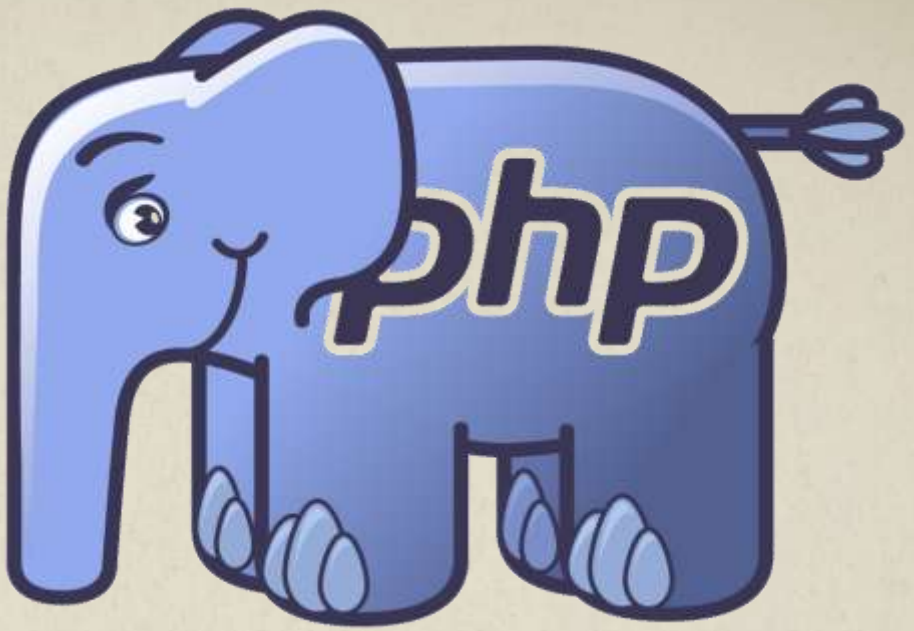


PHP: ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

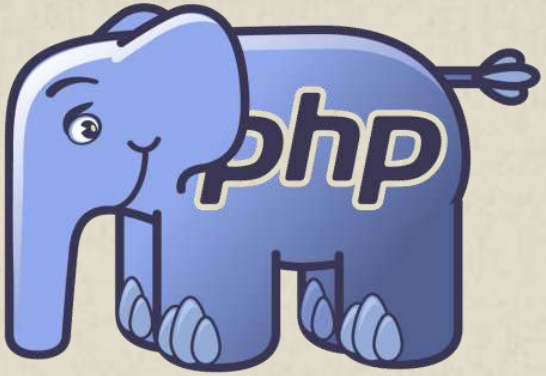
УРОК 1. МОДЕЛИ ДАННЫХ И ООП

ОБРАЩЕНИЕ К «СКЛАДЧИКАМ»

Я считаю складчины – полной ерундой. Невозможно научиться чему-то, просматривая видео. Без домашних заданий, без общения с преподавателями и коллегами. Покупая в складчину видеозаписи курсов, вы вредите прежде всего самим себе, создавая иллюзию «обучения». И поддерживаете каких-то мутных личностей-«организаторов», имеющих свой процент.

Впрочем, дело ваше.

Однако, если вы хотите по-настоящему учиться – приходите. Адрес есть на слайдах. Напишите в поддержку, мол «я складчик, но я хочу учиться». Скидку гарантирую ☺



МОДЕЛИ В MVC

Архитектура - это базовая организация системы, воплощенная в ее компонентах, их отношениях между собой и с окружением, а также принципы, определяющие проектирование и развитие системы. [IEEE 1471]

А если по-русски?

- Компоненты кода («на какие части разбить»)
- Структура кода («где что будет лежать»)
- Отношения между компонентами

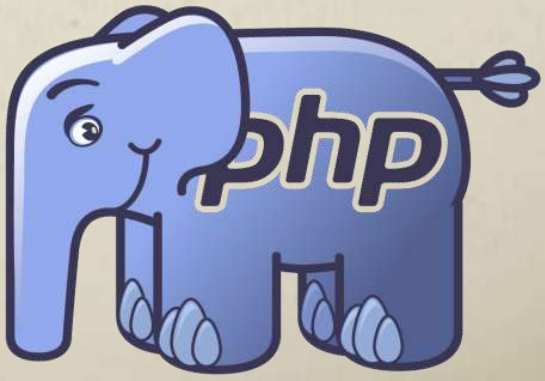
Сложно сразу продумать хорошую архитектуру

- Но делать это нужно!
- К счастью, всё придумано до нас ☺

Паттерны проектирования – «шаблоны» построения программы, следуя которым, можно получить удовлетворительный результат.

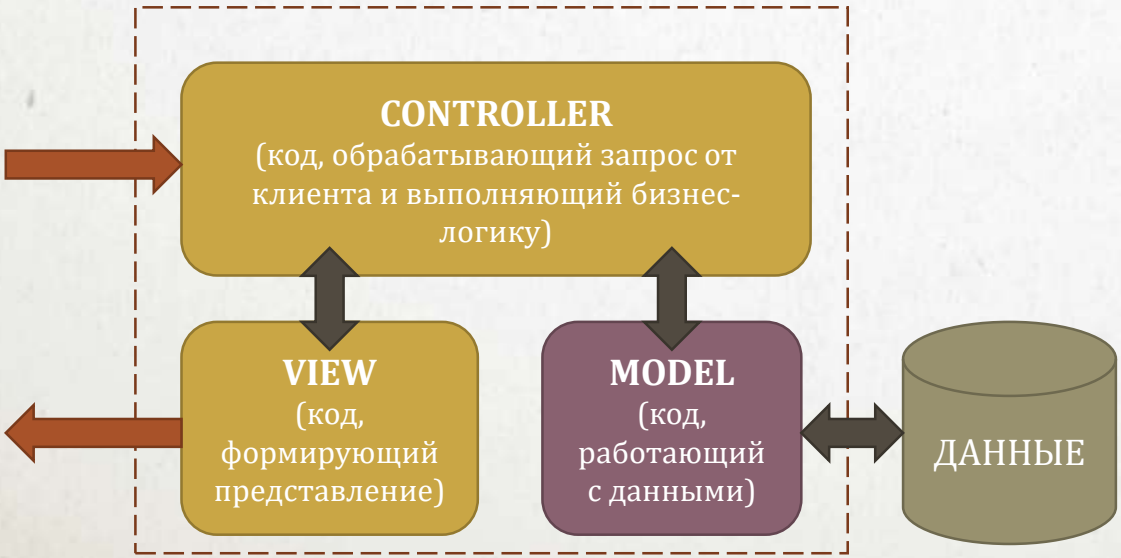
Пожалуй, главный паттерн - **MVC**

АРХИТЕКТУРА

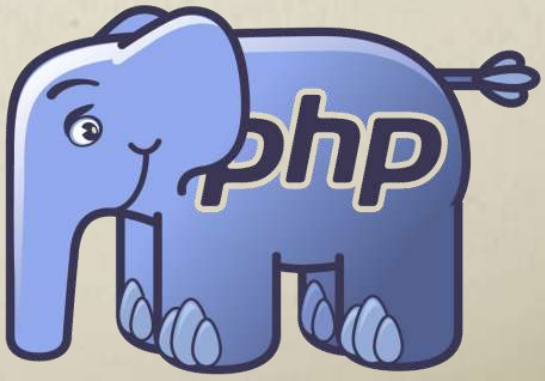


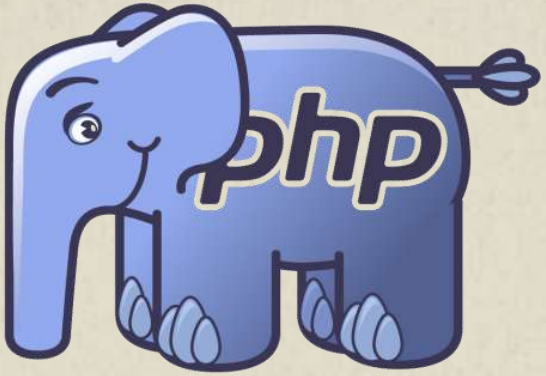
Модель работает с данными

- Ее задача – дать нужные данные тем компонентам, которые их запрашивают
- Модель и только модель, знает, где данные находятся, как они организованы
- Модель и только модель умеет обновлять, вставлять и удалять данные



АРХИТЕКТУРА





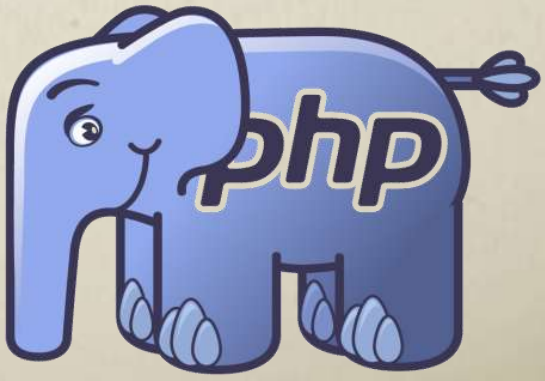
СТРОИМ СИСТЕМУ МОДЕЛЕЙ

Первое, что нам нужно сделать – это собрать в одном классе работу с базой данных

Логично сразу же запрограммировать класс DB, который должен удовлетворять следующим условиям:

- Создает и инкапсулирует соединение с базой данных
- Умеет выполнять ЛЮБЫЕ запросы к БД
- Возвращает результаты запросов
- И данные
- И, конечно, обрабатывает ошибки

КЛАССЫ ДЛЯ РАБОТЫ МОДЕЛЕЙ



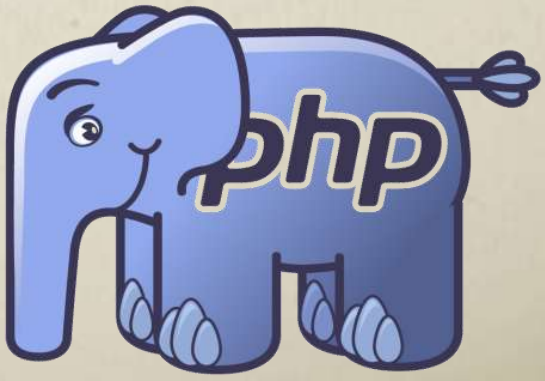
Мы можем определить, доступно ли свойство или метод «извне» или нет. Это называется «видимость»

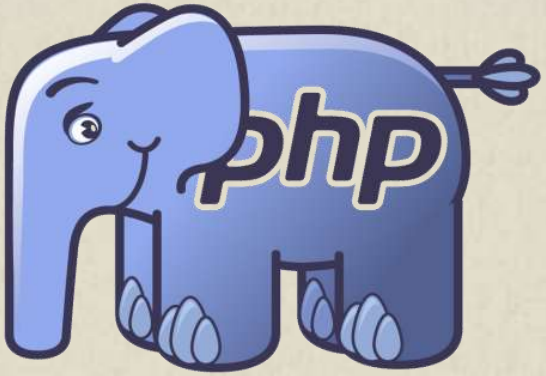
Например:

```
class Item {
  public $color;
  protected $owner;
}

$item = new Item;
$item->color = 'black';
$item->owner = $admin; // ОШИБКА!
```

- Ключевое слово `protected` вместо `public` приводит к тому, что обращение к свойству (или методу!) напрямую вызывает ошибку
- Однако «внутри», то есть в методах объекта, через `$this`, такое свойство/метод по-прежнему доступно
- Совокупность всех публичных свойств и методов называется «**интерфейс**»

НЕМНОГО ТЕОРИИ





МОДЕЛИ ДАННЫХ. ORM.

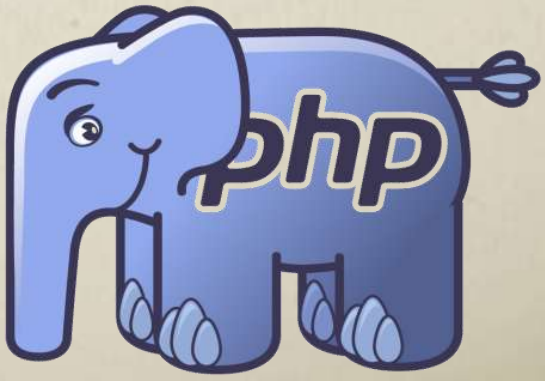
Объекты принято использовать в качестве МОДЕЛЕЙ данных

- **ORM – Object Relational Mapping**
  или, иначе говоря, принцип отображения объектов реального мира (и их связей) на объекты вашего языка программирования
- ООП прекрасно подходит для реализации этого шаблона проектирования.
  - Класс описывает какие объекты данных у нас могут быть
  - Сами данные представлены объектами заданных классов
  - И мы сразу может определить «поведение данных» в виде методов этих объектов

```
class User {
  public $email;
  public $name;
}
```

Задача: «заставить» класс DB возвращать нам не просто данные в массивах, а объекты нужного нам класса! Это пригодится в дальнейшем.

ORM

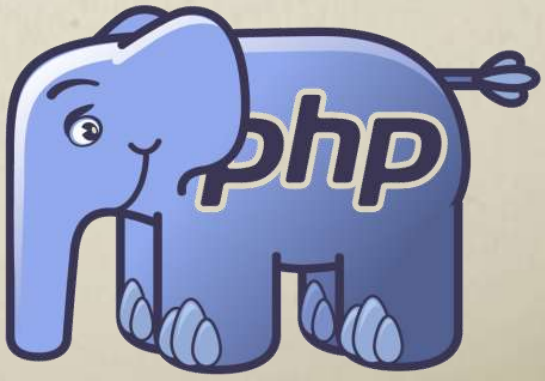


ORM – Object Relational Mapping

- Вообще говоря, было бы круто «заставить» класс User самого искать нужные нам данные в базе. Например так:
  $user = new User;
  $users = $user->findAll();
- Сделать это несложно. Давайте сделаем!
- Однако получается какой-то абсурд… Мы создаем нового пользователя и просим его найти всех пользователей? Что за бред?
- **Решение проблемы – статические методы. Это методы, которые принадлежат КЛАССУ в целом, а не конкретным объектам!**

```
class User {
  public $email;
  public $name;

  public static function findAll() {
    ...
  }

}

$users = User::findAll();
```

НЕМНОГО ТЕОРИИ



Статические методы и свойства

- Кроме методов статическими могут быть и свойства:

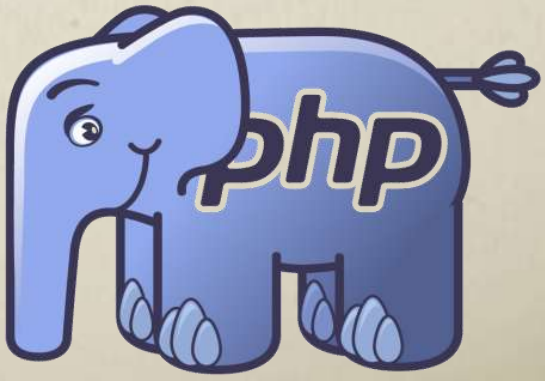
```
class User {

  public static $table = 'users';

  public static function findAll() {
    ...
  }

}

echo User::$table
```

- Однако, часто бывает, что нам в общем-то не нужно изменять некое значение, связанное с классом (имя таблицы в модели). В этом случае применяют константы:

```
class User {
  const TABLE = 'users';
  ...
}

echo User::TABLE
```

NB. *Имя константы принято писать в верхнем регистре!*

НЕМНОГО ТЕОРИИ



Статические методы и свойства

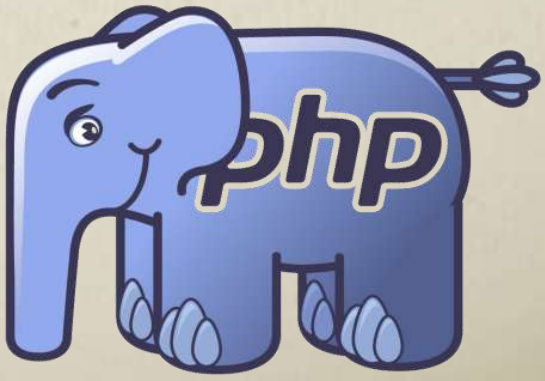
- Иногда требуется обратиться к статическому свойству, методу или классу из самого класса

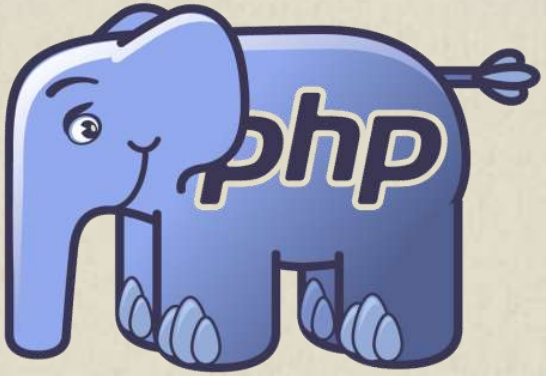
```
class User {

  protected static $table = 'users';

  public static function getTable () {
    return User::$table;
  }

}
```

- Однако такой код неоптимален.
  Представим, что нам нужно переименовать класс User – сколько нужно найти мест! Поэтому применяют слово “self” – оно значит «этот класс»

```
class User {

  public static function getTable () {
    return self::$table;
  }

}
```

КЛЮЧЕВОЕ СЛОВО SELF





АБСТРАКТНЫЕ КЛАССЫ. LSB.

Абстрактный класс – это класс, который не позволяет создание объектов этого класса.

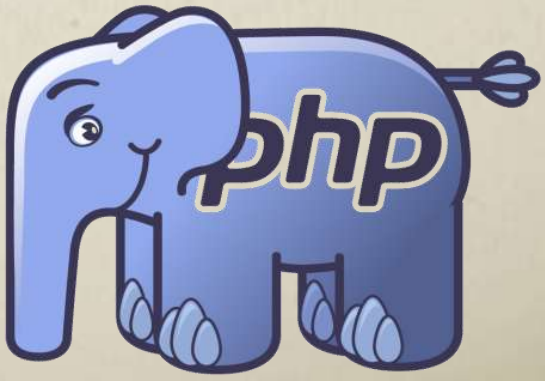
- Используются такие классы, конечно же, для наследования:

```
abstract class Model {
  ...
}

class User extends Model {
  ...
}
```

- В абстрактном классе могут быть как обычные свойства и методы, так и абстрактные методы
  - Они содержат только заголовок
  - Отсутствует собственно код метода
  - Дочерний класс обязан такой метод реализовать в точном соответствии с заголовком из абстрактного класса, иначе будет ошибка

NB. *Любой класс, в котором есть хотя бы один абстрактный метод, обязан быть абстрактным!*

АБСТРАКТНЫЕ КЛАССЫ



- **Проблема:**
  Ключевое слово “self” значит «тот класс, где оно написано»:

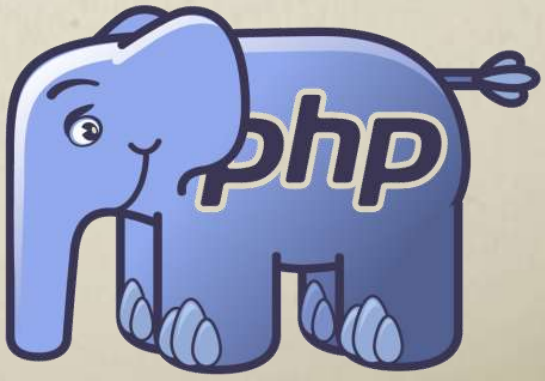
```
abstract class Model {
  public static $table='some table';
  public static function getTable() {
    return self::$table;
  }
}

class User extends Model {
  public static $table='users';
}

echo User::getTable(); // some table
```

- Такое происходит, потому что “self” – это «раннее» связывание, на этапе разбора кода
- Чтобы связаться с тем классом, в котором будет выполняться код, используйте **`static`** вместо **`self`**:

```
return static::$table;
```

- Это называется Late Static Binding (позднее статическое связывание), на этапе выполнения

LSB



ДО ВСТРЕЧИ НА СЛЕДУЮЩЕМ УРОКЕ!

**ВИДЕОЗАПИСЬ, СЛАЙДЫ, ПРЕЗЕНТАЦИЯ
И ДОМАШНЕЕ ЗАДАНИЕ
БУДУТ ВЫЛОЖЕНЫ ДО 10 УТРА СЛЕДУЮЩЕГО ДНЯ**

ProfIT