Guidelines of CodeStyle (PHP)

Version: 0
Updated: 04/06/2020
Author: Josué Silva

1. Introduction

The Coding Style guidelines were developed in order to minimize the different forms of development for programmers. They follow and use PHP-FIG's PSRs (php-fig.org) as a source.

PSRs used:
PSR-1 (Basic Coding Standard): https://www.php-fig.org/psr/psr-1/
PSR-12 (Extended Coding Style): https://www.php-fig.org/psr/psr-12/

Bibliographic reference:
"Clean Code A Handbook of Agile Software Craftsmanship" written by Robert C. Martin and your team.

2. General

2.1. Basic

2.1.1. PSR-1: Tags PHP

Long tags (<? Php?>) Must be used, short tags and / or variations are not allowed.

2.1.2. PSR-1: Character Encoding

Follow the same rules used in the PSR-1.

2.1.3. PSR-1: Side Effects

Follow the same rules used in the PSR-1.

2.1.4. PSR-1: Namespace and Class Names

Follow the same rules used in the PSR-1 com a ressalva do uso de PascalCase nas declarações.

2.1.5. PSR-1: Class Constants, Properties, and Methods

Follow the same rules used in the PSR-1 with the exception of item 4.2 (Properties), in which the camelCase standard must be used.

2.2. Files

Follow the same rules used in the PSR-12.

2.2.1. Vertical Formatting

Following are the suggestions used by Robert C. Martin's Clean Code book. Files should desirably have 200 lines, and a maximum of 500 lines.

2.3. Lines

Follow the same rules used in the PSR-12.

2.4. Identing

Follow the same rules used in the PSR-12

2.5. Keywords and Types

Follow the same rules used in the PSR-12

3. Declare, Namespace e Import

Follow the same rules used in the PSR-12

4. Classes, Properties, and Methods

Follow the same rules used in the PSR-12, Clean Code e S.O.L.I.D. following the guidelines of the book "Clean Code A Handbook of Agile Software Craftsmanship" written by Robert C. Martin and his team.

Whenever starting a new class, the use of parentheses is mandatory even when there are no parameters to be passed to the constructor.

4.1. Extends and Implements

Follow the same rules used in the PSR-12 adding the Open / Closed, Liskov Substitution and Interface Segregation principles of SOLID.

Whenever possible, inheritance and polymorphism should be used to ensure that classes depend on their abstractions and that they respect the principle of being open to extension and closed for modification.

Whenever possible, classes should implement an interface to guarantee the same principle above and still offer the possibility of substitution by derived classes of the same type, thus following the Liskov principle.

There should be segregated interfaces instead of a single interface with multiple methods.

The interfaces must have objective and assertive comments to facilitate the understanding of the purpose of each method. It is mandatory to use this rule following the suggestions in the "Good Comments" section of the book "Clean Code A Handbook of Agile Software Craftsmanship".

4.2. Traits

Follow the same rules used in the PSR-12.

4.3. Properties and Constants

Follow the same rules used in the PSR-12 *adding the "Horizontal Alignment" suggestions from the book "Clean Code A Handbook of Agile Software Craftsmanship".*

Following these suggestions, the use of horizontal alignment for attribution declarations is expressly prohibited.

Forbidden

"

```
public int var1          = 0;
public int variavel2     = 0;
public int var3          = 0;
public int varN1         = 0;
```

"

Allowed

"

```
public int var1 = 0;
public int variavel2 = 0;
public int var3 = 0;
public int varN1 = 0;
```

"

When using PHP versions> 7.4, typing variables is mandatory.

4.4. Methods and Functions

Follow the same rules used in the PSR-12 *adding the "Vertical Ordering" suggestions from the book "Clean Code A Handbook of Agile Software Craftsmanship".*

Each method should always follow SOLID's sole responsibility (SRP) principle. It is strictly prohibited to use methods that have more than one responsibility.

Each method must also follow the SOLID dependency inversion (DIP) principle and, for that reason, the Dependency Injection design pattern should be implemented whenever possible.

The functions called must be in descending order so that the most important functions are always at the highest level of the file, that is, a function that calls another one must be placed first. For example, if method “A” calls method “B”, the code position of method “A” must be before method “B”.

Example:

```
“public function A()
{
        $b = new B();
}

public function B()
{}
“
```

4.5. Method and Function Arguments

Follow the same rules used in the PSR-12 *adding the suggestions for “Function Arguments” from the book “Clean Code A Handbook of Agile Software Craftsmanship”.*

The number of arguments for a function should ideally be zero (none), up to two arguments are allowed, three should be avoided as much as possible and should only be used with a plausible justification. The use of more than three arguments is strictly prohibited.

The use of "Flag Arguments", such as Boolean parameters, is prohibited.

4.6. abstract, final e static

Follow the same rules used in the PSR-12 adding the restriction on the use of static methods, they should only be used using the “Singleton” design pattern.

4.7. Method and Function Calls

Follow the same rules used in the PSR-12.

5. Control Structures

5.1. if, elseif, else

Follow the same rules used in the PSR-12 adding the restriction on the use of elseif and else. The use of these structures is strictly prohibited.

5.2. switch, case

Follow the same rules used in the PSR-12 adding the caveat that the use of this control structure should not be trivialized, that is, it should only be used in situations where there is really no other alternative solution to the problem.

5.3. while, do while

Follow the same rules used in the PSR-12, adding the caveat to avoid using this control structure.

5.4. for

Follow the same rules used in the PSR-12.

5.5. foreach

Follow the same rules used in the PSR-12.

5.6. try, catch, finally

Follow the same rules used in the PSR-12.

6. Operators

6.1. Unary operators

Follow the same rules used in the PSR-12

6.2. Binary operators

Follow the same rules used in the PSR-12 adding the restriction to use the identical operator (===) whenever possible.

6.3. Ternary operators

Follow the same rules used in the PSR-12 by adding the ternary nesting constraint. The insertion of a ternary within another ternary is expressly prohibited.

7. Closures

Follow the same rules used in the PSR-12

8. Anonymous Classes

Follow the same rules used in the PSR-12