



As Novidades do PHP5

Pablo Dall'Oglio
Adianti Solutions

www.adianti.com.br



- I CISL, 2003, Curitiba-PR;
- III SDSL, 2004, Univates, Lajeado-RS;
- O caminho do SW livre, 2005, Crisciúma-SC;
- VI Fórum Int. SW Livre, 2005, POA-RS;



PHP/FI 1 PHP/FI 2

- 1994 Rasmus Lerdorf (Personal Home Pages);
- PHP/FI Form Interpreter (Curriculum Vitae);
- Ferramenta para criar páginas dinâmicas;

PHP 3 PHP: Hypertext Preprocessor

Zend Engine 0.5: Zeev Suraski e Andi Gutmans

- Infraestrutura, Implementa a sintáxe;
- Define as regras de parse;
- OO (arrays associativos);



PHP 4 PHP: Hypertext Preprocessor

- Zend Engine 1.0: Zeev Suraski e Andi Gutmans
- Abstração do Servidor Web;
- Nova API, extensões: PEAR, CLI, PHP-GTK;
- Crescimento revolucionário, mecanismo limitado;

Zend Engine 2.0:

- Muito tempo para ser escrito (escopo grande);
- Novo modelo OO (private, public, abstract);



Considerações

- Extremamente fácil de utilizar;
- Possivelmente a linguagem de script mais utilizada na web;
- Vasta Bibliografia a respeito;
- Muitas funções (arquivos, arrays, strings, SimpleXML, DomXML, Sax);
- Muitas extensões (BD, FTP, LDAP, ZLIB, GD, PDF);
 - Pgsqll, Ora, Mysql, Sqlserver, Sqlite, Firebird, Sybase, Frontbase, Informix, ODBC, Dbase, SapDB, Adabas, DB2, dentre outros.
- Flexível, não-rígida, permite procedural e OO;



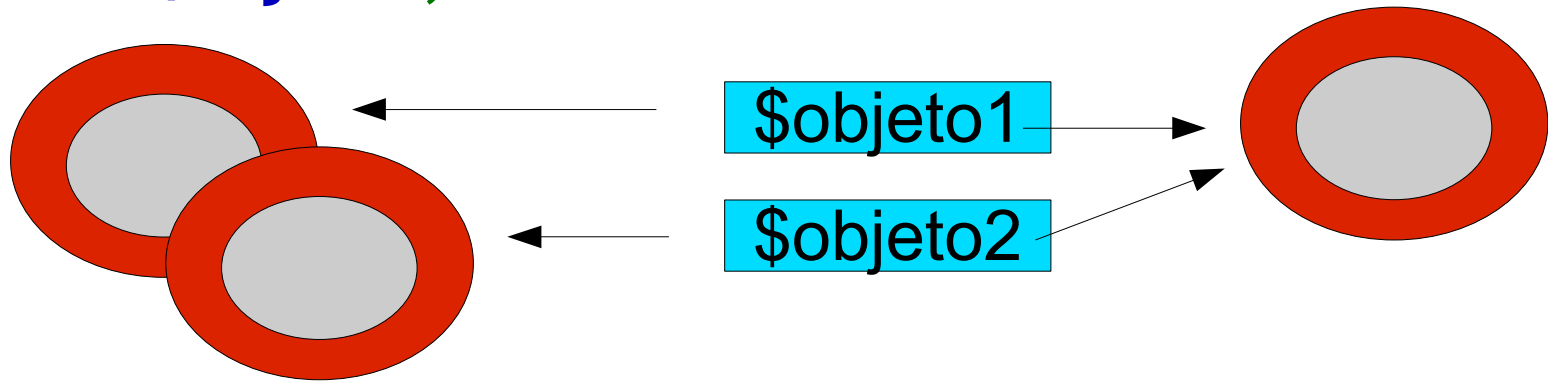
Modelo de Objetos

Zend 1.0: Objetos tratados como valores (int, str);

Em atribuições, parâmetros, os objetos são copiados inteiramente;

Java: O modelo trata o objeto como referência, não valor;

\$objeto1 = \$objeto2;



Novo Modelo: Permite destrutores, controle de duplicação, retorno de valores referência;

Novo Modelo: Quando se cria um objeto, se obtém uma referência (handler, ponteiro), e não o objeto em si;



Para instanciar objetos:

```
<?php
```

```
// criar instancia
```

```
$objeto = new NomeDaClasse;
```

```
// chamada de método
```

```
$objeto->NomeDoMetodo();
```

```
?>
```



Modelo de Objetos

```
<?php
# Classe
class Carro
{
    // Método
    function setNome($valor)
    {
        $this->nome = $valor;
    }
    // Método
    function getNome()
    {
        return $this->nome;
    }
}

# Função
function trocaNome($obj)
{
    $obj->setNome('pálio');
}
?>
```



Modelo de Objetos

Novo Modelo: Somente o ponteiro é enviado para funções, atribuições ou copiado, nunca o mesmo objeto. Trabalha sobre o ponteiro do objeto (by reference);

```
<?php
// criar instancia
$objeto = new Carro;

// atribui nome
$objeto->setNome('gol');

// chamada de função
trocaNome($objeto);

// imprime propriedade
print $objeto->getNome();
?>
```

// PHP4 com & = mesmo resultado

PHP4

gol

PHP5

pálio



Modelo de Objetos

```
<?php

// Instancia a classe
$a = new Carro;

// Atribu um valor para $a
$a->Nome = 'Mille';

// Duplica o objeto
$b = $a;

$b->Nome = 'Pálio';

// Exibe os objetos
echo $a->Nome;
echo $b->Nome;

// Verifica se são iguais
if ($a === $b)
    echo 'São Iguais';
else
    echo 'São Diferentes';

?>
```

PHP4

Mille
Pálio
São Diferentes

PHP5

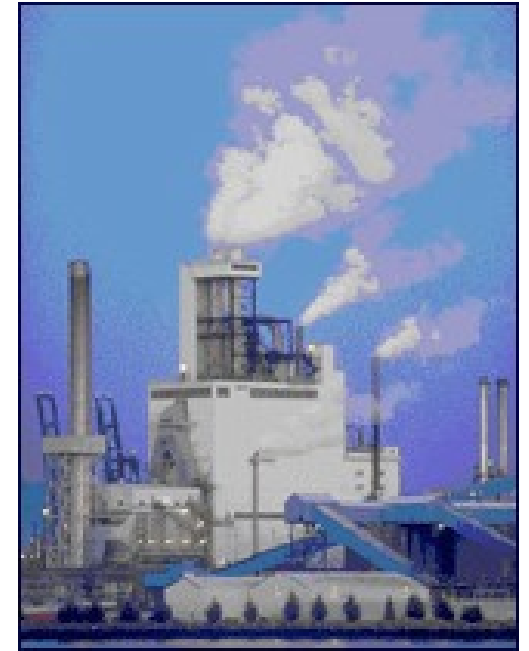
Pálio
Pálio
São Iguais

Factory

Utilizar uma classe central para geração de objetos, novo modelo retorna exatamente a mesma instância e não uma réplica;

```
<?php
// Classe Factory
Class Factory
{
    // Métodos Factory
    function CriarCliente($nome)
    {
        return new Cliente($nome);
    }
}

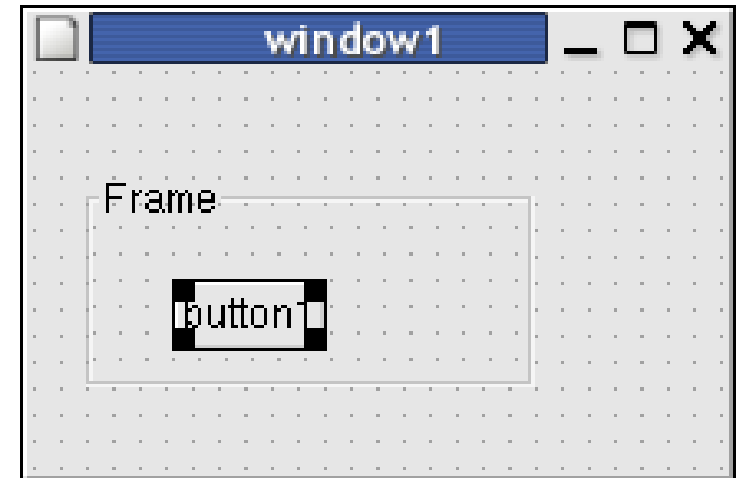
// Instanciação
$Pedro = Factory::CriarCliente('Pedro');
$Maria = Factory::CriarCliente('Maria');
?>
```



Referenciando objetos retornados

Velho modelo:

```
<?
$frame = $botao->get_parent();
$janela = $frame->get_parent();
$janela->set_title('titulo');
?>
```



Novo modelo:

```
<?
$botao->get_parent()->get_parent()->set_title('titulo');
$obj->metodo()->metodo();
func()->metodo()->metodo();
$obj->metodo()->member->metodo();
?>
```



Referenciando objetos retornados

```
<?
# Classe Pessoa

Class Pessoa
{
    function Pessoa($nome)
    {
        $this->nome = $nome;
    }
    function GetNome()
    {
        return $this->nome;
    }
}

# Função CriaPessoa

function CriaPessoa($nome)
{
    return new Pessoa($nome);
}
?>
```



Referenciando objetos retornados

PHP4 :

```
$joao = CriaPessoa('Joao');
```

```
// Imprime o nome
```

```
echo $joao->GetNome();
```

PHP5 :

```
// Imprime o nome
```

```
echo CriaPessoa('Joao')->GetNome();
```



Clonagem



Clonagem

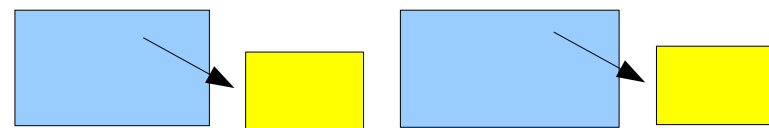
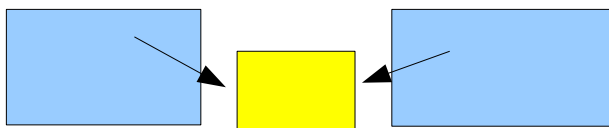
Velho modelo:

Como objeto é tratado como valor, a duplicação é feita bit-a-bit (cópia idêntica), nem sempre o desejável (**GtkWindow->id**);

\$objetoB = \$objetoA;

Se o objeto pai tiver uma propriedade que é um objeto (composição), duplicar o filho ou fazer os 2 apontar para o mesmo ?

Se não existir a função `__clone()`, usa padrão:



\$objetoB = clone \$objetoA;

Pessoa--<>-----Endereço



Clonagem

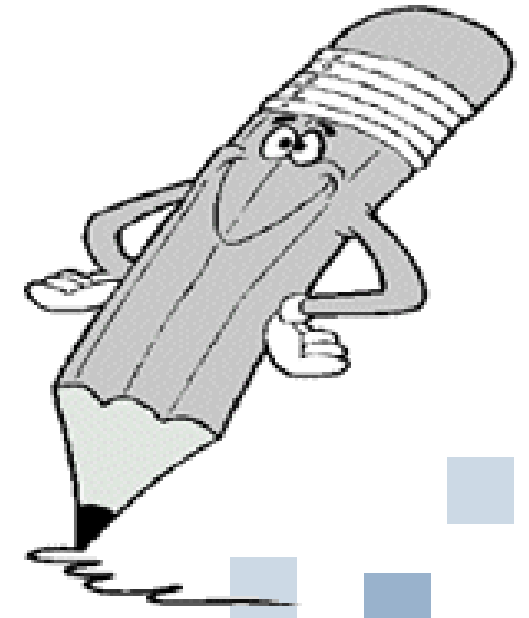
```
<?
// Classe Pessoa
Class Pessoa
{
    static $codigo; // Propriedade estática

    // Método de clonagem
    function __clone()
    {
        $this->codigo = $this->codigo +1;
    }
}

// Instanciar
$a = new Pessoa;
$a->codigo = 4;

// Clonar
$b = clone $a;

// Imprimir
echo $a->codigo; // 4
echo $b->codigo; // 5
?>
```



PHP4

```
class Pessoa
{
    function Pessoa($nome, $idade)
    {
        ...
    }
}
```

PHP5

```
class Pessoa
{
    function __construct($nome, $idade)
    {
        ...
    }
}
```

```
$joao = new Pessoa('joaozinho', 11);
```



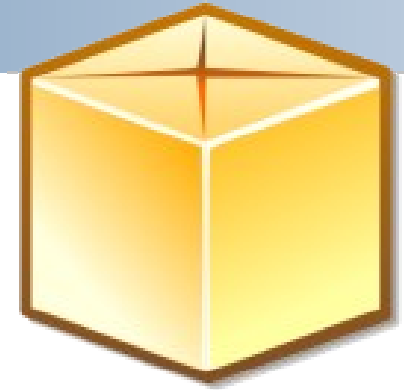
Destruutores

Utilidade: Debug msg's, limpar arquivos temporários, é chamado antes que o objeto seja liberado da memória;

```
<?
class ConexaoBancoDados
{
    // Método construtor
    function __construct($host, $user, $pass)
    {
        $this->conn = mysql_connect($host, $user, $pass);
    }
    // Método destrutor
    function __destruct()
    {
        mysql_close($this->conn);
    }
}
$a = new ConexaoBancoDados('127.0.0.1', 'eu', 'xyz');
?>
```



Destruutores



```
<?
class CestaDeCompras
{
    function __construct()
    {
        echo "construindo...\n";
    }
    function __destruct()
    {
        echo "destruindo...\n";
    }
}

class OutraCestaDeCompras extends CestaDeCompras
{
    function __construct()
    {
        parent::__construct();
        echo "construindo outra cesta\n";
    }
}

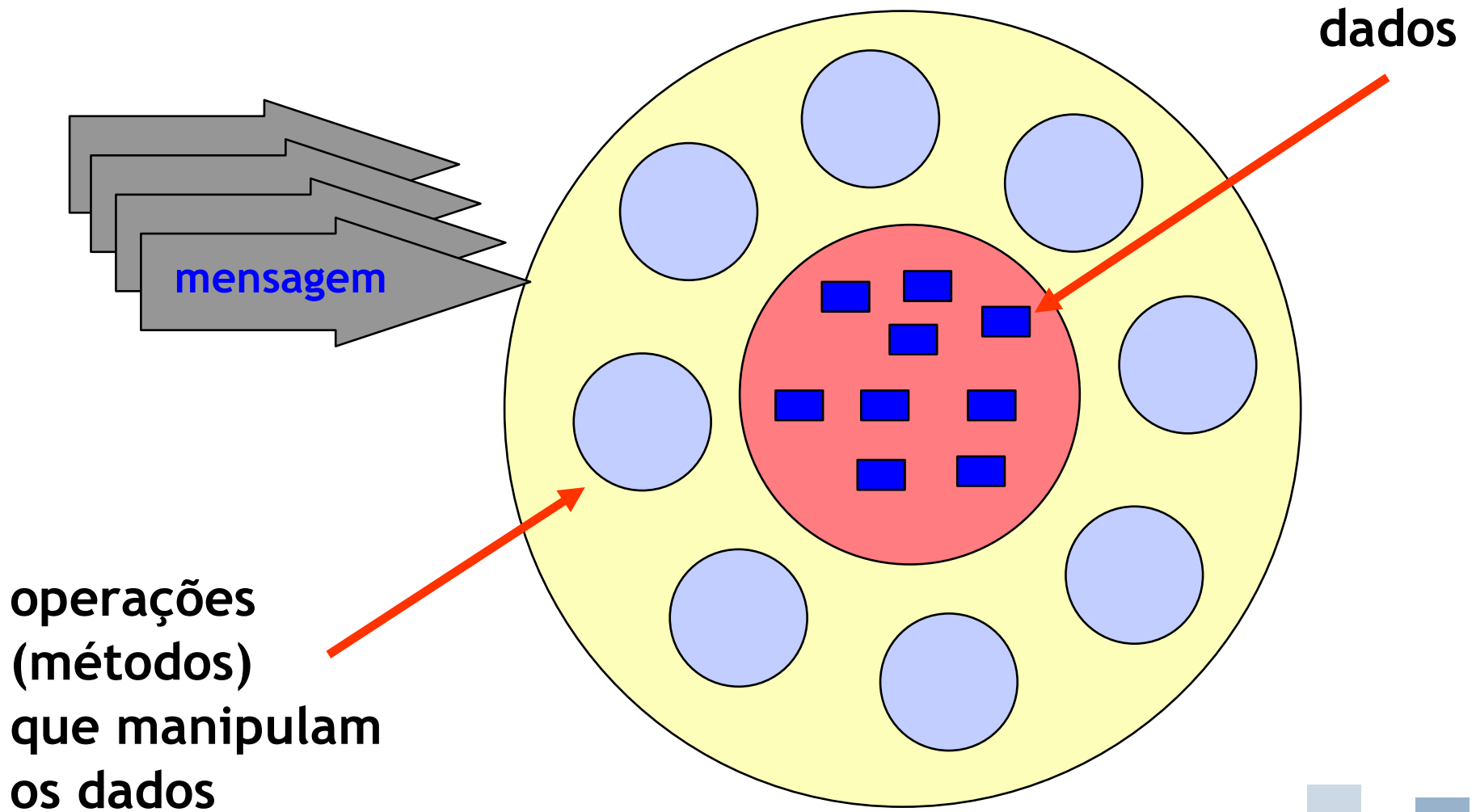
$a = new OutraCestaDeCompras;

?>
```

Resultado

```
construindo...
construindo outra cesta
destruindo...
```

Esquema de um Objeto





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()
```

Novos conceitos, para propriedades e métodos:

- **private;**
- **protected;**
- **public;**



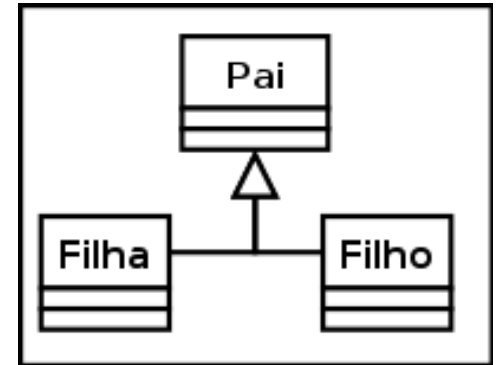
```
<?
class Pessoa
{
    protected $salario;
}
```

Herança

```
class Empregado extends Pessoa
{
    function GetSalario()
    {
        return $this->salario;
    }
}
```

Teste...

```
$joao = new Empregado;
echo $joao->GetSalario(); // ok
echo $joao->salario;     // não
?>
```



Propriedades Estáticas

No ZE 1.0, já existiam métodos estáticos, PHP5 introduz as propriedades estáticas:

```
<?
class Pessoa
{
    // Propriedade estática
    static $qtde_instancias;

    // Retorna Propriedade estática
    static function getQtde()
    {
        return self::$qtde_instancias;
    }
}
?>
```



Propriedades Estáticas

```
<?  
  
// criar instancia  
$joao = new Pessoa;  
Pessoa::$qtde_instancias ++;  
  
// criar instancia  
$maria = new Pessoa;  
Pessoa::$qtde_instancias ++;  
  
// Imprime quantidade  
print Pessoa::GetQtde();  
  
?>
```

Resultado

2

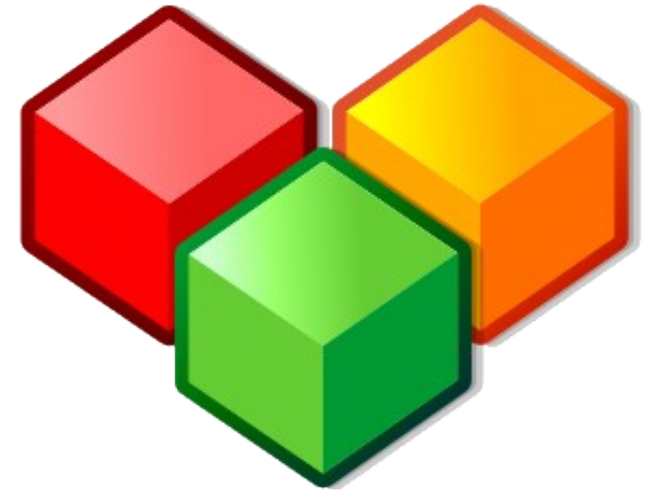


Constantes

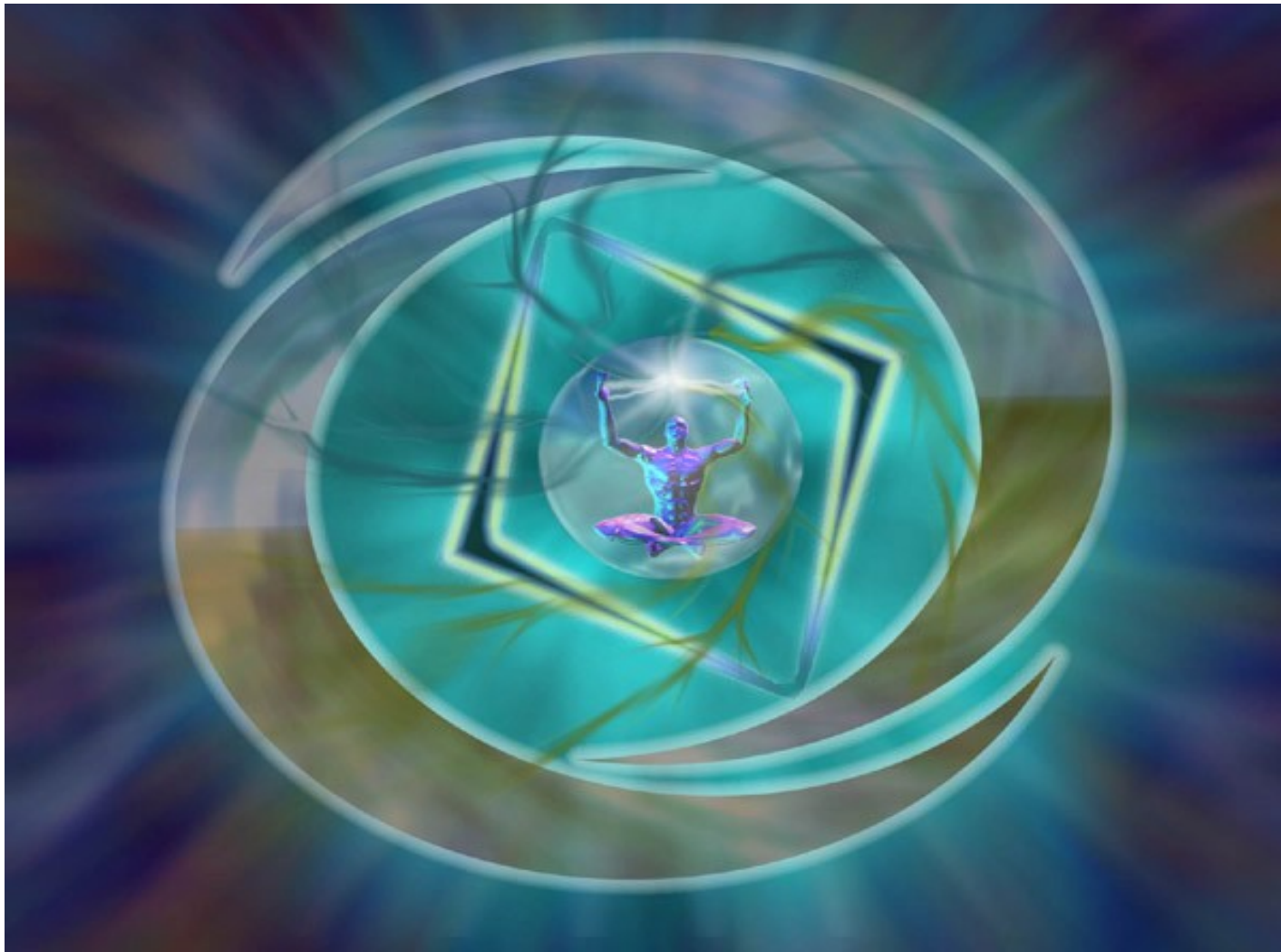
```
<?
// Classe Pessoas
class Pessoas
{
    const Numero = 5;
}

// imprime constante
echo Pessoas::Numero;

// Classe Maquinas
class Maquinas
{
    const Numero = 5;
}
// imprime constante
echo Maquinas::Numero;
?>
```



Interceptadores



__set(), __get() e __call()

```
<?
class MinhaClasse
{
    function __set($name, $value)
    {
        $this->$name = $value;
    }
    function __get($name)
    {
        return $this->$name;
    }

    function __call($metodo, $parametro)
    {
        // ...
    }
}
?>
```



__set()

```
<?php
class Pessoa
{
    private $Nascimento;
    // Retorna a Data
    function GetNascimento()
    {
        return $this->Nascimento;
    }
    // Intercepta a atribuição
    function __set($nome, $valor)
    {
        if ($nome == 'Nascimento')
        {
            $partes = explode('/', $valor);
            if (count($partes)==3)
                $this->$nome = $valor;
        }
    }
}
?>
```



__set()

```
<?php

// Instancia Pessoa
$julia = new Pessoa;

// Atribui Nascimento
$julia->Nascimento = '04/03/1980';
echo $julia->GetNascimento() . "\n";

// Atribui Nascimento
$julia->Nascimento = '8 de março';
echo $julia->GetNascimento() . "\n";

?>
```

Resultado

```
04/03/1980
04/03/1980
```



__get()

```
<?php
class Conta
{
    private $Saldo;
    private $Limite = 500;

    // Altera o Saldo
    function SetSaldo($valor)
    {
        return $this->Saldo = $valor;
    }

    // Intercepta obtenção
    function __get($nome)
    {
        if ($nome == 'Saldo')
        {
            return $this->Saldo + $this->Limite;
        }
    }
}
?>
```



__get()

```
<?php
// instancia
$contaCorrente = new Conta;

// altera saldo
$contaCorrente->SetSaldo(100);
echo $contaCorrente->Saldo;

// altera saldo
$contaCorrente->SetSaldo(260);
echo $contaCorrente->Saldo;

?>
```

Resultado

```
600
760
```

__call()

```
<?php
class Conta
{
    private $Saldo;
    private $Limite = 500;

    // Altera o Saldo
    function SetSaldo($valor)
    {
        return $this->Saldo = $valor;
    }

    // Intercepta obtenção
    function __call($nome, $parametros)
    {
        echo "Método não existente\n";
        echo $nome . ' - ' . $parametros[0];
    }
}
?>
```



```
<?php
// instancia
$contaCorrente = new Conta;

// altera saldo
$contaCorrente->SetSaldo(100);

// altera conjugue
$contaCorrente->SetConjugue('Joana');
?>
```

Resultado

Método não existente
SetConjugue - Joana



Abstract

```
<?
abstract class Pessoa
{
    abstract function GetSalario();
    // sobrecarga obrigatória
}

class Funcionario extends Pessoa
{
    // ...
}
class Estagiario extends Pessoa
{
    // ...
}

$joao = new Funcionario;           // ok
$joao = new Pessoa;               // não
?>
```



```
<?
final class Pessoa
{
    final function GetSalario()
    {
    }
}

class Funcionario extends Pessoa
{
    // ...
}

# Classe final não pode ser herdada.
# Método final não pode ser sobrecarregado.

?>
```



Interfaces

```
<?
interface IAutomovel
{
    // class type hints
    public function Ligar(motor $a);
}

class Palio implements IAutomovel
{
    // public function Ligar(motor $a) { }
}

class Clio implements IAutomovel
{
    // public function Ligar(motor $a) { }
}
?>
```



Manipulação de Erros

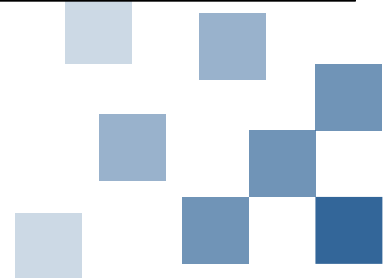
```
<?
// função Hello
function Hello($nome = null)
{
    if ($nome === null)
    {
        throw new Exception('falta parametro');
    }
    else
    {
        echo "Hello $nome\n";
    }
}
}
```

```
// TRY-CATCH
try
{
    Hello();
    Hello('John');
}
catch(Exception $e)
{
    echo "erro: " . $e->getMessage();
}
?>
```



Resultado

#1 erro: falta param.
#2 "Hello John"



```
< Pessoa >  
  < codigo > 5 </ codigo >  
  < nome > Maria da Silva </ nome >  
  < idade > 24 </ idade >  
</ Pessoa >
```

```
<?php  
//carrega o arquivo  
$xml = simplexml_load_file('teste.xml');  
  
// acessa os dados XML  
echo "Codigo: " . $xml->codigo . "\n";  
echo "Nome: " . $xml->nome . "\n";  
echo "Idade: " . $xml->idade . "\n";  
?>
```

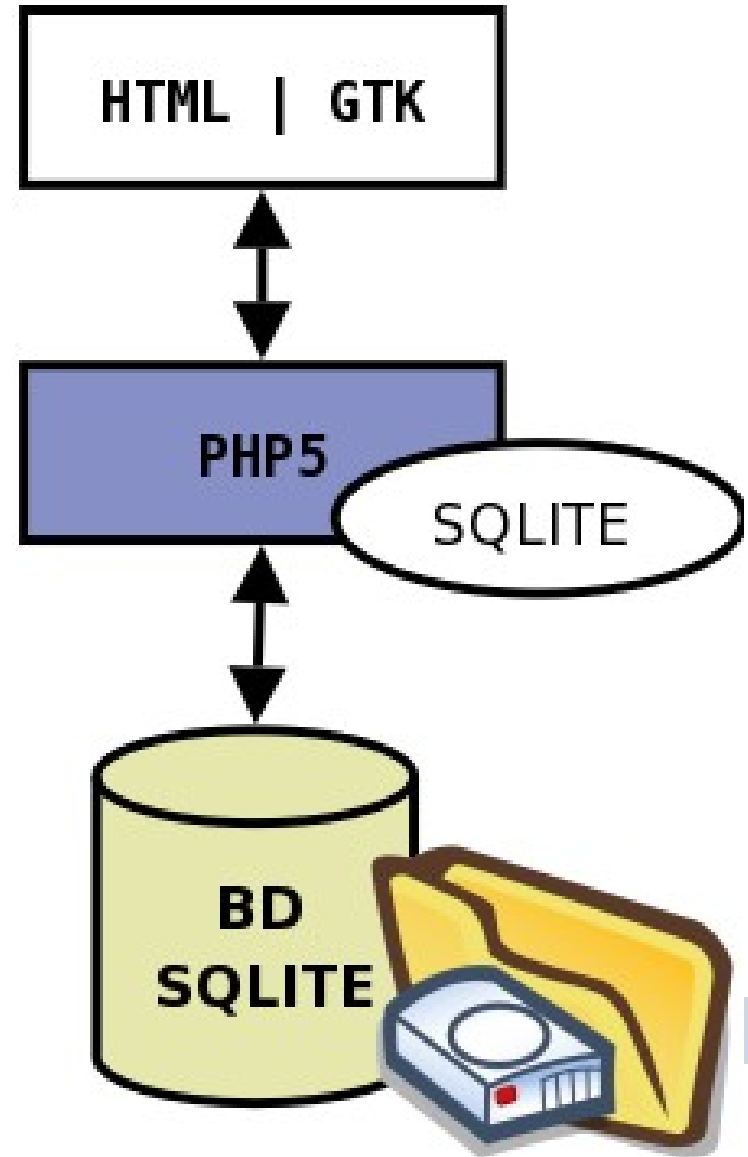
Resultado

```
Codigo: 5  
Nome : Maria da Silva  
Idade : 25
```



SQLite

- Embutido no PHP5;
- Iniciado no ano de 2000 por Richard Hipp;
- Banco de Dados composto por simples arquivos;
- Subselect, triggers, transações, views, 2-3x mais rápido que MySQL, limite de 2 Terabytes, views são read-only, sem foreign keys;



```
<?php
// Abre o BD (cria se não existir);
$db = sqlite_open("cidades.db");

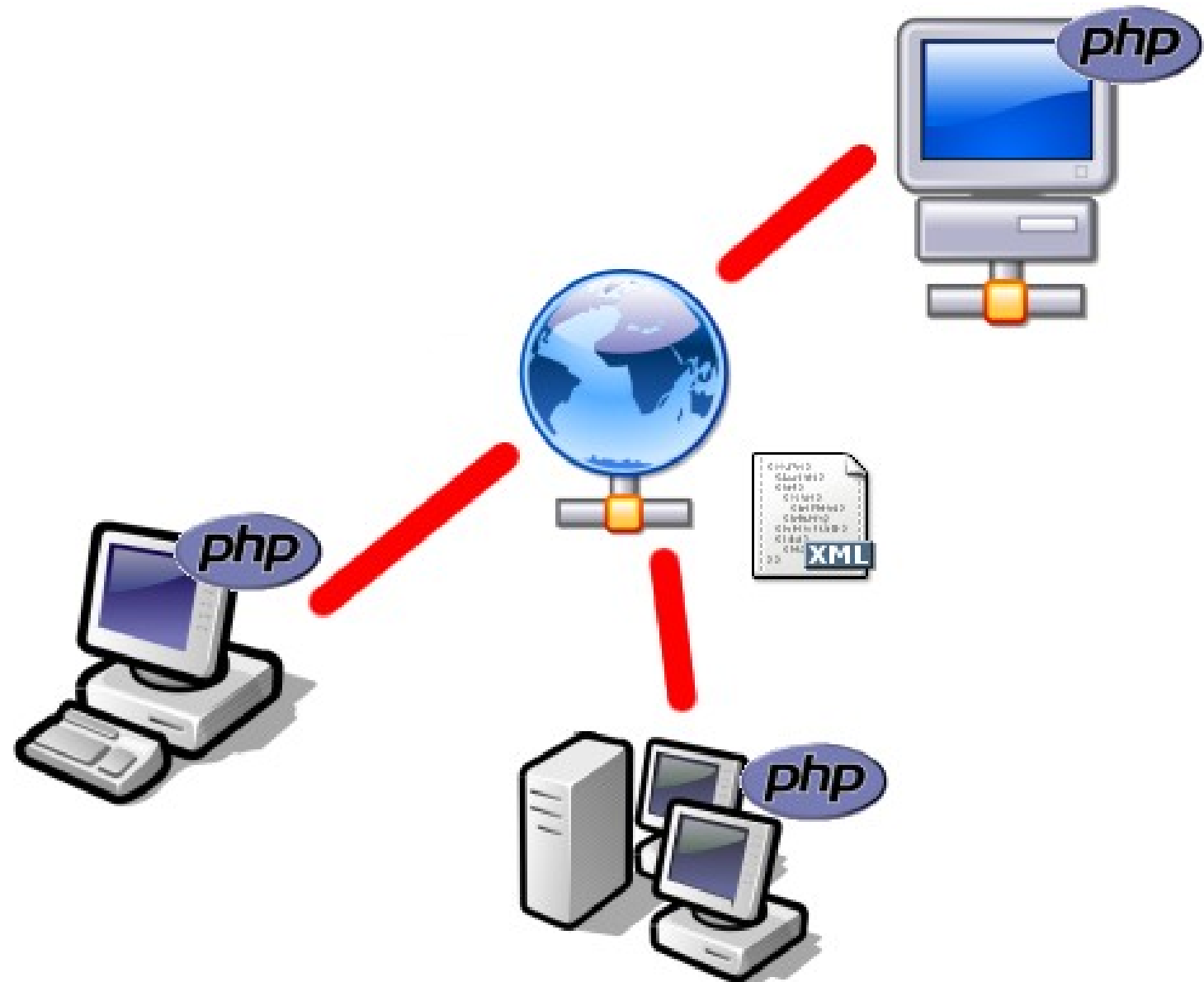
// Consultas
$result = sqlite_query($db, "SELECT * from cidades");

// Retornando os dados
while ($row = sqlite_fetch_array($result))
{
    print_r($row);
}

// Fecha a conexão
sqlite_close($db);
?>
```

City Code	City Description	State Code	State Description
1	Arlington	MA	Massachusetts
2	Cambridge	MA	Massachusetts
3	Lawrence	MA	Massachusetts
4	Somerville	MA	Massachusetts
5	Albany	NY	New York
6	Cortland	NY	New York
7	Hamilton	NY	New York
8	Waterville	NY	New York

Web Services



Web Services

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<definitions name = 'Exemplo'>

  <message name = 'getNomeRequest'>
    <part name = 'codigo' type = 'xsd:string' />
  </message>

  <message name = 'getNomeResponse'>
    <part name = 'resultado' type = 'xsd:string[]' />
  </message>

  <portType name = 'ExemploPortType'>
    <operation name = 'getNome'>
      <input message = 'tns:getNomeRequest' />
      <output message = 'tns:getNomeResponse' />
    </operation>
  </portType>

  ... como o Webservice irá ser codificado
  ... como o Webservice irá ser transportado

</definitions>
```



Web Services

```
<?php
function getNome($codigo)
{
    // conecta ao Banco de Dados
    $id = @pg_connect("dbname=samples user=postgres");

    // realiza consulta ao Banco de Dados
    $result = pg_query($id, "select * from clientes " .
        "where codigo=$codigo");

    $matriz = pg_fetch_all($result);

    if ($matriz == null)
        throw new SoapFault("Server",
            "Cliente nao encontrado");

    // retorna os dados.
    return $matriz[0];
}

// instancia servidor SOAP
$server = new SoapServer("exemplo.wsdl");
$server->addFunction("getNome");
$server->handle();
?>
```



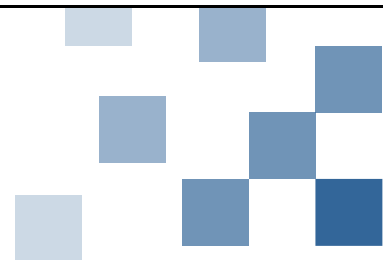
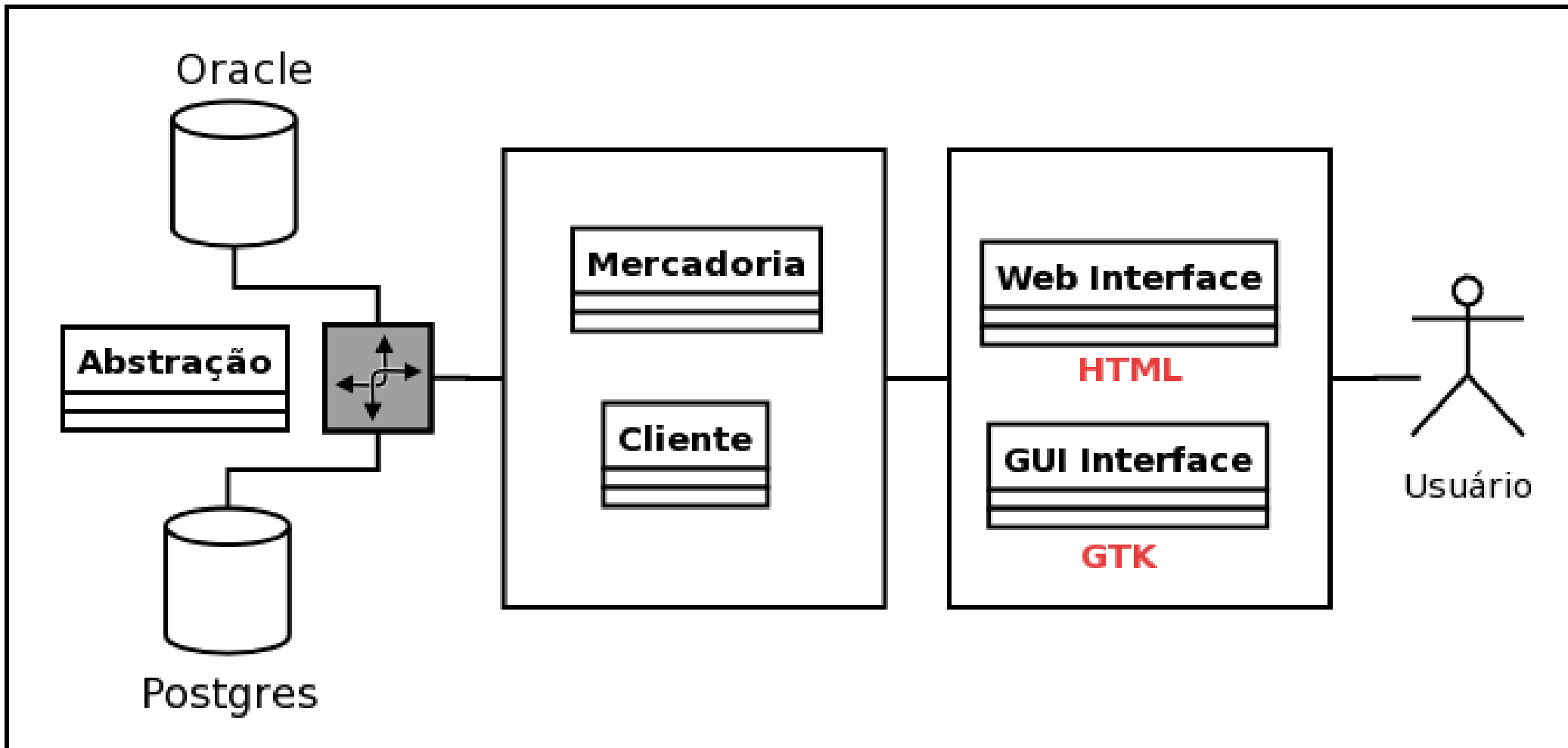
Web Services

```
<?php
// instancia cliente SOAP
$client = new SoapClient("exemplo.wsdl");

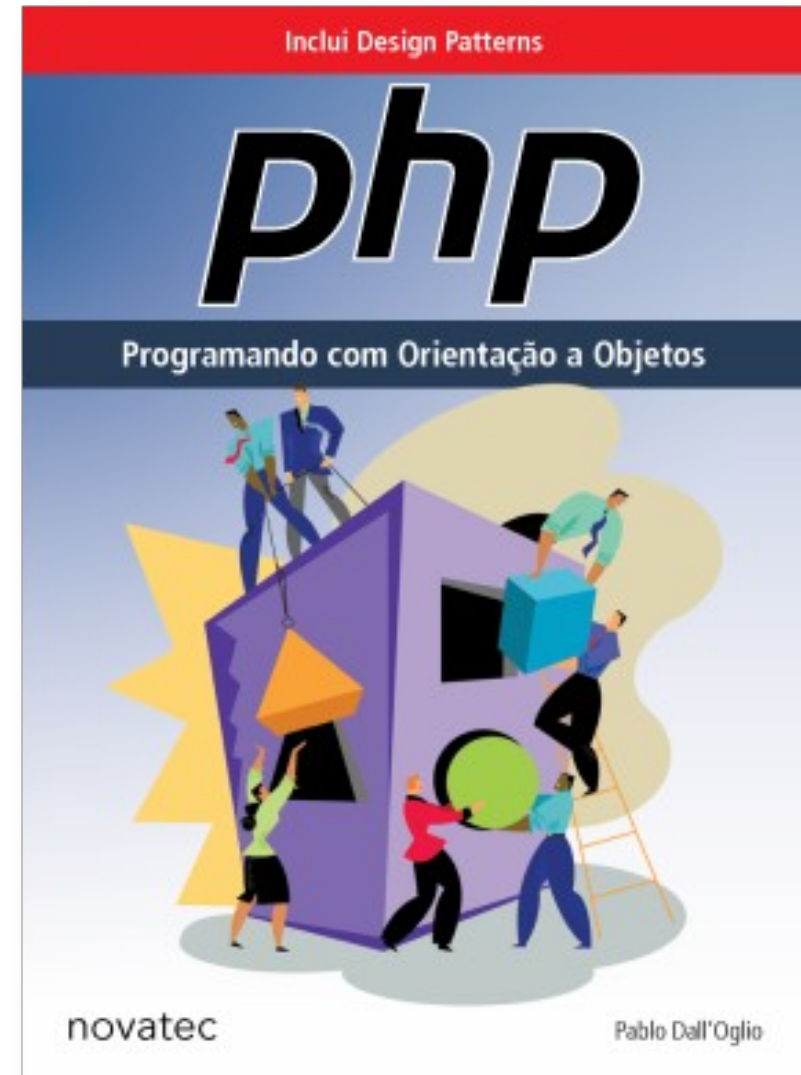
try
{
    // realiza chamada remota de método
    $retorno = $client->getNome(3);

    echo $retorno['codigo'];
    echo $retorno['nome'];
    echo $retorno['telefone']
}
catch (SoapFault $excecao) // ocorrência de erro
{
    echo "Erro: ";
    echo $excecao->faultstring;
}
?>
```





Referências



Obrigado!

E-Mail

pablo@php.net

pablo@dalloaglio.net

URL

<http://www.php-gtk.com.br>

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

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



Creative Commons

- Estes slides estão disponíveis sob a licença não comercial da creative commons 1.0;
- Você pode distribuir, copiar, exibir e realizar outros trabalhos seguindo estas condições:
 - **Atribuição:** Você deve dar os créditos ao autor original;
 - **Não-Comercial:** Você não pode utilizar este trabalho para propósitos comerciais;
 - **Não derivar trabalhos:** Você não pode alterar, transformar, ou construir algo sobre este trabalho.
 - Para todo reuso ou distribuição, você deve deixar explícito para os outros, os **termos da licença** deste trabalho;
 - Estas condições podem ser alteradas se você obter **permissão do autor**;
- O uso justo e outros direitos não são afetados pelas condições acima.

