Coding Style Guide

PHP

JavaScript

CSS / SASS / SCSS

Правила нейминга и оформления программного кода

PHP

```php
<?php
namespace FastFood\Page;

use FastFood\Logic\OrderLogic;
use FastFood\Menu\MenuFactory;
use Vendor\Food\FoodOrderInterface;

/**
 * OrderPage Controller
 *
 * Приём заказов с сайта
 * Отправка писем
 */
class OrderPage extends OrderLogic implements FoodOrderInterface {

    const TYPE_MAIN = "order-type-main";
    const TYPE_VEGETARIAN = "order-type-vegetarian";
    const TYPE_VEGAN = "order-type-vegan";
    const TYPE_GLUTENFREE = "order-type-glutenfree";

    /**
     * @var string
     */
    private OtherClass $menu;

    /**
     * Comment about functionality
     */
    public function go() {
        $this->menu = MenuFactory::create($_GET["type"]);
        $this->sampleMethod($this->menu->getName(), $this->menu->getSize());
        $output = $this->longMethodWithManyArgs(
            $this->getDefaultSize(),
            123,
            "example"
        );
```

```php
    }

    /**
     * Comment with parameters PHP Doc
     *
     * @param string $name Call method name
     * @param int $size Call method name
     * @return bool|void true - on success, void or false - on decline
     *
     * @throws Exception If element in array is not an integer
     */
    private function sampleMethod(string $name, int $size = 0) {
        $maxSize = OtherClass::getMaxSize($name);

        $arr = ['a1', 'b2', 'c3'];

        if ($size === 0) {
            $this->menu->fillEmpty(OtherClass::getDefault());
        } elseif ($size > $maxSize) {
            $size = $maxSize;
        } else {
            switch ($size) {
            // family style set
            case OtherClass::SIZE_INDIVIDUAL:
                $this->menu->setStyle("individual");
                break;

            // default style set
            default:
                $this->menu->setStyle(Menu::SIZE_DEFAULT);
                break;
            }
        }

        if (!$theSwitch) {
            return;
        }
        // or
        if (!$theSwitch) return;
```

```
    }

    // Simple comment allow for function
    public function longMethodWithManyArgs(
        int $firstNumber,
        int $secondNumber = 0,
        string $exampleString = null,
        array $options = []
    ) {
        $duplicate = MenuFactory::create( $this->menu->getType() );

        // Allow multiline if v.1
        if ($duplicate->getSize() !== 0
        && $duplicate->name === $exampleString) {
            return MenuFactory::createDefault();
        } elseif ($duplicate-
>getSize() !== 0 && $firstNumber > 0) { // and inline if v.2
            return MenuFactory::createDefault();
        } else {
            return false;
        }

        return $duplicate;
    }
}
```

Структура

1. Добавлять пробел после запятой
2. Добавлять пробелы вокруг операторов сравнения и бинарных операторов (&&, ==, ===, || и т.п.) за исключением оператора конкатенации (.)
3. Унарные операторы (!, --, ++ и т.п.) располагать впритык к переменной (без отступа)
4. Применять точное сравнение в операция сравнения, когда не нужен перебор типов - https://www.php.net/manual/en/language.operators.comparison.php - использование identical приоритетно (===, !==)
5. Добавьте пустую строку перед оператором return, если он не внутри блока (например if)

6. Используйте **return null**; когда надо вернуть **null**, и return; когда надо вернуть **void**.
7. Используйте фигурные скобки { } для блоков во всех случаях кроме inline if, но даже для них рекомендовано использовать скобки.
8. Определяйте один класс на файл, за исключением когда классы не для создания экземпляров извне, которые не относятся к PSR-0 и PSR-4.
9. Объявите наследование класса и все реализованные интерфейсы в одной строке с именем класса;
10. Объявляйте свойства класса перед методами;
11. Сначала объявите открытые методы (public), затем защищенные (protected) и, наконец, закрытые (private). Исключением из этого правила являются конструктор класса и методы setUp () и tearDown () тестов PHPUnit, которые всегда должны быть первыми методами, повышающими читабельность;
12. Объявите все аргументы в одной строке с именем метода / функции, независимо от количества аргументов;
13. Используйте круглые скобки при создании экземпляров классов независимо от количества аргументов, которые имеет конструктор - new MyClass();
14. Строки сообщений об исключениях и ошибках должны быть объединены с использованием sprintf;
15. Не используйте пробелы вокруг [квадратных скобок];
16. Добавьте оператор use для каждого класса, который не является частью глобального пространства имен;
17. Когда теги PHPDoc, такие как @param или @return, включают в себя null и другие типы, всегда помещайте null в конец списка типов.
18. Для объявления и описания массивов использовать квадратные скобки [‘a’,’b’], и не использовать array(‘a’,’b’);
19. Для всех методов, возвращающих единственный тип данных, мы указывает этот тип данных при определении метода, в том числе и для типа void (method():string, method():void и т.п.), в случае возвращения mixed (нескольких типов) значений, эти типы мы описываем в PHP Doc в формате /* @return int|void Comment.
20. Не ставить закрывающий тэг “?>” в конце PHP файлов.
21. Кодировка для всех файлов «UTF-8 без BOM»

Соглашение об именах

1. Используйте camelCase для переменных PHP, имен функций и методов, аргументов (например, $acceptContentTypes, hasSession());
2. Используйте snake_case для параметров конфигурации

3. Используйте пространства имен для всех классов PHP и UpperCamelCase для их имен (например, ConsoleLogger);
4. Константы должны использовать UPPERCASE_UNDERSCORED
5. Добавляйте префикс Abstract для всех абстрактных классов.
6. Суффикс интерфейсов - Interface;
7. Суффикс для трейтов – Trate;
8. Суффикс для исключений – Exception;
9. Используйте UpperCamelCase для именования файлов PHP (например, EnvVarProcessor.php) и lower-case с дефисами для именования шаблонов и веб-ресурсов (section-layout.phtml, index.scss);
10. В комментариях PHP Doc используйте такие типы данных: bool (вместо boolean или Boolean), int (вместо integer или Integer), float (вместо Real или Double).

Скобки

1. Открывающие скобки { для классов и методов всегда с новой строки, для всех остальных блоков - всегда должны быть в конце той же строки.

```
class MyClass
{
        Method()
        {
                if ( a === b ) {
                foreach ( $arr as $item ) {
```

2. Закрывающие скобки всегда должны быть помещены в начале новой строки (})
3. Один уровень отступа должен применяться внутри и только внутри фигурных скобок

Шаблоны .phtml

1. В случаях присваивания переменных и вызовов функций прямо в шаблоне, в частях кода где не используется закрытие php тэга ?>, рекомендовано использовать фигурные скобки { }. Во всех остальных случаях – альтернативный синтаксис или фигурные скобки.
2. При использовании альтернативного синтаксиса точку с запятой в конце блока ставить не обязательно <? endif ?>, <? endforeach ?>, <? endfor ?>

3. Запрещается использовать альтернативный синтаксис в «одну строку», когда в одной строке и открытие и закрытие блока.
4. Запрещается использовать альтернативный синтаксис вне блоков содержащих чистый html.
5. Для вывода текста рекомендовано использовать тэг **<?=** вместо **<?php echo** или **<? echo**.'
6. Рекомендовано использовать short tags синтаксис <? ?>, вместо <?php ?>.

Пример применения блоков и стилей в шаблоне

```
<?php
foreach ($arr as $item) {
    $item->a = $b;
}

?>

<div>
    <div> <!-- recommended -->
    <? foreach ($arr as $item) { ?>
        <div><?=$item->title?></div>
    <? } ?>
    </div>

    <div> <!-- good -->
    <? foreach ($arr as $item): ?>
        <div><?=$item->title?></div>
    <? endforeach ?>
    </div>

    <div><!-- recommended -->
    <? foreach ($arr as $item) {
        $item->iterator++;
        $item->renderPart();
    } ?>

    <div> <!-- VERY BAD -->
    <? foreach ($arr as $item):
        $item->iterator++;
        $item->renderPart();
    endforeach ?>
    </div>
</div>
```

Документация

1. Добавьте блоки PHPDoc для всех классов, методов и функций;
2. Сгруппируйте аннотации вместе, чтобы аннотации одного типа сразу следовали друг за другом, а аннотации другого типа разделялись одной пустой строкой;
3. Опустите тег @return, если метод ничего не возвращает;
4. Аннотации @package и @subpackage не используются;
5. Не вставляйте блоки PHPDoc в одну строку, даже если они содержат только один тег;
6. Для описания простых функций можно использовать вместо PHP Doc – простой комментарий - //comment.

CSS

Пример SCSS:

```scss
/**
 SCSS
**/

.block { /* Блок */
    font-size: 13px;
    font-style: italic;
    font-weight:bold;

    position: absolute;
    left:10px;
    top:10px;
    right:50%;
    bottom:30px;

    border:1px solid red;
    background-color: #fff;

    &_wrapper { /* Элемент */
        padding:10px;
    }
```

```
    &_list { /* Элемент */
        margin-top:10px;

        & > li {
            margin-left:10px;
        }
    }

    &_input-form { /* Элемент */
        border-radius:10px;

        &_blue { /* Модификатор */
            background-color: blue;
        }

        &_enabled {
            display:block;
        }
    }
}
```

Пример CSS (откомпилированный пример выше):

```
/**
 CSS
**/

@charset "UTF-8";

.block {
  /* Блок */
    font-size: 13px;
    font-style: italic;
    font-weight: bold;
    position: absolute;
    left: 10px;
    top: 10px;
    right: 50%;
    bottom: 30px;
    border: 1px solid red;
```

```
    background-color: #fff;
}

.block_wrapper {
    /* Элемент */
    padding: 10px;
}

.block_list {
    /* Элемент */
    margin-top: 10px;
}

.block_list > li {
    margin-left: 10px;
}

.block_input-form {
    /* Элемент */
    border-radius: 10px;
}

.block_input-form_blue {
      /* Модификатор */
    background-color: blue;
}

.block_input-form_enabled {
    display: block;
}
```

БЭМ (Блок, Элемент, Модификатор) — компонентный подход к веб-разработке. В его основе лежит принцип разделения интерфейса на независимые блоки. Он позволяет легко и быстро разрабатывать интерфейсы любой сложности и повторно использовать существующий код, избегая «Copy-Paste».

Мы будем использовать упрощенную версию БЭМ – нашу, Си-Рейтовскую!

Блок

Функционально независимый компонент страницы, который может быть повторно использован.

Название блока характеризует смысл («что это?» — «меню»: menu, «кнопка»: button), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).

```
<!-- Верно. Семантически осмысленный блок `error` -->
<div class="error"></div>
<div class="error-message"></div>
<div class="error-message-block"></div>

<!-- Неверно. Описывается внешний вид -->
<div class="red-text"></div>
```

Блок не должен влиять на свое окружение, т. е. блоку не следует задавать внешнюю геометрию (в виде отступов, границ, влияющих на размеры) и позиционирование.

Вложенность

- Блоки можно вкладывать друг в друга.
- Допустима любая вложенность блоков.

```
<!-- Блок `header` -->
<header class="header">
    <!-- Вложенный блок `logo` -->
    <div class="logo"></div>

    <!-- Вложенный блок `search-form` -->
    <form class="search-form"></form>
</header>
```

Элемент

Составная часть блока, которая не может использоваться в отрыве от него.

Особенности:

- Название элемента характеризует смысл («что это?» — «пункт»: item, «текст»: text), а не состояние («какой, как выглядит?» — «красный»: red, «большой»: big).
- Структура полного имени элемента соответствует схеме: имя-блока_имя-элемента. Имя элемента отделяется от имени блока одним подчеркиванием (_).

```
<!-- Блок `search-form` -->
<form class="search-form">
    <!-- Элемент `input` блока `search-form` -->
    <input class="search-form_input">

    <!-- Элемент `button` блока `search-form` -->
    <button class="search-form_button">Найти</button>
</form>
```

Принципы работы с элементами

Вложенность

Элементы можно вкладывать друг в друга.

Допустима любая вложенность элементов.

Элемент — может быть частью блока или быть частью другого элемента.

Принадлежность

Элемент — всегда часть блока и не должен использоваться отдельно от него.

Необязательность

Элемент — необязательный компонент блока. Не у всех блоков должны быть элементы.

Когда создавать блок, когда — элемент?

Создавайте блок

Если фрагмент кода может использоваться повторно и не зависит от реализации других компонентов страницы.

Создавайте элемент

Если фрагмент кода не может использоваться самостоятельно, без родительской сущности (блока).

Исключение составляют элементы, реализация которых для упрощения разработки требует разделения на более мелкие части — подэлементы.

Модификаторы

Сущность, определяющая внешний вид, состояние или поведение блока либо элемента.

Особенности:

Название модификатора характеризует внешний вид («какой размер?», «какая тема?» и т. п. — «размер»: size-s, «тема»: theme-islands), состояние («чем отличается от прочих?» — «отключен»: disabled, «фокусированный»: focused) и поведение («как ведет себя?», «как взаимодействует с пользователем?» — «направление»: directions-left-top).

Имя модификатора отделяется от имени блока или элемента одним подчеркиванием (_).

```
<!-- Блок `search-form` -->
<form class="search-form">
    <!-- Элемент `input` блока `search-form` -->
```

```
    <input class="search-form_input search-form_input_enabled">

    <!-- Элемент `button` блока `search-form` -->
    <button class="search-form_button_blue">Найти</button>
</form>
```

Формирование название класса или ID элемента

1. Первым указывается имя блока или нескольких блоков
2. Вторым указывается имя элемента или элементов
3. Последним идёт список модификаторов

Блок_Элемент1_ЭлементN_Модификатор1_МодификаторN

Блок1_Элемент1_Блок2_Элемент2_Модификатор1_Модификатор2

circles_wrapper блок_элемент

circles_wrapper_selected блок_элемент_модификатор

circles_input блок_элемент

circles_input_error блок_элемент_модификатор

circles_list блок_элемент

circles_list_item блок_элемент_элемент или блок_блок_элемент

circles_list_item_blue блок_элемент_элемент_модификатор

circles_list_item_blue_bold блок_элемент_элемент_модификатор_модификатор

cicrles_blue блок_модификатор

Соглашение об именах

1. В CSS названиях разрешается использовать маленькие (прописные) латинские буквы, цифры, - и _.

2. Название блока, элемента и модификатора может содержать только латинские буквы, цифры и - (дефис). Пример: block, element-name, modificator-is-bold

3. Разделителем между блоками, элементами и модификаторами является _ (нижнее подчеркивание). Пример: block_element_modificator.

JavaScript

Максимальная разрешенная версия для использования: ECMAScript 2015 (ES6).

Для продакшена компилируется в es5 используя Babel.

Разрешенные библиотеки: React, JQuery (deprecated минимизируем использование).

Code Style Guide:

- English - https://github.com/airbnb/javascript
- Русский - https://github.com/leonidlebedev/javascript-airbnb

Code Style Guide для React`a:

- English - https://github.com/airbnb/javascript/tree/master/react
- Русский - https://github.com/leonidlebedev/javascript-airbnb/tree/master/react