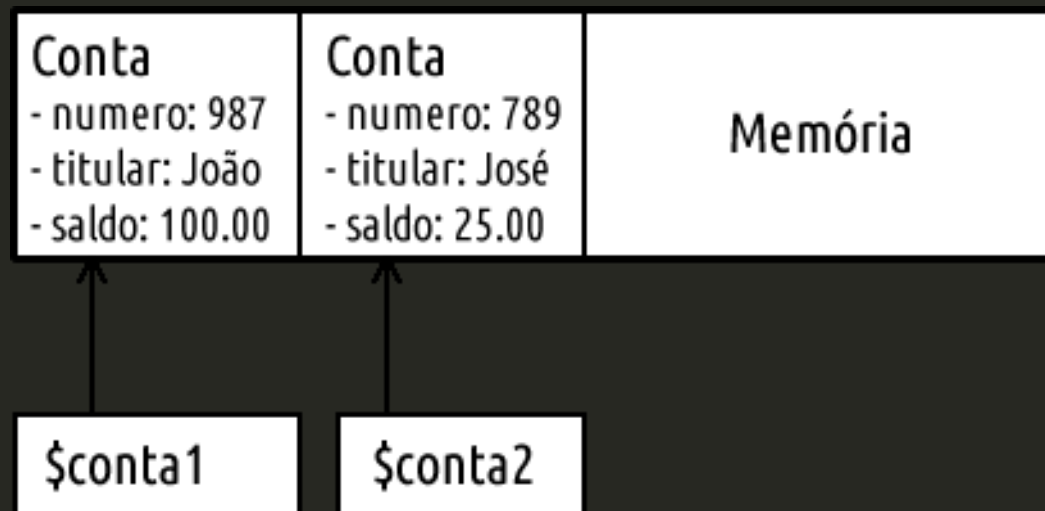
6

7

8

9

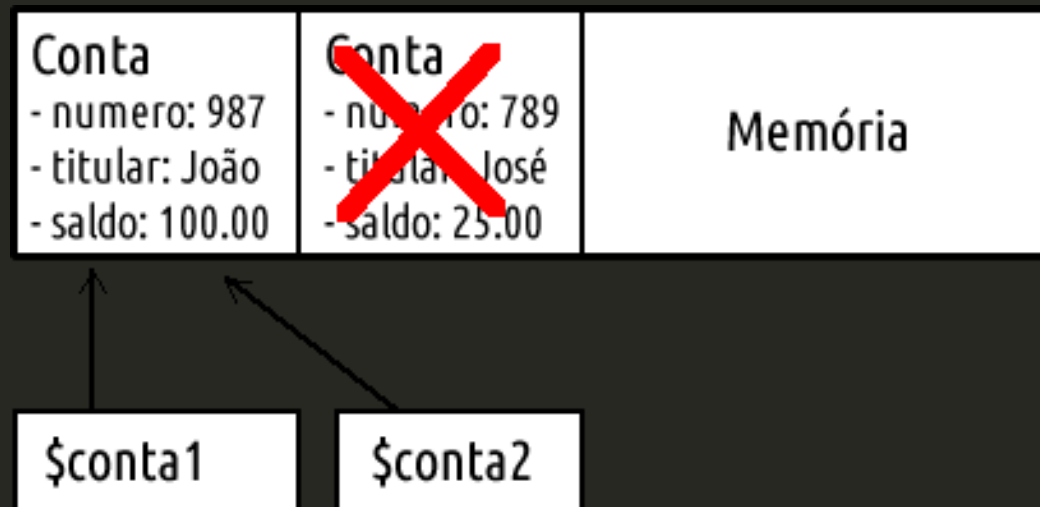
10



```

1 $conta1 = new Conta( );
2 $conta2 = $conta1;
3 //Apenas 1 new logo, apenas uma conta em
4 memória
5

```






- \$conta1 e \$conta2 vão guardar uma **referência** (um **endereço**) que identifica em que posição da memória aquela **Conta** está;
- Dessa maneira, usaremos essa referência seguida da seta ( -> ) para acessar os dados da conta, por exemplo:
  - \$conta1->saldo;
  - \$conta1->saca(300.00);



## O que exatamente faz o new?!

- O new executa uma série de tarefas, que veremos mais adiante.
- Mas, para melhor entender as referências, saiba que o new, depois de alocar a memória para esse objeto, devolve uma "flecha", isto é, um valor de referência.
- Quando você atribui isso a uma variável, essa variável passa a se referir para esse mesmo objeto.

A faint, dark silhouette of an elephant is visible in the background on the left side of the slide, partially obscured by the text.

**PROGRAMAÇÃO DE  
COMPUTADORES**

**O MÉTODO  
TRANSFERE**

- Como fazer um método que transfere dinheiro entre duas contas?
- Podemos ficar tentados a criar um método que recebe dois parâmetros:
  - \$conta1 e \$conta2 do tipo Conta.
- **Mas cuidado: assim estamos pensando de maneira procedural.**

```
1 class Conta {  
2     // ... outros atributos e métodos  
3  
4     function transfere(Conta $destino, $valor ){  
5         $this->saldo      = $this->saldo - $valor;  
6         $destino ->saldo = $destino ->saldo +  
7     $valor;  
8     }  
9 }  
10
```

- O que é refatorar?
- Usando os métodos existentes;
- Alterando o nome do método:  
**transferePara;**

- Para deixar a chamada do método mais natural, com mais sentido, vamos mudar seu nome para **transferePara**;
- \$conta1->transferePara(\$conta destino, 50);
- A leitura deste código seria "Conta1 transfere para conta\_destino 50 reais".

```
1  class Conta {  
2      // ... outros atributos e métodos  
3  
4      function transferePara(Conta $destino, $valor ){  
5          if($this->saldo >= $valor){  
6              $this->saldo = $this->saldo - $valor;  
7              return true;  
8          }  
9          else { return false; }  
10     }
```




## Atenção

Pode ser confuso pensar em como os objetos estão na memória para poder tirar as conclusões de o que ocorrerá ao executar determinado código.

Com tempo, você adquire a habilidade de rapidamente saber o efeito de atrelar as referências.

É importante, nesse começo, você estar sempre pensando no estado da memória. E realmente lembrar que *"uma variável nunca carrega um objeto, e sim uma referência para ele"* facilita muito.

A faint, dark green silhouette of an elephant is visible in the background on the left side of the slide.

**PROGRAMAÇÃO DE  
COMPUTADORES**

**CONTINUANDO COM  
OS ATRIBUTOS**

- Se precisarmos de mais atributos para nossa Classe devemos nos perguntar:
  - São **MESMO** esses atributos dessa Classe?
- Se a resposta for NÃO, então devemos criar uma nova Classe para esses atributos;

```
1 class Conta{  
2  
3     public $numero;  
4     public $dono;  
5     public $cpf;  
6     public $telefone;  
7     public $saldo;  
8     public $limite;  
9     //....  
10 }
```

- Se analisarmos melhor, veremos que uma **Conta** não possui cpf, telefone;
- Quem possui tais atributos é o cliente;
- Nesse caso devemos criar uma classe **Cliente** e fazer uma **composição**.

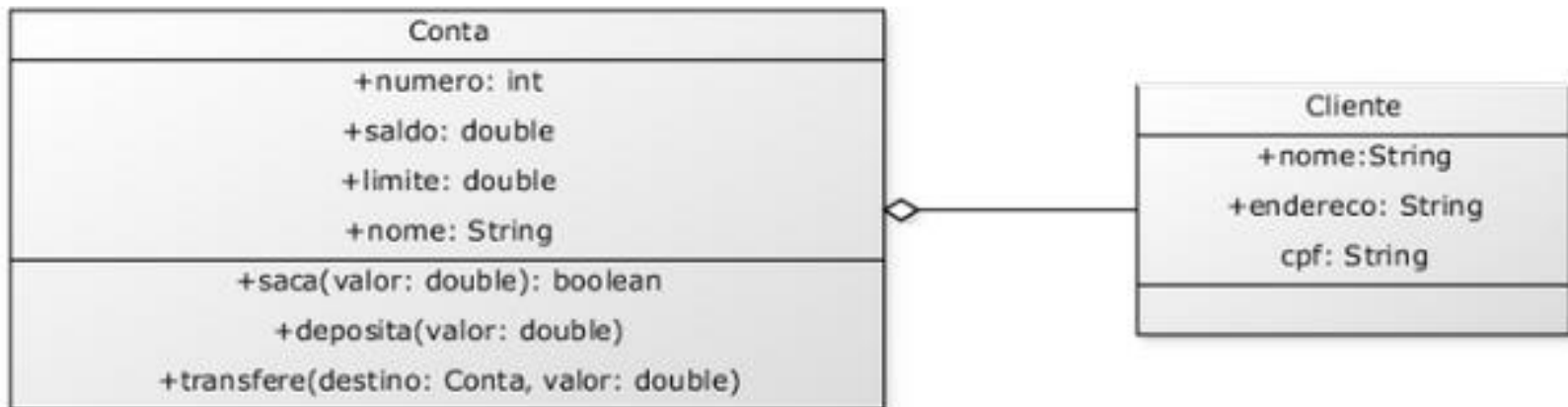


Diagrama de Classes: agregação

- Um sistema orientado a objetos é um conjunto de classes;
- Comunicam-se entre si por meio de métodos;
- Delegação de responsabilidades para a classe mais apta.



EXERCÍCIOS

MANIPULAÇÃO DE  
ARQUIVOS

Jefferson de Oliveira Chaves



1. Modele um funcionário. Ele deve ter o nome do funcionário, o departamento onde trabalha, seu salário, a data de entrada no banco (String) e seu RG (String).

A ideia aqui é apenas modelar, isto é, só identifique que informações são importantes e o que um funcionário faz.

**Desenhe no papel tudo o que um Funcionario tem e tudo que ele faz.**

2. Crie um método **recebeAumento** que aumenta o salario do funcionário de acordo com o parâmetro passado como argumento.

Crie também um método **calculaGanhoAnual**, que não recebe parâmetro algum, devolvendo o valor do salário multiplicado por 12.

3. Transforme o modelo acima em uma classe. Teste-a, usando um arquivo **teste\_funcionario.php**

Esse é um processo incremental.

Procure desenvolver seus exercícios, passo a passo para não descobrir só no fim do caminho que algo estava muito errado.

4. Crie um método `mostra()`, que não recebe nem devolve parâmetro algum e simplesmente imprime todos os atributos do nosso funcionário.