Orientação a Objetos

Classe

```php
<?php
    class Produto {
        public $descricao;
        public $estoque;
        public $preco;
    }
```

```php
$p1 = new Produto;
$p1->descricao = 'Chocolate';
$p1->estoque   = 10;
$p1->preco     = 7;

$p2 = new Produto;
$p2->descricao = 'Café';
$p2->estoque   = 20;
$p2->preco     = 4;
// output objeto inteiro
var_dump($p1);
var_dump($p2);
```

Métodos

```
<?php
    class Produto {
        public $descricao;
        public $estoque;
        public $preco;

        public function aumentarEstoque($unidades) {
            if (is_numeric($unidades) AND $unidades >= 0) {
                $this->estoque += $unidades;
            }
        }

        public function diminuiEstoque($unidades) {
            if (is_numeric($unidades) AND $unidades >=0) {
                $this->estoque -= $unidades;
            }

            public function reajustarPreco($percentual) {
                if (is_numeric(percentual) AND $percentual >= 0) {
                    $this->preco *= (1 + ($percentual/100));
                }
            }
        }
    }
```

Métodos

```
$p1 = new Produto;
$p1->descricao = 'Chocolate';
$p1->estoque   = 10;
$p1->preco     = 8;
echo "O estoque de " . $p1->descricao . " é " .  $p1->estoque . "<br>\n";
echo "O preço de " . $p1->descricao . " é " . $p1->preco ."<br>\n";

$p1->aumentarEstoque(10);
$p1->diminuiEstoque(5);
$p1->reajustarPreco(50);
echo "O estoque de " . $p1->descricao . " é " .  $p1->estoque . "<br>\n";
echo "O preço de " . $p1->descricao . " é " . $p1->preco ."<br>\n";
```

Private - sem getters and setters

```
class Produto {
        private $descricao;
        private $estoque;
        private $preco;
    }
```

```
$p1 = new Produto;
$p1->descricao = 'Chocolate';
$p1->estoque   = 10;
$p1->preco     = 8;
```

Result: Fatal Error: Cannot access private property Produto::$descricao in objeto3.php on line 10

Private - com getters and setters

```
class Produto {
        //exemplo de private com getters and setters
        private $descricao;
        private $estoque;
        private $preco;

        public function setDescricao($descricao) {
            if (is_string($descricao)) {
                $this->descricao = $descricao;
            }
        }

        public function getDescricao() {
            return $this->descricao;
        }
```

Private - com getters and setters

```
public function setEstoque($estoque) {
            if (is_numeric($estoque) AND $estoque > 0) {
                $this->estoque = $estoque;
            }
        }

        public function getEstoque() {
            return $this->estoque;
        }

    }
     $p1 = new Produto;
            $p1->setDescricao('Chocolate');
            $p1->setEstoque(10);
            echo "Descrição: " . $p1->getDescricao() . '<br>' . PHP_EOL;
            echo "Estoque: " . $p1->getEstoque() . '<br>' . PHP_EOL;
```

Métodos construtores e destrutores

No PHP um método construtor deve ter o nome __construct(), pois é uma convenção da linguagem.

```
class Produto {
        private $descricao;
        private $estoque;
        private $preco;

        public function __construct($descricao, $estoque, $preco) {
            if (is_string($descricao)) {
                $this->descricao = $descricao;
            }
            if (is_string($estoque) AND $estoque > 0) {
                $this->estoque = $estoque;
            }
            if (is_string($preco) AND $preco > 0) {
                $this->preco = $preco;
            }
        }
```

Métodos construtores e destrutores

No PHP um método construtor deve ter o nome __construct(), pois é uma convenção da linguagem.

```
    public function getDescricao() {
        return $this->descricao;
    }
    public function getEstoque() {
        return $this->estoque;
    }
    public function getPreco() {
        return $this->preco;
    }
}
$p1 = new Produto('Chocolate', 10, 5);
    echo 'Descrição: ' . $p1->getDescricao() . '<br>' . PHP_EOL;
    echo 'Estoque: ' . $p1->getEstoque() . '<br>' . PHP_EOL;
    echo 'Preço: ' . $p1-getPreco() . '<br>' . PHP_EOL;
```

Métodos construtores e destrutores

PHP também implementa o conceito de método destrutor. Um destrutor é um método especial com o nome __destruct() (convenção da linguagem) executado automaticamente quando o objeto é desalocado da memória, o que pode acontecer em algumas circunstâncias, como: quando atribuímos o valor NULL ao objeto; quando o programa é finalizado, e então todos os objetos são desalocados automaticamente. como exemplos de utilização, o método destrutor pode ser utilizado para finalizar conexões, apagar aquivos teporários e desfazer outras operações criadas durante o ciclo de vida do objeto.

Métodos construtores e destrutores

```
<?php
        class Produto {
            private $descricao;
            private $estoque;
            private $preco;

            public function __construct($descricao, $estoque, $preco) {
                $this->descricao = $descricao;
                $this->estoque   = $estoque;
                $this->preco     = $preco;

                echo "CONSTRUÍDO: Objeto " . $descricao . ", estoque " . ", preco " . $preco . "<br>\n";
            }

            public function __destruct() {
                echo "DESTRUÍDO: Objeto " . $this->$descricao . ", estoque " . $this->$estoque . ", preco
" . $this->$preco . "<br>\n";
            }
        }
```

Métodos construtores e destrutores

```
$p1 = new Produto('Chocolate', 10, 5);
    unset($p1);
    $p2 = new Produto('Café', 100, 7);
    unset($p2);
```

Conversão de tipos

No PHP podemos criar objetos sem ter uma classe definida. Isso pe possível porque no PHP existe uma classe predefinida chamada stdClass (Standard Class), que é uma classe vazia (sem atributos e métodos). Essa é utilizada também quando realizamos conversões de tipo, como de array para objeto, como será demonstrado no exemplo. Criamos uma objeto $produto da classe stdClass.

Inicialmente serão definidos alguns atributos. Ao final utilizaremos o print_r() para exibir a estrutura do objeto criado.

Conversão de tipos

```
<?php
        $produto = new StdClass;
        $produto->descricao = 'Chocolate Amargo';
        $produto->estoque   = 100;
        $produto->preco     = 7;

        print_r($produto);


        Result:
        stdClass Object (
            [descrica] => Chocolate Amargo
            [estoque] => 100
            [preco] => 7
        )
```

Conversão de tipos

Podemos realizar operações de tipo (casting), como criar um objeto a partir de um vetor e vice-versa, por meio da utilização de um operador que utiliza o tipo de destino entre parênteses. Neste exemplo vamos declarar um objeto $produto, com alguns atributos. Em seguida vamos convertê-lo em array ($vetor1) por meio da operação de casting (array). Mais adiante criaremos o $vetor2, um array criado pela sintaxe resumida de criação de vetores (utilizando parênteses []). Em seguida vamos convertê-lo em object ($product2) por meio da operação de castinho (object).

Conversão de tipos

```
<?php
        $produto = new StdClass;
        $produto->descricao = 'Chocolate Amargo';
        $produto->estoque   = 100;
        $produto->preco     = 7;

        $vetor1 = (array) $produto;
        echo $vetor1['descricao'] . "<br>\n";

        $vetor2 = ['descricao' => 'Café', 'estoque' => 100, 'preco' => 7];
        $produto2 = (object) $vetor2;
        echo $produto2->descricao . "<br>\n";

        Result:
        Chocolare Amargo
        Café
```

Conversão de tipos

Outra possibilidade que o PHP nos oferece é utilizar variáveis variantes para declarar as propriedades de um objeto. Neste exemplo criaremos um vetor simples com alguns índices definidos, como descricao, estoque e preco. Em seguida vamos declarar um objeto ($objeto) vazio (StdClass). Posteriormente percorreremos o vetor ($produto) acessando a chave e o valor de cada posição a cada iteração. Dentro do laço de repetição preencheremos os atributos do objeto com a sintaxe ($objeto->$chave). Neste caso a variável $chave será traduzida pelo seu conteúdo (descricao, estoque, preco) a cada passada do loop. Note que a única diferença oara a tribuição normal é a presença do cifrão ($) na frente da variável de propriedade.

Conversão de tipos

```
<?php
    $produto = array();
    $produto['descricao'] = 'Chocolate Amargo';
    $produto['estoque'] = 100;
    $produto['preco'] = 7;

    $objeto = new stdClass;

    foreach ($produto as $chave =>$valor) {
        $objeto->$chave = $valor;
    }

    print_r($objeto);

    Result:
    stdClass Object (
        [descricao] => Chocolate Amargo
        [estoque] => 100
        [preco] => 7
    )
```

Relacionamento entre Objetos

Veremos os principais tipos de relacionamento entre objetos: associação, composição e agregação. A herança será tratada separadamente por ser de maior complexidade.

Obs: a partir deste ponto vamos abandonar uma prática utilizada até então, que é a de mesclar a declaração da classe com a sua utilização no mesmo arquivo.

Como essa não é uma boa prática, a partir deste ponto as classes serão armazenadas isoladamente no subdiretório classes.

Relacionamento entre Objetos - Associação

Associação é a relação mais comum entre objetos. Na associação um objeto contém uma referência a outro objeto. Essa referência funciona como um apontamento em que um objeto terá um atributo que apontará para a posição da memória onde o outro objeto se encontra, podendo executar seus método. A forma mais comum de implementar uma associação é ter um objeto como atributo de outro.

Conceitualmente existe uma associação entre um produto e seu fabricante, em que um produto está relacionado a um fabricante e, por sua vez, um fabricante pode fabricar diferentes produtos.

Ex figura 2.5

Relacionamento entre Objetos - Associação

classes/Fabricante.php

```
<?php
class Fabricante {
    private $nome;
    private $endereco;
    private $documento;

    public function __construct($nome, $endereco, $documento) {
        $this->nome      = $nome;
        $this->endereco  = $endereco;
        $this->documento = $documento;
    }

    public function getNome() {
        return $this->nome;
    }
}
```

classes/Produto.php

```
<?php
class Produto {
    private $descricao;
    private $estoque;
    private $preco;
    private $fabricante;

    public function __construct($descricao, $estoque,
$preco) {
        $this->descricao = $descricao;
        $this->estoque   = $estoque;
        $this->preco     = $preco;
    }

    public function getDescricao() {
        return $this->descricao;
    }

    public function setFabricante(Fabricante $f) {
        $this->fabricante = $f;
    }

    public function getFabricante() {
        return $this->fabricante;
    }
}
```

Relacionamento entre Objetos - Associação

A partir dos objetos $p1 e $f1 criados, associação é estabelecida entre os dois objetos no momento da execução do método setFabricante(), que recebe a instância $f e a armazena no atributo $this->fabricante do objeto Produto. por fim serão impressos alguns atributos para demonstrar a relação.

associacao.php

```
<?php
require_once 'classes/Fabricante.php';
require_once 'classes/Produto.php';

//criação dos objetos
$p1 = new Produto('Chocolate', 10, 7);
$f1 = new Fabricante('Chocolate Factory', 'Willy Wonka Street', '15423586572');

//associação
$p1->setFabricante($f1);

echo 'A descrição é ' . $p1->getDescricao() . '<br>\n';
echo 'O fabricante é ' . $p1->getFabricante()->getNome() . '<br>\n';

Result: A descrição é Chocolate
        O fabricante é Chocolate Factory
```

Relacionamento entre Objetos - Composição

A composição é uma relação entre objetos de duas classes conhecida como relação todo/ parte. O relacionamento tem nome porque conceitualmente um objeto(todo) contém outros objetos (partes). A composição permite combinar diferentes tipos de objetos em um objeto mais complexo.

Colocar figura 2.6 do livro

Relacionamento entre Objetos - Composição

classes/Caracteristica.php

```
<?php
class Caracteristica {
    private $nome;
    private $valor;

    public function __construct($nome, $valor) {
        $this->nome  = $nome;
        $this->valor = $valor;
    }

    public function getNome() {
        return $this->nome;
    }

    public function getValor() {
        return $this->valor;
    }

}
```

classes/Produto.php

```
<?php
class Produto {
    //...
    private $caracteristicas;
    public function
addCaracteristica($nome, $valor) {
        $this->caracteristicas[] = new
Caracteristica($nome, $valor);
    }

    public function getCaracteristicas() {
        return $this->caracteriscas;
    }
}
```

Relacionamento entre Objetos - Composição

composicao.php

```
<?php
require_once 'classes/Produto.php';
require_once 'classes/Caracteristica.php';

//criação dos objetos
$p1 = new Produto('Chocolate', 10, 7);

//composição
$p1->addCaracteristica('Cor', 'Branco');
$p1->addCaracteristica('Peso', '2.6 kg');
$p1->addCaracteristica('Potência', '20 watts RMS');

echo 'Produto: ' . $p1->getDescricao() . "<br>\n";
foreach ($p1->getCaracteristicas() as $c) {
    echo ' Característica: ' . $c->getNome() . ' - ' . $c->getValor() . "<br>\n";
}

Result:
Produto: Chocolate
    Característica: Cor - Branco
    Característica: Peso - 2.6kg
    Característica: Potência - 20 watts RMS
```

Relacionamento entre Objetos - Agregação

Agregação também é um tipo de relação entre objetos todo/parte. Na agregação um objeto agrega outro objeto, ou seja, torna um objeto externo parte de si mesmo pela utilização de u dos seus métodos. Assim o objeto "todo" poderá utilizar funcionalidades do objeto agregado.

Colocar figura 2.7 do livro

Relacionamento entre Objetos - Agregação

classes/Cesta.php

```
<?php
class Cesta {
    private $time;
    private $itens;

    public function __construct() {
        $this->time  = date('Y-m-d H:i:s');
        $this->itens = array();
    }

    public function addItem(Produto $p) {
        $this->itens[] = $p;
    }

    public function getItens() {
        return $this->itens;
    }

    public function getTime() {
        return $this->time;
    }

}
```

Relacionamento entre Objetos - Agregação

agregacao.php

```
<?php
require_once 'classes/Cesta.php';
require_once 'classes/Produto.php';

//criação da cesta
$c1 = new Cesta;

//agregação dos produtos
$c1->addItem($p1 = new Produto('Chocolate', 10, 5));
$c1->addItem($p1 = new Produto('Café', 100, 7));
$c1->addItem($p1 = new Produto('Mostarda', 50, 3));

foreach($c1->getItens() as $item) {
    echo 'Item: ' . $item->getDescricao() . "<br\n>";
}
```

Result:
Item: Chocolate
Item: Café
Item: Mostarda

obs: Caso em algum momento o programador não respeite a regra de passar um objeto da classe Produto ao método addItem(), uma mensagem de erro será emitida:
Fatal error: Argument 1 passed to Cesta::addItem() must be an instance of Produto, instance of XYZ given...

Herança

Colocar figura 2.8 do livro

Herança

classes/Conta.php

```php
<?php
class Conta {
    protected $agencia;
    protected $conta;
    protected $saldo;

    public function __construct($agencia, $conta, $saldo) {
        $this->agencia = $agencia;
        $this->conta   = $conta;
        if ($saldo >= 0) {
            $this->saldo = $saldo;
        }
    }

    public function getInfo() {
        return "Agência: {$this->agencia}, Conta: {$this->conta}";
    }

    public function depositar($quantia) {
        if ($quantia > 0) {
            $this->saldo += $quantia;
        }
    }

    public function getSaldo() {
        return $this->saldo;
    }

}
```

Herança

classes/ContaPoupança.php

```
<?php
class ContaPoupança extends Conta {
    function retirar($quantia) {
        if ($this->saldo >= $quantia) {
            $this->saldo -= $quantia;
        }
        else {
            return false; //retirada não permitida
        }
        return true; //retirada permitida
    }
}
```

classes/ContaCorrente.php

```
<?php
class ContaCorrente extends Conta {
    protected $limite;

    public function __construct($agencia, $conta, $saldo,
$limite) {
        parent::__construct($agencia, $conta, $saldo);
        $this->limite = $limite;
    }

    public function retirar($quantia) {
        if (($this->saldo + $this->limite) >= $quantia) {
            $this->saldo -= $quantia; //retirada permitida
        } else {
            return false; // retirada não permmitida
        }
        return true;
    }
}
```

Polimorfismo

poli.php

```php
<?php
require_once 'classes/Conta.php';
require_once 'classes/ContaPoupanca.php';
require_once 'classes/ContaConrrente.php';

//criação dos objetos
$contas = array();
$contas[] = new ContaCorrente(6677, "CC.1234.56", 100, 500);
$contas[] = new ContaPoupanca(6678, "PP.1234.57", 100);

//percorre as contas
foreach ($contas as $key => $conta) {
    echo "Conta: " . $conta->getInfo() . "<br>\n";
    echo "Saldo Atual: " . $conta->getSaldo() . "<br>\n";
    $conta->depositar(200);
    echo "Depósito de: 200 <br>\n";
    echo "Saldo atual: " . $conta->getSaldo() . "<br>\n";

    if ($conta->retirar(700)) {
        echo "Retirada de: 700 <br>\n";
    } else {
        echo "Retirada de: 700 (não permitida) <br>\n";
    }

    echo "Saldo atual: " . $conta->getSaldo() . "<br>\n";
}
```

Abstração - Classes abstratas

classes/conta.php

```
<?php
abstract class Conta {
    //...
}
```

classe_abstrata.php

```
<?php
require_once 'classes/Conta.php';
$conta = new Conta;
```

Result:

Fatal error: Cannot instantiate abstract class Conta...

Abstração - Classes finais

classes/ContaPoupanca.php

```
<?php
final class ContaPoupanca extends Conta {
    // ...
}
```

```
classe_final.php
<?php
require_once 'classes/Conta.php';
require_once 'classes/ContaPoupanca.php';

class ContaPoupancaUniversitaria extends ContaPoupanca {
    // ...
}
Result:
Fatal error: Class ContaPoupancaUniversitaria may not inherit from final class (ContaPoupanca)...
```

Abstração - Métodos abstratos

classes/Conta.php

```
<?php
abstract class Conta {
    //...
    abstract function retirar($quantia);
}
```

metodo_abstract.php

```
<?php
require_once 'classes/Conta.php';
class ContaSalario extends Conta {
    //...
}
```

Result:

Fatal error: Class ContaSalario contains 1 abstract method and must therefore be declared abstract or implement the remaining methos (Conta::retirar)...

Abstração - Métodos finais

classes/ContaCorrente.php

```
<?php
class ContaCorrente extends Conta {
    //...
    public final function retirar($quantia) {
        //...
    }
}
```

metodo_final.php

```
<?php
require_once 'classes/Conta.php';
require_once 'Classes/ContaCorrente.php';

class ContaCorrenteEspecial extends ContaCorrente {
    public function retirar($quantia) {
        $this->saldo -= $quantia;
    }
}
```

Result:

Fatal error: Cannot override final method ContaCorrente::retirar() in...

Encapsulamento - Public

public.php

```php
<?php
class Pessoa {
    public $nome;
    public $endereço;
    public $nascimento;
}

$p1 = new Pessoa;
$p1->nome = 'Maria da Silva';
$p1->endereço = 'Rua Bento Gonçalves';
$p1->nascimento = '01 de Maio de 2015';
$p1->telefone = '(51) 1234-5678'; //definida em tempo de execução, por padrão é public;

print_r($p1);
```

Encapsulamento - Private

private.php

```
<?php
class Pessoa {
    private $nome;
    private $endereco;
    private $nascimento;
}

$p1 = new Pessoa;
$p1->nome = 'Maria da Silva';
$p1->endereco = 'Rua Bento Gonçalves';
$p1->nascimento = '01 de Maio de 2015';

Result:
Fatal error: Cannot access private property Pessoa::$nome...
```

Encapsulamento - Private

private2.php

```
<?php
class pessoa {
    private $nome;
    private $endereco;
    private $nascimento;

    public function __construct($nome, $endereco) {
        $this->nome = $nome;
        $this->endereco = endereco;
    }

    public function setNascimento($nascimento) {
        $partes = explode('-', $nascimento);
        if (count($partes)==3) {
            if (checkdate($partes[1], $partes[2], $partes[0])) {
                $this->nascimento = $nascimento;
                return true;
            }
            return false;
        }
        return false;
    }
}
```

```
$p1 = new Pessoa('Maria da Silva', 'Rua Bento Gonçalves');

    if ($p1->setNascimento('01 de Maio de 2015')) {
        echo "Atribuiu 01 de Maio de 2015<br>\n";
    } else {
        echo "Não abribuiu 01 de Maio de 2015<br>\n";
    }

    if ($p1->setNascimento('2015-12-30')) {
        echo "Atribuiu 2015-12-30<br>\n";
    } else {
        echo "Não atribuiu 2015-12-30";
    }

    Result:
    Não atribuiu 01 de Maio de 2015
    Abribuiu 2015-12-19
```

Encapsulamento - Protected

protected.php

```
<?php
class Pessoa {
    private $nome;

    public function __construct($nome) {
        $this->$nome;
    }
}

class Funcionario extends Pessoa {
    private $cargo, $salario;

    public function contrata($c, $s) {
        if (is_numeric($s) AND $s > 0) {
            $this->cargo = $c;
            $this->salario = $s;
        }
    }

    public function getInfo() {
        return "Nome: {$this->nome}, Salário: {$this-
>salario}";
    }
}
```

```
$p1 = new Funcionario('Maria da Silva', 'Rua Bento
Gonçalves');
    $p1->contrata('Auxiliar administrativo', 1600);

    echo $p1->getInfo();

    Result:
    Notice: Undefined property: Funcionario::$nome
in...
    Nome: , Salário: 1600
```

Encapsulamento - Protected

Para permitir que o atributo $nome devemos declarar como protected:

protected.php

```
class Pessoa {
    protected $nome;
    //...
}
```

Membros de Classe

Como vimos anteriormente, a classe é uma estrutura-padrão para criação dos objetos. Ela permite que armazenemos valores nela de duas formas: constantes de classe e propriedades estáticas. Esses atributos são comuns a todos os objetos da mesma classe.

Membros de Classe - Constantes

O PHP permite definir constantes globais define(). Agora veremos que ele também permite definir constantes de classe. Uma constante de classe é contida por sua classe, ou seja, podemos ter diferentes constantes de mesmo nome, porém pertencendo a classes diferentes. Neste exemplo veremos como se dá a declaração de uma constante dentro de uma classe pelo operador const.

Constantes de classe podem ser utilizadas para declarar valores imutáveis que somente fazem sentido dentro de uma classe. No exemplo a seguir definiremos um vetor de gêneros contendo as descrições para os gêneros masculino e feminino. O método construtor receberá as informações de nome e gênero. Enquanto o método getNome() retornará o nome, o método getNomeGenero() retornará o nome do gênero correspondente, que será obtido a partir do array contido na constante GENEROS. A constante poderá ainda ser acessada de forma externa ao contexto da classe pela sintaxe Class::CONSTANTE, e dentro da classe pela sintaxe self::CONSTANTE.

O operador self representará a própria classe.

Membros de Classe - Constantes

constante_classe.php

```php
<?php
class Pessoa {
    private $nome;
    private $genero;
    const GENEROS = array('M' => 'Masculino', 'F' => 'Feminino');

    public function __construct($nome, $genero) {
        $this->nome = $nome;
        $this->genero = $genero;
    }

    public function getNome() {
        return $this->nome;
    }

    public function getNomeGenero() {
        return self::GENEROS[$this->genero];
    }
}

$p1 = new Pessoa('Maria da Silva', 'F');
$p2 = new Pessoa('Carlos Pereira', 'M');

echo 'Nome:   ' . $p1->getNome() . "<br>\n";
echo 'Genero: ' . $p1->getNomeGenero() . "<br\n>";
echo 'Nome:   ' . $p2->getNome() . "<br\n>";
echo 'Genero: ' . $p2->getNomeGenero() . "<br>\n";

print_r(Pessoa::GENEROS);
```

Membros de Classe - Atributos estáticos

Atributos estáticos são atributos que pertencem a uma classe, não a um objeto específico; são dinâmicos como os atributos de um objeto, mas estão relacionados à classe. Como a classe é a estrutura comum a todos os objetos dela derivados, atributo estáticos são compartilhados entre todos os objetos de uma mesma classe.

Para demonstrar a utilidade de um atributo estático, vamos criar uma classe chamada Software. Cada objeto dessa classe terá como atributos id e nome. Além disso haverá um atributo estático chamado contador. A cada instância criada de Software, incrementaremos o atributo estático contador e verificaremos ao final da execução se ele conservou seu valor. Um atributo estático conserva seu valor em nível de classe, ou seja, seu valor não está vinculado a um objeto específico. Enquanto para acessar atributos de objetos utilizamos a forma ($this->atributo), para acessar atributos estáticos utilizamos a forma (self::$atributo), quando acessado de dentro da classe, e a forma (Classe::$atributo), quando acessado de fora da classe. Além disso pe importante declarar o modificador static na frente de seu nome.

Membros de Classe - Atributos estáticos

propriedade_estatica.php

```
<?php
class Software {
    private $id;
    private $nome;
    public static $contador;

    function __construct($nome) {
        self::$contador ++;

        $this->id = self::$contador;
        $this->nome = $nome;
        echo "Software" . $this->id . " - " . $this->nome "<br>\n";
    }
}

//cria novos objetos
new Software('Dia');
new Software('Gimp');
new Software('Gnumeric');

echo 'Quantidade criada = ' . Software::$contador;

result:
Software 1 - Dia
Software 2 - Gimp
Software 3 - Gnumeric
Quantidade criada = 3
```

Membros de Classe - Métodos estáticos

Vimos como manipular um atributo estático, que é um atributo armazenado em nível de classe. Entretanto, para demonstrar o seu funcionamento, devemos declará-lo como public, o que nem sempre é uma boa opção, visto que ele pode ser acidentalmente modificado por fora da classe.

Para manipular atributos estáticos, podemos utilizar métodos estáticos. Métodos estáticos podem inclusive ser executados diretamente a partir da classe sem a necessidade de criar um objeto para isso. Eles não devem referenciar propriedades internas pelo operador $this, pois esse operador é utilizado para referenciar instâncias da classe (objetos), mas não a própria classe; são limitados a chamar outros métodos estático da classe ou utilizar apenas tributos estáticos. Para executar um método estático, basta utilizar a sintaxe self::metodo() de dentro da classe ou Classe::metodo() de fora dela.

Membros de Classe - Métodos estáticos

No exemplo a seguir reescreveremos o exemplo anterior, declarando o atributo estático contador como private. COmo o atributo foi remarcado como private, não será mais possível acessá-lo do contexto externo à classe pela sintaxe Software::$contador, o que ocasionaria um erro do seguinte tipo:

Fatal error: Cannot access private property Software::$contador

Para manipularmos um atributo estático, será preciso criar um método estático. Para tal, criaremos o método getContador(). Um método estático será definido pelo modificador static na frete de seu nome.

Membros de Classe - Métodos estáticos

metodo_estatico.php

```php
<?php
class Software {
    private $id;
    private $nome;
    private static $contador;

    function __construct($nome) {
        self::$contador ++;
        $this->id = self::$contador;
        $this->nome = nome;
        echo "Software " . $this->id . " - " . $this->nome . "<br>\n";
    }

    public static function getContador() {
        return self::$contador;
    }

    //cria novos objetos
    new Software('Dia');
    new Software('Gimp');
    new Software('Gnumeric');

    echo 'Quantidade criada = ' . Software::getContador();

    result:
    Software 1 - Dia
    Software 2 - Gimp
    Software 3 - Gnumeric
    Quantidade criada = 3
}
```