*Лекция 6: Объекты и классы в PHP*

#### **Классы и объекты**

Начнем с основных понятий объектно-ориентированного программирования – *класса* и *объекта*. Существует множество определений этих понятий. Мы дадим следующее: ***объект*** – это структурированная переменная, содержащая всю информацию о некотором физическом предмете или реализуемом в программе понятии, *класс* – это описание таких *объектов* и действий, которые можно с ними выполнять.

В PHP *класс* определяется с помощью следующего синтаксиса:

class Имя\_класса{

var $имя\_свойства;

/\*список свойств\*/

function имя\_метода( ){

/\* определение метода \*/

}

/\*список методов\*/

}

Имена *свойств* *объектов* *класса* объявляются с помощью ключевого слова var, *методы*, применимые к *объектам* данного *класса*, описываются функциями. Внутри определения *класса* можно использовать ключевое слово *this* для обращения к текущему представителю *класса*.

Например, нам нужно создать *класс*, описывающий категорию статей. У каждой статьи имеются такие *свойства*, как название, автор и краткое содержание. Какие действия мы хотим совершать со статьями? Возможно, нам понадобится задавать значения перечисленным *свойствам* статьи, отображать статью в браузере. Тогда определение этого *класса* может выглядеть следующим образом:

<?

class Articles { // Создаем класс Статей

var $title;

var $author;

var $description;

// метод, который присваивает значения

// атрибутам класса

function make\_article($t, $a, $d){

$this->title = $t;

$this->author = $a;

$this->description = $d;

}

//метод для отображения экземпляров класса

function show\_article(){

$art = $this->title . "<br>" .

$this->description .

"<br>Автор: " . $this->author;

echo $art;

}

}

?>

Итак, для описания физических *объектов* типа "статья" мы создали *класс* с именем *Articles*, состоящий из трех переменных, содержащих характеристики статьи, и двух функций для создания конкретной статьи и для ее отображения.

Как известно, работая с PHP, можно периодически переключаться в режим HTML. В этом случае программа состоит из нескольких кусков (блоков) кода. Определение *класса* нельзя разносить по разным блокам php-кода и тем более по разным файлам. То есть если написать:

<?php

class Articles { // Начало описания класса

var $title;

?>

<?php

// продолжение описания класса

function show\_article(){

// содержание метода

}

} // конец описания класса

?>

то программа не будет работать корректно.

Несколько замечаний по поводу имен *классов*. Имя *класса* должно удовлетворять правилам именования *объектов* в языке PHP, но есть ряд имен, которые зарезервированы разработчиками для своих целей. В первую очередь это имена, начинающиеся с символа подчеркивания "\_". Для создания *классов* и функций нельзя использовать такие имена. Кроме того, зарезервировано имя stdClass, поскольку оно используется внутри движка PHP.

##### **Инициализация переменных**

Часто некоторым атрибутам *класса* бывает необходимо присваивать значения сразу после создания представителя *класса*. Когда мы создавали *класс* статей, для присваивания значений атрибутам ( *свойствам* ) *класса* мы использовали специальную функцию make\_article(). Вообще говоря, мы поступили не совсем верно, потому что занялись изобретением велосипеда. Специально для задания начальных значений атрибутам *класса* существует два стандартных *метода*. В *PHP4* можно инициализировать значения с помощью оператора var или с помощью функции *конструктора*. С помощью var можно инициализировать только константные значения. Для задания не константных значений используют функцию *конструктор*, которая вызывается автоматически, когда *объект*конструируется из *класса*. Функция- *конструктор* должна иметь имя, совпадающее с именем всего *класса*, в котором она определена.

Приведем пример. Допустим, при создании *объекта* "статья" мы хотим установить его *свойства* следующим образом: автора – равным строке "Иванов", название и краткое содержание – соответствующим элементам глобального массива $*\_POST*, а дату публикации статьи – текущей дате. Тогда следующее описание *класса* не является корректным в *PHP4*:

<?

class Articles { // Создаем класс Статей

var $title= $\_POST["title"];

var $author = "Иванов";

var $description = $\_POST["description"];

var $published = date("Y-m-d");

// метод, который присваивает значения

// атрибутам класса

}

?>

А вот такое *описание класса в PHP4* будет работать так, как нужно:

<?

class Articles { // Создаем класс Статей

var $title;

var $author = "Иванов";

var $description;

var $published;

// метод, который присваивает значения

// атрибутам класса

function Articles(){

$this->title = $\_POST["title"];

$this->description = $\_POST["description"];

$this ->published = date("Y-m-d");

}

//метод для отображения экземпляров класса

function show\_article(){

$art = $this->title . "<br>" .

$this->description .

"<br>Автор: " . $this->author;

echo $art;

}

}

?>

Отметим, что в *PHP3* и *PHP4* *конструкторы* работают по-разному. В *PHP3* функция становилась *конструктором*, если она имела то же имя, что и *класс*, а в *PHP4* – если она имеет то же имя, что и *класс*, в котором она определена. Разница в подходах видна, когда один *класс* расширяет другой и происходит *наследование* *свойств* и *методов* *базового класса*. Но об этом мы поговорим чуть позже.

В *PHP5**конструктор* *класса* именуется \_\_construct. Кроме того, в *PHP5* появились и *деструкторы* – функции, которые вызываются автоматически перед уничтожением *объекта*. В *PHP5* функция-*деструктор* должна быть названа \_\_destruct.

##### **Объекты**

В одной из первых лекций мы упоминали о существовании в PHP такого типа данных, как *объект*. ***Класс*** – это описание данных одного типа, данных типа *объект*. *Классы* являются как бы шаблонами для реальных переменных. Переменная нужного типа создается из *класса* с помощью оператора new. Создав *объект*, мы можем применять к нему все *методы* и получать все *свойства*, определенные в описании *класса* . Для этого используют такой синтаксис: $имя\_объекта->название\_свойства или $имя\_объекта->название\_метода(*список аргументов*). Заметим, что перед названием *свойства* или *метода* знак $ не ставят.

<?php

$art = new Articles;

// создаем объект $art

echo ($art ->title);

// выводим название объекта $art

$another\_art = new Articles;

// создаем объект $another\_art

$another\_art->show\_article();

// вызываем метод для

// отображения объекта в браузер

?>

Пример 6.1. Доступ к методам и свойствам объекта

Каждый из *объектов* *класса* имеет одни и те же *свойства* и *методы*. Так, у *объекта* $*art* и у *объекта* $another\_art есть *свойства*title, description, author и *методы* *Articles*(), show\_article(). Но это два разных *объекта*. Представим себе *объект* как директорию в файловой системе, а его характеристики – как файлы в этой директории. Очевидно, что в каждой директории могут лежать одинаковые файлы, но тем не менее они считаются различными, поскольку хранятся в разных директориях. Точно так же *свойства* и *методы* считаются различными, если они применяются к разным *объектам*. Чтобы получить нужный файл из директории верхнего уровня, мы пишем полный путь к этому файлу. При работе с *классами* нужно указывать полное имя функции, которую мы хотим вызвать. Директорией верхнего уровня в PHP будет пространство глобальных переменных, а путь указывается с помощью разделителя ->. Таким образом, имена $*art*->title и $another\_art->title обозначают две разные переменные. Переменная в PHP имеет только один знак доллара перед именем, поэтому нельзя писать $*art*->$title. Эта конструкция будет рассмотрена не как обращение к *свойству* title *объекта* $*art*, а как обращение к *свойству*, имя которого задано переменной $title (например, $*art*->"" ).

<?php

$art->title = "Введение в Internet";

// так можно установить

// значение свойства объекта

$art->$title = "Введение в Internet";

// так нельзя установить

// значение свойства объекта

$property = "title";

$art->$property = "Введение в Internet";

// так можно установить значение

// свойства объекта

?>

Пример 6.2. Установка значений свойств

Создавая *класс*, мы не можем знать, какое имя будет иметь *объект* этого *класса*, тем более что *объектов* может быть много и все могут иметь разные имена. Соответственно мы не знаем, как обращаться к *объекту* внутри определения *класса*. Для того чтобы иметь доступ к функциям и переменным внутри определения *класса*, нужно использовать псевдопеременную $this. Например, $this->titleвозвращает значение *свойства* title у текущего *объекта* данного *класса*. Иногда эту переменную предлагают читать как "мое собственное" (к примеру, по отношению к *свойству* ).

#### **Наследование**

##### **extends**

Механизм *наследования* – очень важная часть всего *объектно-ориентированного подхода*. Попытаемся объяснить его суть на примере. Допустим, мы создаем описание человека. Очевидно, что сделать это мы можем по-разному, в зависимости от того, для чего нужно это описание. Можно описать человека как программиста: он знает такие-то языки программирования, операционные системы, участвовал в стольких-то проектах. Однако если человек программист, то он не перестает быть человеком вообще, т.е. он имеет имя, фамилию, место жительства и т.п. Если перевести наши рассуждения в термины объектно-ориентированного программирования, то можно сказать, что мы описали два *класса* – *класс* людей и *класс* программистов, каждый со своими *свойствами* и *методами*. Причем *класс*программистов, очевидно, обладает всеми *свойствами* *класса* людей и при этом имеет свои специфические характеристики, т.е. *класс*программистов является подклассом *класса* людей. Так, если у человека вообще есть имя, то у программиста оно тоже должно быть, но не наоборот. Кроме программистов можно выделить еще множество *классов* по профессиональной принадлежности людей. И все они будут подклассами *класса* людей. Часто на практике удобно определять общий *класс*, который может использоваться сразу в нескольких проектах (например, *класс* людей или личностей), и адаптировать его для специфических нужд каждого проекта (например, как *класс* программистов). Как это можно реализовать? С помощью механизма расширений. Любой *класс* может быть расширением другого *класса*. Расширяющий (или производный) *класс*, кроме тех *свойств* и *методов*, которые описаны в его определении, имеет все функции и *свойства* основного ( *базового класса* ). В нашем примере *класс* программистов – расширяющий, а *класс* всех людей – базовый. Из *класса* нельзя удалить никакие существующие *свойства* и функции, *класс* можно только расширить. *Расширяющий класс* в *PHP4* всегда зависит только от одного *базового класса*, поскольку множественное *наследование* в PHP не поддерживается. Расширяются *классы* в PHP с помощью ключевого слова *extends*.

<?php

class Person { // определяем класс Личности

var $first\_name; // имя личности

var $last\_name; // фамилия личности

function make\_person($t,$a){

// метод устанавливает

// значения имени и фамилии объекта

$this->first\_name = $t;

$this->last\_name = $a;

}

function show\_person(){

// метод отображает информацию о личности

echo ("<h2>" . $this->first\_name . " " .

$this->last\_name . "</h2>");

}

}

class Programmer extends Person{

// определяем класс

// Programmer, расширяющий Person

var $langs = array ("Lisp");

// константным массивом

// задать переменную в var можно

function set\_lang($new\_lang){

// метод добавляет еще

// один язык к списку известных

$this->langs[] = $new\_lang;

}

}

?>

Пример 6.3. Использование механизма наследования

*Класс* Programmer имеет те же переменные и функции, что и *класс* Person, плюс переменную $langs, в которой содержится список изученных программистом языков, и функцию set\_lang для добавления еще одного языка к списку изученных. Создать представителя *класса* программистов можно обычным способом с помощью конструкции new. После этого можно устанавливать и получать список языков, которые знает программист, и в то же время можно использовать функции, заданные для *класса* Person, т.е. устанавливать и получать имя и фамилию программиста и отображать сведения о нем в браузере:

<?php

$progr = new Programmer;

$progr->set\_lang("PHP");

// методы, определенные для

// класса Programmer

print\_r ($progr->langs);

// методы, определенные для класса Person

$progr->make\_person("Bill","Gates");

$progr->show\_person();

?>

Отношения, в которых состоят созданные нами *классы* Person и Programmer, называют также *отношениями родитель–потомок*. *Класс* Person – родитель, а его потомки, такие как *класс* Programmer, создаются, основываясь на нем, с помощью расширений. Любой *класс* может стать родительским и соответственно породить потомков.

Порядок определения *классов* имеет значение. Нельзя сначала определить *класс* Programmer, *расширяющий класс* Person, а уже потом сам *класс* Person. *Класс* должен быть определен перед тем, как он будет использоваться (расширяться).

#### **Конструкторы**

Теперь, после знакомства с механизмом *наследования* в PHP, мы можем прокомментировать различие между *конструкторами* *PHP4* и *PHP3* и более подробно рассказать о *конструкторах* вообще. Напомним, что в *PHP3* ***конструктор*** – это функция, имя которой совпадает с именем *класса*. А в *PHP4* – функция, имя которой совпадает с именем *класса*, в котором она определена.

<?php

class Programmer extends Person{

// определяем класс

// Programmer, расширяющий Person

var $langs = array ("Lisp");

function Programmer(){

// этот конструктор будет

// работать и в PHP3, и в PHP4

$this->make\_person("Иван","Петров");

}

}

?>

Пример 6.4. Использование конструктора

Здесь функция Programmer() является *конструктором*, т.е. выполняется сразу после создания любого представителя *класса*Programmer, задавая ему имя "Иван" и фамилию "Петров". *Конструкторы*, как и любые другие функции, могут иметь аргументы. В этом случае, создавая представителя *класса*, нужно указать значения этих параметров. Аргументы *конструктора* могут иметь и значения по умолчанию. Если все аргументы имеют значения по умолчанию, тогда можно создавать экземпляр *класса* без параметров.

<?php

class Programmer extends Person{

// определяем класс

// Programmer, расширяющий Person

var $langs = array ("Lisp");

function Programmer($n = "Иван",

$f = "Петров"){

// это конструктор

$this->make\_person($n,$f);

}

}

$default\_progr = new Programmer();

// создаст программиста Ивана Петрова

$new\_progr = new Programmer("Вася",

"Сидоров");

// создаст программиста Васю Сидорова

print\_r($new\_progr);

/\* выведет информацию о переменной

$new\_progr, т.е. свойства объекта

и их значения \*/

?>

Пример 6.5. Использование конструктора

Приведенные примеры будут работать и в *PHP3*, и в *PHP4*, конечно если дописать в них определение *базового класса* Person. Допустим, ситуация немного другая: *конструктор* имеется только у *базового класса* Person:

<?php

class Person { // определяем класс Личности

var $first\_name;

var $last\_name;

function Person($t,$a){ // конструктор

$this->first\_name = $t;

$this->last\_name = $a;

}

/\* ... \*/

}

class Programmer extends Person{

// определяем класс

// Programmer, расширяющий Person

var $langs = array ("Lisp");

function set\_lang($new\_lang){

$this->langs[] = $new\_lang;

}

}

$new\_progr = new Programmer("Вася",

"Сидоров");

?>

Что произойдет в этом случае при создании *объекта* *класса* Programmer, будет ли автоматически вызвана какая-либо функция? В *PHP3* ничего не произойдет, поскольку в этом *классе* нет функции с именем Programmer() (здесь *конструктор* – это функция, имя которой совпадает с именем *класса* ). В *PHP4* будет вызван *конструктор* *базового класса*, если он существует, т.е. вызовется функция Person() из *класса* Person (здесь *конструктор* – функция, имя которой совпадает с именем *класса*, в котором она определена).

Еще одна ситуация – в *базовом классе* есть функция, имя которой совпадает с именем *расширяющего класса*, а в *расширяющем классе*нет *конструктора*.

<?php

class Person { // определяем класс Личности

var $first\_name;

var $last\_name;

function Person($t,$a){ // конструктор

$this->first\_name = $t;

$this->last\_name = $a;

}

function Programmer($new\_lang){

echo "Я – программист";

}

}

class Programmer extends Person{

// определяем класс

// Programmer, расширяющий Person

var $langs = array ("Lisp");

function set\_lang($new\_lang){

$this->langs[] = $new\_lang;

}

}

$new\_progr = new Programmer("Вася",

"Сидоров");

?>

В этом случае *PHP3* вызовет в качестве *конструктора* функцию Programmer() из описания *класса* Person. Поскольку *конструктор* – это функция, у которой то же имя, что и у *класса*. И неважно, определена ли эта функция в самом *классе* или она наследуется из *базового класса*. В *PHP4* *класс* Programmer не будет иметь своего *конструктора*, поэтому вызовется *конструктор* *базового класса*.

Ни в PHP 3, ни в PHP 4 *конструктор* *базового класса* не вызывается автоматически из *конструктора* порожденного *класса*.

#### **Оператор ::**

Иногда внутри описания *класса* возникает необходимость сослаться на функции или переменные из *базового класса*. Бывает, что нужно ссылаться на функции в *классе*, ни один представитель которого еще не создан. Как быть в таком случае? В *PHP4* для этого существует специальный оператор "::"

Например, вот так можно вызвать в описании *класса* Programmer функцию show\_name() из *базового класса* Person и функцию say\_hello(), заданную в описании *класса* Programmer, когда ни один *объект* этого *класса* еще не был создан:

<?php

class Person { // определяем класс Личности

var $first\_name;

var $last\_name;

function Person($t,$a){ // конструктор

$this->first\_name = $t;

$this->last\_name = $a;

}

function show\_name(){

// метод отображает информацию о личности

echo ("Меня зовут, " .

$this->first\_name . " " .

$this->last\_name . "!<br>");

}

}

class Programmer extends Person{

// определяем класс

// Programmer, расширяющий Person

function set\_lang($new\_lang){

// метод добавляет еще

// один язык к списку известных

$this->langs[] = $new\_lang;

Person::show\_name();

// вызываем функцию из базового класса

echo "И я знаю теперь еще и " .

$new\_lang;

}

function show\_name(){

echo ("Я программист, " .

$this->first\_name . " " .

$this->last\_name . "!<br>");

}

function say\_hello(){

echo "Привет!<br>";

}

}

Programmer::say\_hello();

// вызываем функцию, когда ни

// один объект ее класса еще не создан

$new\_progr = new Programmer("Вася","Сидоров");

$new\_progr->set\_lang("PHP");

?>

В результате работы этой программы получим следующее:

Привет!

Меня зовут Вася Сидоров!

И я знаю теперь еще и PHP

С помощью команды Programmer::say\_hello(); мы вызываем функцию say\_hello *класса* Programmer как таковую, а не как *метод*, применяемый к *объекту* данного *класса*. В этот момент переменных *класса* нет. Поэтому функции, вызываемые до создания *объекта*, не могут пользоваться переменными *класса* и конструкцией *this*, но могут пользоваться локальными и глобальными переменными.

В определении *класса* Programmer мы переопределили функцию show\_name(), поэтому вызвать функцию show\_name() из *базового класса* Person можно только с помощью оператора "::" Вообще говоря, внутри определения *класса* мы можем вызывать любые *методы* и *свойства*, заданные в его *базовом классе* с помощью обычного $this, если только порожденный *класс* не переопределяет эти *свойства* и *методы*, как в нашем примере.

#### **Оператор parent**

В приведенном выше примере, обращаясь к *базовому классу*, мы использовали его имя (мы писали Person::show\_name() ). Это не совсем удобно, потому что имя *класса* или иерархия *классов* может измениться, и тогда придется переписывать код описаний всех *классов* с тем, чтобы привести используемые в них имена в соответствие с новой иерархией. Чтобы избежать подобной ситуации, вместо имени *базового класса* нужно использовать ключевое слово parent (например, parent::show\_name() ). Parent ссылается на *класс*, прописанный после *extends* в объявлении вашего *класса*. Поэтому если вдруг иерархия *классов* изменится, то достаточно будет внести изменения в имена, указанные после *extends* в описаниях *классов*.

#### **Объектная модель PHP5**

Кроме нового названия для *конструкторов* и появления *деструкторов* в *PHP5* произошло еще достаточно много изменений. Мы не будем обсуждать их подробно, только опишем в общих чертах. Основное изменение – это передача значений параметров *класса* по ссылке и присвоение *объектов* по ссылке, а не по значению, как это было в *PHP4*. В *PHP5* если создаются две равные переменные типа *объект*, то они указывают на одно значение и изменяются одновременно (мы приводили похожий пример с переменными *строкового типа*). В связи с этим появился новый механизм для создания копий *объектов* – так называемое *клонирование*. В *PHP4* все *методы* и переменные *класса* доступны извне, т.е. они всегда являются открытыми. В *PHP5* переменные и *методы* можно делать открытыми (доступными отовсюду), закрытыми (доступными только внутри *класса* ) и защищенными (доступными внутри *класса* и в его производных *классах* ). Кроме того, появилась возможность создавать *интерфейсы* и *абстрактные классы* и многое другое. В целом *объектная модель в PHP5*значительно усовершенствована для более точного соответствия объектно-ориентированной *парадигме программирования*.

#### **Решение задачи**

Итак, мы хотели по выбору пользователя генерировать форму для ввода описания статьи или человека и отображать данные, введенные в эту форму. Попробуем решить эту задачу, используя *объектно-ориентированный подход*. Для начала создадим форму, где пользователь выбирает, что он хочет создать, – описание статьи или человека (точнее, это будут две формы):

<form action="task1.php">

Создать описание статьи: <input type=submit

name=art\_create

value="Create Article">

</form>

<form action="task1.php">

Создать описание личности: <input

type=submit name=pers\_create

value="Create Person">

</form>

Теперь напишем файл для обработки этих форм. В нем создадим два *класса* – статьи и личности. У каждого *класса* имеется *метод* для инициализации его переменных и *метод* для отображения *объектов* данного *класса*. При решении задачи будут использованы две функции, встроенные в PHP для работы с *классами* и *объектами*. Это функция *get\_class* (объект), возвращающая имя *класса*, экземпляром которого является *объект*, переданный ей в качестве параметра. И функция *get\_class\_vars* (имя *класса* ), которая возвращает массив всех *свойств* *класса* и их значений по умолчанию. Аналогично можно получить массив имен всех *методов* *класса*: *get\_class\_methods* (имя *класса* )

<?php

// Создаем классы Статей и Личностей.

// Статья имеет заголовок, автора и

// описание. Личность имеет имя, фамилию

// и e-mail

class Article {

var $title;

var $author;

var $description;

// метод, который присваивает значения

// атрибутам класса

function Article($t="Название отсутствует",

$a="Автор отсутствует",

$d="Описание отсутствует"){

$this->title = $t;

$this->author = $a;

$this->description = $d;

}

//метод для отображения экземпляров класса

function show(){

$art = "<h2>$this->title</h2><font

size=-1>$this->description</font><p>Автор:

$this->author</p>";

echo $art;

}

}

// Определение класса Личностей

class Person {

var $first\_name;

var $last\_name;

var $email;

//метод, который присваивает значения атрибутам класса

function Person($t="Имя не введено",

$a="Фамилия не введена",$d="Email не указан"){

$this->first\_name = $t;

$this->last\_name = $a;

$this->email = $d;

}

//метод для отображения экземпляров класса

function show(){

$art = "<h2>$this->first\_name</h2><font

size=-1>$this->last\_name</font><p>e-mail:

$this->email</p>";

echo $art;

}

}

// Далее следует собственно создание и отображение

// экземпляров выбранного класса

if (isset($\_GET["art\_create"])){ //Если была выбрана статья

$art = new Article; // создаем представителя класса статей

$art\_vars = get\_class\_vars(get\_class($art)); //какие

// аргументы этого класса нужно задать

Make\_form($art,$art\_vars,"art\_create"); //вызов функции

// создания формы

if (isset($\_GET["create\_real"])){ Show\_($art\_vars); }

// если данные этой формы отправлены, то вызываем

// функцию показа

}

//то же самое, если была выбрана личность

if (isset($\_GET["pers\_create"])){

$art = new Person;

$art\_vars = get\_class\_vars(get\_class($art));

Make\_form($art,$art\_vars,"pers\_create");

if (isset($\_GET["create\_real"])){ Show\_($art\_vars); }

}

// функция создания формы

function Make\_form($art,$art\_vars,$glob){

$str = "<form>"; // html код формы записывается

// в строку $str

//перебираем список переменных класса объекта $art

foreach ($art\_vars as $var\_name => $var\_value){

$str .="$var\_name<input type=text name=$var\_name><br>";

//создаем элемент формы с именем свойства класса

}

$str .= "<input type=hidden name=$glob>"; // чтобы не

// забыть, что мы создаем

$str .= "<input type=submit name=create\_real

value='Create and Show'></form>";

echo "$str"; // выводим форму

}

// функция показа объекта

function Show\_($art\_vars){

global $art; //используется глобальное имя объекта

$k = count($art\_vars); //число свойств класса

// (переменных в форме)

$p=0; //вспомогательная переменная

foreach ($art\_vars as $name => $value){

$p++;

if ($\_GET["$name"]=="") $val= $art->$name;

else $val = $\_GET["$name"];

if ($p<>$k) $par .='"'. $val.'",';

else $par .='"'. $val.'"';

}

$const=get\_class($art);

$par = '$art->'.$const ."(" .$par.");";

// теперь $par представляет собой php-код для вызова

// метода класса $art, изначально

// записанного в $par

// например,

// $art->Person('Vasia','Petrov','vas@intuit.ru');

eval($par); // функция eval выполняет код,

// содержащийся в $par

$art->show();

}

?>

Листинг 6.6. Использование объектно-ориентированного подхода

#### **Заключение**

Итак, мы изучили основные понятия объектной модели, используемой в языке PHP. Были описаны правила создания *классов* и их представителей-объектов, способы задания начальных значений переменным *класса*, способы получения значений *свойств* и вызов *методов* *классов*.