Стандарты разработки

Table of contents

Базовые принципы

Именование переменных

В основе именования лежит венгерская нотация. Именование переменных должно быть осмысленным,
чтобы по имени было стало понятно назначение **переменной** или **метода**.

Переменные

CArticle, COffer, CProduct - классы
oElement, oSection, oCatalog - объекты
arFields, sPath, iIndex, fPrice, bExists - промежуточные переменные для выстраивания логики
ID, CATALOG, PROPERTY, INDEX - входные или выходные переменные
если принять любой метод, компонент или модуль за чёрный ящик, то переменные такого типа используются для передачи в него или в качестве результата

Методы

GetList, Add, Update, Delete, CreateElementByID - основные публичные public методы, которые используются в классах
_getPrice, _loadStack, _rebaseOnto - вспомогательные приватные private методы, используются внутри логики классов

Документирование кода

Описанием требуется снабжать ключевые входые точки, а также логические блоки, условия, циклы.

Методы

@arParams, @return - входые и выходные переменные

//----------начало логического
*//***----------конец логического*

//комментарий к сущности, методу, циклу или др.

пример:

```
/*
	добавить заявку
	@arParams
		COIN_ID - монета из каталога
		FIELDS - параметры заявки (форма)
	@return
		PRODUCT_ID - идентификатор продукта
*/
public function Add($COIN_ID,$FIELDS)
{
	global $USER;
```

```
        //----------------------добавить элемент
        //поля элемента
        $arFieldsElement = Array(
                "IBLOCK_ID" => self::IBLOCK_ID,
                "NAME" => "Заявка ".date("d.m.Y")
        );
        //свойства элемента
        $arFieldsElement["PROPERTY_VALUES"]=Array(
                "USER_TO" => $USER->GetID(),
                "TYPE" => $this->arEnum["TYPE"][ $FIELDS["TYPE"] ],
                "STATUS" => $this->arEnum["STATUS"]["ACT"],
                "CML2_LINK" => $COIN_ID
        );
        $oElement = new \CIBlockElement();
        $PRODUCT_ID = $oElement->Add($arFieldsElement);
        //***----------------------добавить элемент

        //----------------------добавить продукт
        //поля каталога
        $arFieldsCatalog = Array(
                "ID" => $PRODUCT_ID,
                "QUANTITY" => $FIELDS["LOTS_QUANTITY"]*$FIELDS["UF_LOT_SIZE"]

        );
        \CCatalogProduct::Add($arFieldsCatalog);
        //***----------------------добавить продукт

        //----------------------добавить цены
        //поля цены
        $arPrice = Array(
                "PRODUCT_ID" => $PRODUCT_ID,
                "CATALOG_GROUP_ID" => BASE_PRICE_CODE,
                "PRICE" => $FIELDS["PRICE"],
                "CURRENCY" => "RUB",
                "QUANTITY_FROM" => false,
                "QUANTITY_TO" => false
        );
        \CPrice::Add($arPrice);
        //***----------------------добавить цены

        return $PRODUCT_ID;
}
```

Компоненты (скрипты)

компонент или скрипт также является точкой входа, в начале кода нужно добавить краткое описание решаемой задачи или назначение компонента

```
/*
        пройти таблицу b_file с сортировкой по NAME
        SELECT ORIGINAL_NAME,HEIGHT,WIDTH,FILE_SIZE,COUNT(ORIGINAL_NAME) as CNT
FROM `b_file` GROUP BY ORIGINAL_NAME,FILE_SIZE HAVING CNT>1
        //----
        нашли идентичные файлы, теперь нужно перезаписать SUBDIR значением 1го из списка,
остальные файлы удалить через unlink
        SELECT ID,SUBDIR,ORIGINAL_NAME,HEIGHT,WIDTH,FILE_SIZE FROM `b_file` WHERE
ORIGINAL_NAME='1_150.jpg' AND FILE_SIZE='26539' ORDER BY ID ASC
*/
```

ООП

Что такое "объектно-ориентированный" подход? Он подразумевают, что какая-то деятельность направлена на определенный объект. Объектами в жизни выступают все окружающие нас предметы: автомобили, книги, стол, часы.

Рассмотрим такой объект, как телевизор. Внутри этого объекта находятся множество других объектов: микросхемы, провода, электронно-лучевая трубка и так далее. Но при взаимодействии с телевизором мы об этом даже и не задумываемся. В этом заключается первый принцип ООП - инкапсуляция. Инкапсуляция тесно связана со вторым принципом - абстракцией, без которой не возможно сокрытие кода.

Мы также знаем, что, нажав на определенную кнопку, мы включим телевизор, а удерживая другую - увеличим или уменьшим громкость. При этом от объекта мы получаем только результат его работы, не задумываясь о его внутренних процессах. Это составляет второй принцип - абстракцию. Если немного углубиться, то с помощью абстракции создаются шаблоны для управления объектами. Очень простой пример – очень часто в жизни вы встречаетесь с плюсом и минусом. Причём вы чётко знаете что произойдёт с объектом, если вы примените то или другое действие. В нашем примере это была громкость. Но сколько разных объектов подвержены этим же операциям. Будь то оценка в школе, полярность тока, сложение в математике, понижение и повышение температуры. “+” и “-” это и есть шаблонные операции.

Наконец, третий принцип, составляющий парадигму ООП, называется наследованием (полиморфизмом). Он заключается в том, что, например, цветной телевизор произошел от черно-белого, а телевизор с плазменным экраном - от обыкновенного. При этом каждый потомок наследовал свойства и функции предшественника, дополняя их своими, качественно новыми. Наследование позволяет расширить возможности объекта, не создавая при этом новый объект с нуля. В переводе на более сложный язык полиморфизм - выполнение различного кода классов-потомков через обращение к интерфейсу, или, если такой конструкции в языке нет, к классу-родителю.

Стоит добавить что класс, который состоит более, чем из 1000 строк кода – уже плохой, следует в этом случает обратиться к наследованию. Для реализации на PHP можно брать большие ограничения, что связано со спецификой конкретного языка.

MVC

В шаблоне MVC, как следует из названия, есть три основных компонента: **Модель**, **Представление**, и **Контроллер**.

Представление (вид) отвечает за отображение информации, поступающей из системы или в систему.

Модель предоставляет данные и реагирует на команды контроллера, изменяя свое состояние. Является «сутью» системы и отвечает за непосредственные алгоритмы, расчёты и тому подобное внутреннее устройство системы. Так называемый черный ящик.

Контроллер интерпретирует действия пользователя, оповещая модель о необходимости изменений. Является связующим звеном между «**представлением**» и **моделью** системы, посредством которого и существует возможность произвести разделение между ними. **Контроллер** получает данные от пользователя и передаёт их в «модель». Кроме того, он получает сообщения от **модели**, и передаёт их в **представление**.

Применительно к интернет-приложениям бытует мнение, что части **контроллер** и **представление** объединены, потому что за отображение и одновременно за ввод информации отвечает браузер. С этим можно согласиться, а можно не соглашаться и выделить-таки контроллер в отдельную часть, что мы и сделаем.

Итак, условимся:

Представление. Модуль вывода информации. Это может быть шаблонизатор или что-либо подобное, цель которого является только в выводе информации в виде HTML на основе каких-либо готовых данных.

Контроллер. Модуль управления вводом и выводом данных. Данный модуль должен следить за переданными в систему данными (через форму, строку запроса, cookie или любым другим способом) и на основе введённых данных решить:

· Передавать ли их в **модель**

· Вывести сообщение об ошибке и запросить повторный ввод (заставить модуль представление обновить страницу с учётом изменившихся условий)

Кроме того, **контроллер** обязан определять тип данных, полученных от модели (есть ли это готовый результат, отсутствие оного, либо сообщение об ошибке) и передавать информацию в модуль представления.

Модель. Модуль, отвечающий за непосредственный расчёт чего-либо на основе полученных от пользователя данных. Результат, полученный этим модулем, должен быть передан в контроллер, и не должен содержать ничего, относящегося к непосредственному выводу (то есть должен быть представлен во внутреннем формате приложения).

Проще говоря – разделяйте логику и шаблонизаторы.

Компонент 2.0

DRY

Этот принцип лежит в основе любой оптимизации кода. Для этого были придуманы циклы, процедуры, функции. Данный пункт не требует пояснения, основной постулат здесь **«не используйте copy-paste»**.

1с-Битрикс

Структура файлов

Javascript

/js/ - файлы JavaScript

../objects/ - объекты (псевдоклассы), используемые в дальнейшем в компонентах

../jquery/, *../bootstrap/*, *../**[LIBRARY_JS]**/* - если используются сторонние библиотеки, все файлы, которые имеют отношение к данной библиотеке должны лежать здесь

*/local/components/**[NAMESPACE]**/**[COMPONENT_NAME]**/templates/**[TEMPLATE_NAME]**/script.js* - путь для js файла, работающий в логике конкретного компонента

LIBRARY_JS - название сторонней библиотеки, например jquery
NAMESPACE - пространство имён компонентов проекта, например dquad (обычно одно на проект)
COMPONENT_NAME - название компонента, например catalog.section
TEMPLATE_NAME - название шаблона, например .default (по умолчанию)

HTML

/_html/ - свёрстанные макеты, которые в дальнейшем будут интегрированы в логику
../_html/index.html - список доступных страниц (обычно с указанием номера раздела в соответсвии с ТЗ)
../_html/template/ - повторяющиеся блоки страниц (соблюдения принципа DRY)

CSS
/css/ - файлы стилей css
../css/style.css - основной файл стилей, кастомизация шаблона
../css/into.css - дополнительный файл стилей, кастомизация страниц (с обозначением разделов комментариями)
../css/[**VIEW_SIZE**].css - кастомизация адаптивной вёрстки для альтернативных разрешений

*/local/components/[**NAMESPACE**]/[**COMPONENT_NAME**]/templates/[**TEMPLATE_NAME**]/style.css* - дополнительный файлы стилей, подключаемый автоматически
если требуется внести коррективы в вёрстку (используется в качестве заплаток, в общем случае не используется)

VIEW_SIZE - адаптивное разрешение, например 540, 768, 990, 1170

PHP

*/local/templates/[**SITE_TEMPLATE**]* - основные шаблоны сайта
../.default/include/ - повторяющиеся блоки логики, использующие в подключении несколько компонентов или наоборот очень простую логику
*../.default/components/bitrix/[**COMPONENT_NAME**]/* - шаблоны, использующие компоненты ядра Битрикс, требующие только адаптации шаблона

/local/php_interface/ - папка для хранения собственных классов, функции, констант, событий
../init.php - константы сайта
*../[**SITE_ID**]/init.php* - константы сайта (в случае многосайтовости)
../_event_handlers.php - обработчики событий (подчинённый файлу init.php)
/include/classes/ - классы в порядке дерева namespace (для старой версии */include/dquad_classes/*)
например для класса **\DQuad\Catalog\CBid** путь будет */dquad/catalog/CBid.php*
/include/functions/ - функции (по возможности использовать классы)
/include/functions/index.php - подключение групп функции
*/include/functions/_[**GROUP_FUNCTIONS**].php* - группа функции одного назначения

/local/components/dquad/ - кастомизированные или собственные компоненты, на базе которых строится сайт
../component.php - базовый файл логики, модель (MVC)
или
../class.php - базовый класс, наследуется от CBitrixComponent и должен содержать определенный набор методов (см. Битрикс d7, с 2014г)
*../_[**LINE_BLOCK**].php* - подчинённый файлу component.php блок, содержит линейный логический блок работы с данными, например выборка цен
*../templates/[**TEMPLATE_NAME**]/template.php* - шаблон представления, обычно содержит простейшие циклы, условия и html код
*../_[**TPL_BLOCK**].php* - подчинённый файлу template.php блок, введён для лучше читаемости и семантики
../ajax/ - дополнительные скрипты, которые нужны компоненту для реализации логики ajax

/tools/ - временные и тестовые скрипты, не имеющие правильного оформления

SITE_TEMPLATE - символьный код шаблона, например **base, into, main**
SITE_ID - символьный код сайта (см. Битрикс)
GROUP_FUNCTIONS - группы функций одного назначения, например **format**
LINE_BLOCK - название логического блока, например **prices**
TPL_BLOCK - название логического блока, например **offerList**

Примечание: папка /local/ используется с 2014г, в старых проектах вместо неё используется папка /bitrix/ (подробнее)

GIT

Ревизии

1. Необходимо подготовить копию для размещения на сервере
    исключить из корневой директории
    /upload/

    помимо файлов вёрстки в репозиторий попадает папка /local/
    файлы
    .htaccess
    urlrewrite.php

2. Исключённые директории с сохранением иерархии сохранить отдельно предварительно

3. При работе с проектом необходимо делать ревизию после завершения логического блока работ.

4. При поднятии ключевой ревизии необходимо добавлять комментарии по след. шаблону:
*[тип события] Remove/Add/Update/Fix
*[цель] В свободном виде написать что были исправлено и какой получился результат
*[время] Примерное время на разработку до текущего обновления
[недоработки] Что по твоему мнению разработано в модуле некачественно и хотелось бы переделать, однако времени оказалось недостаточно
[SQL] В свободном виде написать какие изменения были внесены в базу данных

Пример:

```
Add:        dquad.catalog.section
Goal:       разработан модуль каталога, который отображает список элементов с постраничной навигацией, просмотр доступен по адресу http://vlando.ru/catalog/951/
Resource:   4H
Flaw:       Не был переписан модуль компонента битрикс, который съедает много ресурсов и выполняет лишние запросы, критично при кол-ве элементов > 5 на странице, не интегрирована вёрстка, необходимо привлечь верстальщика
SQL:        добавлены свойства в IBLOCK=10, PRP = [68,69,70,85], удалены лишние PRP [50-55]
```

Пример (более простая итерация):

```
Update:         CSS/*
Goal:       неправильные отступы в каталоге в ie6, не подгружается png
Resource:   20m
```

Примечание: если ревизия промежуточная, комментарий можно добавлять в сокращенном виде, например

```
FIX: исправлены ошибки подключения js
```

HTML

Общие требования:
- Chrome, FF, Opera, Safari (версии не старше 3лет)
- валидность w3c
- отсутсвие inline подстановок
- адаптивность (ключевые точки 1170, 990, 768, 540, 320)
- комментарии блоков вида <!-- name --> <!--/ name -->, аналогично в css, js
- спрайты
- Все что можно сделать без Javascript, делать без него.
- jquery (если вам требуется)
- @font-face шрифты
- комментирование кода (не менее 10% от объёма должен составлять текст комментариев)
(удобства интеграции)
- строгое выравнивание (автовыравнивание отступов)
- локаничная инкапсуляция блоков для интеграции (программист не должен собирать будущий фильтр из 2х и более кусков, разделенным, например постраничной навигацией)
- SSI(или альтернатива) для соблюдения принципа DRY, блоки не должны повторяться copy-paste
- контроль версий git (аккаунт выдаётся)
- использование абсолютных путей относительно корня
- максимальное кол-во строк (css, js) 500

Расположение файлов:
- html файлы должны лежать в одной папке /_html/,
в файле index.html перечислены все доступные страницы
то же самое касается css, images или js, у каждого своя папка
у меня это папки /js/, /css/
- если вы готовите шаблон для компонента, у Битрикс есть стандарт расположения файлов
\local\components\[namespace]\[component.name]\templates\[template.name]\
 - style.css
 - script.js
которые будут подключены автоматически

т.о. если кому-то потребуется поправить вёрстку, у него будет только 2 варианта где искать код - папка шаблона или базовая папка /css/

пример вёрстки на соответствие текущим стандартам http://coinclub.online/_html/main.html