МАКСИМ КУЗНЕЦОВ
ИГОРЬ СИМДЯНОВ

bhv®

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ на PHP

КЛАССЫ И ОБЪЕКТЫ

ИСКЛЮЧЕНИЯ

ОТРАЖЕНИЯ

FRAMEWORK

ОБЪЕКТНО-ОРИЕНТИРОВАННАЯ CMS

ПОДВОДНЫЕ КАМНИ ООП

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Максим Кузнецов
Игорь Симдянов

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ на PHP

Санкт-Петербург
«БХВ-Петербург»
2008

УДК 681.3.068+800.92PHP
ББК 32.973.26-018.1
К89

Кузнецов, М. В.

К89 Объектно-ориентированное программирование на PHP /
М. В. Кузнецов, И. В. Симдянов. — СПб.: БХВ-Петербург, 2008. —
608 с.: ил. + CD-ROM — (Профессиональное программирование)

ISBN 978-5-9775-0142-2

Книга предоставляет наиболее полное описание объектно-ориентированных возможностей PHP.

Предполагается, что читатель знаком с базовыми возможностями PHP, языком разметки HTML и приемами работы с СУБД MySQL. Даны основы объектно-ориентированного подхода: классы, специальные методы классов, инкапсуляция, наследование и полиморфизм, интерфейсы, статические, константные и final члены класса, особенности клонирования и длительного хранения объектов, обработка исключений и др. Рассмотрена практика объектно-ориентированного программирования на примерах — от построения собственного Framework (набора классов, облегчающих разработку Web-приложений) и до создания собственной объектно-ориентированной системы управления контентом (CMS). На прилагаемом компакт-диске находятся исходные коды всех Web-приложений, рассматриваемых в книге.

Для Web-разработчиков

УДК 681.3.068+800.92PHP
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Ирина Артемьева*
Компьютерная верстка	*Ольги Сергиенко*
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 24.09.07.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 49,02.
Тираж 2500 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 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 978-5-9775-0142-2

Оглавление

Введение

Начинающие и опытные программисты часто задаются вопросом: что дает объектно-ориентированное программирование? Освоив его, получу ли я какое-то конкурентное преимущество по сравнению с другими разработчиками? Если для объектно-ориентированных языков программирования, таких как Java или C++, ответ однозначен, то в Web-среде, больше ориентированной на процедурный стиль, ответить на этот вопрос не так просто.

Между тем все большее количество библиотек переходит на объектно-ориентированный интерфейс, вынуждая разработчиков обращаться к объектно-ориентированным возможностям PHP. Введение в пятой версии PHP полноценной объектно-ориентированной модели еще больше подогревает интерес к этой методологии.

Книга, которую вы держите в руках, позволяет найти ответ на вопросы: зачем нужно объектно-ориентированное программирование в PHP, когда его следует применять, а когда его применение не целесообразно и даже вредно.

Зачастую использование объектно-ориентированного подхода к месту и не к месту не обязательно делает проект успешным. Программирование новичка в стиле объектно-ориентированного программирования часто напоминает передвижение по минному полю — если не знать где мины, достичь конца проекта невозможно. Само по себе объектно-ориентированное программирование не является панацеей — это рабочая технология, которая позволяет:

- увеличить процент повторно используемого кода;
- оперировать при программировании понятиями и объектами реального мира (договор, заключение договора, распечатка договора, поиск договора), а не низкоуровневыми компьютерными терминами (файлы, строка, стандартный поток вывода), что позволяет создавать более крупные проекты с меньшим количеством ошибок и в более сжатые сроки.

Предлагаемая книга рассматривает объектно-ориентированное программирование применительно к PHP, раскрывая его методологическую часть. Будет определено, в каком случае следует взять за основу объектно-ориентированный подход в PHP, а в каком — лучше от него отказаться. На примере построения большого Web-приложения (CMS) демонстрируется, как добиться повторного использования кода в реальных проектах. Рассматриваемая система управления содержимым сайта (CMS) используется сотрудниками нашей студии SoftTime (**http://www.softtime.ru**) для построения сайтов (например, на рассмотренной в данной книге объектно-ориентированной CMS построен сайт нашего нового проекта **http://www.bipsi.ru**).

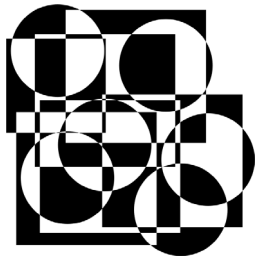
Дополнительные материалы можно также найти на группе сайтов IT-студии SoftTime, сотрудниками которой являются авторы книги:

- **http://www.softtime.ru** — головной сайт нашей студии;
- **http://www.softtime.biz** — услуги, предоставляемые студией;
- **http://www.softtime.org** — различные проекты студии;
- **http://www.softtime.mobi** — мобильные версии материалов и форумов.

Обновленные версии приложений и SoftTime FrameWork можно загрузить по адресу **http://www.softtime.ru/info/downloads.php**, а обсудить вопросы, которые могут возникнуть по мере чтения материала книги, можно на форуме авторов **http://www.softtime.ru/forum/**.

Благодарности

Авторы выражают признательность сотрудникам издательства "БХВ-Петербург", благодаря которым наша рукопись увидела свет, а также посетителям форума **http://www.softtime.ru/forum/** за интересные вопросы и конструктивное обсуждение.



ГЛАВА 1

Введение в объектно-ориентированное программирование

Объектно-ориентированное программирование на сегодняшний день является одной из самых популярных методик создания программного обеспечения. Однако чтобы использовать ее по назначению и с наибольшей отдачей, необходимо четко представлять себе причины возникновения этой технологии и область ее применения.

1.1. Причины возникновения объектно-ориентированной технологии

Развитие технологий программирования, как метко заметил Дейкстра, диктуется тезисом "Разделяй и властвуй". Любые удачные технологии предполагают, что чем короче код программы, тем легче его создавать, отлаживать и сопровождать, а простая программа подвержена ошибкам в гораздо меньшей степени, чем сложная.

На заре компьютерной эры программа представляла собой один поток, который обрабатывал один массив данных (рис. 1.1).

Замечание

На рисунках в данном разделе поток обозначается стрелкой, в то время как массив данных — квадратом.

Со временем сложность программ и предъявляемых к ним требований возросли, и такой способ организации данных оказался неприемлемым. Был предложен структурный подход, при котором массив данных становился доступен из любой точки программы, однако основной поток программы разбивался на несколько процедур. Отдельную небольшую процедуру, пусть даже

использующую общие данные, разрабатывать гораздо проще, чем большой объем кода (рис. 1.2).

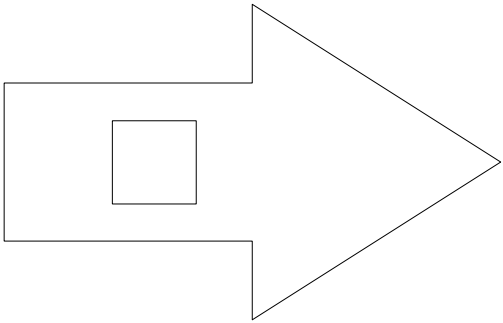


Рис. 1.1. Один поток обрабатывает один массив данных

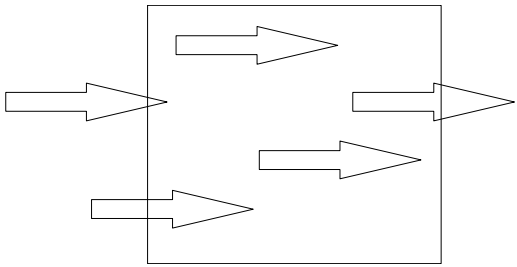


Рис. 1.2. Несколько потоков обрабатывают один массив данных

Каждая из процедур обладает локальными переменными, срок жизни которой определяется продолжительностью работы процедуры. Одни процедуры могут вызывать другие, однако массив данных в программе остается общим и доступным для всех процедур. Такой подход позволил создавать еще более значительные программные комплексы. В результате объем данных вырос еще больше, а для его обработки потребовались еще более гибкие способы масштабирования программ, учитывающие разбиение данных.

Ответом на все возрастающую сложность стало появление объектно-ориентированного подхода в программировании: программа разбивается на несколько массивов данных, каждый из которых имеет свои собственные процедуры, а также процедуры, которые взаимодействуют с другими массивами данных.

В результате сложная задача разбивается на ряд более простых подзадач, а разработчики получают более гибкий способ управления проектом: редактировать один огромный монолитный блок кода гораздо сложнее, чем совокупность небольших, слабо связанных между собой блоков.

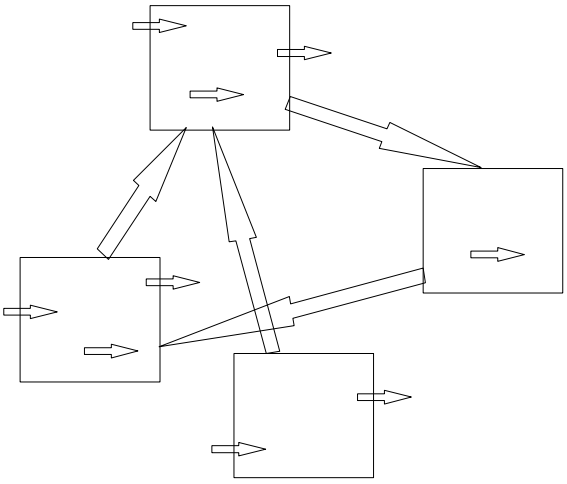


Рис. 1.3. Несколько массивов данных обрабатываются отдельным потоком

Замечание

Примечательно, что подход к программированию, использующий слабо связанные, взаимодействующие между собой блоки, был принят при создании операционной системы UNIX, в которой для решения задачи несколько программ выстраиваются в цепочку, тогда как в других операционных системах, таких как Windows, каждая программа представляет собой монолит, практически не взаимодействующий с другими программами. Именно блочный подход в UNIX объясняет его долголетие (около 40 лет): цепочки программ можно перестраивать, использовать элементы старых цепочек для построения новых, благодаря чему операционная система очень быстро приспосабливается к новому аппаратному обеспечению. Точно так же и объектно-ориентированный подход позволяет создать программное обеспечение с более гибкими и легко модифицируемыми свойствами.

1.2. Какая программа является объектно-ориентированной?

Объектно-ориентированная технология не привязана к языку программирования. Даже если язык не поддерживает ее в явном виде, с его помощью можно создать объектно-ориентированную систему. Именно так создавались первые версии операционной системы Windows: несмотря на то что язык C (не путать с C++) не содержал конструкций, поддерживающих объектно-ориентированный подход, разработчикам удалось организовать код таким образом, что он, по сути, был объектно-ориентированным.

Конечно, гораздо удобнее создавать объектно-ориентированные программы, если объектно-ориентированная методология поддерживается на уровне язы-

ка программирования. Такая поддержка избавляет от необходимости программирования рутинных операций и использования сложных нестандартных библиотек.

С другой стороны, синтаксиса объектно-ориентированного программирования недостаточно, чтобы называть программу объектно-ориентированной. Можно привести множество примеров прикладных программ (в том числе и на PHP), использующих классы, объекты, но не являющихся по своей природе объектно-ориентированными. Объект в таких приложениях выступает в качестве удобного контейнера, представляющего все приложение, организация кода в этом случае опять же сводится к одному блоку данных и одному обрабатывающему их потоку.

Независимо от привязки к языку программирования, объектно-ориентированный подход имеет ряд общих принципов, а именно:

- возможность создавать *абстрактные типы данных*, позволяющая наряду с предопределенными типами данных (таких 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 подобная операция становится невозможной;
- *инкапсуляция*, ограничивающая взаимодействие пользователя абстрактных типов данных только их интерфейсом 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 и 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.

Замечание

Программа, отвечающая только одному принципу, не обязательно является объектно-ориентированной. Например, абстрактные типы данных появились задолго до объектно-ориентированного подхода. Если один из приведенных выше принципов оказывается лишним в объектной схеме программы — это повод задуматься: а требуется ли вообще для решения этой задачи объектно-ориентированный подход?

Следовать этим нехитрым правилам можно без поддержки объектно-ориентированного подхода на уровне языка программирования. Доказательством служит операционная система Windows 95, разработанная на языке C, благодаря которой корпорация Microsoft заняла ведущие позиции в мире.

Замечание

Язык программирования C поддерживает абстрактные типы данных, однако инструментальная поддержка инкапсуляции, наследования и полиморфизма в нем отсутствует.

На самом деле, на каком бы языке программирования мы ни работали, мы имеем дело с абстрактными данными — программируя систему документооборота, мы в любом случае будем иметь дело с документами и волей-неволей абстрагироваться от машинных данных.

Инкапсуляция существует с момента создания первых структурных программ: модули и функции также скрывают реализацию, получая на входе данные и выдавая на выходе результат; пользующийся ими программист может не вникать в принципы их работы.

Наследование тоже в какой-то мере использовалось и до появления объектно-ориентированного подхода: программисты всегда стремились использовать ранее наработанный код в новых проектах.

Полиморфизм можно организовать в любой программе, для этого следует просто правильно организовать именование процедур и функций, так чтобы имя предполагаемой процедуры можно было "вычислить" автоматически.

Тем не менее, за последние 20 лет почти каждый "живой" язык программирования был снабжен дополнительными ключевыми словами в целях поддержки объектно-ориентированного подхода. Для чего же понадобились новые обозначения уже существующих концепций?

1.3. О важности обозначений

Без сомнения, каждому, читающему эти строки, знакомо выражение:

$$2 + 2 = 4.$$

Благодаря всеобщему образованию, таблица умножения также не вызывает затруднений:

$$6 \times 8 = 48.$$

Однако в античные времена доказательство последнего утверждения занимало множество листов и тянуло на кандидатскую диссертацию.

Благодаря логике, созданной Аристотелем, а затем развитой арабами и отточенной схоластами, на сегодняшний день у нас есть научный язык и система математических обозначений. Решение сложнейших для античности задач сейчас не является ни доблестью, ни даже правилом хорошего тона — это неотъемлемая часть культуры. При этом таблица умножения усваивается нами на подсознательном уровне — мы не вычисляем ответ, а просто знаем его и не задумываемся, откуда он берется.

Это не значит, что мыслители древности были менее способными, просто для решения задач они пользовались другим математическим аппаратом. Как известно, существует два способа решения сложной задачи:

- сложное решение задачи простыми способами;
- простое решение задачи сложными способами.

В последнем случае под сложным способом понимается не запутанный, а более высокотехнологичный и высокоэффективный способ. Как правило, на освоение такого способа требуется потратить время.

Поиск строки в тексте по сложному условию может занимать десятки и сотни строк кода с использованием строковых функций. При помощи регулярных выражений задача зачастую решается в одну строку кода. Однако для того чтобы решать задачу при помощи регулярных выражений, их следует освоить, на что уходит немало времени. Многие не без успеха используют вместо регулярных выражений строковые функции (то есть решают сложную задачу простыми способами). Однако программисты, потратившие время на изучение регулярных выражений, практически никогда больше не решают задачи поиска при помощи строковых функций.

Объектно-ориентированный подход тоже требует длительного усвоения, однако взамен предоставляет преимущества, а именно позволяет создавать простые решения для сложных задач, практически не разрешимых в рамках структурного подхода.

Рассмотрим пример PHP-программы. В листинге 1.1 перемножаются два числа: число 6, которое хранится в таблице `number`, и число 8, которое, в свою очередь, хранится в таблице `blocks`.

Листинг 1.1. Умножение двух чисел, находящихся в разных таблицах базы данных

```
<?php
  // Адрес сервера MySQL
  $dblocation = "localhost";
  // Имя базы данных на хостинге или локальной машине
  $dbname = "test";
  // Имя пользователя базы данных
  $dbuser = "root";
  // и его пароль
  $dbpasswd = "";

  // Устанавливаем соединение с базой данных
  $dbcnx = @mysql_connect($dblocation, $dbuser, $dbpasswd);
  if (!$dbcnx) {
   exit( "<P>В настоящий момент сервер базы данных не доступен,
             поэтому корректное отображение страницы невозможно.</P>" );
  }
  // Выбираем базу данных
  if (! @mysql_select_db($dbname, $dbcnx) ) {
    exit( "<P>В настоящий момент база данных не доступна,
              поэтому корректное отображение страницы невозможно.</P>" );
  }

  // Извлекаем число из таблицы number
  $query = "SELECT num FROM number";
  $num = mysql_query($query);
  if(!$num) exit("Ошибка обращения к таблице number");
  if(!mysql_num_rows($num)) exit("Отсутствуют записи в таблице number");
  $number = mysql_result($num, 0);

  // Извлекаем число из таблицы blocks
  $query = "SELECT num FROM blocks";
  $blk = mysql_query($query);
  if(!$blk) exit("Ошибка обращения к таблице blocks");
  if(!mysql_num_rows($blk)) exit("Отсутствуют записи в таблице blocks");
  $blocks = mysql_result($blk, 0);

  echo $number * $blocks;
?>
```

Как можно заметить, для умножения числа 6 на 8 вместо одной строки понадобилось 38 строк кода. Нетрудно догадаться, какого размера достигнет законченное приложение, если будет реализовано подобным образом.

Задача объектно-ориентированного подхода состоит в сокращении кода, так чтобы реализация промежуточных операций не отвлекала программиста от решения поставленной задачи (листинг 1.2).

Листинг 1.2. Объектно-ориентированный подход

```
<?php
  $number = new num("number");
  $blocks = new num("blocks");
  echo $number.num * $blocks.num;
?>
```

Важно понимать, что сразу 38 строк преобразовать в пять не удастся. Если сложить все строки, которые необходимы для функционирования кода из листинга 1.1, его объем наверняка превысит код из листинга 1.2. Сэкономить можно только на очень больших приложениях и, возможно, на их комплексах.

1.4. Терминология

Ключевыми понятиями объектно-ориентированного подхода являются объект и класс, которые находятся в тесной связи с понятиями переменной и ее типа. В листинге 1.3 приведены примеры объявления целочисленной переменной `$val` и строковой переменной `$str`.

Замечание

Язык PHP не является строго типизированным, одну и ту же переменную можно использовать для хранения целого числа, строкового значения и числа с плавающей точкой. В связи с этим при объявлении переменной не требуется указывать ее тип. Однако это вовсе не означает, что тип переменной отсутствует вовсе — просто переменные принимают тип своего текущего значения. Изменить тип переменной можно при помощи функций `is_bool()`, `is_float()`, `is_int()`, `is_numeric()` и `is_string()`.

Листинг 1.3. Объявление целочисленной и строковой переменных

```
<?php
  $val = 1;
  $str = "Hello world";
?>
```

Переменная и ее тип — это низкоуровневые машинные понятия. Переменная представляет собой область памяти, в которой хранится ее значение, а тип — выделяемый для ее хранения объем памяти. Используя множество переменных и обрабатывающих их функций, можно моделировать объекты и процессы реального мира. Однако мало построить объекты, необходимо еще определить способы их взаимодействия. 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 разработать свой мини-язык путем создания классов и их объектов. Переменными такого мини-языка программирования выступают 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. После того как класса разработан, можно объявить несколько экземпляров объекта или даже их массив.

Замечание

Элементы объектно-ориентированного программирования начали вводиться практически сразу после становления структурного подхода. Понятие классов и объектов впервые введены в языке программирования Simula 67. Считается, что окончательно объектно-ориентированный подход сформировался в языке программирования Smalltalk, который и явился прообразом всех современных объектно-ориентированных языков.

Таким образом, объект — это своеобразная макропеременная, поведение которой определяется программистом при помощи класса, выступающего в качестве типа макропеременной. В результате программист поднимается на более высокий уровень абстракции, оперируя не низкоуровневыми компьютерными терминами (переменная, функция, число, строка), а высокоуровневыми конструкциями (объект, его поведение, например, договор, клиент и т. п.).

Таким образом, не прибегая к сложным инструментам вроде создания специализированного языка высокого уровня, программист *расширяет* уже существующий объектно-ориентированный язык программирования, вводя в него новые типы данных (классы), переменные которого (объекты) обладают необходимым для решения текущей задачи поведением.

Г. Буч определяет объектно-ориентированное программирование следующим образом: *объектно-ориентированное программирование* — это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Каждый из классов разрабатывается отдельно и включает в себя переменные, которые называют *членами класса*, и функции, которые называют *методами класса*. В качестве члена класса может выступать объект другого класса, таким образом, можно создавать действительно сложные объекты.

1.5. Повторное использование кода

Классы должны обеспечивать повторное использование кода. В идеале класс, созданный в одном приложении, должен свободно извлекаться из него и встраиваться в другое приложение. Однако добиться этого на практике достаточно сложно. Без тщательного проектирования и жесткого контроля классы норовят связаться друг с другом всеми возможными способами (рис. 1.4).

Замечание

На рисунках в данном разделе класс обозначается квадратом, а его интерфейс, позволяющий ему взаимодействовать с другими классами, — стрелкой.

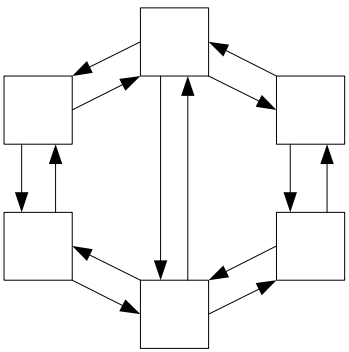


Рис. 1.4. Сильно связанная система

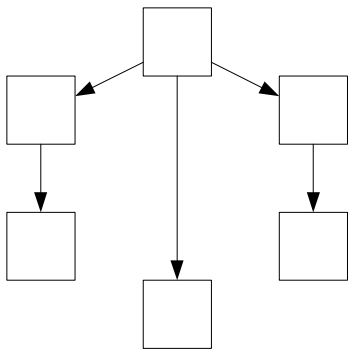


Рис. 1.5. Слабо связанная система

В результате извлечь класс для повторного использования в другом приложении можно только "с мясом" (то есть вместе с ворохом связанных с ним классов). Одна из важнейших задач разработчика состоит в проектировании системы, которая имела бы по возможности как можно меньше связей (так называемое слабое связывание) (рис. 1.5).

Дело в том, что проявить всю свою силу объектно-ориентированный подход может только в том случае, если классы слабо связаны между собой и могут разрабатываться независимо друг от друга.

Чем меньше связей между отдельными подсистемами, тем надежнее система и тем проще ее сопровождать, т. к. ее общая сложность в этом случае уменьшается. Когда каждая подсистема взаимодействует с другими подсистемами,

программисту необходимо знать особенности их работы и влияние вносимых изменений на другие части системы; когда же такое взаимодействие сведено к минимуму, программисту приходится учитывать меньшее количество связей, и вероятность совершения ошибок уменьшается.

Замечание

В 1956 г. Миллер в статье "The Magical Number Seven, Plus or Minus Two" показал, что человек способен удерживать в краткосрочной памяти семь объектов плюс-минус два. Это означает, что для эффективной работы число связей, которые должен учитывать программист при разработке того или иного блока, не должна превышать 5—9. (Со статьей "The Magical Number Seven, Plus or Minus Two" можно ознакомиться по адресу **http://www.well.com/user/smalin/miller.html**.)

Объектно-ориентированный подход предоставляет разработчику две возможности повторного использования кода:

- *композиция* — включение в класс любого количества объектов произвольного типа. Наряду с переменными встроенных типов, класс может содержать объекты других типов точно так же, как объекты реального мира могут содержать в своем составе другие объекты. Например, объект "государство" содержит объекты "области" и "края", которые, в свою очередь, включают в качестве объектов "населенные пункты". Объект "населенный пункт" может содержать объекты "улицы", в которые могут входить объекты "дома" и, наконец, последние могут включать в себя объекты "квартиры";
- *наследование* — расширение существующего класса путем наследования от него производного класса. Класс "транспортное средство" может быть расширен до класса "автомобили", который будет включать в себя все члены и методы обобщенного транспортного средства; в свою очередь "автомобиль" может быть расширен до классов "легковой автомобиль" и "грузовой автомобиль".

На рис. 1.6 схематически представлена схема композиции. В объект класса может входить набор некоторых объектов, которые также могут включать другие объекты.

Наследование позволяет расширять уже существующий класс, при этом получаемый новый класс автоматически содержит все члены и методы старого класса. Старый класс при этом называют базовым классом, а новый — производным классом. На рис. 1.7 представлена схема наследования.

Наследование позволяет избежать утомительной работы по реализации методов и членов для похожих классов, которые отличаются друг от друга только незначительными деталями.

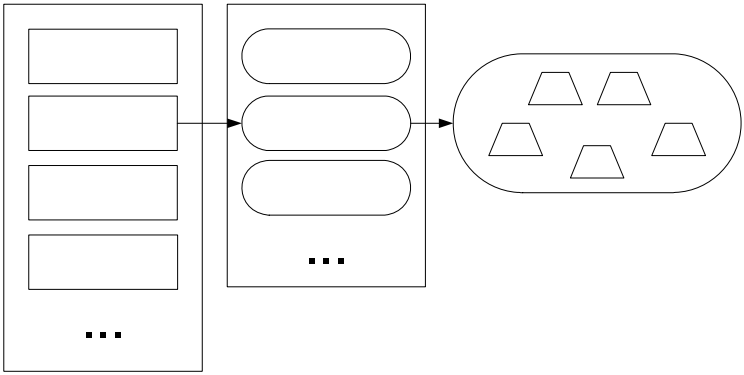


Рис. 1.6. Композиция

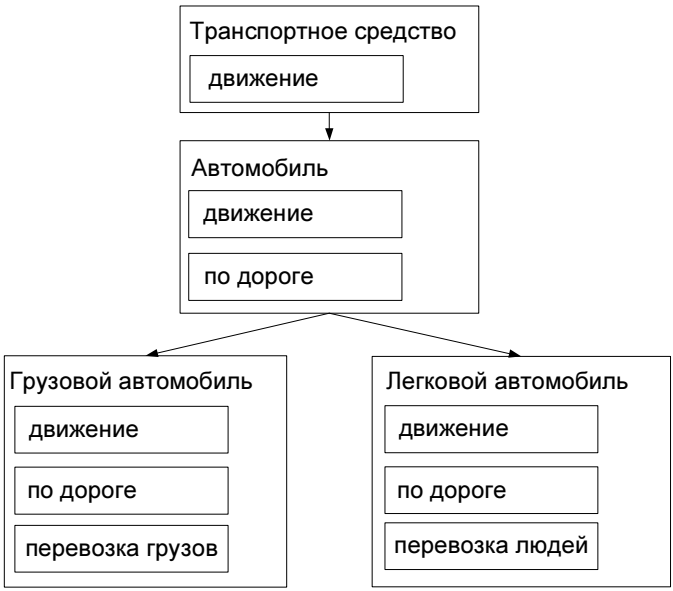


Рис. 1.7. Наследование

1.6. Недостатки объектно-ориентированного подхода

Обычно книга, посвященная какой-либо технологии, рассказывает только о ее достоинствах и сильных сторонах. Автор, негативно относящийся к использованию какого-либо подхода, обычно не берется за написание книги и даже за детальное изучение технологии, если обнаруживает в ней крити-

ческое количество недостатков. В результате вокруг технологии формируется радужный ореол, который разрушается, когда в проект уже вложено значительное количество сил, средств и времени. Поэтому следует упомянуть о подводных камнях объектно-ориентированного подхода в PHP и сразу развеять ряд устойчивых мифов.

Объектно-ориентированный подход — это не панацея. Не стоит думать, что, используя объектно-ориентированный подход, вы автоматически избавляетесь от ошибок проектирования. Запутаться в плохо спроектированных классах так же легко, как разработать неудачное структурное приложение. Если вы используете объектно-ориентированный подход просто ради его использования и не можете ответить, какие преимущества он предоставит вам по сравнению со структурным подходом, вы ничего не получите, кроме увеличения времени разработки, снижения читабельности программы и ее эффективности.

Снижение производительности. За любую функциональность следует расплачиваться либо производительностью, либо читабельностью, либо эффективностью, либо всем вместе взятым. Например, операционные системы реального времени менее эффективны своих "неточных" аналогов. Причина этого заключается в том, что для обеспечения точного момента срабатывания система вынуждена простаивать вплоть до наступления временного репера: в данном случае мы расплачиваемся производительностью за точность. Аналогично обстоит дело и с объектно-ориентированным подходом — создание и удаление объектов требует времени и памяти. Следует отметить, что только один язык не теряет эффективности при использовании объектно-ориентированного подхода — это C++. В PHP эффективность уменьшается в среднем на 30%.

Процедурные интерфейсы. За годы развития языков программирования и компьютерных технологий было разработано большое количество разнообразных интерфейсов, начиная с сетевых протоколов и заканчивая API СУБД. До настоящего времени объектно-ориентированные интерфейсы не получили широкого распространения в области Web-программирования, поскольку наибольшей популярностью пользуются реляционные СУБД, а объектно-ориентированные СУБД существуют лишь в концептуальной форме. В результате довольно часто приходится преобразовывать объект в линейную форму (например, перед помещением в базу данных или для отправки по Сети) и затем снова его восстанавливать. Это не прибавляет разработке приложений ни удобства, ни эффективности, ни скорости.

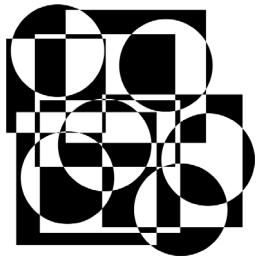
В PHP элементы объектно-ориентированного подхода не отлажены до конца. В настоящий момент существует некоторое количество ошибок, связанных с реализацией объектно-ориентированного подхода в PHP. Они не

являются критичными и их можно обойти, однако это зачастую утомительно и требует дополнительного времени и увеличивает объем кода, что снижает эффективность объектно-ориентированного подхода по сравнению со структурным, который более или менее отлажен. Ряд ошибок реализации и способы их обхода описываются в следующих главах.

1.7. Объектно-ориентированное программирование в PHP

Объектно-ориентированное программирование в PHP появилось сравнительно недавно, начиная с четвертой версии. Однако можно утверждать, что в четвертой версии PHP объектно-ориентированная модель была реализована в крайне ограниченных объемах, отсутствовал деструктор, спецификаторы доступа, исключения. Начиная с пятой версии PHP, объектно-ориентированная форма получила современное воплощение, поэтому в данной книге будет представлена объектно-ориентированная модель PHP 5.0 и 5.1.

ГЛАВА 2



Объекты и классы

Ключевыми конструкциями объектно-ориентированной технологии являются классы и объекты. В данной главе будет подробно рассмотрен процесс их создания и использования.

2.1. Объявление класса

Класс объявляется при помощи ключевого слова `class`, после которого следует уникальное имя класса и тело класса в фигурных скобках. В теле класса объявляются методы и члены. В листинге 2.1 приводится общий синтаксис объявления класса.

Листинг 2.1. 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()`. Объявление класса `cls` в листинге 2.2 является ошибочным.

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. В отличие от `require()`, при отсутствии включаемого файла инструкция `include()` генерирует предупреждение, однако не останавливает работу скрипта, в то время как `require()` 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 включаемый файл будет включен лишь один раз, повторный вызов инструкции `include_once()` или `require_once()` игнорируется. Это особенно удобно для включения библиотек функций и классов, повторное объявление которых вызывает ошибку.

Листинг 2.2. Недопустимое объявление класса

```
<?php
  class cls
  {
    // Реализация класса во внешнем файле
    include "cls_include.php";
  }
?>
```

Такое поведение может обескуражить опытных программистов, привыкших иметь дело с объектно-ориентированным программированием на других языках. Язык C++, например, позволяет объявлять лишь прототипы методов, вынося реализацию в отдельный файл. Таким образом, внешний программист имеет дело только с интерфейсом класса, и у него отсутствует соблазн разбираться в его внутренней реализации, нарушая принцип инкапсуляции (сокрытия реализации).

Впрочем, имеется способ обойти подобное ограничение, т. к. в методах класса не запрещается использование конструкций `include()`, `include_once()`, `require()`, `require_once()`. Забегая вперед, представим листинг 2.3, демонстрирующий, как реализация метода `test()` класса `cls()` размещается в отдельном файле test_cls.php.

Замечание

Более подробно использование конструкций `include()`, `include_once()`, `require()`, `require_once()` обсуждается в *разделе 2.4*.

Листинг 2.3. Корректное использование в классе инструкции `include()`

```
<?php
  class cls
  {
    function test()
    {
      // Реализация метода во внешнем файле
      include "test_cls.php";
    }
  }
?>
```

Нельзя обойти стороной размещение в приложении и самих классов. Очевидно, что классы в объектно-ориентированном приложении будут использоваться множеством файлов, поэтому разумно выделить их в отдельный файл, который бы включался в остальные файлы при помощи инструкций `include_once()` или `require_once()`. Важно с самого начала использовать для включения инструкции с суффиксом `once`, чтобы предотвратить повторное включение самого файла. В небольшом прозрачном приложении практически невозможно ошибиться и допустить ошибку, связанную с повторным включением файла с объявлением класса, однако в большие порталы входит множество более мелких приложений, и не исключен случай попытки повторного ввода класса.

Файлы, описывающие классы, могут носить произвольные имена. В отличие от других языков программирования, требующих, чтобы содержащий класс файл имел то же название, что и сам класс, в PHP имя класса никак не связано с именем файла, в котором класс размещен. Объектно-ориентированное приложение включает множество классов, поэтому для простоты ориентирования лучше сразу продумать систему именования файлов. В данной книге каждый класс будет размещаться в отдельном файле, имя которого будет начинаться с ключевого слова `class`, после чего будет следовать название файла. Например, файл для класса из листинга 2.3 можно назвать class.test.php. Позже мы дадим пример механизма наследования, чтобы показать, как класс `derived` наследует класс `base`; имя файла данного класса будем называть class.base.derived.php.

Замечание

PHP предоставляет разработчику автоматическое средство для автоматической загрузки классов в момент создания объектов. Это достигается за счет функции `__autoload()`, которая рассматривается в *разделе 3.6*.

2.2. Абстрактные типы данных. Создание объекта

Согласно одной из концепций объектно-ориентированного программирования, классы выступают как настраиваемые пользователем типы данных. Программист получает возможность разрабатывать программу не при помощи компьютерных абстракций (переменных, функций, потоков и т. п.), а при помощи абстракций предметной области программы.

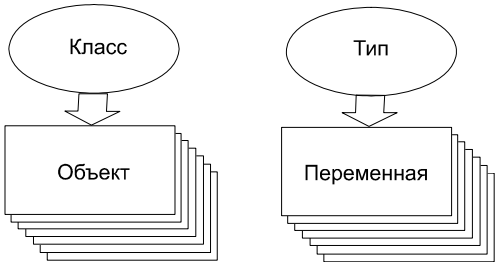


Рис. 2.1. Переменные объявляются при помощи типа, объекты — при помощи класса

Тип переменной является ключевым словом, определяющим область памяти с определенными техническими характеристиками (объем отводимой памяти, способность хранить числа с плавающей точкой, последовательность символов и т. д.). Класс также выступает в качестве своеобразного типа данных, при помощи которого можно объявлять объекты (рис. 2.1), однако в отличие от нескольких предопределенных типов, заложенных в интерпретатор PHP на этапе компиляции, функциональность класса определяет программист, причем количество классов не ограничено.

Аналогично скриптам, которые могут создавать неограниченное количество переменных одного типа, может быть создано неограниченное количество объектов одного и того же класса. В общем случае это может быть массив однотипных объектов.

Объявление объекта класса несколько отличается от объявления переменной. Следует напомнить, что благодаря слабой типизации объявлять переменные и уточнять их тип в PHP вообще не требуется: переменную можно ввести в любой момент, а впоследствии изменить ее тип (листинг 2.4).

Листинг 2.4. В PHP объявление переменной не требуется

```
<?php
  // Указывать тип переменной не обязательно
  $str = "Hello world!";
  $var = "test";
```

```
  // Создаем переменную $test с содержимым "Hello world"
  $$var = $str;
  echo $test;
  // Изменяем тип переменной на числовой
  $var = 5;
  echo "<br>$var";
?>
```

Результатом работы скрипта из листинга 2.4 являются следующие строки:

```
Hello world!
5
```

Для объявления объекта класса следует использовать ключевое слово `new`, которое выделит под него необходимую область памяти. Освобождать данную область памяти самостоятельно (как в языке программирования C++) не требуется, интерпретатор сам ее освободит. Впрочем, если скрипту не хватает памяти, ее можно освободить при помощи конструкции `unset` (см. *раздел 2.17*) в любой момент, не дожидаясь окончания работы скрипта.

Замечание

Оператор `new` является атавизмом, пришедшим в PHP из языка программирования C++. В последнем при помощи этого оператора выделяется память под объекты, которая после завершения работы должна освобождаться при помощи оператора `delete`. В PHP программист напрямую не участвует в процессе управления памятью, однако ключевое слово `new` было оставлено, т. к. PHP не знает заранее размер пользовательского объекта, и ему необходимо явно указать, что сейчас будет производиться создание объекта (а не предопределенной переменной) и требуется найти его класс.

В листинге 2.5 создается пустой класс `emp` и объявляется объект `$obj` данного класса.

Замечание

Обратите внимание, что в PHP после завершающей фигурной скобки класса не требуется точки с запятой (в отличие от C++, где она обязательна).

Листинг 2.5. Создание класса `emp` и объявление объекта `$obj` данного класса

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

Как видно из листинга 2.5, при помощи ключевого слова `new` переменной `$obj` присваивается в качестве значения новый объект класса `emp`. После имени