

um objeto;**é definido;**

Um 'objeto' é uma instância de uma 'classe'. Você define uma classe utilizando o trecho de código abaixo;

```
class ClasseBase{
    //aqui entra o código
}
```

é instanciado;

```
$objetoBase = new ClasseBase;
```

Caso queira, você pode atribuir valores às variáveis (ou atributos) da classe através do método construtor.

```
$objetoBase = new ClasseBase($var1, $var2);
```

possui um construtor;

Trata-se de um 'método mágico' executado toda vez que o objeto é instanciado. Se o construtor não estiver definido, o construtor da classe base (se aplicável) é utilizado.

```
function __construct()
{
    //aqui entra o código
}
```

Caso você esteja atribuindo valores às variáveis do objeto, você precisa do seguinte código;

```
function __construct($var1, $var2) {
    //aqui entra o código
}
```

Você pode ainda utilizar o construtor da classe base junto com o construtor da classe atual. Ao definir um construtor na classe base, você pode processar trechos de código e/ou definir e passar argumentos adicionais ao construtor da classe base. Por exemplo;

```
function __construct($var1, $var2) {
    parent::__construct($var1, $var2, $var3);
}
```

possui um destrutor;

Este 'método mágico' é executado toda vez que o objeto é destruído, geralmente quando o script terminar de ser processado.

```
function __destruct() {
    //aqui entra o código
}
```

pode herdar;

Uma classe pode herdar todos os métodos e atributos de uma outra classe. O nome disso é herança e trata-se de um dos principais conceitos da orientação a objetos.

```
class SubClasse extends ClasseBase {
    //aqui entra o código
}
```

pode ser abstrato;

Este tipo de classe não pode ser instanciado diretamente e precisa ser estendido. Este é outro conceito chave na POO; abstração.

```
abstract class ClasseBase {
    //aqui entra o código
}
```

pode ser final;

Esta é a última classe na hierarquia, ela não pode ser estendida, seus atributos e métodos não podem ser herdados.

```
final class SubClasse extends ClasseBase {
    //aqui entra o código
}
```

pode ser copiado;

Um objeto (uma instância de uma classe) pode ser clonado. A cópia passa a ser uma instância completamente nova do objeto, preservando os valores até o momento exato da cópia.

```
$subObjeto = new SubClasse;
$copia_de_subObjeto = clone $subObjeto;
```

pode ser carregado automaticamente;

O 'método mágico' *autoload* é executado toda vez que um novo objeto é instanciado. É muito usado para o *include* dos arquivos da classe, assim, você não precisa fazer isso nos arquivos de sua aplicação. Por exemplo;

```
function __autoload($class) {
    include_once(CLASS_INCLUDE_PATH.'class.'.$class.'.php');
}
```

utilizando instanceof;

O operador de tipo *instanceof* é utilizado para determinar se uma variável é ou não uma instância (um objeto) de uma classe.

```
$subObjeto = new SubClasse;
if ($subObjeto instanceof SubClasse) {
    //isto é verdadeiro
}
```

utilizando __call();

Mais um 'método mágico'. Este é executado quando um método indefinido ou inacessível é chamado. O nome da função e os argumentos devem ser passados como parâmetros.

```
function __call($funcao, $args) {
    echo 'Você tentou executar um método inválido.<br>';
    echo 'Método: '.$funcao.'<br>';
    echo implode(' ', $args);
    exit;
}
```

um método;**é definido;**

Um método é uma função dentro de uma classe.

```
function subMetodo() {
    echo 'Isto é um método.';
}
```

é acessado;

Deve ser acessado da mesma maneira que você utiliza funções. Se desejar, você pode passar variáveis através de parâmetros. Para chamar um método dentro do escopo da

classe, utilize;

```
$this->subMetodo();
```

E para chamar um método fora do escopo da classe, utilize;

```
$subObjeto = new SubClasse;
$resultado = $subObjeto->subMetodo();
```

Da mesma maneira, você também pode chamar métodos de uma classe base. Você os referencia dentro e fora do escopo da classe utilizando;

```
$subObjeto = new SubClasse;
$resultado = $subObjeto->metodoBase();
```

Dentro do escopo da classe há uma maneira alternativa. Este tipo de chamada geralmente é utilizado para métodos estáticos.

```
$resultado = parent::metodoBase();
```

pode ser *abstrato*;

Ao transformar um método em abstrato você está definindo que o mesmo deve estar presente em todas as subclasses. Caso o método possua argumentos, eles também deverão estar presentes nas subclasses. Por exemplo;

```
abstract function metodoObrigatorio($var1, $var2);
abstract function outroMetodoObrigatorio();
```

pode ser *sobrescrito*;

Você pode sobrescrever um método em uma subclasses, uma vez que ele não seja final (veja abaixo). Por exemplo, o método a seguir pode ser definido na subclasses. Este é outro princípio chave da POO; Polimorfismo.

```
function metodoBase() {
    echo 'Este mesmo método existe também na classe base.';
}
```

pode ser *final*;

Um método final não pode ser sobrescrito nas subclasses.

```
final function metodoBase() {
    echo 'Você não pode criar outro método com este nome';
}
```

pode ser público;

Todos os métodos são públicos, caso não tenha sido especificada uma visibilidade. Este é o padrão do PHP. No entanto, é altamente recomendado que você utilize a *keyword* `public`. Mais um conceito chave da POO; Encapsulamento.

```
public function metodoBase() {
    //aqui entra o código
}
```

pode ser privado;

Caso um método seja configurado como privado, ele só poderá ser acessado dentro do escopo da própria classe.

```
private function metodoBase() {
    //aqui entra o código
}
```

pode ser estático;

Um método estático permite que você acesse atributos estáticos dentro de uma classe e nada mais. Você não precisa instanciar um objeto para acessar um método estático.

```
//defina uma constante estática
static public $estatico = 10;
static function metodoEstatico() {
    //para acessar um atributo estático use a keyword self
    echo self::$estatico;
}
```

Agora, para acessar o método fora do escopo da classe, sem instanciar o objeto (onde `SubClasse` é o nome do objeto) utilize;

```
echo SubClasse::metodoEstatico();
```

indução de tipo;

Este recurso permite que você especifique o tipo dos argumentos que serão passados para um método. Você pode querer que seu método só receba parâmetros do tipo array ou uma instância de um objeto, por exemplo.

```
public function metodoBase(SubClasse $subObjeto) {
    echo 'Variável precisa ser uma instância de SubClasse';
}
```

Para especificar que o parâmetro passado seja um array, utilize;

```
public function metodoBase(array $vetor){
    echo 'A variável precisa ser um vetor';
}
```

utilizando `func_num_args()`;

Esta função nativa do PHP retorna o número de argumentos passados para uma função/método. Por exemplo;

```
public function metodoBase($var1, $var2){
    if (func_num_args() != 2){
        return false;
    } else {
        //aqui entra o código
    }
}
```

um atributo;

é definido;

Um atributo nada mais é do que uma variável dentro do escopo da classe. Para definir um atributo utilize:

```
$subVar = True;
```

é acessado;

Para referenciar um atributo ou definir um valor para o mesmo dentro da classe, utilize;

```
//referencia um atributo
print $this->subVar;

//define um valor para o atributo
$this->subVar = False;
```

E para fazer isso fora do escopo da classe, utilize;

```
//referencia um atributo
print $subObjeto->subVar;
```

```
//define o valor de um atributo
$subObjeto->subVar = False;
```

pode ser público;

Assim como os métodos, a visibilidade padrão dos atributos é pública. Isto significa que eles podem ser acessados e modificados em qualquer lugar dos seus aplicativos. No entanto, seguindo as boas práticas de desenvolvimento, utilize sempre a *keyword public*.

```
public $subVar = True;
public $outraVar;
```

pode ser privado;

Um atributo privado só pode ser acessado dentro do escopo da classe. Não pode ser acessado nem mesmo por suas subclasses.

```
private $subVar = True;
private $outraVar;
```

pode ser protegido;

Um atributo protegido só pode ser acessado dentro do escopo da classe e de suas subclasses.

```
protected $subVar = True;
protected $outraVar;
```

pode ser estático;

Você não precisa instanciar um objeto para acessar um atributo estático. Para definir um atributo como estático utilize;

```
static public $estatico = 10;
```

Para acessar o atributo fora do escopo da classe sem instanciar o objeto (onde SubClasse é o nome do objeto) utilize;

```
echo SubClasse::$estatico;
```

pode ser constante;

Aconteça o que acontecer, os atributos constantes nunca sofrem alterações. São geralmente utilizados como referências na passagem de argumentos para um método. Por exemplo, ao invés de passar o valor 1, utilize a constante STATUS_PENDENTE, que tem algum significado, facilitando a leitura do código.

```
const STATUS_PENDENTE = 1;
```

Para acessar a constante você utiliza o nome da classe, da mesma forma que faz quando

acessa um método estático.

```
echo SubClasse::STATUS_PENDENTE;
```

uma **interface**;

é definida;

Uma interface é um 'manual de instruções' ou um 'modelo' de como as classes devem ser utilizadas. É parecido com definir um método abstrato na classe base, mas permite uma flexibilidade maior entre múltiplas classes. É geralmente utilizado para tarefas mais internas do aplicativo, como registro de logs e tratamento de exceções;

```
interface Modelo {
    //aqui entra o código
}
```

pode conter métodos;

Isto significa que os métodos definidos e suas assinaturas devem ser implementados em algum lugar na hierarquia de classes que utiliza a interface.

```
interface Modelo {
    public function trataXML($xml);
}
```

pode conter constantes;

Você pode declarar constantes da interface, que podem ser utilizadas tanto dentro do escopo da classe como fora. Elas não podem ser sobrescritas por nenhuma das classes que utilize a interface.

```
interface Modelo {
    const nome = 'Teste';
}
```

Para acessar estas constantes, utilize;

```
echo Modelo::nome;
```

é implementada;

Ao definir uma classe, você utiliza a *keyword implements* para dizer que a classe precisa seguir o modelo da interface. Mais de uma interface pode ser implementada.

```
class ClasseBase implements Modelo{
    //aqui entra o código
}
```

pode ser *estendida*;

Interfaces também podem herdar métodos e constantes de uma interface base.

```
interface Modelo extends ModeloPrincipal{  
    //aqui entra o código  
}
```

misc;

php5 oop;

<http://www.php.net/manual/en/oo5.intro.php>

versão original acmultimedia

<http://blog.acmultimedia.co.uk/2010/03/object-oriented-programming-using-php-oo/>

traduzido por davi ferreira

<http://www.daviferreira.com/blog/>