Web-технологии

ООП в PHP

Кулаков Кирилл Александрович

Определение базового класса

- Синтаксис

```
class MyClass
{
    // свойство класса
    public $var = 'значение';
    // метод класса
    public function displayVar() {
        echo $this->var;
    }
}
```

Создание объекта

- Создание экземпляра класса должно быть после описания класса

```
include_once("MyClasses.php");
$one_instance=new MyClass(); // объект типа MyClass
```

- Свойства и методы доступны с помощью ->

```
$имя_объекта->имя_свойства = "значение";
$имя_объекта->имя_метода(arg1, arg2, ....);
```

- При определении свойств не требуется var (предупреждение)
- Переменная $this – ссылка на объект
- Все объекты представляются по ссылке

Пример (использование ссылок)

```
class test {
    public $A=10;
    function show_A ($name_A) {
        echo $this→$name_A."\n"; # к свойству
    }
}
$a=new test;
$a->show_A('A');
$method_name='show_A';
$a→$method_name('A'); # к методу
```

Клонирование

- clone – создание независимой копии объекта

```
$two_instance=clone $one_instance;
```

- Метод __clone() – управление значениями клонированного объекта

```
class MyClass
{
    public $var = 10;
    public function displayVar() {
        echo $this->var;
    }
    public function __clone() {
        $this->var=0;
    }
}
```

Клонирование

- Любые свойства, являющиеся ссылками на другие переменные, останутся ссылками
- PHP7: обращение к свойствам/методам только что склонированного объекта

```
<?php
$dateTime = new DateTime();
echo (clone $dateTime)->format('Y');
?>
```

Константы

- Константы внутри класса определяются ключевым словом const
- Обращение внутри класса
  - по имени
  - self::constant
- Обращение вне класса
  - имя_класса :: константа
  - $переменная_объект :: константа

Пример. Обращение к константе класса

```
class test {
    const A=10;
    function show_A () {
        echo "A =".self::A."\n";  # внутри класса
    }
}
echo test::A."\n"; # через класс
$a=new test;
$a->show_A();
echo $a::A."\n"; # через объект (PHP > 5.3.0)
```

Статические свойства

- Статические свойства принадлежат классу, а не объекту
- Не зависят от состояния объекта
- Описание с помощью static

```
public static $my_static = 'foo';
```

- Доступ внутри класса

```
self::$my_static
```

- Доступ вне класса

```
Имя_класса ::$my_static
```

Статические методы

- Статические методы не связаны с определенным экземпляром объекта
- Вызываются без создания экземпляра класса
- Предназначены для вызова вне контекста объекта
- Не имеют доступа к переменной $this
- Описание с помощью static

```
public static function $my_method() {...}
```

- Вызов метода

```
Имя_класса ::$my_method();
```

Конструктор класса

- Функция, вызываемая при создании объекта
- Метод __construct() – инициализация свойств класса

```
class UserDB {
    private $_dbFileName;
    private $_dbContent = array();
    function __construct ($dbfilename) { # описание аргумента
    $this->_dbFileName = $dbfilename; # обращение к аргументу
    }
    …
}
```

- parent::__construct() – вызов родительского конструктора в теле конструктора потомка

Пример

```
class test {
    public $A;
    function __construct($val) {
        $this->A=$val;
    }
    function show_A () {
        echo $this->A."\n";
    }
}
$a=new test (10); # передача параметра при создании
$a->__construct(20); # передача параметра при явном вызове
$a->show_A();
```

Деструктор класса

- Функция, вызываемая при удалении объекта
- Метод __destruct() вызывается:
  - при явном удаление объекта (unset)
  - при завершении работы скрипта
  - при удалении всех ссылок на объект
  - при вызове exit()

```
function __destruct() {
  print "Удаление объекта" . get_class($this) . "\n";
}
```

parent::__destruct() – вызов родительского деструктора в теле деструктора потомка

Контроль доступа внутри объекта

- Уровни доступа
  - public – доступ из любого места сценария
  - protected – доступ внутри наследуемых, родительских классов, внутри экземпляра класса
  - private – доступ внутри экземпляра класса ($this)
- При описании свойств и методов класса необходимо указывать модификатор
- При описании методов класса работает умолчание: если модификатор не указан, то предполагается public
- PHP7.1: константы класса могут быть объявлены как public, private или protected (по умолчанию public)

Пример

```
class A {
    private $str;
    function mymethod($b) {
        echo "Вы передали параметр b = $b \n“;
    }
}
$obj = new A();
$obj -> mymethod("парам");  # Работает
$obj -> str = "строка";   # Fatal error
```

Пример

```
class MyClass
{
    public const MY_PUBLIC = 'public';
    protected const MY_PROTECTED = 'protected';
    private const MY_PRIVATE = 'private';
}

$myclass = new MyClass();
MyClass::MY_PUBLIC; // Работает
MyClass::MY_PROTECTED; // Fatal Error
MyClass::MY_PRIVATE; // Fatal Error
```

Видимость из других объектов

- Объекты, которые имеют общий тип (наследуются от одного класса), имеют доступ к элементам с модификаторами private и protected друг друга

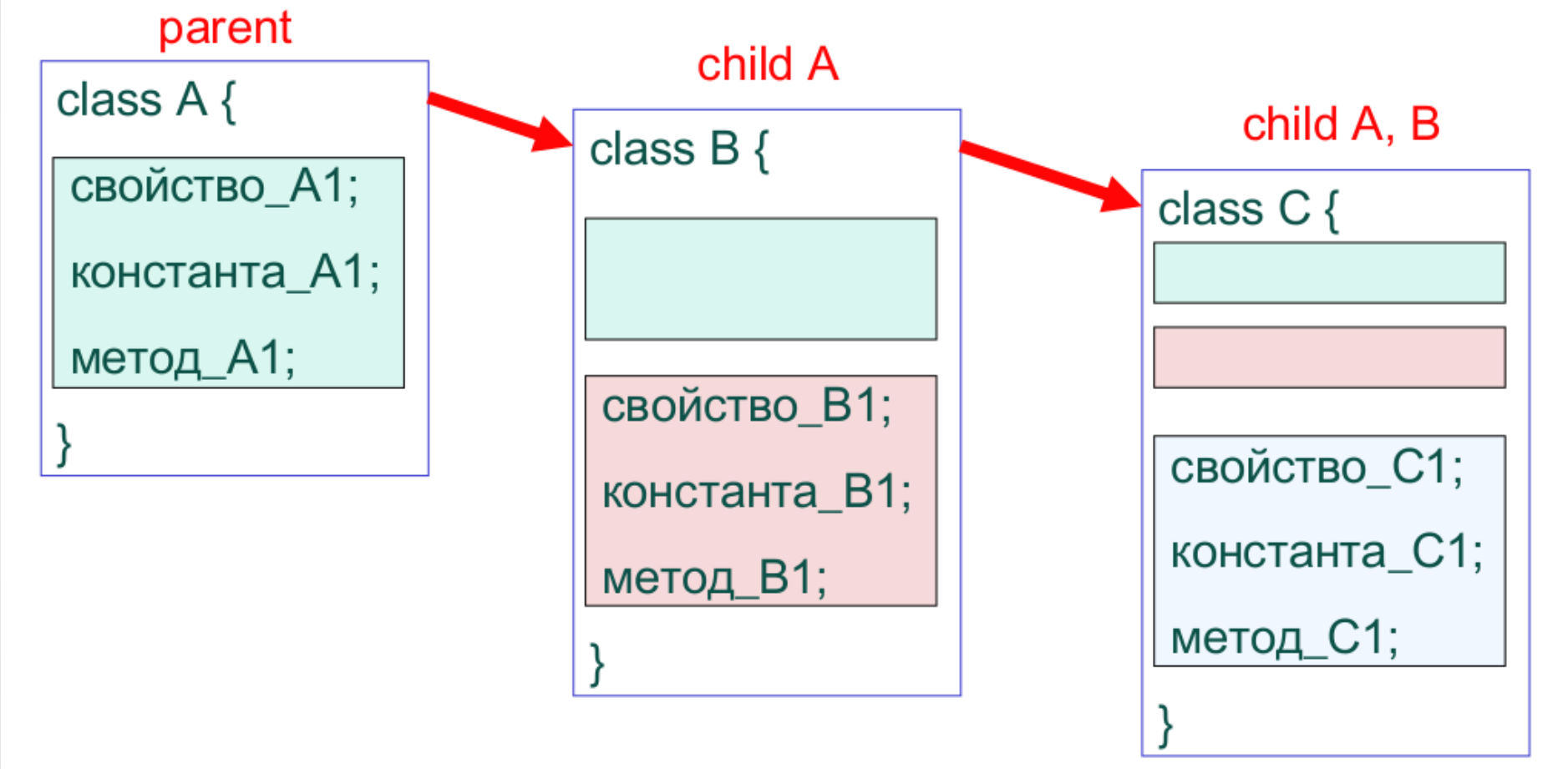
```
class Test {
    private $foo;
    public function __construct($foo) { $this->foo = $foo; }
    private function bar()  { echo 'Доступ к закрытому методу.'; }
    public function baz(Test $other)  {
        $other->foo = 'hello';
        var_dump($other->foo);
        $other->bar();
    }
}
$test = new Test('test');
$test->baz(new Test('other'));
```

Указание типов

- Определение типа в прототипе функции аналогично обычным функциям
- PHP5 варианты: имя_класса, self, массив, NULL
- PHP7 доп.варианты: callable, bool, float, int, string

```
class A {
    public function test(MyClass $my_class) {
        echo $my_class->var;
    }
    public function print_mas(array $mas) {
        print_r($mas);
    }
}
```

Наследование



Наследование

- Синтаксис

```
class A {
...
}
class B extends A {
...
}
class C extends B {
...
}
```

Пример: предпочтение локального (self) конструктора

```
class A {
    private $var_A=15;
    function __construct() { $this->var_A=0; }
    function show_A() { echo $this->var_A."\n"; }
}
class B extends A {
    public $var_B=25;
    function __construct() { $this->var_B=1; }
    function show_B() { echo $this->var_B."\n"; }
}
```

Пример: вызов родительского конструктора

```
class A {
    private $var_A=15;
    function __construct() { $this->var_A=0; }
    function show_A() { echo $this->var_A."\n"; }
}
class B extends A {
    public $var_B=25;
    function __construct() {
        $this->var_B=1;
        parent::__construct();
    }
    function show_B() { echo $this->var_B."\n"; }
}
```

Пример: отличия private и protected

```
class A {
    private $var_A=15;
}
class B extends A {
    public $var_B=25;
    function show(){
        echo $this->var_A."\n";
        echo $this->var_B."\n";
    }
}
$b=new B;
$b->show();
```

Пример: перегрузка метода класса

```
class A {
    function show() {
        echo "Класс A\n";
    }
}
class B extends A {
    function show() {
        echo "Класс_ B\n";
    }
}
$b=new B;
$b→show(); # Класс_ B
```

Управление наследованием с помощью final

- final class имя_класса {}

наследование

- final public function имя_метода() {}

перегрузка

Абстрактные классы

- Класс, который содержит по крайней мере один абстрактный метод, должен быть определен как абстрактный
- Нельзя создать экземпляр абстрактного класса
- Абстрактные методы несут лишь описательный смысл и не могут включать реализации
- При наследовании все абстрактные методы должны быть определены в классе-потомке
- Сигнатуры методов (типы и обязательные аргументы) должны совпадать
- PHP>5.4: Сигнатуры конструкторов должны совпадать

Пример

```
abstract class AbstractClass {
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);
    public function printOut() {
        print $this->getValue() . "\n";
    }
}
class ConcreteClass1 extends AbstractClass {
    protected function getValue() {
        return "ConcreteClass1";
    }
    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}
```

Интерфейсы

- Интерфейс – средство для определения набора методов
- Синтаксис:

```
interface test {
    public function f_1($var);
    public function f_2();
}
```

- Реализация интерфейса:

```
class A implements test {
    public function f_1($var) { return $var;}
    public function f_2() { echo "f_2";}
}
```

- Проверка реализации интерфейса:

```
$a = new A;
var_dump($a instanceof test); # bool (true)
```

Автоматическая загрузка объектов

- Включение модуля-описания класса

```
include_once("MyClassName.php");
```

- __autoload() – автоматическая загрузка (deprecated)
- Пример:

```
function __autoload($class_name) {
    require_once $class_name . '.php';
}
```

Автоматическая загрузка объектов (PHP > 5.3)

- spl_autoload_register — Регистрирация функции в качестве реализации метода __autoload()
- spl_autoload — Реализация по умолчанию метода __autoload()
- spl_autoload_call — Попытка загрузить описание класса всеми зарегистрированными методами __autoload()

```
function my_autoloader($class) {
    include 'classes/' . $class . '.class.php';
}
spl_autoload_register('my_autoloader');
spl_autoload_register(function ($class) {
    include 'classes/' . $class . '.class.php';
});
```

Сравнение объектов

- Оператор равенства ($a == $b):
  - TRUE – если $a и $b содержат одинаковые свойства, одинаковые значения и принадлежат одному классу
- Оператор эквивалентности ($a === $b):
  - TRUE – если $a и $b представляют один и тот же объект
- Пример:

  $o = new A; $p = new A; $r = new B;

  $q = $o;

  $o==$p – TRUE; $o===$p – FALSE

  $o==$q – TRUE; $o===$q – TRUE

  $o==$r – FALSE; $o===$r – FALSE

Полезные функции

- class_exists – был ли объявлен класс
- get_class - возвращает имя класса, к которому принадлежит объект
- get_parent_class - возвращает имя родительского класса для объекта или класса
- interface_exists - определен ли интерфейс
- is_a - принадлежит ли объект к данному классу
- method_exists - существует ли метод в данном классе
- property_exists - содержит ли объект (класс) указанное свойство