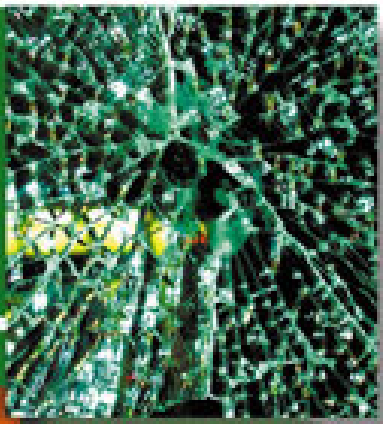
МАКСИМ КУЗНЕЦОВ
ИГОРЬ СИМДЯНОВ

bhv®

PHP

практика создания Web-сайтов

2-е издание



- ОБЪЕКТНО-ОРИЕНТИРОВАННЫЕ ВОЗМОЖНОСТИ PHP
- РАБОТА С БАЗАМИ ДАННЫХ
- «ХИТРОСТИ» PHP
- ТОНКОСТИ РАБОТЫ С HTTP-ПРОТОКОЛОМ
- СИСТЕМА УПРАВЛЕНИЯ СОДЕРЖИМЫМ САЙТА (CMS)
- РАЗРАБОТКА ДИНАМИЧЕСКИХ WEB-ПРИЛОЖЕНИЙ
- СОЗДАНИЕ ДИНАМИЧЕСКИХ ИЗОБРАЖЕНИЙ

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Максим Кузнецов
Игорь Симдянов

PHP
практика создания Web-сайтов

2-е издание

Санкт-Петербург
«БХВ-Петербург»
2008

УДК 681.3.06
ББК 32.973.26-018.2
К89

Кузнецов, М. В.

К89 PHP. Практика создания Web-сайтов / М. В. Кузнецов, И. В. Симдянов. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2008. — 1264 с.: ил.

ISBN 978-5-9775-0203-0

Рассматривается создание большого количества Web-приложений, входящих в состав полнофункционального Web-сайта. Попутно подробно обсуждаются все вопросы, с которыми может столкнуться Web-разработчик, начиная с создания инструментария для быстрой разработки Web-приложений и последних нововведений языка программирования PHP и заканчивая вопросами безопасности и особенностями программирования клиент-серверных приложений.

Книга ориентирована на читателей, знакомых с языком разметки HTML и базовыми возможностями языка программирования PHP. Второе издание полностью переработано, учтены нововведения версий PHP 5.1 и 6.0.

Для программистов и Web-разработчиков

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Ирина Артемьева*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 28.03.08.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 101,91.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

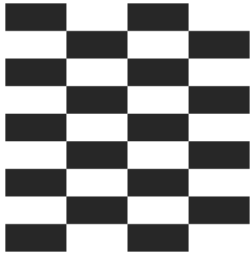
Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.002108.02.07
от 28.02.2007 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0203-0

Оглавление

Глава 1



Объектно-ориентированные возможности PHP

Последние два десятилетия в IT-индустрии получил широкое распространение объектно-ориентированный подход. Его введение связано со все возрастающим объемом программных систем, с которыми приходится сталкиваться разработчикам.

> ***Замечание***
>
> Данная глава кратко рассматривает объектно-ориентированные возможности PHP. Всестороннему рассмотрению предмета посвящена наша отдельная книга "Объектно-ориентированное программирование на PHP".

1.1. Введение в объектно-ориентированное программирование

Независимо от языка программирования объектно-ориентированный подход имеет ряд общих принципов, а именно:

- возможность создавать *абстрактные типы данных*, позволяющая наряду с предопределенными типами данных (такими как `integer`, `bool`, `double`, `string`) вводить свои собственные типы данных (классы) и объявлять "переменные" таких типов данных (объекты). Создавая свои собственные типы данных, программист оперирует не машинными терминами (переменная, функция), а объектами реального мира, поднимаясь тем самым на новый абстрактный уровень. Яблоки и людей нельзя умножать друг на

друга, однако низкоуровневый код запросто позволит совершить такую логическую ошибку, тогда как при использовании абстрактных типов данных такая операция становится невозможной;

- *инкапсуляция*, допускающая взаимодействие пользователя с абстрактными типами данных только через их интерфейс и скрывающая внутреннюю реализацию объекта, не допуская влияния на его внутреннее состояние. Память человека ограничена и не может содержать все детали огромного проекта, тогда как использование инкапсуляции позволяет разработать объект и использовать его, не заботясь о внутренней реализации, прибегая только к небольшому числу интерфейсных методов;
- *наследование*, позволяющее развить существующий абстрактный тип данных — класс, создав на его основе новый класс. При этом новый класс автоматически получает возможности уже существующего абстрактного типа данных. Зачастую абстрактные типы данных слишком сложны, поэтому прибегают к их последовательной разработке, выстраивая иерархию классов от общего к частному;
- *полиморфизм*, допускающий построение целых цепочек и разветвленных деревьев наследующих друг другу абстрактных типов данных (классов). При этом весь набор классов будет иметь ряд методов с одинаковыми названиями: любой из классов данного дерева гарантированно обладает методом с таким именем. Этот принцип помогает автоматически обрабатывать массивы данных разного типа.

Абстрактные типы данных необходимы для того, чтобы дать программисту вводить в программу переменные с желаемыми свойствами, так как возможностей существующих в языке типов данных зачастую не хватает. Связи между объектами реального мира зачастую настолько сложны, что для их эффективного моделирования необходим отдельный язык программирования. Разрабатывать специализированный язык программирования для каждой прикладной задачи — очень дорогое удовольствие. Поэтому в языки программирования вводится объектно-ориентированный подход, который позволяет создавать свой мини-язык путем создания классов и их объектов. Переменными такого мини-языка программирования являются программные *объекты*, в качестве типа для которых выступает класс. *Класс* описывает состав объекта — переменные и функции, которые обрабатывают переменные и тем самым определяют поведение объекта (рис. 1.1).

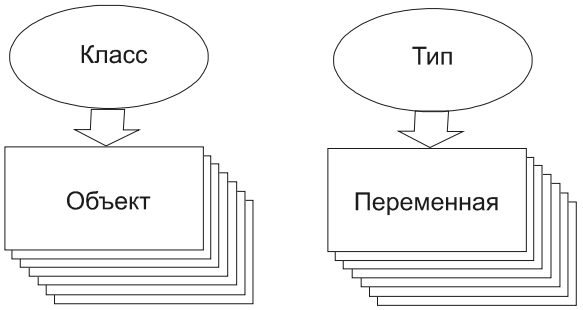


Рис. 1.1. Переменные объявляются при помощи типа, объекты при помощи класса

1.2. Создание класса

Класс объявляется при помощи ключевого слова `class`, после которого следует уникальное имя класса и тело класса в фигурных скобках. В теле класса объявляются переменные и функции класса, которые соответственно называются *методами* и *членами*. В листинге 1.1 приводится общий синтаксис объявления класса.

Листинг 1.1. Объявление класса

```
<?php
  class имя_класса
  {
    // Члены и
    // методы класса
  }
?>
```

Важной особенностью PHP является то, что PHP-скрипты могут включаться в документ при помощи тегов `<?php` и `?>`. Один документ может содержать множество включений этих тегов, однако класс должен объявляться в одном неразрывном блоке `<?php` и `?>`. Попытка разорвать объявление класса приводит к генерации интерпретатором ошибки разбора **Parse error: parse error, unexpected ';', expecting T_FUNCTION.**

Так как прерывать объявление класса недопустимо, его не удастся механически разбить и при помощи инструкций `include()`, `include_once()`, `require()`, `require_once()`. Допускается, однако, использование этих конструкций внутри методов.

Замечание

Напомним, что при помощи инструкций `include()`, `include_once()`, `require()`, `require_once()` можно включать в состав PHP-скриптов другие PHP-скрипты. Это позволяет разбивать объемные многострочные файлы на множество мелких файлов, которые программисту проще воспринять. При отсутствии включаемого файла инструкция `include()` генерирует предупреждение, однако не останавливает работу скрипта, в то время как `require()` в этом случае аварийно завершает работу приложения. Допускается множественное включение файлов друг в друга, что может приводить к запутанным ситуациям и многократному включению файлов в приложение. Суффикс `once` означает, что файл будет включен лишь один раз, и повторный вызов инструкции `include_once()` или `require_once()` игнорируется. Это особенно удобно для включения библиотек функций и классов, повторное объявление которых вызывает ошибку.

1.3. Создание объекта

Объект объявляется при помощи ключевого слова `new`, за которым следует имя класса. В листинге 1.2 создается пустой класс `emp` и объявляется объект `$obj` данного класса.

Замечание

После имени класса могут следовать необязательные круглые скобки.

Листинг 1.2. Создание класса `emp` и объявление объекта `$obj` данного класса

```
<?php
  class emp {}
  $obj = new emp;
?>
```

Замечание

Имена классов, в отличие от имен переменных и объектов, не зависят от регистра, поэтому можно использовать для объявления классов имена `emp`, `Emp()`, `EMP()`. Однако использование вместо объекта `$obj` переменной `$Obj` приведет к ошибке — интерпретатор PHP будет считать, что в программу введена новая переменная.

Объект `$obj` является обычной переменной PHP, правда, со специфичными свойствами. Как и любую другую переменную, объект можно передавать в качестве параметра функции, использовать как элемент массива или присваивать ему новое значение. В листинге 1.3 приводится пример, в котором объекту `$obj` по мере выполнения программы присваивается некоторое числовое значение, и `$obj` становится переменной числового типа.

Листинг 1.3. Объект — это обычная переменная

```
<?php
  class emp {}
  $obj = new emp();
  $obj = 3;
  echo $obj; // 3
?>
```

Объект существует до конца времени выполнения скрипта или пока не будет уничтожен явно при помощи конструкции `unset()` (листинг 1.4). Использование конструкции `unset()` может быть полезным, если объект занимает большой объем оперативной памяти и ее следует освободить, чтобы не допустить переполнения.

Листинг 1.4. Объект — это обычная переменная

```
<?php
  // Объявление класса
  class emp {}
  // Создание объекта класса
  $obj = new emp();
  // Уничтожение объекта
  unset($obj);
?>
```

В отличие от других языков программирования объект, объявленный внутри блока, ограниченного фигурными скобками, существует и вне его, не подвергаясь уничтожению при выходе из блока.

1.4. Инкапсуляция. Спецификаторы доступа

Как было показано в предыдущем разделе, класс может использоваться без методов и членов, однако пользы в этом случае от него не больше, чем от переменной, которая не может хранить значения. Как правило, классы содержат члены и методы. Часть из них будет использоваться внешним разработчиком, часть предназначена только для внутреннего использования в рамках класса. PHP предоставляет специальные ключевые слова — *спецификаторы доступа*, позволяющие указать, какие члены и методы доступны извне, а какие нет. Это позволяет реализовать принцип инкапсуляции.

Открытые члены класса объявляются спецификатором доступа `public` и доступны как методам класса, так и внешнему по отношению к классу коду. *Закрытые* методы и члены класса объявляются при помощи спецификатора `private` и доступны только в рамках класса; обратиться к ним извне невозможно.

> ***Замечание***
>
> Помимо спецификаторов `public` и `private` в PHP поддерживается спецификатор `protected`, используемый при наследовании и подробно рассматриваемый далее. В более ранних версиях для объявления члена класса использовалось ключевое слово `var`. Члены, объявленные с его помощью, были открытыми. В текущих версиях PHP по-прежнему допускается использовать ключевое слово `var` для объявления членов класса, однако это не рекомендуется, а само ключевое слово признано устаревшим. С большой долей вероятности оно будет исключено из PHP 6.

В листинге 1.5 представлен класс "сотрудник" (`employee`), который содержит четыре члена:

- `$surname` — фамилия сотрудника;
- `$name` — имя сотрудника;
- `$patronymic` — отчество сотрудника;
- `$age` — возраст сотрудника.

Листинг 1.5. Класс `employee`

```
<?php
  class employee
  {
    public $surname;
    public $name;
    public $patronymic;
    private $age;
  }
?>
```

Рассмотрим пример. Пусть класс `employee` располагается в файле `class.employee.php`. В листинге 1.6 объявляется объект `$emp` класса `employee`, при этом члены класса получают необходимые значения и выводятся в окно браузера.

Листинг 1.6. Обращение к членам класса `employee`

```
<?php
  // Подключаем объявление класса
  require_once("class.employee.php");

  // Объявляем объект класса employee
  $emp = new employee();

  // Присваиваем значения членам класса
  $emp->surname    = "Борисов";
  $emp->name       = "Игорь";
  $emp->patronymic = "Иванович";
  // $emp->age = 23; // Ошибка

  // Выводим члены класса
  echo $emp->surname." ".$emp->name." ".$emp->patronymic."<br>"
?>
```

Как видно из листинга 2.9, при обращении к члену класса используется оператор `->`, после которого следует ввести имя этого члена. Заметьте, что при обращении к членам класса не добавляется символ `$`.

Обращение к закрытому члену класса `$age` завершится ошибкой **Fatal error: Cannot access private property employee::$age**. На рис. 1.2 приводится схематическое изображение процесса доступа к членам объекта, снабженным разными спецификаторами доступа.

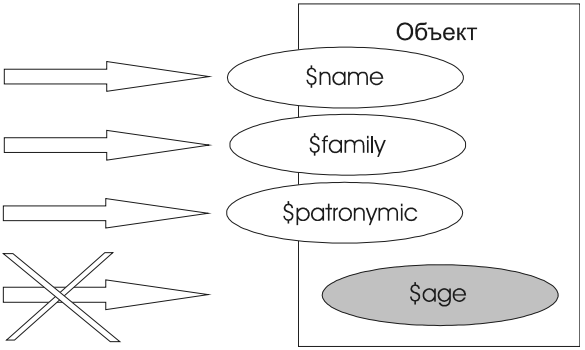


Рис. 1.2. Открытые и закрытые члены объекта

1.5. Методы класса. Член *$this*

Популярность объектно-ориентированного подхода заключается в том, что классы представляют собой не просто контейнеры для хранения переменных (в качестве таких контейнеров в PHP могут выступать ассоциативные массивы), но также позволяют включать методы, обрабатывающие как открытые, так и закрытые члены класса.

Метод класса представляет собой обычную функцию PHP, которую предваряет один из спецификаторов доступа. В листинге 1.7 приводится пример класса `cls_mth`, в состав которого входит метод `show_message()`. Задача метода сводится к выводу в окно браузера сообщения **Hello world!**.

Замечание

В листинге 2.12 можно опустить спецификатор доступа `public` перед методом `show_message()`. В отличие от других объектно-ориентированных языков, если в PHP не уточняется спецификатор, считается, что метод объявлен со спецификатором `public`. Именно потому, что в разных языках программирования используются разные подходы, не рекомендуется опускать спецификатор доступа при объявлении метода.

Листинг 1.7. Объявление метода класса

```
<?php
  class cls_mth
  {
    public function show_message()
    {
      echo "Hello world!";
    }
  }

  $obj = new cls_mth();

  $obj->show_message();
?>
```

В листинге 1.7 метод класса не использует никаких членов, однако, как правило, методы вводят именно для обработки членов класса. Для обращения к любому элементу класса внутри метода следует применять конструкцию `$this->`. Член `$this`, который неявно присутствует в каждом классе, является ссылкой на текущий объект класса.

Вернемся к классу `employee`, определенному в листинге 1.5. Класс содержит закрытую переменную `$age`, определяющую возраст сотрудника. Создадим четыре метода:

- `get_age()` — метод, возвращающий значение закрытого члена `$age`;
- `set_age()` — метод, возвращающий значение закрытого члена `$age` и проверяющий его принадлежность к числовому типу данных и промежутку от 18 до 65;
- `get_info()` — метод, возвращающий фамилию и имя сотрудника;
- `get_full_info()` — метод, возвращающий фамилию, имя и возраст сотрудника.

> ***Замечание***
>
> Следует обратить внимание на то, что открытые методы и члены расположены в начале класса, а закрытые — в конце. Это негласное правило позволяет сосредоточить внимание программиста в первую очередь на открытом интерфейсе.

Листинг 1.8. Использование открытых методов для доступа к закрытому члену класса

```
<?php
  class employee
  {
    // Открытые члены
    public $surname;
    public $name;
    public $patronymic;

    // Открытые методы
    public function get_age()
    {
      return $this->age;
    }
    public function set_age($val)
    {
      $val = intval($val);
      if($val >= 18 && $val <= 65)
      {
         $this->age = $val;
         return true;
      }
      else return false;
    }
    public function get_info()
    {
      return $this->surname." ".$this->name." ".$this->patronymic;
    }
    public function get_full_info()
    {
      return "{$this->get_info()} ({$this->get_age()})";
    }

    // Закрытые члены
    private $age;
  }
?>
```