Padrões de codificação

Padronização de Código PHP

https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md

Exemplo de Código:

```php
<?php
namespace Vendor\Package;

use FooInterface;
use BarClass as Bar;
use OtherVendor\OtherPackage\BazClass;

class Foo extends Bar implements FooInterface
{
    public function sampleFunction($a, $b = null)
    {
        if ( $a === $b ) {
            bar();
        }
        elseif ($a > $b) {
            $foo->bar($arg1);
        }
        else {
            BazClass::bar($arg2, $arg3);
        }
    }

    final public static function bar()
    {
        // method body
    }
}
```

Todos os padrões aqui descritos estão de acordo com psr-2

http://www.php-fig.org/psr/psr-2/

Geral

1. Código deve seguir o padrão PSR-1
2. Código deverá ser indentado com 4 espaços e não tabulações
3. Namespace, quando presente deverá ser declarado logo após a abertura do PHP “<?php”
4. Deve haver uma linha em branco após a declaração do **namespace**, assim como após a declaração do último **use**, cada **use** deverá ser declarado em uma linha
5. Abertura de chaves para classes deverá ser na linha abaixo da declaração, o mesmo vale para métodos.
6. Abertura de chaves para condições e repetições, deverão estar na mesma linha da declaração
7. Sempre declarar a visibilidade de métodos e variáveis (public, private, protected) ; **abstract** e **final** deverão ser declaradas antes da visibilidade; **static** deverá ser declarada depois da visibilidade;
8. Controles de estrutura deverão ter um espaço antes e um depois da abertura/fechamento de parênteses (“if ($arg < 10) { ...”), sem utilizar espaçamentos após a abertura de parênteses e antes do fechamento(“ if ~~($arg < 10)~~ ”)

9. Todos os arquivos deverão utilizar Unix LF como padrão de fim de linha. Todos os arquivos deverão finalizar com uma linha em branco. A tag “?>” deverá ser omitida em arquivos que contenham apenas código PHP.
10. Palavras chave e constantes do PHP (true, false e null) deverão estar em minúsculo

Classes, Propriedades e Métodos

Extends e implements deverão ser declarados na mesma linha do nome da classe

```
1.  class ClassName extends ParentClass implements \ArrayAccess, \Countable
```

Quando haver mais de uma implementação, e o mesmo ultrapassar o limite de 80 caracteres (ver abaixo), utilizar implementações em linhas separadas, como exemplo:

```
1.  <?php
2.  namespace Vendor\Package;
3.
4.  use FooClass;
5.  use BarClass as Bar;
6.  use OtherVendor\OtherPackage\BazClass;
7.
8.  class ClassName extends ParentClass implements
9.      \ArrayAccess,
10.     \Countable,
11.     \Serializable
12. {
13.     // constants, properties, methods
14. }
```

Formatação de linhas

1. Ideal de 80 caracteres por linha
2. Será permitido até 120 caracteres por linha, configurar style Checker para lançar aviso na IDE.
3. Não poderá haver linhas em branco com espaços
4. Não fazer mais do que um statement por linha (“~~$this->foo(); $this->bar() ;~~“)

Nomenclatura de métodos e variáveis

➔ Classes, métodos, variáveis e constantes sempre devem ser declarados na língua inglesa.

➔ Utilizar nomes informativos e significativos, devem revelar porque ela existe, o que ela faz e como ela deve usada.

```
1.  public $d; // elapsed time in days
2.  //vs
3.  public $elapsedTimeInDays;
```

O nome 'd' não significa nada, tanto que um comentário é necessário. Já o segundo nome nos mostra o que será guardado na variável e em qual unidade.
Nomes que revelam a intenção tornam o código muito mais legível, prazeroso de ler e mais fácil de mudar.

➔ Evite desinformação, tão ruim quanto um nome que não significa nada é um nome que parece significar algo, mas, na verdade, significa outra coisa. Exemplo, não se refira a um grupo de usuários como *userList* se a estrutura de dados utilizada para agrupar estes usuários não for uma Lista, use *userGroup* ou apenas *users*.

➔ Utilize sempre nomes pronunciáveis, não se esqueça que você está escrevendo código para seres humanos lerem. Então utilize nomes que possam ser facilmente pronunciados por quem está lendo.

Exemplo, qual das duas é melhor de entender?

```
1.  class DtaRcrd102 {
2.      private $genymdhms;
3.      private $modymdhms;
4.      private final $pszqint = "102";
5.      /* ... */
6.  }
7.
8.  class Customer {
9.      private $generationTimestamp;
10.     private $modificationTimestamp;;
11.     private final $recordId = "102";
12.     /* ... */
13. }
```

➔ Nome de interfaces não devem ser precedidos om um “I” ou qualquer outro tipo de diferenciação, Uma interface que representa um repositório de usuários deve apenas se chamar *UserRepository* e não *IUserRepository* ou *UserRepositoryInterface.* Nomes mais específicos devem ser colocados nas implementações da interface, neste exemplo poderíamos ter uma implementação chamada *RelationalUserRepository* para uma implementação de repositório que usa um SGBD relacional para tanto. Quando tivermos apenas uma implementação 'default' para a interface este implementação pode possuir o sufixo “Impl”. Neste exemplo teríamos *UserRepositoryImpl.*

➔ Nomes de classes devem ser substantivos ou frases com substantivos como *Customer*, *WikiPage*, *Account*, e *AddressParser.* Evite nomes como *Manager*, *Processor*, *Data* ou *Info*, estes são nomes genéricos e quando usados tentem a dar um nome pouco significativo a classe. O nome de uma classe não deve ser um verbo.

➔ Os nomes de métodos devem ser verbos ou frases que possuam verbos. Devem representar uma ação, como: postPayment, deletePage, ou save. Métodos de acesso, métodos de set e predicados devem ser nomeados com o nome da variável mais um prefixo “get”, “set” ou “is”.

```
1.  public class MyClass {
2.
3.      var name;
4.      var enable;
5.
6.      public getName() {
7.          return $this->name;
8.      }
9.
10.     public setName($name) {
11.         $this->name = $name;
12.     }
13.
14.     public isEnable() {
15.         return $this->enable;
16.     }
17.
18. }
```

➔ Use nomes do domínio da solução, quem está lendo o seu código é um programador, então se você está usando um pattern ou um algoritmo conhecido para resolver um problema, use o nome deste pattern ou algoritmo no nome da classe ou do método. Assim ao ler o nome da classse ou do método um programador já sabe o que esperar, já possui uma ideia de o que aquele código deve fazer.

➔ Use nomes do domínio do problema, quando estivermos trabalhando com algo que não possa ser traduzido para a “linguagem de programador” devemos usar nomes que remetam ao domínio do problema, assim quando um outro programador se deparar com o nosso código ele poderá consultar uma pessoa que entenda do domínio do problema, a partir do código.

Fontes:

1. Clean Code – A Hand Book of Agile Software Craftsmanship. Capítulo 2.
2. https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md
3. http://www.php-fig.org/psr/psr-2/
4. https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md