ООП. Модуль 1. Объекты. Базовые понятия

Введение

Добрый день! Меня зовут <имя фамилия>. В текущем курсе мы рассмотрим тему посвященную ООП. Разберем базовые понятия ООП. Основные его принципы. Познакомимся с объектами в языке PHP. Узнаем как их создавать и использовать.

Общие сведения об ООП

По мере роста функционала вашего приложения повышается и его сложность. Со временем появляется потребность в более понятной организации кода вашего приложения, написанного в процедурном стиле. На помощь приходит методология объектно-ориентированного программирования. Сокращенно “ООП”. Данная методология подразумевает представление системы как совокупность объектов, каждый из которых является экземпляром определенного класса. Классы могут образовывать иерархию наследования.

ООП подразумевает реализацию программы на основе абстрактного представления ее основных частей в виде классов и объектов. Это дает возможность разрабатывать сложные программные комплексы большой командой разработчиков. ООП предполагает проектирование системы в целом, создание отдельных компонентов и объединение их в конечный продукт. При этом разные компоненты могут быть реализованы разными командами разработчиков.

Таким образом при проектировании системы нужно ответить на два вопроса:

- из каких частей состоит система
- какая зона ответственности у каждой части

После того как вы определили набор частей системы и разграничили их зоны ответственности, в дальнейшем можно смотреть на нее оперируя ее отдельными частями, не задумываясь об их конкретной реализации. Это повышает уровень абстракции. Вы подходите к рассмотрению системе как к взаимодействию ее отдельных частей. При данном подходе можно сконцентрироваться на реализации функциональности системы как инструмента достижения конкретных бизнес целей.

Основные понятия объекта и класса

Класс - это способ описания сущности (сущность = объект), иначе говоря, класс это своего рода шаблон, который описывает состояние (свойства) и поведение (методы) объектов.

Объект - это экземпляр класса, который содержит уже конкретные значения состояний (свойств) этого класса, и также может выполнять все методы класса.

Если привести аналогию из реальной жизни, то чтобы создать двигатель автомобиля нужно понять из каких деталей он будет состоять и как где какая деталь будет располагаться. Для каждой детали делается чертеж, по сути шаблон для ее производства. Это и есть наши классы. Далее по чертежам изготавливается детали. Это наши объекты. Как по одному чертежу вы можете создать несколько деталей, так в ООП вы можете на основе одного описания класса создать множество одинаковых объектов.

Так что можно сказать, что классы - это чертежи наших деталей. А объекты - это детали, созданные на основе чертежей.

Давайте опишем наш первый объект.

Текст	**Код**
Для объявления класса используется зарезервированное слово “class”. Напишем class и далее должно идти название нашего класса. Давайте создадим класс сообщений. Напишем название класса Cat. После названия класса идет описание самого класса, его функционала. Это называется телом класса. Тело класса заключается в фигурные скобки. согласно стандартам стиля кода, при объявлении класса открывающая фигурная скобка должна размещаться на следующей строке.	<?php class Cat { // Тело класса }
Давайте объявим еще один класс. Важно понимать, что в вашей системе не должно быть объявление нескольких классов с одним названием. В таком случае ваш код просто не запустится.	class User { // Тело класса User }
Мы объявили наши классы, это хорошо. Но, как мы помним, это только чертежи. Для использования нам нужно создать объекты из этих классов. Для создания объекта используется в языке php используется зарезервированная конструкция new и название класса. Наш объект не может существовать сам по себе, поэтому мы его положим в переменную. Для этого	$cat = new Cat();

объявим переменную $cat и присвоим ей значение новым объектом. Пишем new, название нашего класса Cat и круглые скобки. Скобки используются для передачи параметров конструктору, о котором поговорим позже. В данном случае объект создается без параметров.	
Если у нас класс не требует параметров, то скобки можно опустить.	`$cat = new Cat;`
Выведем информацию о нашем объекте используя функцию var_dump. Напишем var_dump и в параметрах функции передадим переменную с нашим объектом.	`var_dump($cat); // Cat Object ()`
запустим наш скрипт. Нам вывелась информация о нашем котике Cat. В данном у него нет никаких особенностей.	`object(Cat)#1 (0) {}`
Давайте создадим объект класса User. И положим его в переменную $user. Пишем $user равно new User.	`$user = new User();`
Выведем информацию через var_dump. Пишем var_dump. В параметрах наша переменная $user, содержащая наш объект User	`var_dump($user); // User Object ()`
Повторно выполним скрипт	`object(Cat)#1 (0) {}` `object(User)#2 (0) {}`

Когда у нас есть описание класса, то мы можем создавать неограниченное количество его экземпляров. Т.е. объектов. Давайте создадим еще одного кота. Положим его в переменную $cat2. Также воспользуемся функцией var_dump для отображения информации о нашем объекте.	$cat2 = new Cat(); *var_dump*($cat2);
запустим наш пример. Как мы видим, идентификатор нашего первого объекта и второго отличаются. Значит, что это два разных объекта. Получается что мы создали два разных объекта на основе одного описания класса.	$ php example.php object(Cat)#1 (0) {} object(User)#2 (0) {} object(Cat)#3 (0) {}
Давайте попробуем скопировать наш объект в новую переменную. Назовем нашу новую переменную $cat3. И присвоим новой переменной нашего первого кота $cat. Выведем информацию о нашей новой переменной через var_dump.	$cat3 = $cat; *var_dump*($cat3);
У объекта в переменной $cat идентификатор идентичен тому, что и у нашего первого объекта $message. Почему же так получилось? Это от того, что в версиях php начиная с 5 объекты копируются по ссылке. Таким образом переменные $cat и $cat3 ссылаются на один и	object(Cat)#1 (0) {} object(User)#2 (0) {} object(Cat)#3 (0) {} object(Cat)#1 (0) {}

тот же объект созданный в начале при инициализации переменной $cat, т.е. они указывают на одного и того же кота. При инициализации переменной $cat3 мы не скопировали сам объект, а только ссылку на него.

Чуть подробнее про ссылочные типы. Существует два типа переменных ссылочные, и указывающие на значение. Первые содержат ссылку на область памяти, в которой лежат наши данные, а вторая указывает прямо на наши данные. В чем разница.

Текст	**Код**
Когда мы оперируем с переменной, которая указывает на значение, то при выполнении операций присваивания, передачи в качестве параметра функции программа копирует значение из переменной и оперирует с копией, в данном случае для переменной $b будет выделена отдельная ячейка в памяти, и в нее будет помещено значение.	<?php $a = 5; $b = $a;
Но когда мы выполняем операции с ссылочной переменной - тогда взаимодействие происходит с объектом, на который указывает наша переменная. (Не обращаем внимание на $a->value - об этом чуть ниже) Скрипт выведет 2 2, Потому что при присваивании $b = $a - новый объект создан не будет, в переменную $b поместиться ссылка, указывающая на тот же объект, что и переменная $a, соответственно все изменения с $b - это изменения для $a.	$a = new StdClass(); $a->value = 1; $b = $a; $b->value++; var_dump($a->value, $b->value); // 2 2

Отсюда следует вывод - передавая переменные содержащие объекты в различные функции и методы программы, или других классов, будет передана ссылка на исходный объект, а следовательно и изменения будут происходить с исходным объектом, а не локальной копией внутри функции.

В данной главе мы рассмотрели простые примеры описания классов и создания объектов. Но пока наши объекты являются пустыми. Они не содержат никаких данных. Для того, чтобы наши объекты были полезны для нашей системы нам нужно добавить в них функциональность. Об этом будет наша следующая глава.

Из чего состоит класс (свойства, методы, константы)

Наши классы в данный момент не умеют ничего делать, т.к. они пустые. Для того чтобы наделить их определенным функционалом нужно добавить в них основные функциональные компоненты. Класс может содержать:

- Свойства;
- Методы;
- Константы;

Давайте рассмотрим каждый компонент отдельно.

Свойства

Каждый объект может хранить в себе определенный набор информации, который в него заложен на этапе объявления класса. Свойства - это как бы прилагательные класса, описывающие какой будет объект. Также можно встретить альтернативные названия “атрибуты”, “поля”. Каждый экземпляр класса, т.е. объект может содержать различные значения для свойств. Другими словами это текущее состояние объекта. Свойствами класса являются его внутренние переменные.

Текст	Код
Добавим новое свойство нашему Коту: кличку, для этого и объявим внутри него свойство $name. Сразу же присвоим ему значение типа “string” с текстом “Вы забыли дать мне кличку”. Создадим экземпляр объекта и присвоим ссылку на него переменной $message. Используем функцию var_dump для вывода информации о нашем объекте.	<?php class Cat { public $name = 'Вы забыли дать мне кличку'; } $cat = new Cat(); *var_dump*($cat);
Как мы видим, у нашего объекта появилось свойство, которое содержит наше сообщение.	object(Cat)#1 (1) { ["name"]=>string(42) "Вы забыли дать мне кличку" }
Для доступа к свойствам объекта используется конструкция “->”. Давайте выведем имя нашего котика. Используем функцию echo для вывода на экран информации. Далее пишем переменную, содержащую ссылку на наш объект, конструкцию доступа к свойствам и название нашего свойства. Добавим к	echo $cat->name; Вы забыли дать мне кличку

нашему сообщению строку с символами переноса строк.	
Через конструкцию доступа к свойствам объекта можно изменять значение свойств объекта. Дадим нашему котику имя. Для этого пишем название нашей переменной, конструкцию доступа и название свойства. Далее конструкцию присваивания и новую строку “Tom”.	$cat>name = "Tom"; echo $cat>name; Tom

Правилом хорошего тона считается объявление свойств класса в самом начале. При чем должны быть объявлены все возможные свойства, если не предполагается их динамическое добавление в процессе использования объекта. Но динамическое объявление считается плохой практикой, т.к. другим людям, просматривающим ваш код, будет сложнее понять как он работает. Не используйте динамическое добавление свойств, если это не требуется.

Для определения свойств можно использовать следующие типы: числа, строки, массивы, константы, выражения и операции со строками. Правила именования не отличаются от правил именования обычных переменных. При объявлении свойств также указывается модификатор доступа, о нем чуть позже, а сейчас укажем модификатор public.

Текст	Код
Давайте опишем наш класс User с использованием всех возможных вариантов значений для свойств. Объявим свойство $age и присвоим ему числовое значение. В этом свойстве будет хранится значение в типом int. Объявим свойство $secondName и присвоим ему строковое значение. В этом свойстве будет хранится значение в типом string. Объявим свойство $bithday и присвоим ему значение которое будет массивом. В этом свойстве будет хранится значение в типом array. Объявим свойство $position и присвоим ему значение константы DEVELOPER. Объявим значение константы в начале нашего скрипта через конструкцию с const. Для объявления свойств возможно использовать выражения. Например для свойства $contractPeriod мы использовали оператор умножения для наглядности. Число 31536000 ничего бы говорила разработчику, который бы читал наш код. Но выражение, которое мы написали показывает, что это количество секунд в году. Также можно использовать конкатинацию строк	<?php const *DEVELOPER* = 'Разработчик'; class User { public $age = 27; // числа public $secondName = 'Иванов'; // строки public $birthday = [10, 10, 1990]; // массивы public $position = *DEVELOPER*; // константы public $contractPeriod = 60 * 60 * 24 * 365; // выражения public $FIO = 'Иванов ' . 'И. И.'; // операции со строками } $mainUser = new User();

при описании свойств. МЫ использовали строки с Фамилией и Инициалами для описания свойства $FIO. Создадим объект User, запишем его в переменную $mainUser и выведем информацию на экран.	*var_dump*($mainUser);
Мы видим все наши свойства, которые мы объявили с их типами.	object(User)#1 (6) { ["age"]=> int(18) ["secondName"]=>string(14) "Иванов" ["birthday"]=>array(3) { [0]=>int(30) [1]=>int(11) [2]=>int(1990) } ["position"]=>string(22) "Разработчик" ["contractPeriod"]=>int(31536000) ["FIO"]=>string(21) "Иванов И. И." }

Для того, чтобы наши объекты имели определенный функционал в теле классов могут объявляться методы. Методами класса называются функции объявленные в теле класса. Так же как и функция, методы могут принимать любое количество аргументов и возвращать, либо не возвращать, одно значение. При наименовании методов используют глаголы, т.к. методы описывают что делают наши объекты.

Текст	Код
Продолжим улучшать нашего Кота Cat. Научим нашего котика разговаривать. Объявим метод sayName() как обычную функцию php, но внутри класса. Данная функция не будет принимать никаких аргументов и не будет ничего возвращать. Внутри функции находится ее тело. Но в контексте метода класса ее называют телом метода. Наша функция say будет выводить на экран строку “Mau”. Далее опишем второй метод sleep. Он уже будет принимать параметр $seconds. Наш котик теперь умеет спать, а когда он спит - то и спит вся наша программа	<?php class Message { public $name = 'Вы забыли дать мне кличку'; public function say() { // тело метода echo 'Mau'; } public function sleep($seconds) {

	`` sleep($seconds); } } ```
Давайте теперь воспользуемся функционалом наших методов. Для начала создадим объект Cat и присвоим ссылку на него в переменную $cat. Для вызова метода используется такая же конструкция как и для доступа к свойствам объекта “->”. Напишем $cat, конструкцию вызова метода объекта и название метода.	``` $cat = new Cat(); $cat->say(); $cat->sleep(2); ```

Константы

Как и обычные константы, константы класса не могу менять свое значение в ходе выполнения кода. Константы используются для обозначения постоянных значений, которые не будут менятся в процессе работы вашего класса. При этом они будут находится в одном месте и в случае изменения требований могут быть модифицированы.

Давайте рассмотрим на примере.

Текст	Код
Константы объявляются в теле класса. Объявим наш класс BlackCat и константу в нем. Пишем зарезервированное слово const и название нашей константы. Для нашего черного кота, константа Цвет - будет черной	class BlackCat { const *COLOR*= 'black'; }
Константы не привязаны к конкретным объектам класса, они принадлежат классу в целом. Поэтому для доступа к константе класса нужно написать название нашего класса и через два двоеточия название нашей константы. Используем var_dump для вывода значения на экран.	*var_dump*(BlackCat::*COLOR*);

запустим наш скрипт. Как мы видим, на экране отобразилось значение нашей константы.	string(5) "black"

Пространства имен

Легче всего понять что такое пространство имен, представив себе директорию с файлами. Где папки для файлов формируют их пространство имен. Так например файл с одним и тем же названием может находиться в разных директориях, но две копии файла не могут находится в одной директории.

Это одно из назначений пространства имен - решение проблемы с одинаковыми именами классов в разных библиотеках и своем коде. Для улучшения читаемости кода и ухода от Ну_Очень_Длинных_Имен. А также для логической группировки классов.

Для того чтобы задать пространство имен используется специальная конструкция namespace, после которой следует произвольное название, либо группа названий разделенных символом “\”, формируя тем самым подпространства имен.

Текст	**Код**
Пример. Конструкция namespace должна быть размещать вначале файла до любого другого кода	<?php namespace App; namespace App\Animals;

Хоть пространство имен определяется и для всего файла, но влияет оно только на: Классы, Интерфейсы, Трейты, Функции и Константы.

Хотя и возможно описывать несколько пространств имен в одном файлу - это делать крайне не рекомендуется.

Полное имя класса

Полным именем класса - складывается из пространства имен и имени класс.

Текст	Код
В этом случае полное имя класса будет App\Cat	<?php namespace App; class Cat {}
В этом случае полное имя класса будет App\Animals\Cat\Cat	<?php namespace App\Animals\Cat; class Cat {}

Для работы с пространствами имен - необходимо точно понимать, как определяется полное имя класса в текущем участке кода. Принцип построения схож, с построением пути в файловой системе. Есть три точки отсчета.

1. Неполные имена классов - если мы используем в коде программе только имя класса - то оно прибавляется к текущему пространству имен.

Текст	Код

Если у нас есть два класса Cat из предыдущего примера, то в этом случае мы создадим объект класса App\Cat	<?php namespace App; $cat = new Cat();
В этом случае полное имя класса будет App\Animals\Cat\Cat	<?php namespace App\Animals\Cat; $cat = new Cat();

2. Полные имена классов - если мы используем в коде подпространства и имя класса. То они прибавляются к текущему пространству имен

Текст	**Код**
В этом случае полное имя класса будет App\Animals\Cat\Cat и будет создан второй кот. Несмотря на то, что мы находимся в пространстве имен App	<?php namespace App; $cat = new Animals\Cat\Cat();

А в этом случае будет ошибка, т.к. полное имя получится App\Animals\Cat - что не будет соответствовать ни одному коту.	<?php namespace App\Animals; $cat = new Cat();

3. Абсолютные имена классов - если имя используемого класса начинается с косой черты \, то имя и будет считаться полным именем класса

Текст	**Код**
В этом случае полное имя класса будет App\Cat и будет создан первый кот.	<?php namespace App\Animals\Cat; $cat = new App\Cat();

Существует магическая константа *__NAMESPACE__* - это строка, которая содержит имя текущего пространства имен. В глобальном пространстве, вне пространства имен, она содержит пустую строку.

Стоит запомнить, что для классов пространство имен всегда отсчитывается относительно текущего пространства имен, для функций же, сначала проверяется существует ли функция в текущем пространстве имен, и если ее нет, то выполняется функция из глобального пространства имен.

Текст	Код
ц	<?php namespace App\Animals\Cat; function trim($str) { echo 'trim'; } trim('line');

Использование пространств имен

К классам можно обращаться тремя разными способами: По абсолютному или относительному имени, после импортирования или по псевдониму.

Как обращаться по имени мы уже рассмотрели выше. А что же такое импортирование.

Можно импортировать пространства имен или класс в свое пространство имен, при этом при импортировании класса - можно указать ему псевдоним (имя класса в текущем пространстве имен)

Текст	Код
Импорт осуществляется используя ключевое слово use При этом при импортировании нет необходимости указывать начальный \ , т.к. рассчет и так строится относительно глобального пространства имен. В этом примере $cat1 - будет объектом App\Cat; $cat2 и $cat3 - будут объектом App\Animals\Cat\Cat; функция trim - будет функцией из глобального пространства имен, а функция catTrim - это переопределенная функция из пространства имен App\Animals\Cat	<?php namespace App; use App\Animals; use App\Animals\Cat\Cat as CatAnimal; $cat1 = new Cat(); $cat2 = new Animals\Cat\Cat(); $cat3 = new CatAnimal(); use App\Animals\Cat\trim as catTrim; trim(‘cat1’); catTrim(‘cat2’);
Т.к. для классов пространство имен полное имя всегда преобразуется относительно текущего пространства имен, то для использования классов из глобального пространство имен - необходимо указывать их абсолютное имя, также и для переопределенных функций.	<?php namespace App\Animals\Cat; function trim($str) { echo 'trim';

Функция trim - будет вызвана из глобального пространства имен. а объект Исключение в этом случае удастся создать.	} \trim('line'); $b = new \Exception('hi');

Указатель $this

Внутри каждого метода объекта класса доступна псевдопеременная $this, которая является ссылкой на текущий объект. С помощью нее можно обращаться к свойствам текущего объекта и вызывать другие методы этого объекта внутри вызываемого метода.

Текст	Код
Создадим класс. И два метода getName() и say(); Создадим нового кота и узнаем как его зовут. Внутри метода getName будет вызван метод say текущего объекта, с использованием переменной $this - который выведет Мяу, а затем будет возвращено значение свойства name текущего объекта. Обращение к текущему свойству класса $name происходит при помощи переменной $this.	<?php namespace App; class Cat { public $name = 'Tom'; public function getName() { $this->say(); return $this->name; } public function say()

	{ echo ‘Mau’; } } $cat = new Cat(); echo $cat->getName(); ‘MauTom’

Конструкторы и деструкторы

У каждого класса есть два особых метода, которые выделяются, и часто используются в коде - это конструкторы и деструкторы.

Конструктор __construct

Конструктор - позволяет выполнить необходимые действия при создании/инициализации объекта, т.е. он будет вызван при использовании конструкции new Cat(). Конструктор может содержать параметры, которые должны быть переданы при создании объекта. Часто в конструкторе происходит инициализация свойств объекта.

Текст	**Код**

Добавим конструктор классу Кота - пусть параметром будет имя кота.
Внутри конструктора установим это имя.

```php
<?php
namespace App;

class Cat
{
    public $name;

    public function __construct($name)
    {
        $this->name = $name;
    }

    public function getName()
    {
        $this->say();
        return $this->name;
    }

    public function say()
    {
        echo 'Mau';
```

	} } $cat = new Cat('Мурзик'); echo $cat->getName(); 'MauМурзик'

Деструктор __destruct

Деструктор - как понятно из название - противоположность конструктору, будет вызван при высвобождении всех ссылок на объект или при завершении скрипта.

Текст	**Код**
Добавим деструктор к классу, и убедимся что он вызовется при завершении скрипта, т.е. выведется строка из деструктора.	<?php namespace App; class Cat { … function __destruct()

```
    {
        echo 'Котик ' . $this->name . ' ушел по своим делам';
    }

}

$cat = new Cat(‘Мурзик’);

‘Котик Мурзик ушел по своим делам’
```