

PHP ORIENTADO A OBJETOS



2º Encontro PHP MG
16 e 17 de Outubro / 2009

Charles Schaefer

Charles Schaefer



- Profissional PHP há 4 anos
- Sócio-Diretor da UaiTI
- Instrutor dos cursos de PHP da PHPPrime / Belo Horizonte
- Certificado ZCE PHP 5

Agenda



- **Elementos Importantes na POO**
- **Sintaxe OO no PHP 5**
- **Recursos POO do PHP 5**
- **Arquitetura MVC – Prática POO no PHP**
- **Adicionais**

PHP 5

Suporte melhorado a OO



- **Visibilidade de membros**
- **Interfaces e Classes e Métodos Abstratos**
- **Métodos e Propriedades estáticos**
- **Método Destrutor (e mudanças no Construtor)**
- **Métodos e Classes “Finais”**
- ***Exceções (try / catch)***
- ***PHP 5.3:***
 - ***Namespaces***

Elementos Importantes em POO



- **Classes**
 - Define características abstratas de uma “coisa” (ou objeto), incluindo seus atributos e comportamentos/ações;
- **Objetos / Instâncias**
 - Um exemplar de uma classe (Beethoven → Cão)
- **Herança**
 - Processo pelo qual uma classe “herda” as características de outra e adiciona as suas próprias

Elementos Importantes em POO



- **Encapsulamento**

- Permite à classe esconder detalhes internos de seu funcionamento do mundo exterior!

- **Polimorfismo**

- “Permite que um objeto de uma classe seja tratado como um objeto de outra classe”!

Se você já trabalhou com OO, provavelmente já se deparou com esses conceitos sem saber seus nomes

POO no PHP

Sintaxe



- **Classes**

```
// classe Autenticacao
class Autenticacao {
    // declaração de atributos
    protected $usuario;
    protected $senha;

    // declaração de método
    public function login($usuario, $senha) {
        // ....
    }
}
```

POO no PHP

Sintaxe



- A variável **\$this**

```
// classe Autenticacao
class Autenticacao {
    // declaração de atributos
    protected $usuario;
    protected $senha;

    // declaração de método
    public function login($usuario, $senha) {
        // atribuindo valor a atributos da classe
        $this->usuario = $usuario;
        $this->senha = $senha;
        // chamando um método da classe
        $this->verificar($usuario, $senha);
    }

    // ....
}
```

POO no PHP

Sintaxe



- Instanciando Objetos

```
require_once 'includes/autenticacao.php';  
require_once 'includes/database.php';  
require_once 'includes/cliente.php';
```

```
// instanciando um objeto Autenticacao  
$auth = new Autenticacao();  
// instanciando um objeto Database  
$db = new Database();  
// instanciando um objeto Cliente  
$cliente = new Cliente();
```

POO no PHP

Sintaxe *



- Trabalhando com Objetos

```
require_once 'includes/autenticacao.php';
```

```
// instanciando um objeto Autenticacao
```

```
$auth = new Autenticacao();
```

```
// Chama o método login()
```

```
$auth -> login($_POST['usuario'], $_POST['senha']);
```

```
// "Lê" um atributo do objeto
```

```
$id = $auth->id;
```

```
// "Lê" um atributo do objeto utilizando um método get*()
```

```
$ultimo_acesso = $auth->getLastAccess();
```

```
// atribui valor a um atributo utilizando um método set*()
```

```
$auth->setIdade((int)$_POST['idade']);
```

Recursos POO no PHP *



- **Herança:** “Processo pelo qual uma classe “herda” as características de outra e adiciona as suas próprias”
- As classes filhas passam a ter os mesmos métodos e atributos das classes pais ...
- ... e ainda os novos métodos e atributos definidos dentro delas!
- Exemplo na pasta *herança*

Recursos POO no PHP



- **Construtores:** permitem que rotinas sejam executadas logo após a instanciação de um objeto da classe.
- **Destrutores:** permitem que rotinas sejam executadas imediatamente antes de um objeto da classe ser “destruído” - removido da memória.

Recursos POO no PHP

Construtores e Destrutores*



```
class DataBase {
    public function __construct($host, $user, $pass) {
        $this->connect($host, $user, $pass);
    }
    public function __destruct() {
        $this->disconnect();
    }
    public function connect($host, $user, $pass) {
        // ...
    }
    public function disconnect() {
        // ...
    }
}

// instancia a classe passando argumentos para o construtor
$db = new DataBase('localhost', 'root', 'p4pm6');
// "destrói" o objeto, fazendo com que o PHP chame o método destrutor
unset($db);
```

Recursos POO no PHP *



- **Classe abstrata**

- Define métodos abstratos que devem ser implementados em classes herdeiras
- Não podem ser instanciadas
- Qualquer classe que tenha um método abstrato deve ser declarada como abstrata

- **Método abstrato**

- Define a assinatura de um método a ser implementado em classes herdeiras

Recursos POO no PHP *



- **Interfaces**

- Permite a criação de modelos de métodos que devem ser implementados nas classes
- Só podem ter assinaturas dos métodos (sem atributos)
- Uma classe que implementa uma ou mais interfaces deve implementar **TODOS** os métodos definidos por ela(s)

Recursos POO no PHP *



- **Autoload**

- Função especial do PHP que permite uma última chance de encontrar uma classe

```
function __autoload($class) {  
    if (file_exists('includes/' . $class . '.php')) {  
        require_once 'includes/' . $class . '.php';  
    } elseif (file_exists('classes/' . $class . '.php')) {  
        require_once 'classes/' . $class . '.php';  
    } elseif (file_exists('libs/' . $class . '.php')) {  
        require_once 'libs/' . $class . '.php';  
    }  
}
```

Entendendo a Teoria

Encapsulamento *



“Permite à classe esconder detalhes internos de seu funcionamento do mundo exterior!”

- O mais comum:
 - Métodos getters e setters
- Exemplos na pasta *encapsulamento*

Cliente
#cpf #email #nome
+setCPF(cpf) +setEmail(email) +setNome(nome) +getCPF() +getEmail() +getNome() #validateCPF(cpf) #formataCPF(cpf)

Entendendo a Teoria

Polimorfismo *

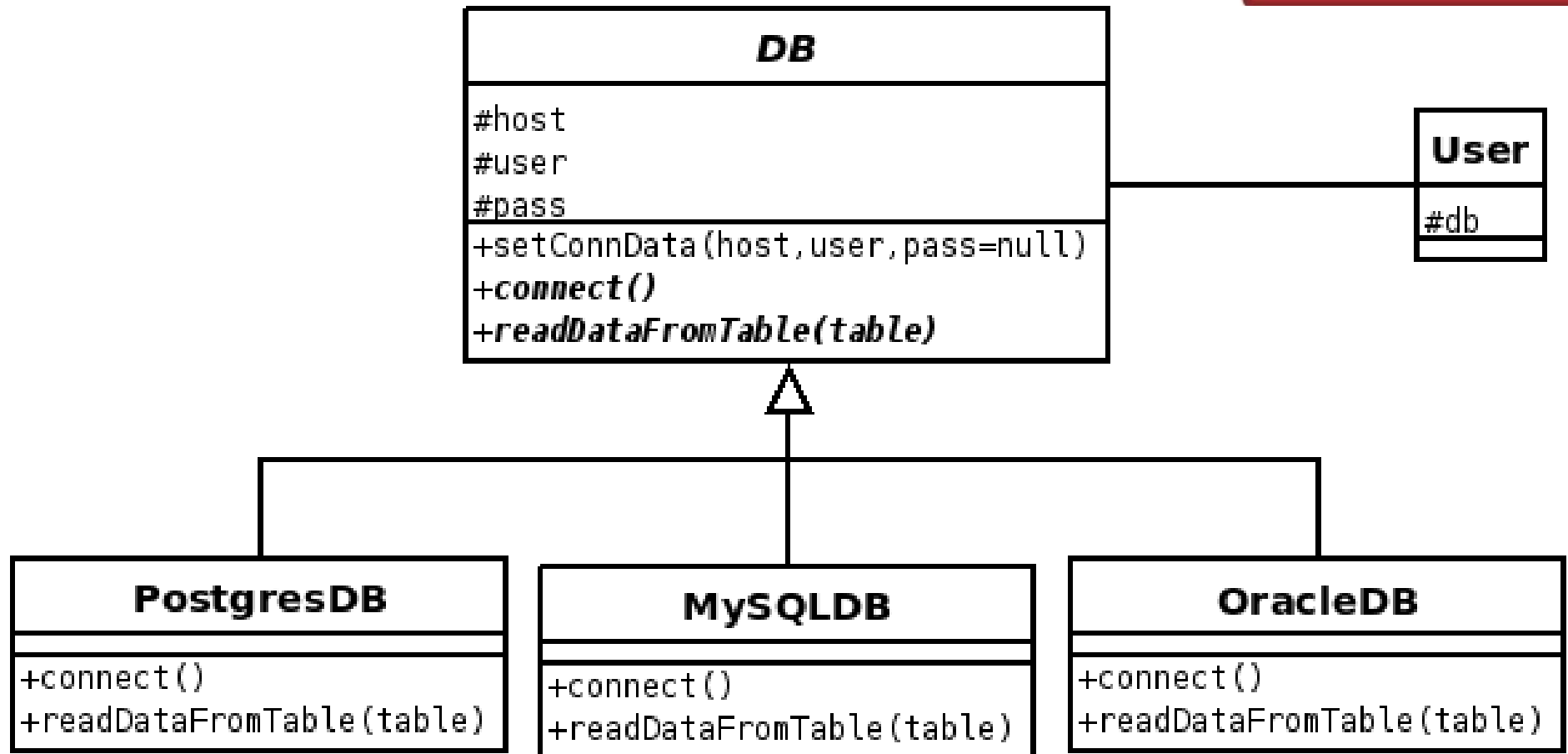


“Permite que um objeto de uma classe seja tratado como um objeto de outra classe”

- **Básico: Herança**
- **Classes Abstratas**
- **Interfaces**
- **Exemplos na pasta *polimorfismo***

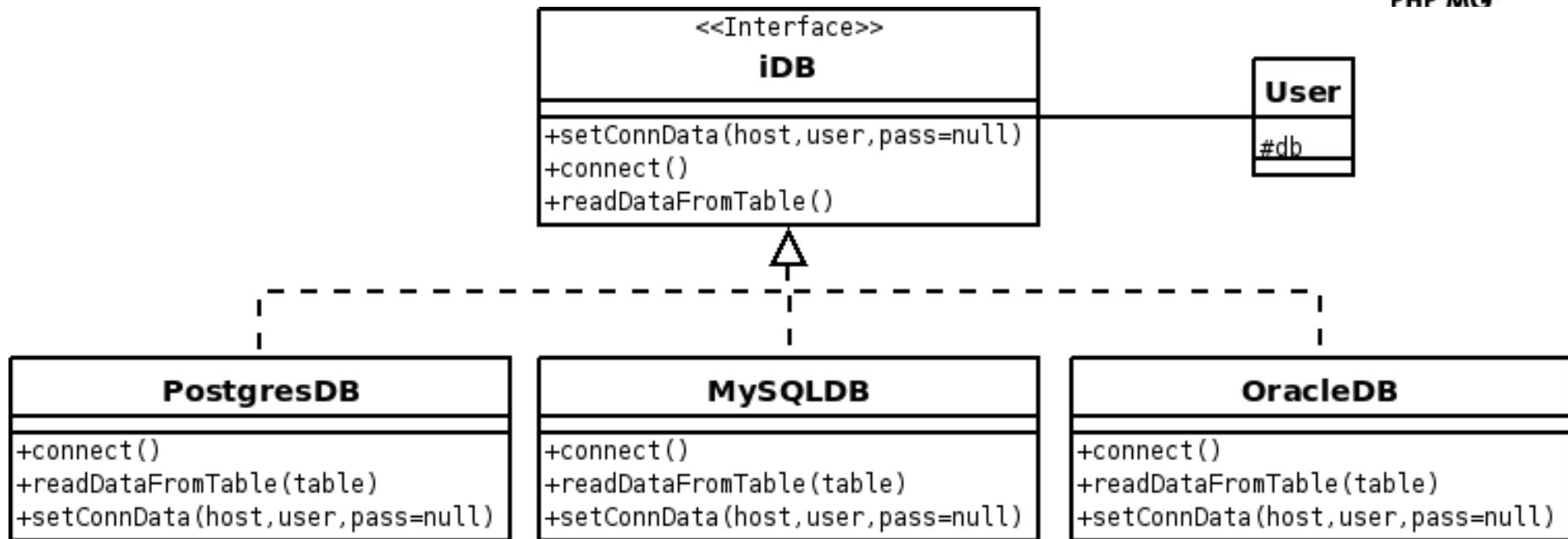
Entendendo a Teoria

Polimorfismo *



Entendendo a Teoria

Polimorfismo *



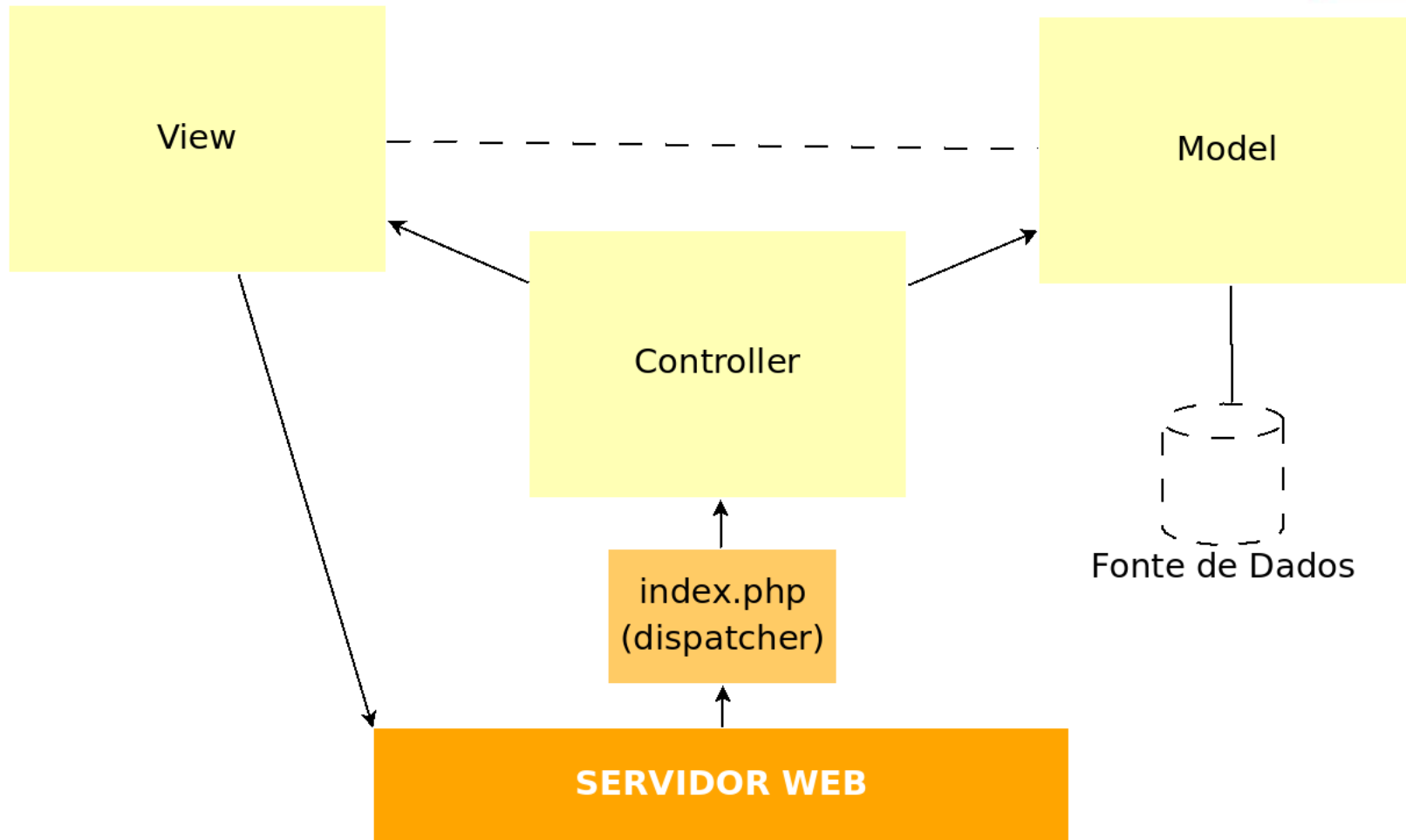
Arquitetura MVC



- **Separa a aplicação em três camadas**
 - **Model**
 - Trata o acesso aos dados
 - Responsável pela lógica de negócios
 - **View**
 - Camada responsável pela lógica de apresentação
 - **Controller**
 - Orquestrador
 - Recebe a requisição e busca os dados do Model
 - Passa para a View os dados recebidos do Model para a apresentação

Arquitetura MVC

PHP



Arquitetura MVC

- **Vantagens da separação em camadas**
 - Facilita a manutenção e extensão
 - Modularização por camada



Arquitetura MVC

PHP



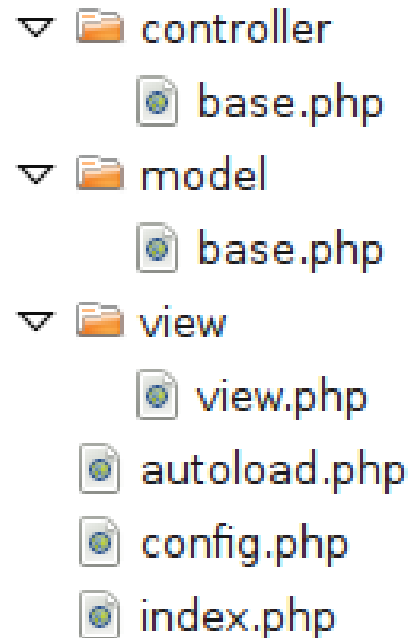
- **Request Dispatcher**

- Um arquivo responsável por descobrir qual o controller e método está sendo requisitado.
- Exemplo na pasta MVC (arquivo index.php)
- Exemplo de acesso: `index.php?c=cliente&ac=view&p=2`
 - Controller: cliente
 - Método: listar()

Praticando

PHP + POO

- Criando uma estrutura MVC básica
- Estrutura de pastas



Praticando

PHP + POO



- Model Base (model/base.php)
- Métodos:
 - `__construct()`
 - `__destruct()`
 - `query($sql)`
 - `select($table, $conditions = array())`
 - `connect()`
 - `disconnect()`

Praticando

PHP + POO



- Controller Base (controller/base.php)
- Métodos:
 - `__construct()`
 - `loadModel($modelName)`
- Atributos
 - `$view, $name, $basePath`
 - `$models`
 - `$models['Cliente'] = new ClienteModel();`

Praticando

PHP + POO



- View(view/view.php)
- Métodos:
 - `__construct()`
 - `assign($var, $val)`
 - `render($tpl_file)`
- Atributos
 - `$vars`, `$basePath`

Os templates serão arquivos .php dentro da pasta view. Ex.:

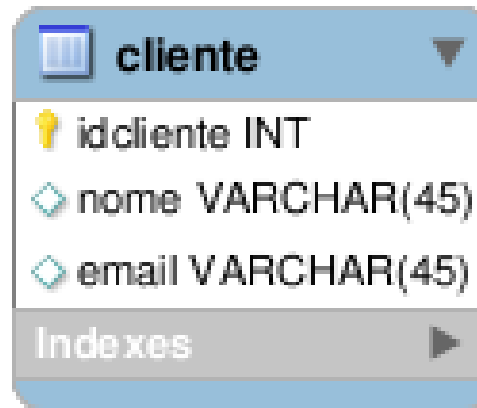
`view/cliente/novo_cliente.php`

Praticando

PHP + POO



- Iniciando a aplicação
- Módulo: Clientes
 - Controller: cliente.php (ClienteController)
 - Model: cliente.php (ClienteModel)
 - Views (cliente): create.php, index.php, view.php
 - Tabela cliente



Praticando

PHP + POO



- Model Cliente:
 - Métodos
 - pegaCliente(\$id)
 - pegaListaCliente()
 - criarCliente()

Praticando

PHP + POO



- Controller Cliente:
 - Métodos
 - Index() → lista de usuários
 - view(\$id = null) → exibe os dados do usuário
 - CriarCliente() → Salva os dados do cliente no banco

Praticando

PHP + POO



- Views Cliente:
 - arquivos
 - Index.php → lista de usuários
 - view.php → exibe os dados do usuário
 - create.php → Salva os dados do cliente no banco

O quê mais?

- **Atributos estáticos**
- **Métodos estáticos**
- **Classes e Métodos finais**
- **Constantes de Classes**

