SENAI CIMATEC SISTEMA FIEB
Federação das Indústrias do Estado da Bahia

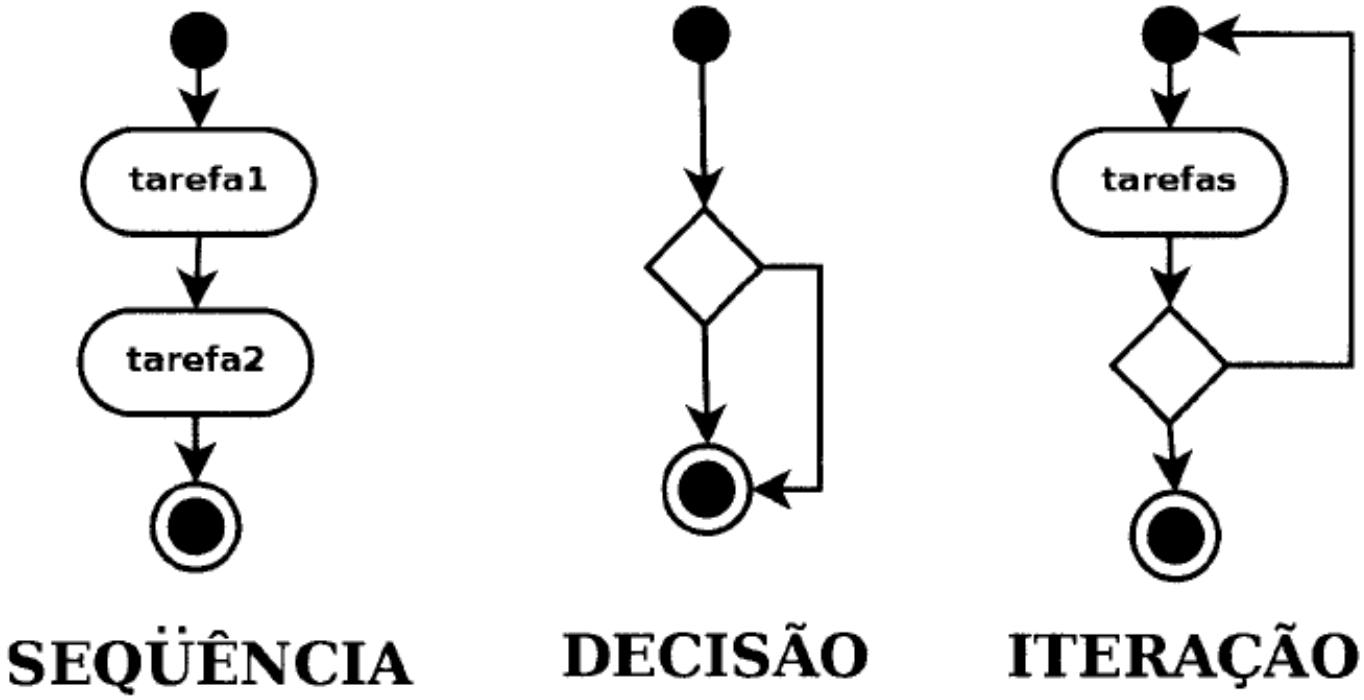
2016.2

Introdução a Programação Orientada a Objetos Utilizando PHP

Docente: Gil Jader – gil.jader@gmail.com

1. Paradigma Estruturado

Na programação estruturada, as unidades do código (funções) se interligam por três mecanismos básicos:



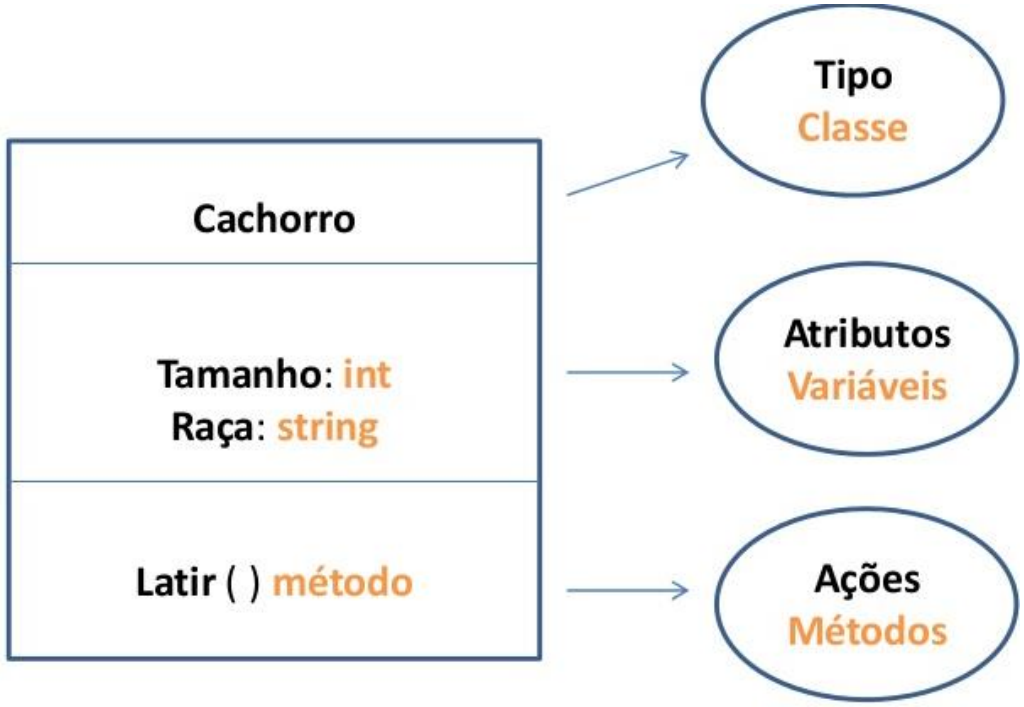
2. Paradigma Orientado a Objetos

A orientação a objetos é um paradigma que representa toda uma filosofia para a construção de sistemas. Em vez de construir um sistema formado por um conjunto de procedimentos e variáveis nem sempre agrupadas com o contexto, lidamos com objetos (uma ótica mais próxima do mundo real).

2. Paradigma Orientado a Objetos

3. Classes

Ao trabalharmos com orientação a objetos é fundamental entender o conceito de classes e objetos.



3. Classes

- A definição de uma classe em PHP começa com a palavra chave ***class***, seguido do **nome da classe**, que pode ser **qualquer nome que não seja uma palavra reservada no PHP**.

- Seguido por um par de chaves **{ }** , que contém a definição dos **atributos** (ou **propriedades**) e **métodos** (ou **comportamentos**) da classe.

Observação: recomenda-se iniciar nomes de classes com a letra maiúscula e, de preferência, evitar a utilização do caractere "_". Por exemplo, utilize `class CestaDeCompras` e não `class cesta_de_compras`.

---

3. Classes

Atributos de uma Classe

- Os atributos são elementos das classes semelhantes as variáveis da lógica de programação estruturada.
- Cada atributo representa uma propriedade do item que está sendo modelado.
- Para cada atributo é definido um intervalo de valores possíveis. Por exemplo, a classe funcionário pode ter os atributos nome, endereço, telefone, número de identidade, salário, cargo.

3. Classes

Métodos de uma Classe

- As classes costumam ter métodos, também conhecidos como operações ou comportamentos.
- Um método é uma operação que pode ser executada sobre os objetos de uma classe. São os métodos que executam todas as funcionalidades do programa, como fazer cálculos, exibir mensagens, gerar relatórios, etc.

3. Classes

Exemplo de classe em PHP:

Produto.class.php

```
<?php
class Produto
{
    var $Codigo;
    var $Descricao;
    var $Preco;
    var $Quantidade;
}
?>
```

4. Objetos

- Um objeto contém exatamente a mesma estrutura e as propriedades de uma classe, no entanto sua estrutura é dinâmica, seus atributos podem mudar de valor durante a execução do programa.

- Para instanciar um objeto em PHP fazemos uso da palavra chave **new**, seguido do nome da classe que desejamos instanciar.

4. Objetos

Exemplo:

objeto.php

```
<?php
// insere a classe
include_once 'classes/Produto.class.php';

// cria um objeto
$produto = new Produto;

// atribuir valores
$produto->Codigo = 4001;
$produto->Descricao = 'CD - The Best of Eric Clapton';

echo $produto;
?>
```

O que acontece aqui?

4. Objetos

Um objeto não foi feito para ser impresso diretamente, mas suas propriedades sim. Entretanto, se tentarmos imprimir um objeto diretamente, o PHP retornará o identificador interno desse objeto na memória.

objeto.php

```
<?php
// insere a classe
include_once 'classes/Produto.class.php';

// cria um objeto
$produto = new Produto;

// atribuir valores
$produto->Codigo = 4001;
$produto->Descricao = 'CD - The Best of Eric Clapton';

echo $produto;
?>
```

Catchable fatal error: Object of class Produto could not be converted to string

Resultado:

```
Object id #1
```

5. A variável $this

- A fim de diferenciar as propriedades de um objeto de variáveis locais, utiliza-se a pseudo variável **$this** para representar o objeto atual e acessar suas propriedades.

$this é uma **referência** para o **objeto** ao chamar um método ou um atributo.

5. A variável $this

- Podemos reescrever a classe Produto do seguinte modo:

Produto.class.php

```
<?php
class Produto
{
    var $Codigo;
    var $Descricao;
    var $Preco;
    var $Quantidade;

    function ImprimeEtiqueta()
    {
        print 'Código:    ' . $this->Codigo . "\n";
        print 'Descrição: ' . $this->Descricao . "\n";
    }
}
?>
```

5. A variável $this

Atividade:

Criem dois objetos para demonstrar a utilização da Classe Produto e o método **ImprimeEtiqueta()**.

5. A variável $this

Criação de dois objetos para demonstrar a utilização da classe produto:

objeto.php

```
<?php
// insere a classe
include_once 'classes/Produto.class.php';

// cria dois objetos
$produto1 = new Produto;
$produto2 = new Produto;

// atribuir valores
$produto1->Codigo = 4001;
$produto1->Descricao = 'CD - The Best of Eric Clapton';

$produto2->Codigo = 4002;
$produto2->Descricao = 'CD - The Eagles Hotel California';

// imprime informações de etiqueta
$produto1->ImprimeEtiqueta();
$produto2->ImprimeEtiqueta();
?>
```

7. Construtores

- Em PHP 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**. Assim, não devemos retornar nenhum valor por meio do método construtor porque o mesmo retorna por definição o próprio objeto que está sendo instanciado.

- Caso não seja definido um método construtor, automaticamente todas as propriedades do objeto serão inicializadas com o valor NULL.

Observação: para definir um método construtor em uma determinada classe basta declarar o método `__construct()`.

Pessoa.class.php

```php
<?php
class Pessoa
{
    # ...
    # conteúdo
    # ...

    /* método construtor
     * inicializa propriedades
     */
    function __construct($Codigo, $Nome, $Altura, $Idade, $Nascimento, $Escolaridade, $Salario)
    {
        $this->Codigo = $Codigo;
        $this->Nome = $Nome;
        $this->Altura = $Altura;
        $this->Idade = $Idade;
        $this->Nascimento = $Nascimento;
        $this->Escolaridade = $Escolaridade;
        $this->Salario = $Salario;
    }
```

8. Destrutores

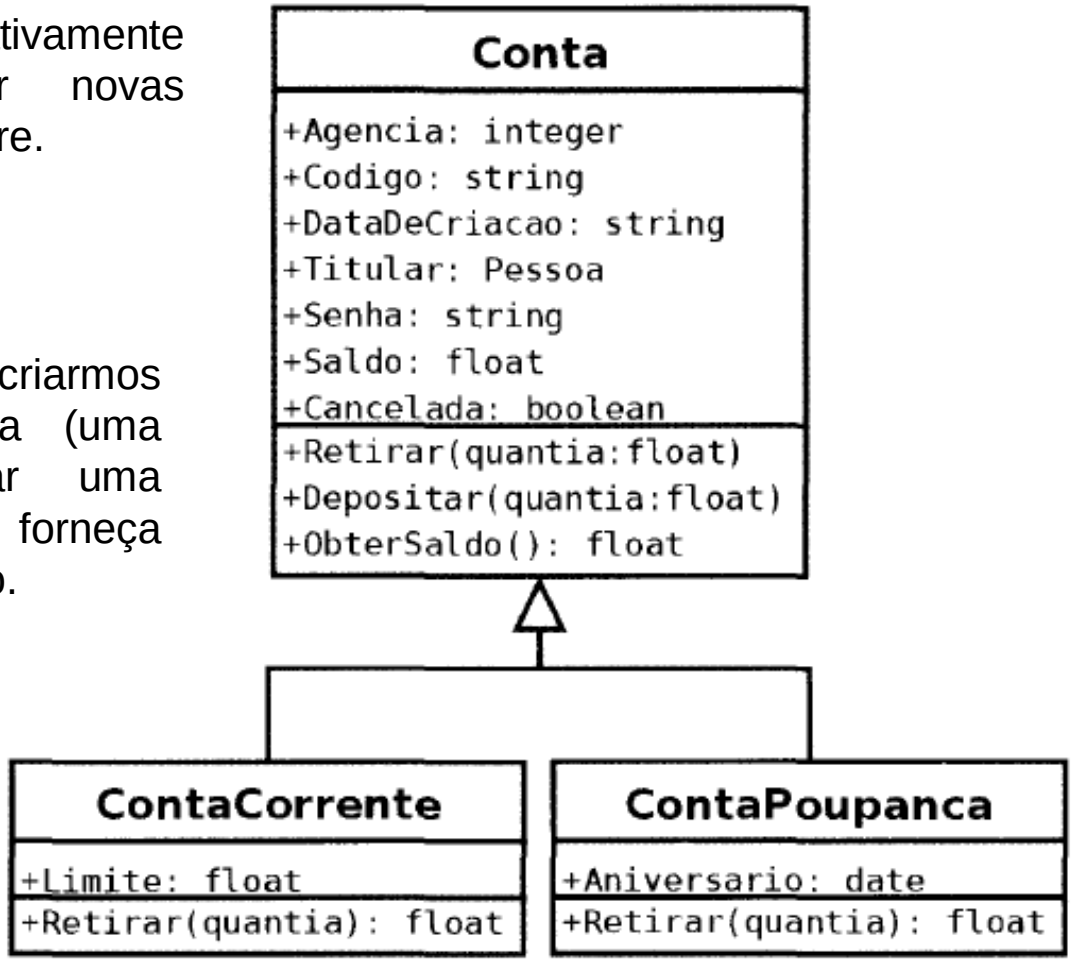
Um destrutor ou finalizador é um método especial executado automaticamente quando o objeto é deslocado da memória, quando atribuímos o valor **NULL** ao objeto, quando utilizamos a função **unset()** sobre o mesmo ou, em última instância, quando o programa é **finalizado**.

Observação: para definir um método destrutor em uma determinada classe basta declarar o método `__destruct()`.

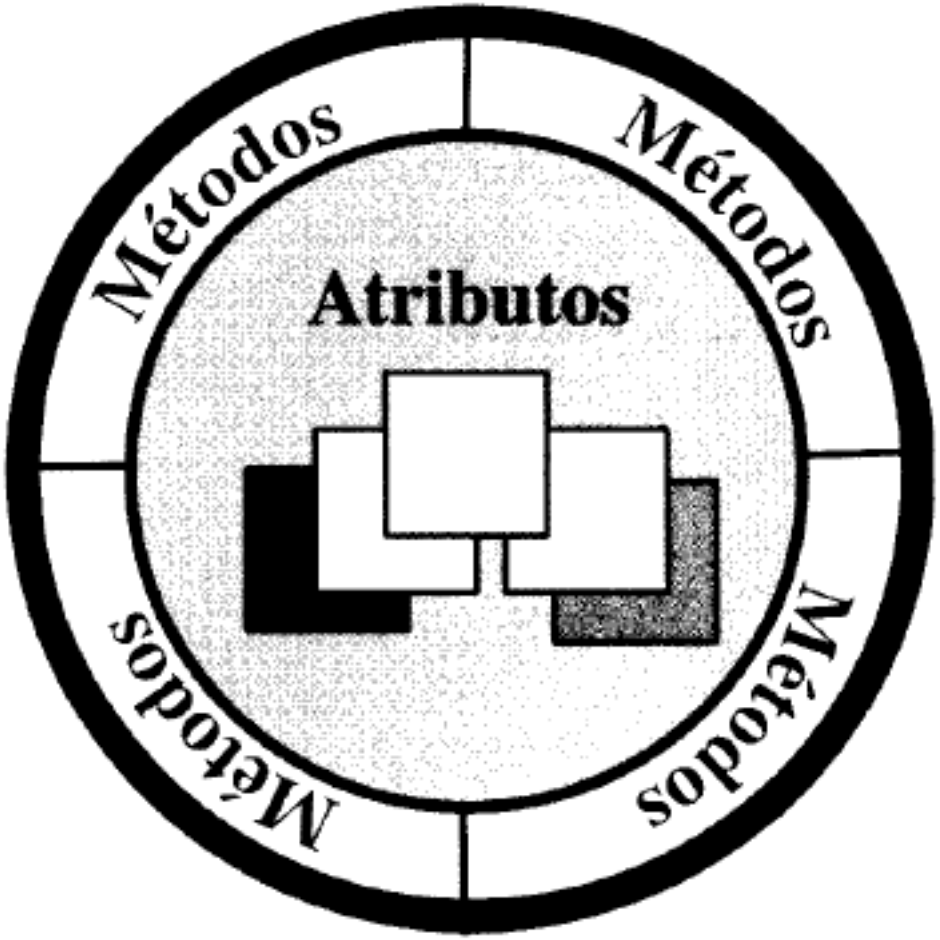
9. Herança

- Este recurso tem aplicabilidade muito grande, visto que é relativamente comum termos de criar novas funcionalidades em um software.

- Utilizando a herança, em vez de criarmos uma estrutura totalmente nova (uma classe), podemos reaproveitar uma estrutura já existente que nos forneça uma base para o desenvolvimento.



10. Encapsulamento



10. Encapsulamento

Visibilidade de propriedades e métodos de uma Classe

A visibilidade é utilizada para indicar o nível de acessibilidade de um determinado atributo ou método, sendo representada à esquerda destes.

Se outros programadores usam nossa classe, nós queremos garantir que erros pelo mau uso não ocorram. Encapsulamento serve para controlar o acesso aos atributos e métodos de uma classe.

Basicamente, usamos quatro tipos de encapsulamento que são divididos em dois níveis:

- Nível de classe ou topo: Quando determinamos o acesso de uma classe inteira que pode ser **public** ou **package-private** (padrão);
- Nível de membro: Quando determinamos o acesso de atributos ou métodos de uma classe que podem ser **public**, **private**, **protected** ou **package-private** (padrão).

10. Encapsulamento

Visibilidade de propriedades e métodos de uma Classe

Exemplo em Java:

```
1. public class MinhaClasse {     //classe public
2.     private int inteiro;       //atributo inteiro private
3.     protected float decimal;   //atributo float protected
4.     boolean ativado;           //atributo booleano package-private
5. }
```

Visibilidade:

Notação visual	Modificador de acesso	A parte é visível...
+	public	dentro da própria classe e para qualquer outra classe
–	private	somente dentro da própria classe
#	protected	somente dentro do próprio pacote e das subclasses em outros pacotes
~	package	somente dentro da própria classe e das classes dentro do mesmo pacote

10. Encapsulamento

Visibilidade de propriedades e métodos de uma Classe

Exemplo de uma classe em UML:

ClienteBanco
+ nome: String ~ CPF: String # idade: int - saldoConta: double
~ ClienteBanco (nome: String, CPF: String) + getSaldoConta () :double + setSaldoConta(saldoConta: double) :void # setIdade (idade: int) :void

10. Encapsulamento

Visibilidade de propriedades e métodos de uma Classe

Exemplo de uma classe em UML:

ClienteBanco
+ nome: String ~ CPF: String # idade: int - saldoConta: double
~ ClienteBanco (nome: String, CPF: String) + getSaldoConta () :double + setSaldoConta(saldoConta: double) :void # setIdade (idade: int) :void

10. Encapsulamento

Visibilidade de propriedades e métodos de uma Classe

Resumo:

Modificador	Classe	Pacote	Subclasse	Globalmente
Public	*Sim*	*Sim*	*Sim*	*Sim*
Protected	*Sim*	*Sim*	*Sim*	*Não*
Sem Modificador (Padrão)	*Sim*	*Sim*	*Não*	*Não*
Private	*Sim*	*Não*	*Não*	*Não*

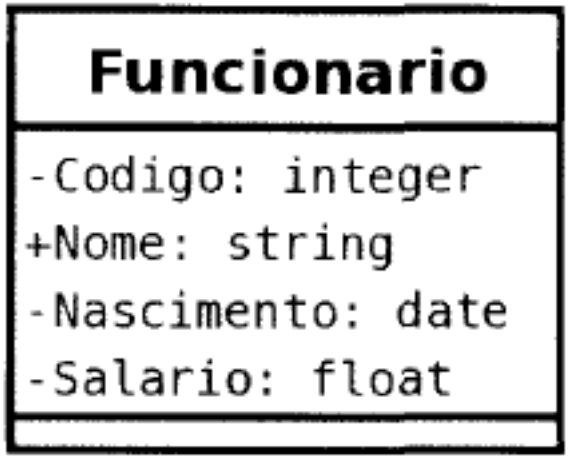
11. Encapsulamento em PHP

Existem três formas de Acesso em **PHP**:

Visibilidade	Descrição
`private`	Membros declarados como `private` somente podem ser acessados dentro da própria classe em que foram declarados. Não poderão ser acessados a partir de classes descendentes nem a partir do programa que faz uso dessa classe (manipulando o objeto em si). Na UML, simbolizamos com um (-) em frente à propriedade.
`protected`	Membros declarados como `protected` somente podem ser acessados dentro da própria classe em que foram declarados e a partir de classes descendentes, mas não poderão ser acessados a partir do programa que faz uso dessa classe (manipulando o objeto em si). Na UML, simbolizamos com um (#) em frente à propriedade.
`public`	Membros declarados como `public` poderão ser acessados livremente a partir da própria classe em que foram declarados, a partir de classes descendentes e a partir do programa que faz uso dessa classe (manipulando o objeto em si). Na UML, simbolizamos com um (+) em frente à propriedade.

11. Encapsulamento em PHP

Demonstração da Visibilidade **Private**:



11. Encapsulamento em PHP

Funcionario.class.php

```php
<?php
class Funcionario
{
    private $Codigo;
    public  $Nome;
    private $Nascimento;
    private $Salario;
}
?>
```

O que acontece aqui?

private.php

```php
<?php
# carrega a classe
include_once 'classes/Funcionario.class.php';

$pedro = new Funcionario;
$pedro->Salario = 'Oitocentos e setenta e seis';
?>
```

Resultado:

```
Fatal error: Cannot access private property Pessoa::$Salario in private.php on line 6
```

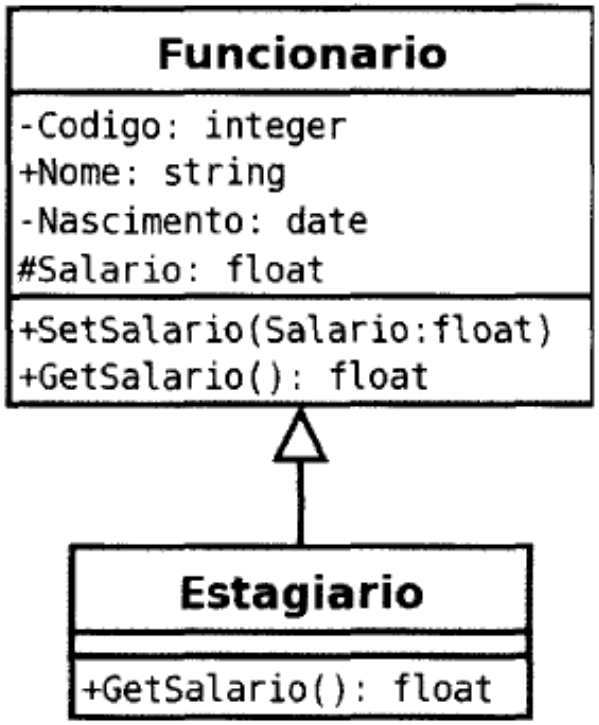
Funcionario.class.php (complemento)

```php
<?php
class Funcionario
{
    private $Codigo;
    public  $Nome;
    private $Nascimento;
    private $Salario;

    /* método SetSalario
     * atribui o parâmetro $Salario à propriedade $Salario
     */
    function SetSalario($Salario)
    {
        // verifica se é numérico e positivo
        if (is_numeric($Salario) and ($Salario > 0))
        {
            $this->Salario = $Salario;
        }
    }

    /* método GetSalario
     * retorna o valor da propriedade $Salario
     */
    function GetSalario()
    {
        return $this->Salario;
    }
}
?>
```

11. Encapsulamento em PHP

Demonstração da Visibilidade **Protected**:

Para demonstrar a visibilidade ***protected***, vamos especializar a classe **Funcionario**, criando a classe **Estagiario**. A única característica exclusiva de um estagiário é que o seu salário é acrescido de 12% de bônus.



11. Encapsulamento em PHP

Demonstração da Visibilidade **Protected**:

Estagiario.class.php

```
<?php
class Estagiario extends Funcionario

{
    /* método GetSalario sobreescrito
     * retorna o $Salário com 12% de bônus.
     */
    function GetSalario()
    {
        return $this->Salario * 1.12;
    }
}
?>
```

11. Encapsulamento em PHP

Demonstração da
Visibilidade **Protected**:

protected.php

```
<?php
# carrega as classes
include 'classes/Funcionario.class.php';
include 'classes/Estagiario.class.php';

$pedrinho = new Estagiario;
$pedrinho->SetSalario(248);
echo 'O Salário do Pedrinho é R$: ' . $pedrinho->GetSalario() . "\n";
?>
```

O que acontece?

Resultado:

```
O Salário do Pedrinho é R$: 0
```

11. Encapsulamento em PHP

Funcionario.class.php (complemento)

```php
<?php
class Funcionario
{
    private   $Codigo;
    public    $Nome;
    private   $Nascimento;
    protected $Salario;

    # ...
    # conteúdo já escrito no exemplo anterior
    # ...
}
?>
```

11. Encapsulamento em PHP

Demonstração da Visibilidade **Public**:

public.php

```
<?php
# carrega as classes
include 'classes/Funcionario.class.php';
include 'classes/Estagiario.class.php';

// cria objeto Funcionario
$pedrinho = new Funcionario;
$pedrinho->nome = 'Pedrinho';

// cria objeto Estagiario
$mariana = new Estagiario;
$mariana->nome = 'Mariana';

// imprime propriedade nome
echo $pedrinho->nome;
echo $mariana->nome;
?>
```

12. Resumo do Conceito de Classe:

- A classe é uma estrutura estática utilizada para descrever objetos mediante atributos (propriedades) e métodos (funcionalidades). A classe é um modelo para criação dos objetos.

- Podem ser classes: entidades do negócio da aplicação **(pessoa, conta, cliente, fornecedor)**, entidades de interface **(janela, botão, painel, frame)**, dentre outras **(conexão com banco de dados, uma conexão SSH, uma conexão FTP, etc)**.

REFERÊNCIAS

- Dall'Oglio, Pablo. Php 5 - PHP: Programando com Orientação a Objetos. 3ª Edição, São Paulo, Novatec, 2015.
- WALLACE, Soares. Php 5 - Conceitos, Programação e Integração com Banco de Dados. 7ª Edição, São Paulo, Erica, 2013.
- http://www.php.net/manual/pt_BR/index.php