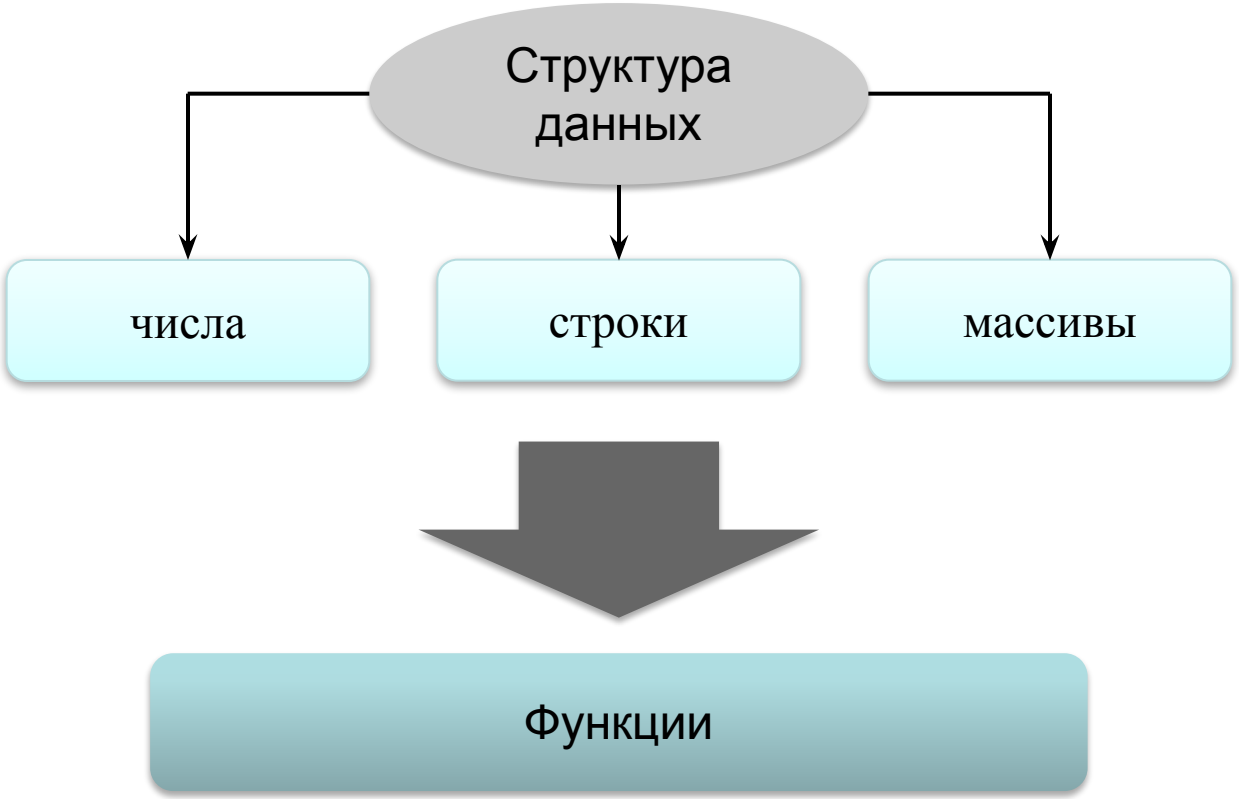
Объектно-ориентированное программирование (ООП)

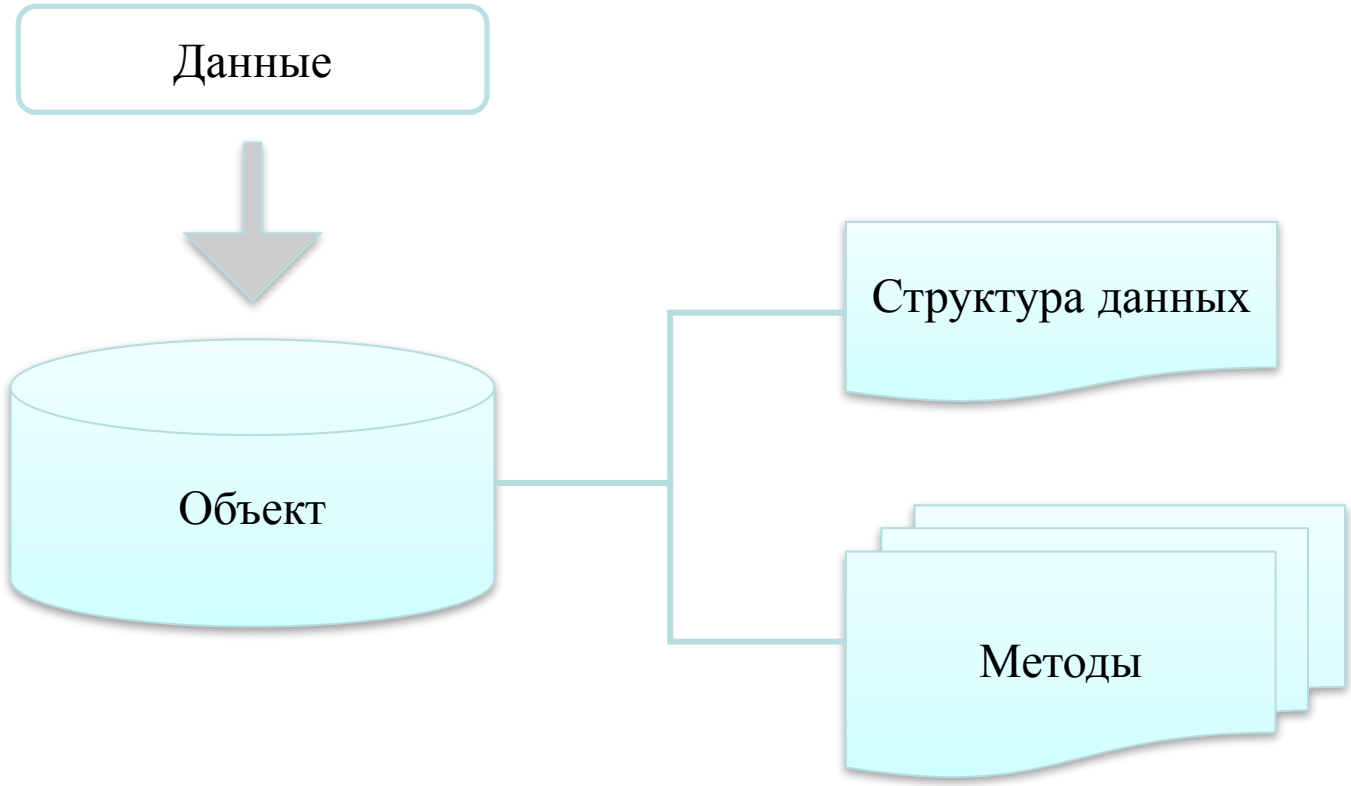
Селезнев Д.А.

Процедурный стиль программирования

Проблема процедурного программирования в том, что данные и функции их обработки не связаны между собой

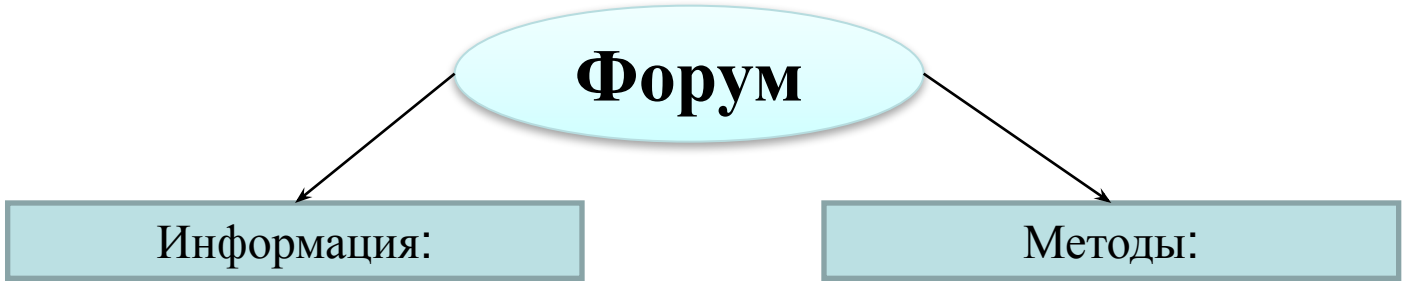


Что такое объектно-ориентированное программирование?



В ООП данные и функции для их обработки (методы) объединены в объекты

Пример



- Имя
- Логин
- Электронный адрес
- Пароль

и т.д

- Регистрация
- Авторизация
- Выход из системы

и т.д.

Объектно-ориентированное программирование (ООП)

Объектно-ориентированное программирование - это стиль кодирования, который позволяет разработчику группировать схожие задачи в классы

Объектно-ориентированное программирование основано на:

Инкапсуляции
Полиморфизме
Наследовании

Основные понятия ООП

Инкапсуляция – это механизм, объединяющий данные и обрабатывающие их функции (обычно называемые методами), как единое целое.

Когда код и данные связываются вместе подобным образом, создается **объект**. Иными словами, **объект** — это элемент, поддерживающий инкапсуляцию. Основной единицей инкапсуляции является **класс**, который описывает данные и методы, т.е. определяет форму объекта.

Полиморфизм - это свойство системы использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта. В более общем смысле понятие полиморфизма нередко выражается следующим образом: "один интерфейс — множество методов". Это означает, что для группы взаимосвязанных действий можно разработать общий интерфейс.

Наследование позволяет одному объекту приобретать свойства другого объекта, не путайте с копированием объектов. При копировании создается точная копия объекта, а при наследовании точная копия дополняется уникальными свойствами, которые характерны только для производного объекта.

Классы и объекты в PHP

Класс - это базовое понятие в объектно-ориентированном программировании. **Класс** можно рассматривать как своего рода "контейнер" для логически связанных данных и функций обрабатывающих их. Если сказать проще, то класс - это своеобразный тип данных.

Экземпляр класса - это **объект**. **Объект** - это совокупность переменных (свойств) и функций (методов) для их обработки. Свойства и методы называются членами класса. Вообще, объектом является все то, что поддерживает инкапсуляцию.

Если класс можно рассматривать как тип данныхЕсли класс можно рассматривать как тип данных, то объект — как переменную (по аналогии). Скрипт может одновременно работать с несколькими объектами одного класса, как с несколькими переменными.

Структура класса

Описание классов в PHP начинаются служебным словом **class**, за которым следует **имя класса** и **{ }:**

```
class MyClass
{
    // описание членов класса – свойств  и методов класса
}
```

Объект класса (новый экземпляр класса) создается с использованием оператора **new**:

```
$obj1 = new MyClass;
$obj2 = new MyClass;
```

Скрипт может одновременно работать с несколькими объектами одного класса, как с несколькими переменными

Данные описываются с помощью служебного слова **var**.

Пример класса

```
// Создаем новый класс  MyClass :
class MyClass {
// данные (свойства):
var $name;
var $addr;
// методы:
 function Name() {
 echo "<h3>John</h3>";
 }
}
// Создаем объект класса  MyClass:
$object = new  MyClass;
```

Доступ к членам класса

Мы рассмотрели, каким образом описываются классы и создаются объекты. Теперь нам необходимо получить доступ к членам класса, для этого в PHP предназначен оператор ->.

Специальный указатель **$this** применяется для обозначения объекта.

$this - это ссылка на объект, её нельзя использовать в статических методах или для статических свойств, для этого существует ключевые слова self и parent.

Чтобы получить доступ к членам класса (свойствам и методам) внутри класса, необходимо использовать указатель **$this**, который всегда относится к текущему объекту.

```
function Setname($name)
{
 $this->name = $name;
 $this->Getname();
}
```

Обратите внимание на отсутствие знака доллара перед **name**.

Пример

```
// Создаем новый класс Coor:
    class Coor {


// данные (свойства):
    var $name;


// методы:
     function Getname() {          // метод  Getname()
     echo $this -> name;
     }
     function Setname($name) {      // метод  Setname()
     $this -> name = $name;
     }
    }


    $object = new Coor;              // Создаем объект класса Coor:
    $object -> Setname("Nick");      // для изменения имени - метод Setname()
    $object -> Getname();            // для доступа -  Getname()


    // Сценарий выводит 'Nick'
```

Свойства и методы класса живут в разделенных "пространствах имен", так что возможно иметь свойство и метод с одним и тем же именем.

```
class Foo
{
    var $bar = 'свойство';

     function bar() {
        return 'метод';
    }
}

$obj = new Foo();
echo $obj -> bar;
echo "<br>";
echo $obj -> bar();
```

Подведем промежуточные итоги

Объявление класса должно начинаться с ключевого слова **class** (подобно тому, как объявление функции начинается с ключевого слова function).

Каждому объявлению свойства, содержащегося в классе, должно предшествовать ключевое слово **var**. Свойства могут относиться к любому типу данных, поддерживаемых в PHP.

После объявлений свойств следуют объявления методов, очень похожие на типичные объявления пользовательских функций. Методу также можно передавать параметры.

Области видимости свойств и методов

Доступ к свойствам и методам определяется через модификаторы:

public (общедоступные) – доступ как внутри класса, так и вне класса;

protected (защищённые) – доступ только внутри класса или внутри производных классов (классов, которые расширяют базовый класс, содержащий метод с директивой **protected**);

private (частные) – доступ только внутри класса, в котором они определены.

Методы, где определение модификатора отсутствует, определяются как **public**.

Свойства определенные с помощью **var**, будут объявлены как **public**.

static (статические) – доступ без инициализации класса. Статические свойства сохраняют свои значения на протяжении работы всего скрипта.

Пример

```
class MyClass
{
    public      $public = 'Public';
    protected $protected = 'Protected';
    private     $private = 'Private';

    public function  printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}
$obj = new MyClass();
echo $obj -> public;    // Работает
echo "<br>";
$obj -> printHello();   // Выводит Public, Protected и Private
```

Определение свойств класса

Определение свойств класса или инициализация объекта - это присвоение свойствам объекта первоначальные значения.

```
class MyClass
{
    $prop1="Свойство объекта";
}

$obj=new MyClass;
echo $obj->prop1;    // Выводим свойство
```

Инициализация объектов

Имя класса Coor и он содержит два свойства: имя человека и город его проживания. Можно написать метод который будет выполнять инициализацию объекта, например Init():

```php
<?php
// Создаем новый класс Coor:
class Coor {
// данные (свойства):
var $name;
var $city;

// Инициализирующий метод:
 function Init($name) {
 $this->name = $name;
 $this->city = "London";
 }

}

// Создаем объект класса Coor:
$object = new Coor;
// Для инициализации объекта сразу вызываем метод:
$object->Init();
?>
```

Конструкторы

Довольно часто при создании объекта требуется задать значения некоторых свойств. К счастью, разработчики технологии ООП учли это обстоятельство и реализовали его в концепции конструкторов.

Конструктор представляет собой метод, который задает значения некоторых свойств (а также может вызывать другие методы). Конструкторы вызываются автоматически при создании новых объектов.

В версиях до PHP5 имя метода конструктора совпадало с именем класса к которому он относится, а начиная с версии PHP5 имя метода конструктора необходимо называть **__construct()** (**это 2 подчеркивания перед словом __construct()**).

Раньше создание объекта и инициализация свойств выполнялись раздельно. Конструкторы позволяют выполнить эти действия за один этап.

Конструктор автоматически вызывается при создании объекта. Давайте попробуем создать класс, который будет содержать метод **__construct()**:

```
class MyClass
{
public  function  __construct()
   {
     echo  "Я только что был создан!";
   }
}
  $myObject = new MyClass();  // выведет "Я только что был создан!"
```

Пример Конструктора

```
Class Product{

  private $title;
  private $price;
  private $discount;


  public function __construct($title,
       $price, $discount)
  {
    $this->title = $title;
    $this->price = $price;
    $this->discount = $discount;
  }

  public function getProduct()
  {
    echo  'Название товара: '.$this->title.'<br>';
    echo  'Цена товара: '.$this->price .' $<br>';
    echo  'Скидка: '.$this->discount .'%<br>';
  }
}
$product = new Product('Мастер создания
       форм', 20, 25);
$product->getProduct();
```

Деструкторы

Подобно конструкторам в PHP существуют деструкторы, которые вызываются строго перед тем, как объект удаляется из памяти.

Это очень полезный метод для корректной очистки свойств класса (например, для правильного закрытия соединения с базой данных).

Примечание: PHP автоматически удаляет объект из памяти, когда не остается ни одной переменной, указывающей на него.

Например, если вы создадите новый объект и сохраните его в переменной **$myObject**, а затем удалите ее с помощью метода **unset($myObject)**, то сам объект также удалится. Также, если вы создали локальную переменную в какой-либо функции, она (вместе с объектом) удалится, когда функция завершит работу.

В отличии от конструкторов, в деструкторы нельзя передавать никакие параметры!

Пример Деструктора

Чтобы создать деструктор, добавьте в класс метод ___destruct().

```
class MyClass
{
 public function __destruct()
  {
   echo "Я деструктор. Объект был удален. Пока!";
   }
}
$myObject = new MyClass();

// для явного вызова деструктора и удаления объекта можно использовать функцию unset( )

unset($myObject);    // отобразит "Я деструктор. Объект был удален. Пока!"

echo “А теперь завершается работа сценария”;
```

Необходимость в вызове деструкторов возникает лишь при работе с объектами, использующими большой объем ресурсов, поскольку все переменные и объекты автоматически уничтожаются по завершении сценария.

```
class MyClass
{
 public function __destruct()
{
   echo "Я деструктор. Объект был удален. Пока!";
 }
}
$myObject = new MyClass();

exit(); // отобразит  "Я деструктор. Объект был удален. Пока!"
```

Магические методы в PHP

Конструктор и деструктор – это так называемые «магические методы».

Магические методы - это специальные методы, которые вызываются, когда над объектом производятся определенные действия.

Все методы начинающиеся со знака __ (два подчеркивания перед именем метода), PHP считает «магическими».

Полный список магических методов смотрите в руководстве по PHP

Вложенные объекты

Свойства объектов сами могут быть объектами.
Тогда говорят, что объект вложен в другой объект.

```
class Room
{
  public $name;
  function __construct($name=‘безымянная’)
  {
    $this -> name = $name;
  }
}
class House
{
  public $room;
}
$home = new House;
$home -> room[] = new Room(‘спальня’);
$home -> room[] = new Room(‘кухня’);
print($home -> room[1] -> name);
```

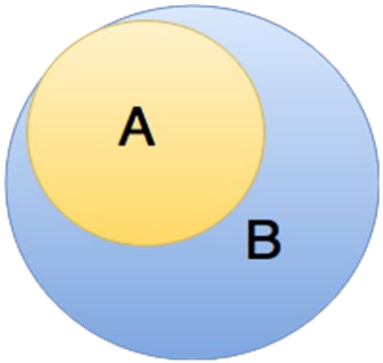
В этом примере объект $home содержит массив вложенных объектов room.

Наследование классов в PHP

Это механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса. Потомок может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами.

Наследование - это не просто создание точной копии класса, а расширение уже существующего класса, чтобы потомок мог выполнять какие-нибудь новые, характерные только ему функции.

Схематичное изображение наследования классом B класса A



Чтобы создать новый класс, наследующий поведение существующего класса, надо использовать ключевое слово **extends** в его объявлении. Например:

```
class классA extends классB
  {
  .......
  }
```

Ключевое слово **extends** говорит о том, что создаваемый класс является лишь "расширением" класса **A**, и не более того. То есть **B** содержит те же самые свойства и методы, что и **A**, но, помимо них и еще некоторые дополнительные, "свои".

Теперь "часть A" находится прямо внутри класса **B** и может быть легко доступна, наравне с методами и свойствами самого класса **B**.

Обратите также внимание: мы можем теперь забыть, что B унаследовал от A некоторые свойства или методы — снаружи все выглядит так, будто класс B реализует их самостоятельно.

Немного о терминологии: родительский класс A принято называть базовым классом, а дочерний класс B - производным от A. Иногда базовый класс также называют суперклассом, а производный - подклассом.

Имейте в виду, что производный класс имеет только одного родителя.

Класс-наследник (подкласс) может переопределить свойства и методы родителя – это называется **перегрузкой**.

Пример наследования

```php
<?php
class Parent {
  function parent_funct() { echo "<h1>Это родительская функция</h1>"; }
  function test () { echo "<h1>Это родительский класс</h1>"; }
}

class Child extends Parent {
  function child_funct() { echo "<h2>Это дочерняя функция</h2>"; }
  function test () { echo "<h2>Это дочерний класс</h2>"; }
}

$object = new Parent;
$object = new Child;

$object->parent_funct(); // Выводит 'Это родительская функция'
$object->child_funct(); // Выводит 'Это дочерняя функция'
$object->test(); // Выводит 'Это дочерний класс'
?>
```

Дочерний класс (подкласс) Child наследует все методы и свойства суперкласса Parent

Обращение к элементам классов. Оператор разрешения области видимости (::)

Оператор разрешения области видимости или просто "двойное двоеточие" - позволяет обращаться к константам, к статическим свойствам и методам класса, к переопределенным свойствам и методам класса.

При обращении к этим элементам извне класса, необходимо использовать имя этого класса.

Начиная с версии PHP 5.3.0, стало возможным обратиться к классу с помощью переменной.

Для обращения к свойствам и методам внутри самого класса используются ключевые слова **SELF, PARENT, STATIC.**

Использование :: вне объявления класса

При обращении к этим элементам извне класса, необходимо использовать имя этого класса.

Начиная с версии PHP 5.3.0, стало возможным обратиться к классу с помощью переменной.

```
class MyClass
{
    const CONST_VALUE = 'Значение константы';
}

$classname = 'MyClass';
echo $classname :: CONST_VALUE;  // Начиная с версии PHP 5.3.0
echo MyClass :: CONST_VALUE;
```

Использование :: внутри объявления класса

Для обращения к свойствам и методам внутри самого класса используются ключевые слова ***self***, ***parent*** и ***static***.

```
class OtherClass extends MyClass
{
    public static $my_static = 'статическая переменная';

    public static function doubleColon() {
        echo parent :: CONST_VALUE . "\n";
        echo self :: $my_static . "\n";
    }
}

$classname = 'OtherClass';
echo $classname :: doubleColon();   // Начиная с версии PHP 5.3.0
OtherClass :: doubleColon();
```