

# Objetos em Ação

Ivo Nascimento

[ivo.nascimento@ianntech.com.br](mailto:ivo.nascimento@ianntech.com.br)

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

# O que é POO

**POO : Programação Orientada a Objeto** (object oriented programming).

Paradigma de programação no qual a unidade de programação é a classe a partir da qual os objetos são criados (instanciados).

Uma classe pode ser definida como um tipo definido pelo programador.

# Benefícios da POO

- Organização/Centralização de Código.
- Reutilização de Código.
- Facilidade de manutenção.
- Visão real do problema.
- Integridade(proteção/validação) dos dados.

# onde pode ser aplicada?

A POO é um paradigma que pode ser aplicado a qualquer problema proposto em sistemas. Desde definição de pequenos problemas à problemas de grande complexidade.

# Definição de Objeto

**Objeto** - representação de um conceito ou coisa física com as definições de suas ações(métodos) e características(propriedades).

**Classe** - Definição programática de determinado problema.

**Instância** – Ocorrência específica de um objeto.

# exemplo (objeto: pessoa)

## propriedades

- nome
- data de nascimento
- altura
- peso
- estado civil

## métodos

- setNome / getNome
- setDtNasc /  
getDtNasc
- setAltura / getAltura
- setPeso / getpeso
- setEC / getEC

# o objeto

```
class pessoa {  
    private $Nome = null;  
    private $DtNasc = null;  
    private $Altura = null;  
    private $Peso = null;  
    private $EC = null;  
    private $Sexo = null;// pq  
    null?  
    public function  
    setNome($nome){  
        $this->Nome = $nome;  
    }  
    public function getNome(){  
        return $this->Nome;  
    }  
}
```

```
    public function  
    setDtNasc($data) {  
        // aqui validamos a data  
        if ($this->isDate($data))  
            $this->DtNasc = $data;  
    }  
    private function isDate($data) {  
        // logicoao de validação  
    }  
    public function  
    setAltura($altura) {  
        if ($altura > 0)  
            $this->Altura = $altura  
    }  
}
```

# decompondo um Objeto

## **propriedade**

são características comuns a todos os objetos daquela classe.

Toda pessoa tem um nome, uma altura, um peso e uma data de nascimento

## **métodos**

são ações comuns a todos os objetos daquela classe.

Toda pessoa em algum momento informa seu nome, , sua altura, seu peso ou sua data de nascimento.

# visibilidade

**private** – somente dentro da classe em que é definido

**public** – por que a define, pelo software que instancia e pelas sub classe

**protected** – somente pela classe em que é definido e pela subclasse.

obs.: métodos/propriedades sem definição de visibilidade são public.

# setters e getters

Métodos set e get são comumente utilizados para inserção de dados e leitura de dados em propriedades que são de visibilidade private.

São também utilizados para a validação dos dados a serem inseridos.

# Encapsulamento

Somente podem ser manipulados aqueles métodos e propriedades que o programador da classe definiu dessa maneira. Isso devido ao ocultamento das características internas.

Isso permite:

- proteção do código
- melhorias na classe
- validação das informações

# Programando com o objeto

```
<? // usando o objeto pessoa
  $Pessoa = new pessoa();
  $Pessoa->setNome('Zé Carioca');
  $Pessoa->setAltura(1.90);
echo 'O nome do único personagem da Disney
genuinamente brasileiro é '$Pessoa->getNome().' , sua
altura é de'.$Pessoa->getAltura();
// um uso mais realístico seria ler os registro de um banco de dados
de pessoas e inseri-los num array destes objeto;
$Pessoa2 = new pessoa();
$Pessoa2->setNome('pato Donalds');
$peessoas = array($Pessoa, $Pessoa2);
  //print_r($peessoas);
  foreach ($peessoas as $p){ echo $p->getNome()."\n";}
?>
```

# Herança

O conceito de herança de objetos é um dos pilares da melhoria de qualidade e aproveitamento daqueles que utilizam esta técnica.

Consiste em especializar uma classe com intuito de dar finalidade mais específica.

conceitos:

**SuperClasse**

**SubClasse**

# Especializando a classe pessoa

Uma pessoa, na vida real, pode ter muitas particularidades que não seriam cobertas pelo class pessoa.

Uma pessoa pode ser:

estudante

analista de sistemas

instrutor

palestrante

# A sub class estudante

<?

```
class estudante extends pessoa{  
  private $curso = null;  
  private $ano = null;  
  private $faculdade = null;  
  private $nome=null;  
  public static $quantidade = 0;  
  function __construct()  
    {  
      self::$quantidade++;  
    }  
  public function setNome($nome){  
    parent::setNome($nome);  
  }  
}
```

?>

# A sub class Analista

```
<?  
class analista extends pessoa{  
private $Empresa = null;  
private $Salario = null;  
public static $quantidade = 0;  
function __construct(){  
    self::$quantidade++;  
}  
public function setEmpresa($empresa){  
    $this->Empresa = $empresa;  
}  
}  
?>
```

# Composição

A composição é a ação de acoplar objetos em busca de uma unidade válida e que atenda especificações maiores.

# objeto Empresa

```
<?  
class empresa  
{  
private $nome;  
private $endereco= array('rua'=>"','Numero'=>"','bairro'=>"");  
function setNome($nome){  
    $this->nome = $nome;  
}  
function getNome(){ return $this->nome;}  
}  
?>
```

```
class pessoa
{
public $nome;
private $empresa=null;
function __construct(){ self::$quantidade++;}
function setNome($nome){
    $this->nome = $nome;
}
function getNome(){ return $this->nome;}
function setEmpresa($emp){ $this->empresa = $emp;}
function getNomeEmpresa(){ return $this->empresa->getNome(); }
}

require_once('empresa.php');
$empresa =new empresa();
$Pessoa = new pessoa();
    $empresa->setNome('tempo real');
    $Pessoa->setNome('Zé carioca');
$Pessoa->setEmpresa($empresa);
$Pessoa->getNomeEmpresa();
echo $empresa->getNome();
```

# Usando Empresa e Pessoa

# final (última palavra)

```
“final” class ClasseBase {  
    public function teste() {  
        echo "ClasseBase::teste() chamado\n";  
    }  
  
    final public function maisTeste() {  
        echo "ClasseBase::maisTeste() chamado\n";  
    }  
}  
  
class ClasseFilha extends ClasseBase {  
    public function maisTeste() {  
        echo "ClasseFilha::maisTeste() called\n";  
    }  
}
```