PSR-1 Basic Coding Standard

The more basic parts of PHP coding standards are defined in PSR-1. For example:

- Only `<?php` or `<?=` are allowed for PHP tags
- Files must be in UTF-8 without BOM(Byte Order Mark)
- Namespaces and class names must follow the standards in PSR-0 and PSR-4
- Class names must be defined in `UpperCamelCase`
- Class variables must be defined in `UPPER_SNAKE_CASE`
- Method names must be defined in `camelCase`

Standard functions in PHP are defined in snake_case, but in PSR-1, method names must be defined in camelCase. There are no explicit rules for variable and property names, so We can use whichever style We like, but it is noted that they should be consistent. For example, defining normal properties in camelCase and static properties in UpperCamelCase like below:

```
class Something
{
    public $normalPropterty;
    public static $StaticProperty;
}
```

PSR-2 Coding Style Guide

PSR-2 is an extension of the PSR-1 coding standard. Some examples of its contents are:

- We must follow PSR-1 coding standards
- 4 spaces must be used for indents. Using tabs is not allowed
- There is no limit to line length, but it should be under 120 characters, and best if under 80
- We must put a newline before curly braces for classes and methods
- Methods and properties must be defined with `abstract`/`final` first, followed with `public`/`protected`, and finally `static`.
- We must not put a newline before curly braces in conditional statements
- We must not put any spaces before `(` and `)` in conditional statements

CakePHP and Symfony, which will be explained later on in this article, are based on this PSR-2 standard.

Defining Classes

We must put a newline before `{` in class definitions.
Also, `extends` and `implements` must be written on the same line as the class name.

```
class ClassName extends ParentClassName implements Interface1,
Interface2
{
    // Class definition
}
```

If there are too many interfaces for one line, We should put a newline after `implements` and write one interface per line like below.

```
class ClassName extends ParentClassName implements
    Interface1,
    Interface2,
    Interface3,
    Interface4
{
    // Class definition
}
```

Since there are quite a few standards that recommend writing `{` on the same line like `class ClassName {`, this may be a style which We haven't seen before.

Defining Properties

In PSR-2, We must not omit `public`/`protected`/`private` modifiers. In PHP, properties become `public` if these are omitted, but because it is hard to tell if one purposely omitted these modifiers or they just forgot, We should always explicitly write `public`. The `static` keyword comes next. We must not use `var` when defining properties because We can't add any modifiers to `var`.

```
class ClassName
{
    public $property1;
    private $property2;
    public static $staticProperty;
}
```

Additionally, We must not define two or more properties with one statement. We *can* define properties in the way shown below but it is prohibited in PSR-2.

```
class ClassName
{
     private $property1, $property2;
}
```

Methods

Like properties, We must have either one of `public`/`protected`/`private` and `abstract`/`final` comes after them if used. `static` is the last modifier. We must not put any spaces before and after braces, and We must put a newline before curly braces. Also, We must not put any whitespaces before commas in arguments, and We must put one whitespace after them.

```
class ClassName
{
    abstract protected function abstractDoSomething();    final
public static function doSomething($arg1, $arg2, $arg3)
    {
        // ...
    }
}
```

If there are too many arguments, We can put a newline after `(` and write one argument per line. In this case, We can't write multiple

variables on one line. Also, We should write `)` and `{` on the same line, separated by a whitespace.

```
class ClassName
{
    public function doSomething(
        TypeHint $arg1,
        $arg2,
        $arg3,
        $arg4
    ) {
        // ...
    }
}
```

Please note that We must not put a newline before `{` in closures.

```
$closure = function ($a, $b) use ($c) {
    // Body
};
```

Conditional Statements

For conditional statements,

- We must put one whitespace before `(`
- We must not put any whitespaces after `(`
- We must not put any whitespaces before `)`
- We must put one whitespace after `)`

Also, use `elseif` rather than `else if`.

```
if ($condition1) {
    // ...
} elseif ($condition2) {
    // ...
} else {
```

```
    // ...
}
```

Be careful, `else if` and `elseif` are not the complete same things. `elseif` is one statement by itself, but `else if` on the other hand is interpreted as an `if` statement in the `else` of the first `if`.

```
if ($condition1) {
    // ...
} else if ($condition2) {
    // ...
} else {
    // ...
}
```

The syntax above is actually interpreted like below:

```
if ($condition1) {
    // ...
} else {
    if ($condition2) {
        // ...
    } else {
        // ...
    }
}
```

For `switch` statements, `case` statements must be indented once from `switch`, and bodies for the `case`s must be indented once from `case`. When not `break`ing after any kind of operations in `case`, We must write a comment.

```
switch ($condition) {
    case 0:
        echo 'First case, with a break';
        break;
    case 1:
        echo 'Second case, which falls through';
        // no break
    case 2:
    case 3:
    case 4:
```

```
        echo 'Third case, return instead of break';
        return;
    default:
        echo 'Default case';
        break;
}
```