

PHP: Programando com Orientação a Objetos

Pablo Dall'Oglio

Adianti Solutions

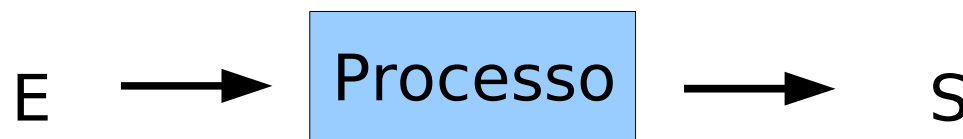
www.adianti.com.br



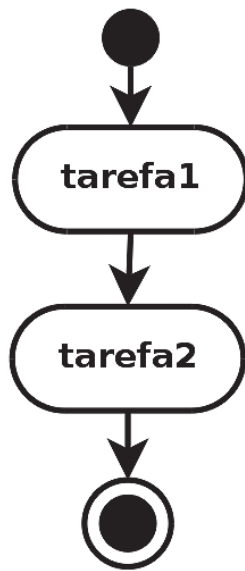
- Conceitos de Orientação a Objetos;
- Classes, objetos, propriedades, métodos;
- Métodos construtores e destrutores;
- Associação, agregação, composição e herança;
- Encapsulamento e polimorfismo;
- Exemplos práticos.



- Vamos entender como surgiu a Orientação a Objetos;
- Como era a programação antes da O.O. ?
- O paradigma que reinava era a programação estruturada;
- A programação estruturada introduziu conceitos importantes na engenharia de software em sua época;
- É baseada fortemente na modularização;
 - A idéia da modularização é dividir o programa em unidades menores conhecidas por procedimentos ou funções, que são construídas para desempenhar uma tarefa específica;



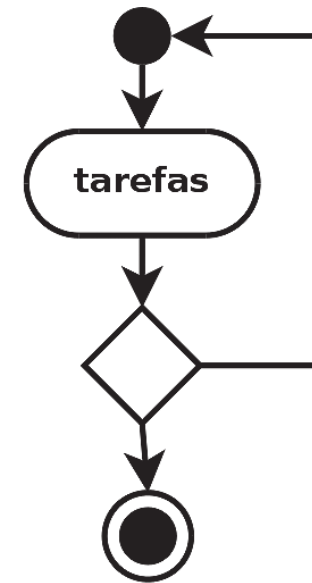
- Na programação estruturada, as unidades do código (funções) se interligam por meio de três mecanismos básicos: seqüência, decisão e iteração, como ilustrado na figura a seguir:



SEQÜÊNCIA



DECISÃO



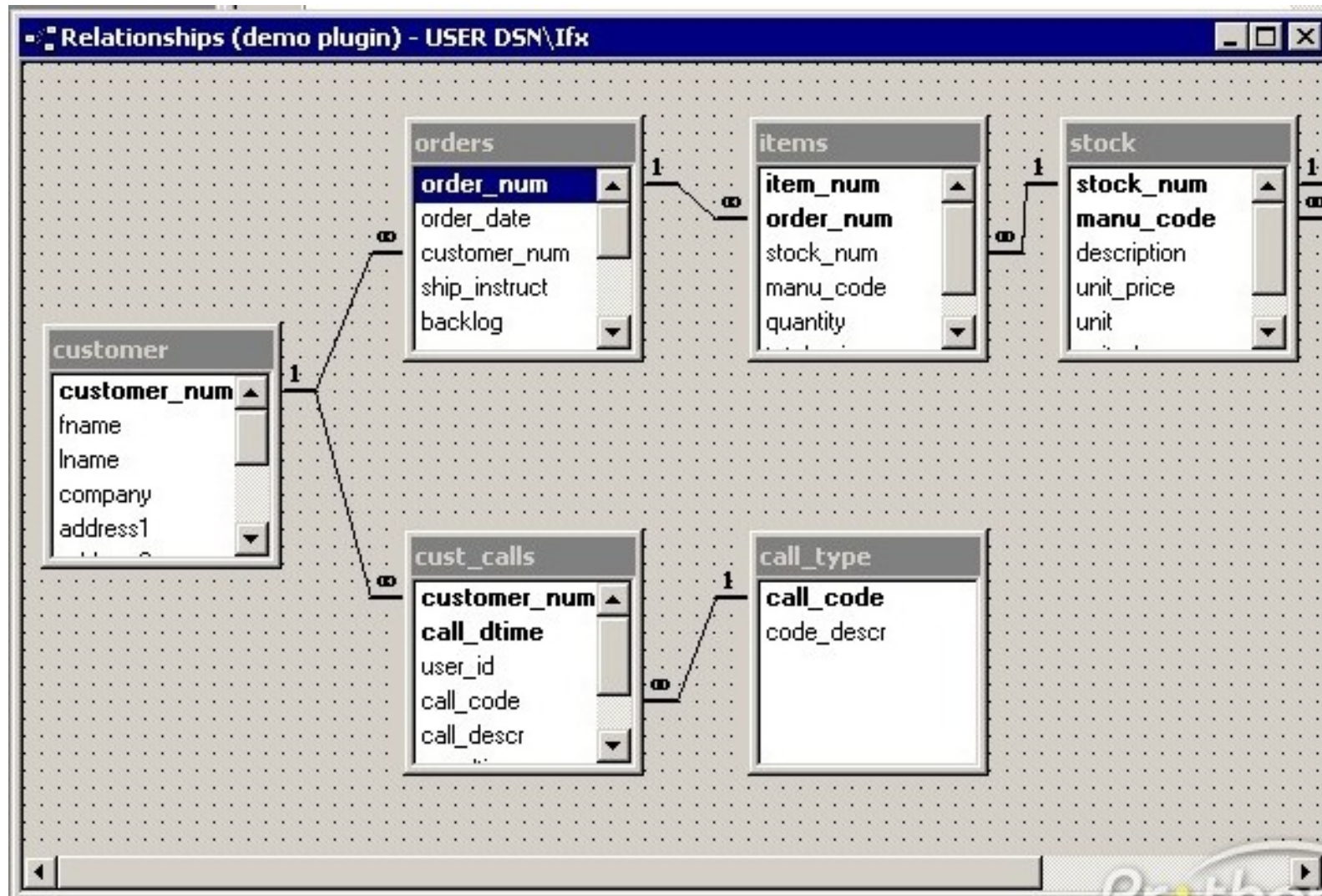
ITERAÇÃO



- As funções ou procedimentos e podem ser executados várias vezes;
- As funções podem receber parâmetros fazendo com que o resultado do seu processamento interno varie de acordo com os argumentos (parâmetros) de entrada;
- É possível executar uma função sob diversas circunstâncias diferentes;
- Estes itens configuram um importante conceito da engenharia de software: o **REUSO**;
- Além disto, uma boa modularização deve ter **alta coesão e baixo acoplamento**;

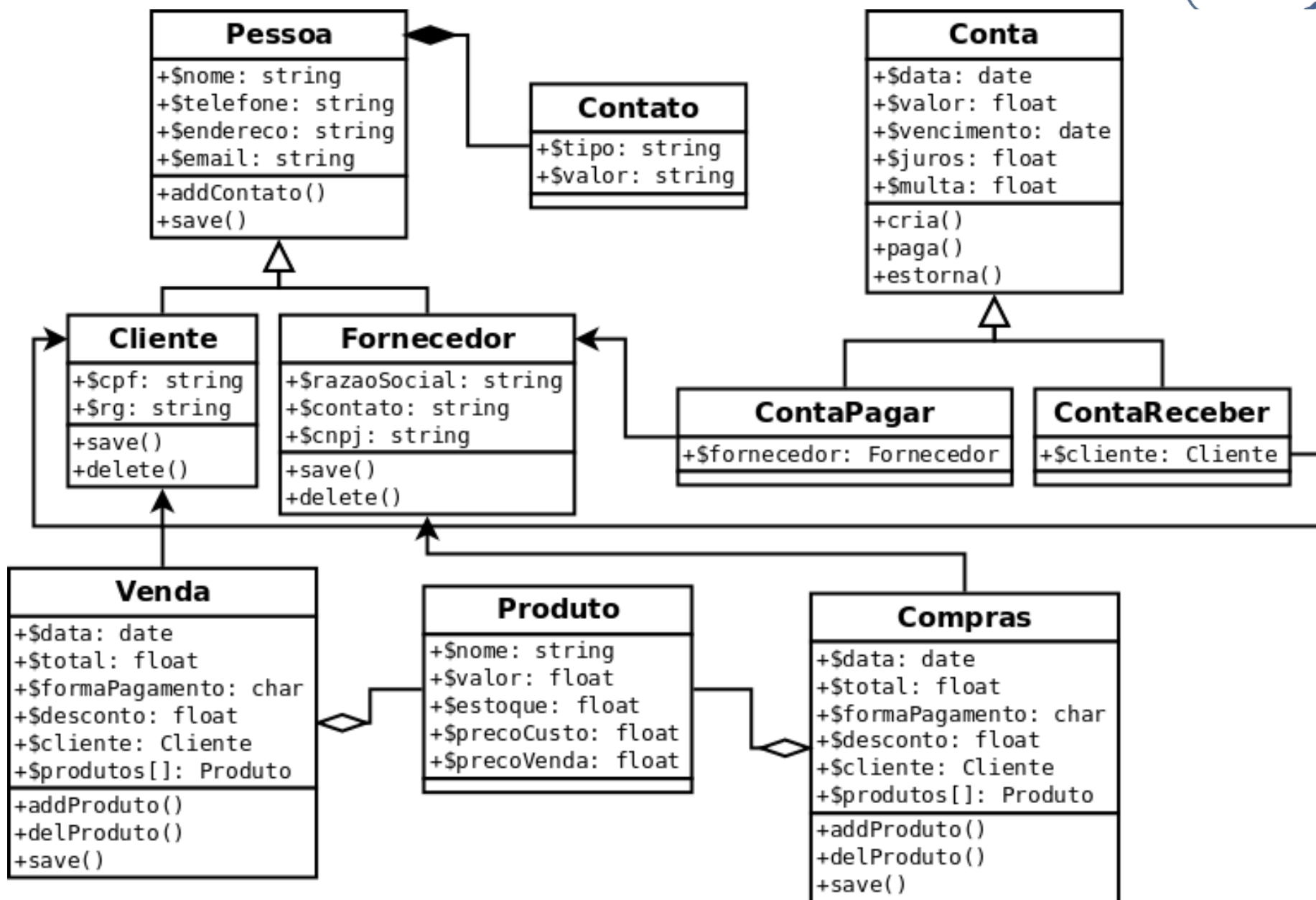


Análise Estruturada: Em primeiro lugar se pensa nos dados e nas estruturas que irão armazená-los (E.R.).



Análise O.O.: Em primeiro lugar se pensa nos conceitos, seus atributos, seu comportamento e nos seus relacionamentos.





- O sistema é organizado através de um conjunto de objetos;
- Uma entidade que possui **atributos**, **comportamento** e se **relaciona** com outros objetos por meio de **mensagens**;
- Um objeto pode ser algo **concreto** (pessoa, bicicleta, um pedido) ou **abstrato** (uma botão, uma janela, um arquivo);
- O objeto possui **responsabilidade** sobre si (*encapsulamento*);
- Os objetos propiciam maior **compreensão** do mundo real;
- A orientação a objetos leva à um baixo grau de acoplamento;
- Algumas linguagens Orientadas a Objetos:
 - Smalltalk;
 - C++;
 - Java;
 - PHP;





Pessoa

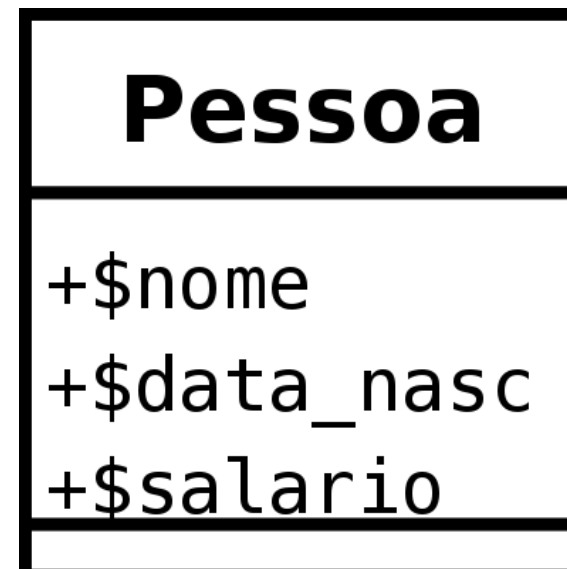
```
+CTPS: integer  
+CPF: integer  
+RG: integer  
+Nome: string  
-Salario: float  
-Cargo: string  
-DtContrata: date  
-DtDemissao: date
```

```
+Admitir(Data)  
+Demitir(Data)  
+Promover(Cargo, Salario)  
+GetSalario()
```

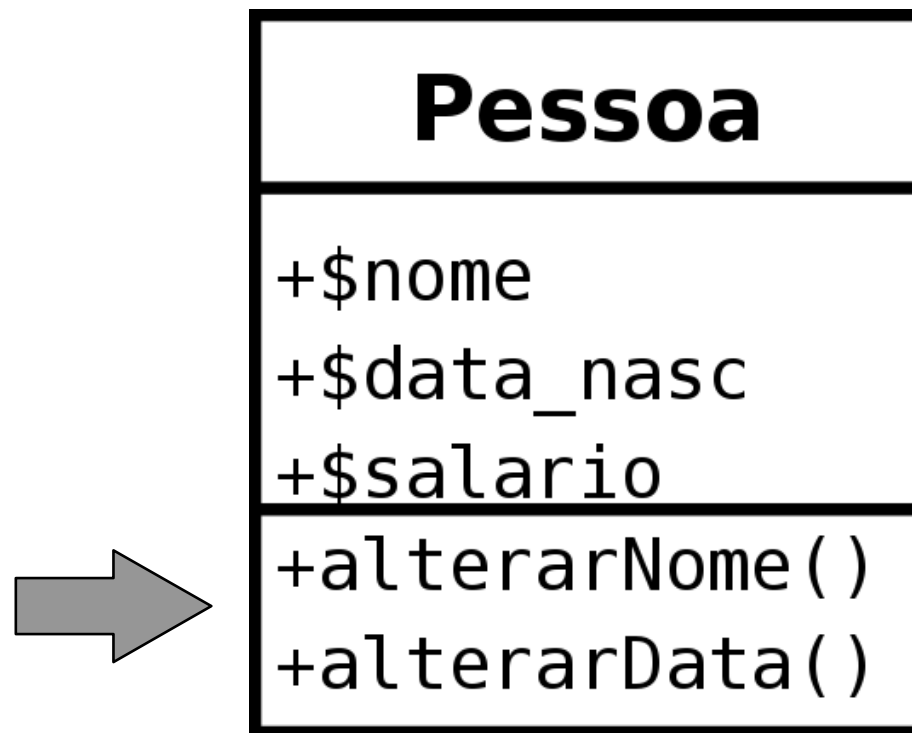
- A classe é uma estrutura estática utilizada para descrever (**moldar**) objetos;
- A classe é um modelo (**template**) para criação de objetos;
- Podem ser classes: entidades do **negócio** da aplicação (pessoa, conta, cliente), entidades de **interface** (janela, botão, painel, frame, barra), dentre outras (conexão com BD, um arquivo XML, um Web Service);
- Um grupos de objetos é descrito por uma classe;
- Um objeto é uma instância de uma classe.
- Exemplo:
 - Classe: Fatura / Objeto: Fatura no. 5470
 - Classe: Pessoa / Objeto: João, Maria, etc...



- Propriedades são **Atributos**, características de um objeto;
- Os atributos definem a identidade de um objeto;
- **Exemplo:** Nós, seres humanos, somos definidos pelo conjunto de nossos atributos: físicos (altura, cor da pele, cor do cabelo), psicológicos (personalidade, humor, empatia), de trabalho (capacidade, especialização, criatividade), de estudo (nível de escolaridade), dentre outros.



- Os métodos são **operações** que definem o comportamento de um objeto;
- Os métodos definem como o objeto irá se **relacionar** com o mundo externo;
- É por meio de um método, que **solicitamos** que um objeto faça algo.



```
<?php
class Pessoa
{
    // define os atributos
    public $Codigo;
    public $Nome;
    public $Altura;
    public $Idade;

    // define os métodos
    function setNome($nome)
    {
        $this->Nome = $nome;
    }

    function Crescer($centimetros)
    {
        $this->Altura += $centimetros;
    }

    function Envelhecer($anos)
    {
        $this->Idade += $anos;
    }
}
?>
```

← Atributos

← Métodos

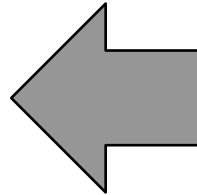
← Métodos

← Métodos



```
<?php
// inclui a classe
include 'pessoa.class.php';
```

```
// instancia o objeto
$maria = new Pessoa;
```



Instanciando o objeto

```
// define atributos
$maria->Altura = 1.7;
$maria->Idade = 28;
```

```
// executa métodos
$maria->setNome('Maria da Silva');
$maria->Crescer(0.1);
$maria->Envelhecer(1);
```

```
// imprime o objeto
var_dump($maria);
?>
```

output

```
object(Pessoa)#1 (4) {
  ["Codigo"]=> NULL
  ["Nome"]=>
    string(14) "Maria da Silva"
  ["Altura"]=> float(1,8)
  ["Idade"]=> int(29)
}
```

```
<?php
// instancia um objeto (janela)
$janela = new GtkWindow;
// define o tamanho da janela
$janela->set_size_request(300,200);

// instancia um rótulo de texto
$label = new GtkLabel('Olá Mundo');
// adiciona o rótulo à janela
$janela->add($label);
// exibe a janela
$janela->show_all();

// controle Gtk
Gtk::Main();
?>
```



- Um **construtor** é um método especial utilizado para definir o **comportamento inicial** de um objeto, ou seja, o comportamento no momento de sua criação;
- O método construtor é executado automaticamente no momento em que instanciamos um objeto por meio do operador **new**;
- Um método **destrutor** ou finalizador é um método especial executado automaticamente quando o objeto é **desalocado** da memória, quando atribuímos o valor **NULL** ao objeto, quando utilizamos a função **unset()** sobre o mesmo ou, em última instância, quando o programa é finalizado;
- O método destrutor pode ser utilizado para **finalizar** conexões, apagar arquivos temporários criados durante o ciclo de vida do objeto, dentre outros.



```
<?php
class Pessoa
{
    public $Codigo;
    public $Nome;

    // método construtor
    function __construct($codigo, $nome)
    {
        $this->Codigo = $codigo;
        $this->Nome = $nome;
    }

    // método destrutor
    function __destruct()
    {
        echo "desalocando {$this->Nome}\n";
    }
}
```

```
$maria = new Pessoa(27, 'Maria da Silva');
$joana = new Pessoa(28, 'Joana Maranhão');
var_dump($maria, $joana);
unset($maria);
unset($joana);
?>
```

output

```
object(Pessoa)#1 (2) {
    ["Codigo"]=> int(27)
    ["Nome"]=>
    string(14) "Maria da Silva"
}

object(Pessoa)#2 (2) {
    ["Codigo"]=> int(28)
    ["Nome"]=>
    string(14) "Joana Maranhão"
}

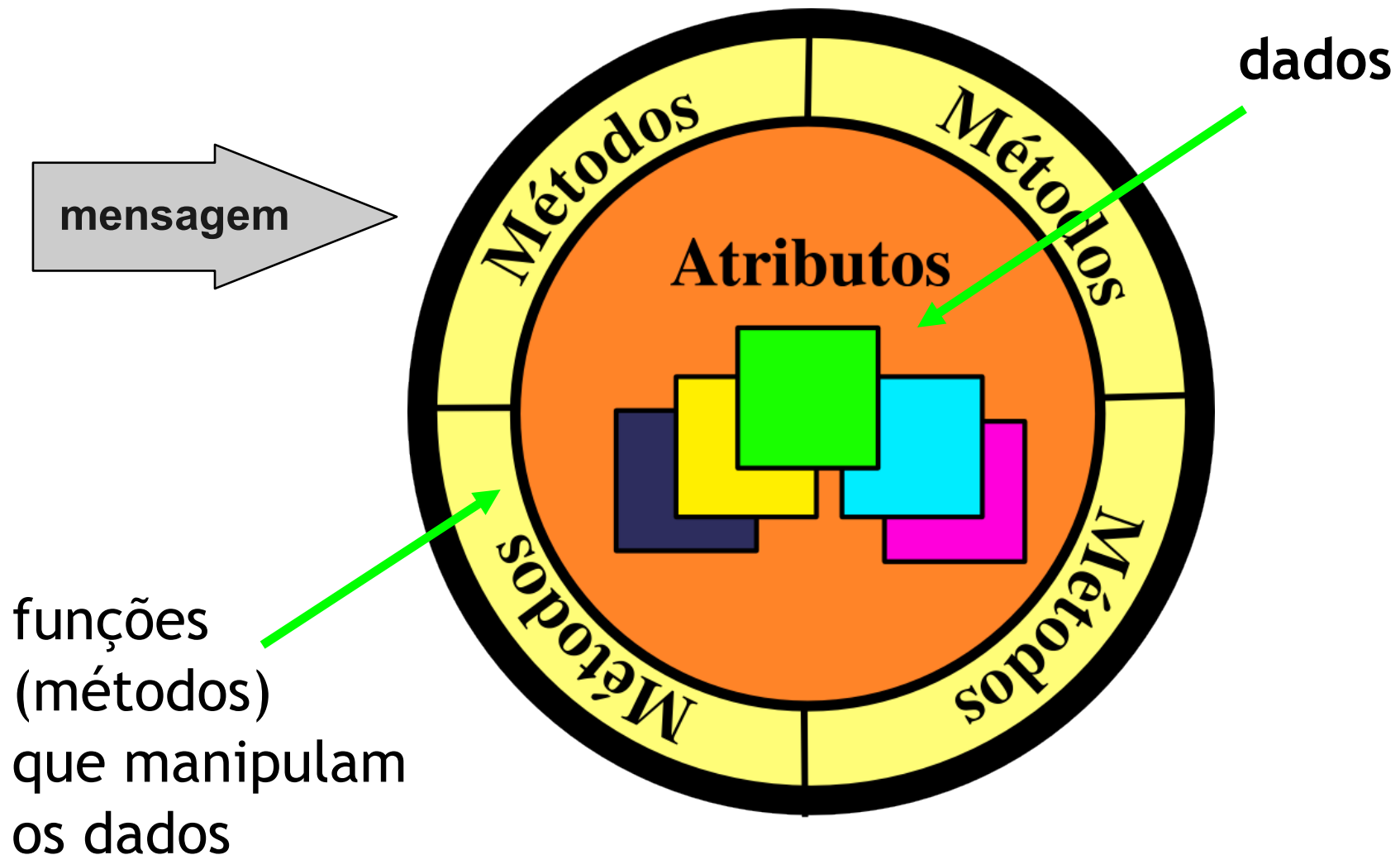
desalocando Maria da Silva
desalocando Joana Maranhão
```

Método construtor

Método destrutor

- As classes são orientadas ao assunto, ou seja, cada classe é responsável por um **assunto** diferente e é responsável sobre o mesmo, ou seja, deve proteger o seu acesso;
- A proteção ao acesso ao seu conteúdo se dá por meio de mecanismos como o de **encapsulamento**;
- O encapsulamento visa **separar** os aspectos externos de um objeto dos detalhes internos daquele objeto.
- É uma forma de proteger certos atributos, evitando que os mesmos contenham valores **inconsistentes** ou sejam manipuladas indevidamente;
- **Exemplo:** Datas, Valores Numéricos;





```
<?php
class Pessoa
{
    private $Codigo;
    public $Nome;
    private $Altura;

    function __construct($codigo)
    {
        $this->Codigo = $codigo;
    }

    function setAltura($altura)
    {
        $this->Altura = $altura;
    }
}
```

```
$maria = new Pessoa(27);
$maria->Nome = 'Maria da Silva';
$maria->setAltura(1.7);
```

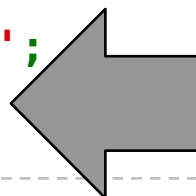
```
$joana = new Pessoa(28);
$joana->Nome = 'Joana Maranhão';
$joana->Altura = 1.8;
?>
```

output

Fatal error: Cannot access private property Pessoa::\$Altura in exemplo.php on line 25

Tipos de visibilidade

PRIVATE :: SOMENTE PRÓPRIA CLASSE
PROTECTED :: CLASSE E DESCENDENTES
PUBLIC :: DE QUALQUER PONTO



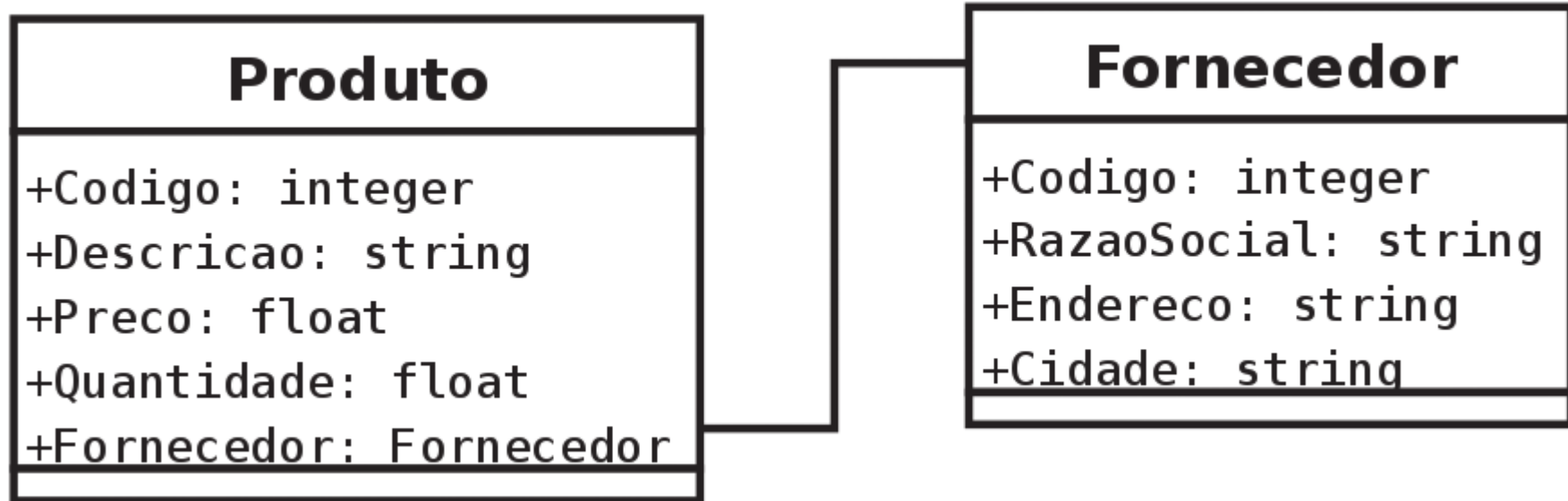
Acesso não permitido



- Associação é a relação mais comum entre dois objetos;
- Na associação, um objeto possui uma **referência** à outro objeto, podendo visualizar seus atributos ou mesmo acionar uma de suas funcionalidades (métodos);
- A forma mais comum de implementar uma associação é ter um objeto como **atributo** de outro;
- No exemplo a seguir, criamos um objeto do tipo **Produto** e outro do tipo **Fornecedor**;
- Um dos atributos do produto é o fornecedor;
- Leia-se um objeto - **está relacionado com** - outro objeto;



Exemplo em diagrama UML

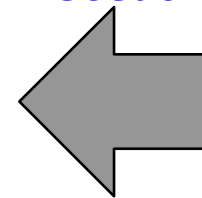


```
<?php
class Produto
{
    public $Codigo;
    public $Descricao;
    public $Preco;
    public $Fornecedor;

    // exibe os dados do produto
    function ExibeDados()
    {
        echo 'Codigo: ' . $this->Codigo . "<br>\n";
        echo 'Descricao: ' . $this->Descricao . "<br>\n";
        echo 'Preço: ' . $this->Preco . "<br>\n";
        echo 'Fornecedor:' . $this->Fornecedor->Nome . "<br>\n";
    }

    // atribui um fornecedor ao produto
    function setFornecedor(Fornecedor $fornecedor)
    {
        $this->Fornecedor = $fornecedor;
    }
}
?>
```

```
<?php
class Fornecedor
{
    public $Nome;
    public $Telefone;
    public $Endereco;
}
?>
```



Associação interna

```
<?php
include_once 'classes/Produto.class.php';
include_once 'classes/Fornecedor.class.php';
```

```
$macarrao = new Produto;
```

```
$macarrao->Codigo      = 7;
$macarrao->Descricao   = 'Macarrão instantâneo';
$macarrao->Preco       = 1.29;
$macarrao->Quantidade  = 10;
```

```
$isabela = new Fornecedor;
$isabela->Nome        = 'Massas Isabela';
$isabela->Telefone    = '(51) 1234-5678';
$isabela->Endereco    = 'Rua das Massas';
```

```
// associacao
$macarrao->setFornecedor($isabela);
```

```
// exibe as informacoes
$macarrao->ExibeDados();
?>
```

output

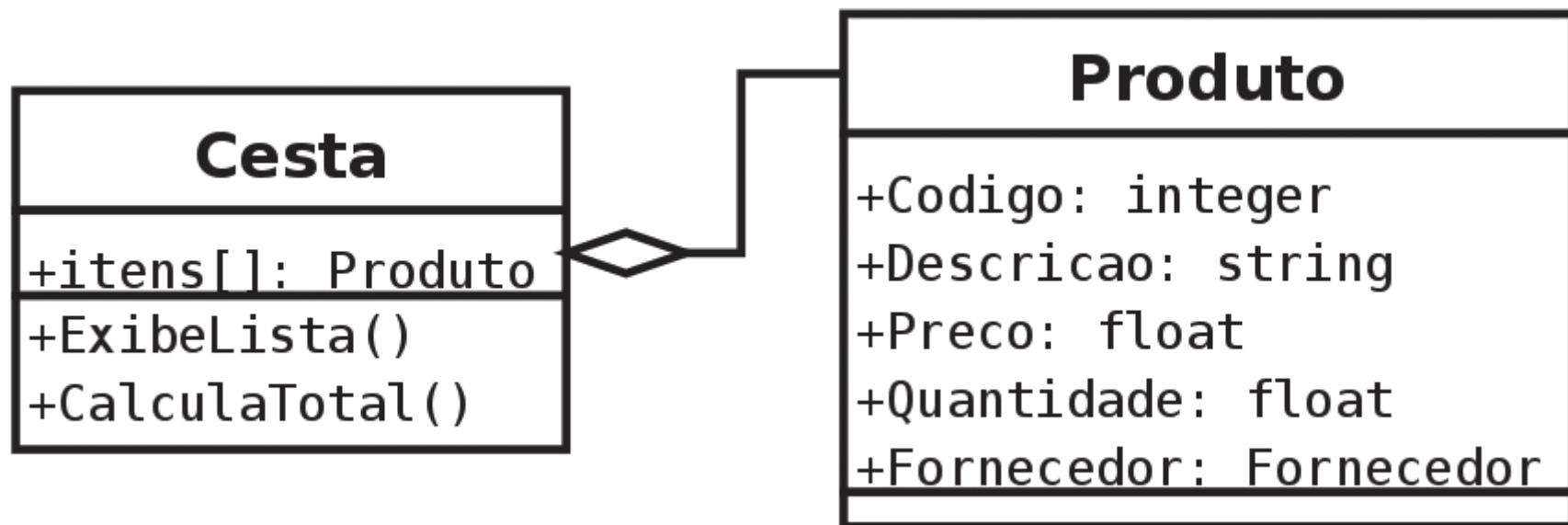
```
Codigo: 7
Descricao: Macarrão instantâneo
Preço: 1,29
Fornecedor:Massas Isabela
```

Método de associação

- A agregação é o tipo de relação entre objetos conhecida como **todo/parte**;
- Na agregação, um objeto **agrega** outro objeto, ou seja, referencia objeto(s) externo dentro de si;
- O objeto-pai poderá agregar uma ou muitas **instâncias** de um outro objeto e poderá utilizar **funcionalidades** do objeto agregado;
- A forma mais simples de implementar agregações é utilizando **arrays**;
- Criamos um array como **atributo** do objeto, sendo que o papel deste array é armazenar inúmeras instâncias de um outro objeto;
- Leia-se um objeto - **contém** - instâncias de outros objetos.



Exemplo em diagrama UML



```
<?php
class CestaDeCompras
{
    private $Produtos; // colecao de objetos (array)
    function AdicionarItem(Produto $produto)
    {
        $this->Produtos[] = $produto;
    }
    function ExibeLista()
    {
        foreach ($this->Produtos as $produto)
        {
            echo $produto->Descricao . "<br>\n";
        }
    }
    function CalculaTotal()
    {
        $total = 0;
        foreach ($this->Produtos as $produto)
        {
            $total += $produto->Preco;
        }
        return $total;
    }
}
?>
```

← Método de agregação

```
<?php
class Produto
{
    public $Codigo;
    public $Descricao;
    public $Preco;
}
```

```
$chocolate = new Produto;
$chocolate->Codigo=4;
$chocolate->Descricao = 'Chocolate Sensação';
$chocolate->Preco = 1.2;
```

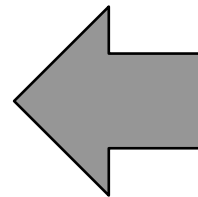
```
$picanha = new Produto;
$picanha->Codigo=7;
$picanha->Descricao = 'Picanha Bovina';
$picanha->Preco = 18;
```

```
$cesta = new CestaDeCompras;
$cesta->AdicionarItem($chocolate);
$cesta->AdicionarItem($picanha);
```

```
$cesta->ExibeLista();
echo 'Total : ';
echo $cesta->CalculaTotal();
?>
```

output

```
Chocolate Sensação
Picanha Bovina
Total : 19,2
```



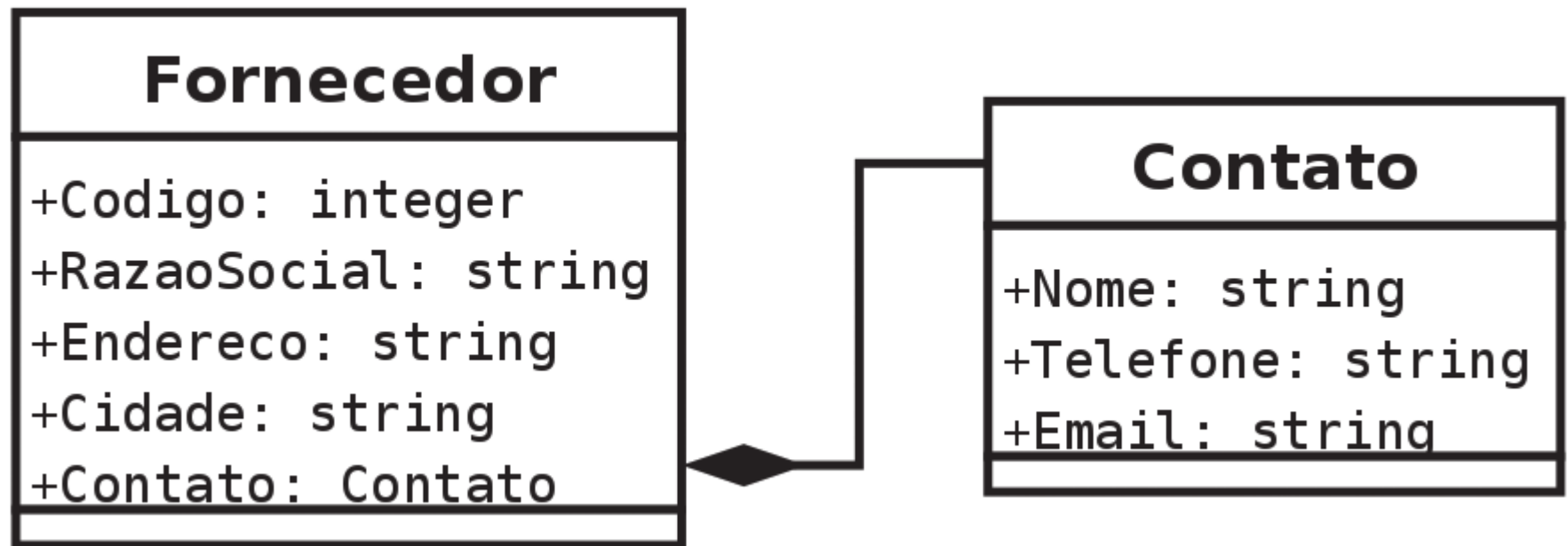
Chamada à agregação



- Composição também é uma relação **todo/parte**;
- A diferença em relação à agregação é que, na composição, o objeto-pai ou “todo” é responsável pela **criação** e **destruição** de suas partes;
- O objeto-pai realmente “possui” a(s) instância(s) de suas partes. Diferentemente da agregação, na qual o “todo” e as “partes” são **independentes**;
- Na agregação, ao destruímos o objeto “todo” as “partes” permanecem na memória, por terem sido criadas fora do **escopo** da classe “todo”. Já na composição, quando o objeto “todo” é destruído, suas “partes” também são;
- Leia-se um objeto - **é composto de** - outros objetos;



Exemplo em diagrama UML

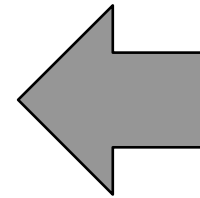


```
<?php
class Fornecedor
{
    public $Codigo;
    public $RazaoSocial;
    public $Contatos;

    function addContato($nome, $telefone, $email)
    {
        $contato = new Contato;
        $contato->Nome      = $nome;
        $contato->Telefone  = $telefone;
        $contato->Email     = $email;
        $this->Contatos[]  = $contato;
    }

    function ImprimeContatos()
    {
        foreach ($this->Contatos as $contato)
        {
            var_dump($contato);
        }
    }
}
?>
```

```
<?php
class Contato
{
    public $Nome;
    public $Telefone;
    public $Email;
}
?>
```



Método de composição



```
<?php
include_once 'classes/Fornecedor.class.php';
include_once 'classes/Contato.class.php';

// instancia novo fornecedor
$fornecedor = new Fornecedor;
$fornecedor->Codigo          = 10;
$fornecedor->RazaoSocial    = 'Produtos Bom Gosto S.A.';

// atribui informações de contato
$fornecedor->addContato('Mauro', '51 1234-5677', 'mauro@bomgosto.com.br');
$fornecedor->addContato('Maria', '51 1234-5678', 'maria@bomgosto.com.br');
$fornecedor->addContato('Joana', '51 1234-5679', 'joana@bomgosto.com.br');

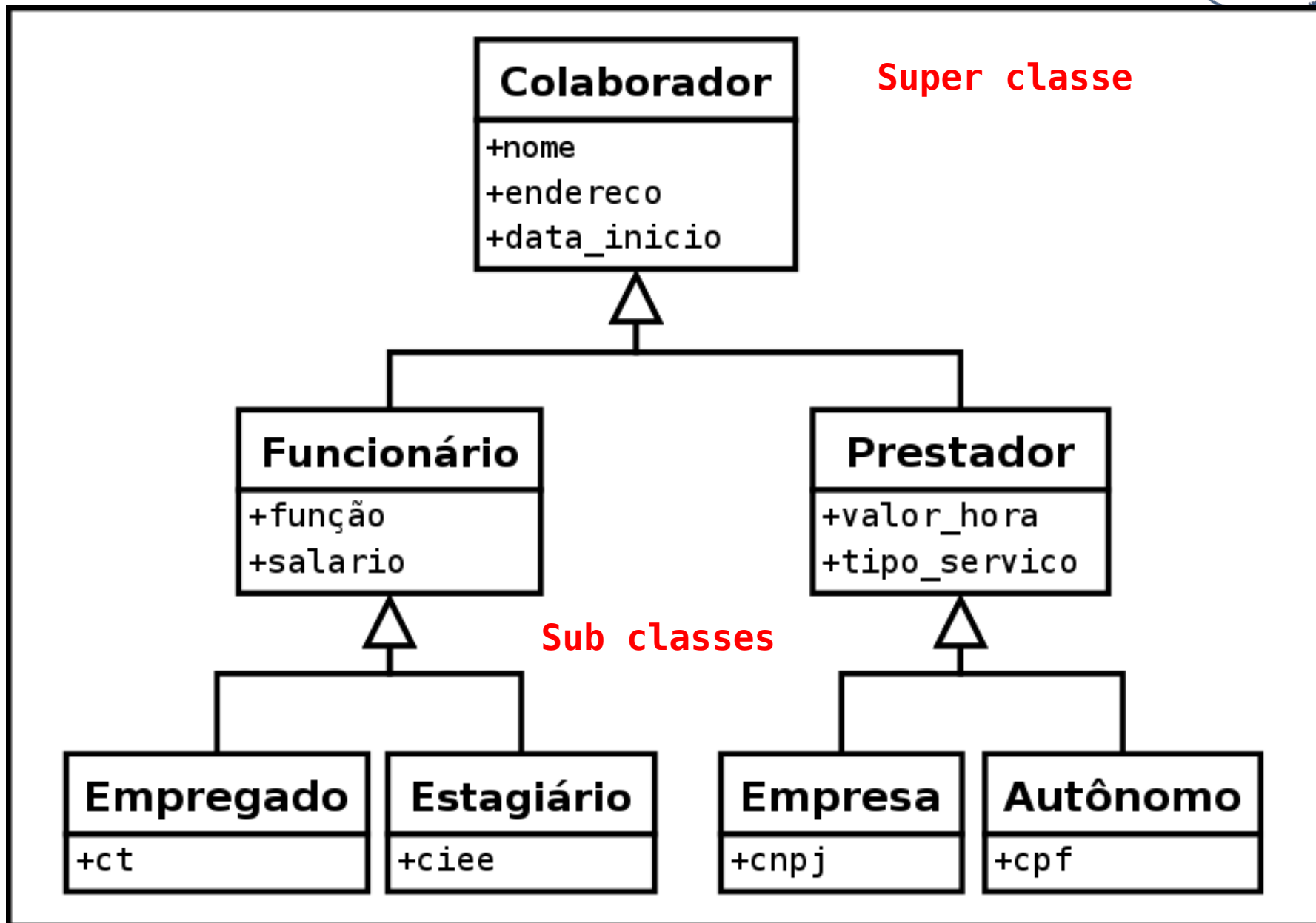
// debug
var_dump($fornecedor);
?>
```

output

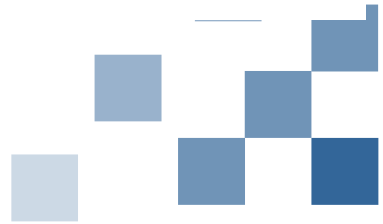
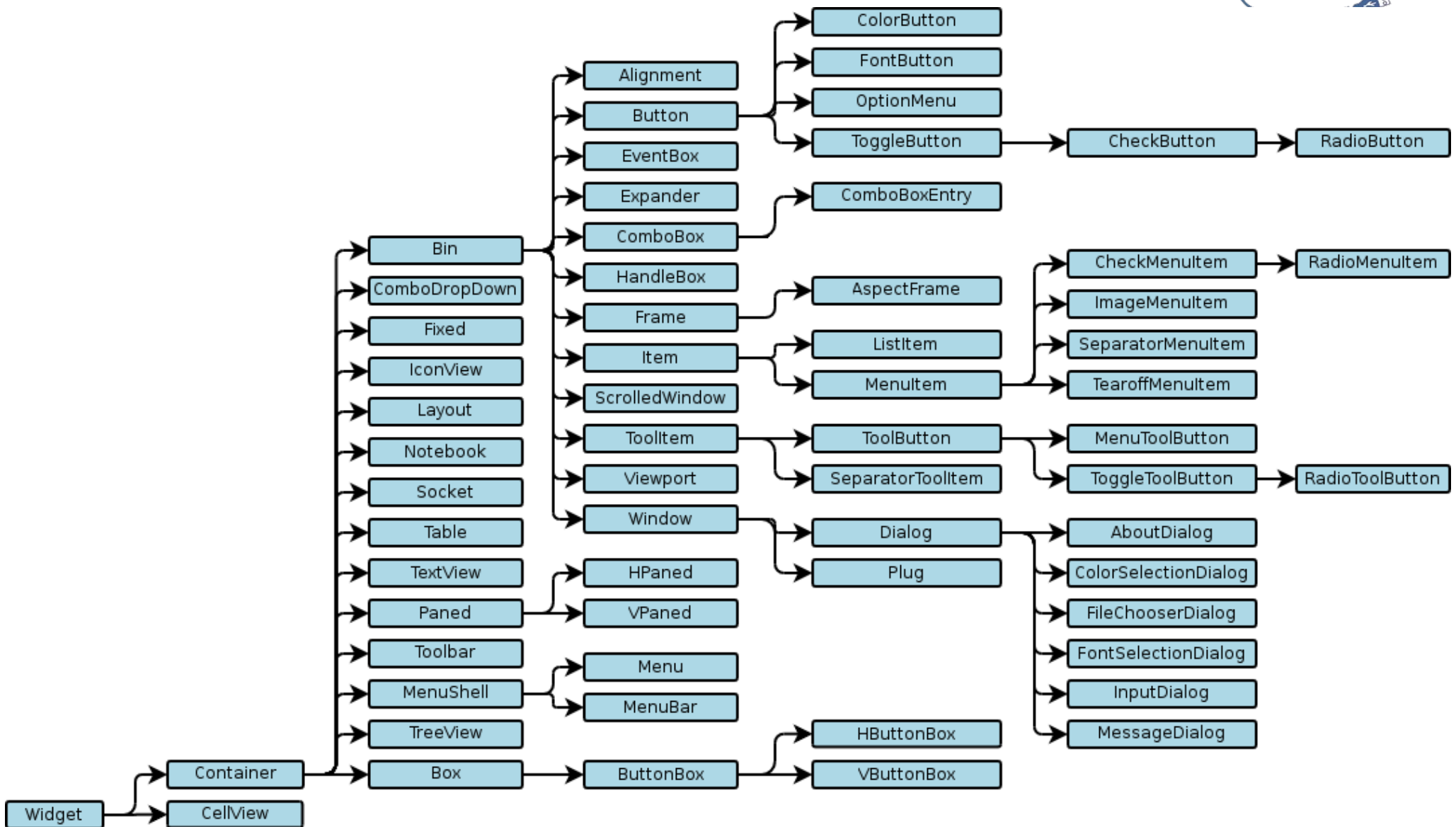
```
Fornecedor Object
(
    [Codigo] => 10
    [RazaoSocial] => Produtos Bom Gosto S.A.
    [Contatos] => Array
        [0] => Contato Object
            [Nome] => Mauro
            [Telefone] => 51 1234-5677
            [Email] => mauro@bomgosto.com.br
        [1] => Contato Object ...
```

- Um dos maiores benefícios na utilização da orientação a objetos é o **reuso**;
- A possibilidade de reutilizar partes de código nos dá maior agilidade no dia-a-dia, além de eliminar a necessidade de eventuais **duplicações** ou reescritas de código;
- Uma herança é um tipo de relacionamento que permite **especializar** uma classe, criar versões refinadas dela;
- Na herança, as classes são organizadas em **hierarquias**;
- A herança é uma forma de reutilizar componentes de software aperfeiçoando-os ou adicionando características específicas;





Hierarquia de Classes



- A herança permite o **compartilhamento** de atributos e métodos entre as classes de uma hierarquia;
- Cada subclasse **herda** todas as propriedades (atributos e métodos) de suas ancestrais;
- Uma subclasse pode **estender** ou **redefinir** a estrutura e/ou o comportamento de sua super classe;
- Leia-se um objeto - **é tipo de** - outro objeto;
- É um poderoso instrumento de reusabilidade, pois:
 - permite que atributos e operações comuns à hierarquia sejam especificados apenas uma vez;
 - Permite que novas classes sejam criadas contendo apenas a diferença entre ela e a classe-pai.



```
<?php
abstract class Conta
{
    private $Agencia;
    private $Numero;
    protected $Saldo;

    function __construct($agencia, $numero, $saldo)
    {
        $this->Agencia = $agencia;
        $this->Numero = $numero;
        $this->Saldo = $saldo;
    }

    function Depositar($valor)
    {
        $this->Saldo += $valor;
    }

    function getSaldo()
    {
        return $this->Saldo;
    }
}
?>
```

Uma classe abstrata
não pode ser instanciada
diretamente.



```
<?php
final class ContaCorrente extends Conta
{
    public $Limite;

    function Retirar($valor)
    {
        if ($this->Saldo + $this->Limite > $valor)
        {
            $this->Saldo -= $valor;
        }
    }
}
```

Uma classe final não
pode ser estendida.

← Classe filha

```
final class ContaPoupanca extends Conta
{
    function Retirar($valor)
    {
        if ($this->Saldo > $valor)
        {
            $this->Saldo -= $valor;
        }
    }
}
?>
```

← Classe filha



```
<?php
include_once 'classes/Conta.class.php';
include_once 'classes/ContaPoupanca.class.php';
include_once 'classes/ContaCorrente.class.php';

$conta1 = new ContaPoupanca('1', '123', 100);
$conta2 = new ContaCorrente('2', '456', 200);
$conta2->Limite = 500;

echo 'saldo conta1: ' . $conta1->getSaldo(); // exibir 100
echo 'saldo conta2: ' . $conta2->getSaldo(); // exibir 200

$conta1->Depositatar(300);
$conta2->Depositatar(500);

echo 'saldo conta1: ' . $conta1->getSaldo(); // exibir 400
echo 'saldo conta2: ' . $conta2->getSaldo(); // exibir 700

$conta1->Retirar(500); // saldo maximo e 400, nao deve permitir
$conta2->Retirar(1199); // 200+500+500(limite) saldo = 1200 (permitir)

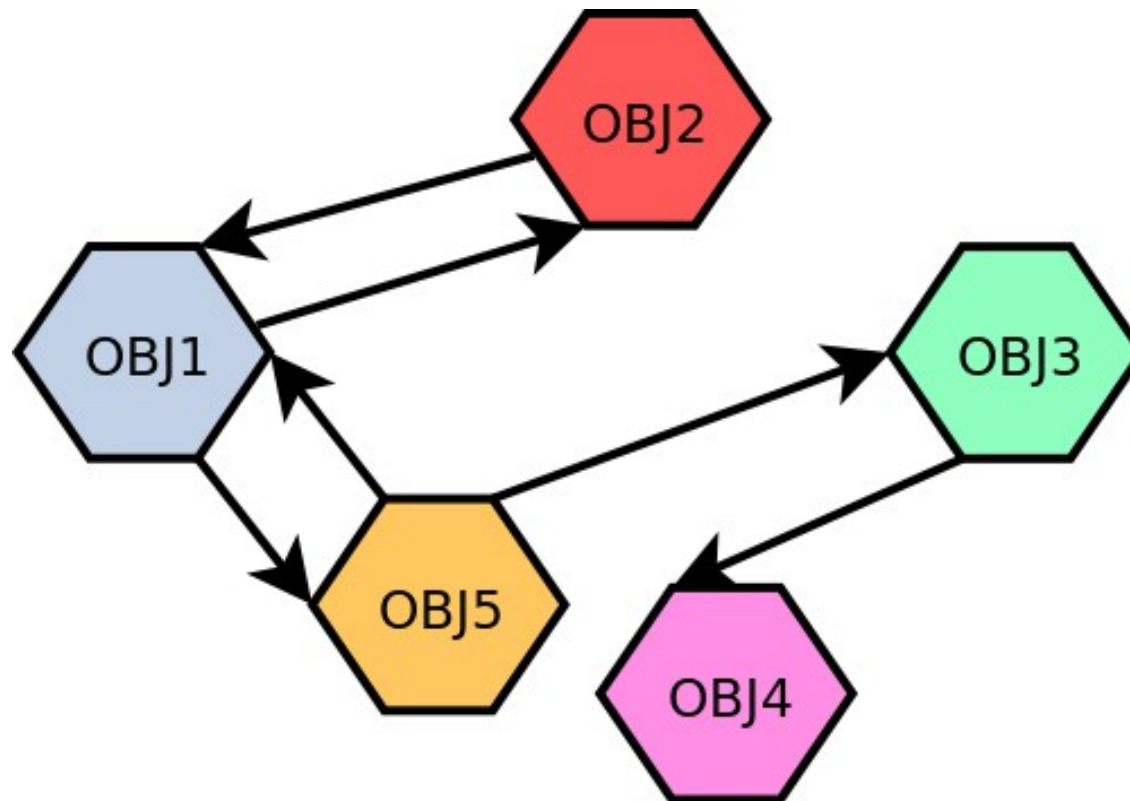
echo 'saldo conta1: ' . $conta1->getSaldo(); // exibir 400
echo 'saldo conta2: ' . $conta2->getSaldo(); // exibir -499
?>
```

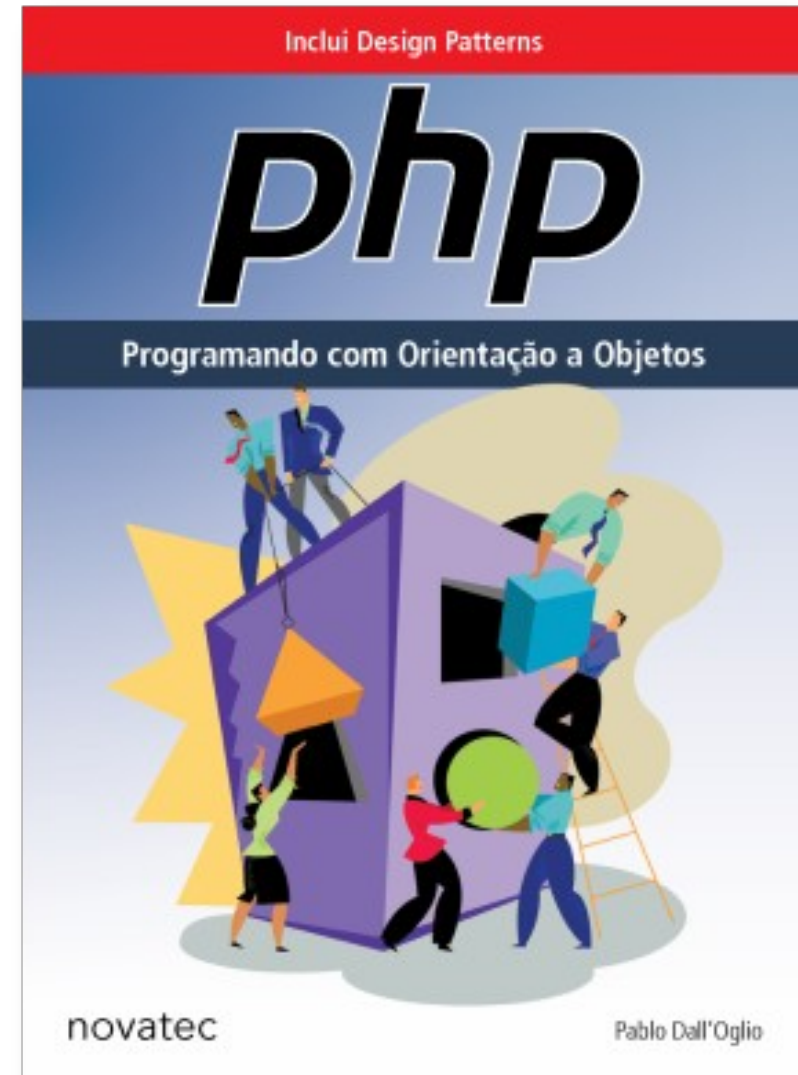
output

```
saldo conta1: 100
saldo conta2: 200
saldo conta1: 400
saldo conta2: 700
saldo conta1: 400
saldo conta2: -499
```



- Um sistema OO é modelado, implementado e efetivamente funciona como um conjunto de objetos que interagem entre si.





Obrigado!



E-Mail

pablo@php.net

pablo@dalloaglio.net

URL

<http://www.adianti.com.br>

<http://www.pablo.blog.br>

