ГЛАВА 6

Объекты

В этой главе вы научитесь определять, создавать и использовать объекты в PHP. ООП представляет элегантные архитектуры, упрощает сопровождение кода и улучшает возможности повторного использования его элементов. Методология ООП оказалась настолько полезной, что сегодня мало кто отважится создать новый язык, в котором бы отсутствовали возможности ООП. В PHP поддерживаются многие полезные средства ООП, которые вы научитесь использовать. Также мы рассмотрим базовые концепции ООП и такие высокоуровневые темы, как интроспекция и сериализация.

Объекты

ООП подчеркивает фундаментальную связь между данными и кодом, работающим с ними, и предоставляет возможность проектирования и реализации программ на основе этой связи. Например, программная система интернет-форума обычно хранит данные о разных пользователях. В процедурном языке программирования каждый пользователь представлен структурой данных, для работы с которой есть набор функций (создание пользователя, получение его данных и т. д.). В объектно-ориентированном языке каждый пользователь представлен *объектом* — структурой данных с присоединенным программным кодом, где данные и код рассматриваются как единое целое. Объект как объединение кода и данных становится модульной единицей для разработки приложений и повторного использования кода.

В гипотетической архитектуре форума объекты могут представлять не только пользователей, но и сообщения и обсуждения. Объект, представляющий пользователя, содержит имя и пароль данного пользователя, а также код нахождения всех сообщений этого пользователя. Объект сообщения знает, какому обсуждению сообщение принадлежит, и содержит код публикации нового сообщения, ответа на существующее сообщение и вывода сообщения. Объект обсуждения представляет собой набор объектов сообщений и содержит код вывода структу-

ры ветвей обсуждения. Впрочем, это всего лишь один из возможных способов распределения необходимой функциональности по объектам. Например, в альтернативном варианте архитектуры код публикации новых сообщений может храниться в объекте пользователя, а не в объекте сообщения.

Проектирование систем ООП — сложная тема, о которой написано множество книг. К счастью, какой бы вариант проектирования вы ни выбрали, этот вариант можно реализовать на PHP. Начнем с введения некоторых ключевых терминов и концепций, которые необходимо знать перед началом изучения этой методологии программирования.

Терминология

Иногда создается впечатление, что в каждом объектно-ориентированном языке существует собственный набор терминов для одних и тех же концепций. В этом разделе описаны термины, используемые в PHP, однако учтите, что в других языках эти термины могут иметь другой смысл.

Вернемся к интернет-форуму. Для каждого пользователя необходимо хранить одни и те же атрибуты, и для структуры данных каждого пользователя должны вызываться одни и те же функции. При разработке программы вы определяете поля и функции, которые должны храниться для каждого пользователя, — в терминологии ООП проектируете *класс* пользователя, то есть паттерн для построения объектов.

Объектом называется экземпляр, созданный на основе класса. В нашем примере это структура данных конкретного пользователя с присоединенным кодом. Объекты и классы отчасти похожи на значения и типы данных: например, есть один тип данных «целое число», но много возможных целых чисел, и аналогичным образом программа может определять один класс пользователя, но создавать множество идентичных пользователей.

Данные, связанные с объектом, называются его *свойствами*. Функции, связанные с объектом, называются его *методами*. Определяя класс, вы определяете имена его свойств и пишете код его методов.

Отладка и сопровождение программ значительно упрощаются при использовании *инкапсуляции*. Концепция инкапсуляции заключается в том, что класс предоставляет коду, использующему его объекты, специальные методы (интерфейс), чтобы внешний код не обращался напрямую к структурам данных этих объектов. Если вы знаете, где следует искать ошибки (весь код, изменяющий структуры данных объекта, находится внутри класса), то можете свободно заменять реализации класса без изменения кода, который работает с этим классом, при условии, что интерфейс остается неизменным.

В любой нетривиальной архитектуре ООП обычно используется *наследование*. Определяя новый класс с использованием наследования, вы указываете, что он похож на существующий класс, но содержит новые или измененные свойства и методы. Исходный класс называется *суперклассом* (а также родительским, или базовым, классом), а новый класс называется *подклассом* (дочерним, или производным). Наследование является формой повторного использования кода, поскольку код суперкласса используется повторно, а не копируется в подкласс. Любые усовершенствования и изменения в суперклассе автоматически передаются подклассу.

Создание объекта

Создавать объекты и работать с ними гораздо проще, чем определять классы, поэтому мы сначала рассмотрим создание объектов.

Для создания объекта заданного класса используется ключевое слово `new`:

```
$object = new Class;
```

Если предположить, что класс `Person` уже был определен, создание объекта `Person` выглядит так:

```
$moana = new Person;
```

Не заключайте имя класса в кавычки, иначе компилятор выдаст сообщение об ошибке:

```
$moana = new "Person"; // не работает
```

Некоторые классы позволяют передавать аргументы при вызове `new`. В документации класса должно быть сказано, может ли он получать аргументы. Если аргументы передаются, то команда создания объекта выглядит примерно так:

```
$object = new Person("Sina", 35);
```

Имя класса необязательно жестко фиксировать в программе. Его также можно передать в переменной:

```
$class = "Person";
$object = new $class;
// эквивалентно
$object = new Person;
```

Если заданный класс не существует, происходит ошибка времени выполнения.

Переменные, в которых хранятся ссылки на объекты, ничем не отличаются от любых других и используются точно так же. Механизм обращения к пере-

менным, имена которых хранятся в других переменных, также работает с объектами:

```
$account = new Account;
$object = "account";
${$object}->init(50000, 1.10); // то же, что $account->init
```

Обращение к свойствам и методам

После того как объект будет создан, для обращения к его методам и свойствам используется запись `->`:

```
$объект->имя_свойства $объект->имя_метода([аргумент, ... ])
```

Пример:

```
echo "Moana is {$moana->age} years old.\n"; // обращение к свойству
$moana->birthday(); // вызов метода
$moana->setAge(21); // вызов метода с аргументами
```

Методы работают так же, как функции (но только с текущим объектом), то есть они могут получать аргументы и возвращать значение:

```
$clan = $moana->family("extended");
```

В определении класса с помощью модификаторов доступа `public` и `private` можно указать, какие методы и свойства являются открытыми (общедоступными), а какие доступны только из самого класса. Так обеспечивается инкапсуляция.

Механизм косвенного обращения работает и с именами переменных:

```
$prop = 'age';
echo $moana->$prop;
```

Статическими называются методы, вызываемые для класса в целом, а не для объектов. Такие методы не могут обращаться к свойствам. Имя статического метода состоит из имени класса, за которым следуют два символа «`:`» и имя функции. Например, следующий фрагмент вызывает статический метод `p()` в классе HTML:

```
HTML::p("Hello, world");
```

При объявлении класса статические свойства и методы обозначаются модификатором доступа `static`.

Созданные объекты передаются по ссылке, то есть вместо копирования всего объекта (и лишних затрат времени и памяти) передается ссылка на объект. Пример:

```
$f = new Person("Pua", 75);

$b = $f; // $b и $f указывают на один объект
$b->setName("Hei Hei");

printf("%s and %s are best friends.\n", $b->getName(), $f->getName());
Hei Hei and Hei Hei are best friends.
```

Чтобы создать настоящую копию объекта, воспользуйтесь оператором `clone`:

```
$f = new Person("Pua", 35);
$b = clone $f; // создание копии
$b->setName("Hei Hei");// изменение копии
printf("%s and %s are best friends.\n", $b->getName(), $f->getName());
Pua and Hei Hei are best friends.
```

Когда вы создаете копию объекта оператором `clone` и класс объявляет метод `__clone()`, то этот метод будет вызван сразу же после создания копии. Эта возможность может пригодиться, когда объект удерживает внешние ресурсы (скажем, дескрипторы файлов), чтобы копия получила новые ресурсы вместо скопированных старых.

Объявление класса

Чтобы спроектировать программу или библиотеку в ООП, необходимо определить классы при помощи ключевого слова `class`. Определение класса включает имя класса, а также свойства и методы класса. Регистр символов в именах классов не учитывается, и эти имена должны соответствовать правилам идентификаторов PHP. Имя класса `stdClass` зарезервировано (вместе с рядом других). Синтаксис определения класса:

```
class имя_класса [ extends суперкласс ] [ implements интерфейс,
 [интерфейс, ... ] ] {
 [ use трейт, [ трейт, ... ]; ]

 [ visibility $свойство [ = значение ]; ... ]

 [ function имя_функции (аргументы) [: тип] {
 // код
 }
 ...
 ]
}
```

Объявление методов

Метод представляет собой функцию, определенную внутри класса. Хотя PHP не устанавливает специальных ограничений, многие методы работают только

с данными того объекта, в котором находится метод. Имена методов, начинающиеся с двух символов подчеркивания (__), могут в будущем использоваться PHP (и уже используются для методов сериализации объектов `__sleep()` и `__wakeup()`, описанных далее в этой главе, и не только), поэтому выбирать такие имена методов не рекомендуется.

Внутри метода переменная `$this` содержит ссылку на объект, для которого был вызван метод. Например, если вы используете вызов `$moana->birthday()`, внутри метода `birthday()` переменная `$this` хранит то же значение, что и переменная `$moana`. Методы используют переменную `$this` для обращения к свойствам текущего объекта и вызова других методов для этого объекта.

Ниже приведено простое определение класса `Person` с использованием переменной `$this`:

```
class Person {
 public $name = '';

 function getName() {
 return $this->name;
 }

 function setName($newName) {
 $this->name = $newName;
 }
}
```

Как видите, методы `getName()` и `setName()` используют `$this` для чтения и присваивания свойства `$name` текущего объекта.

Для объявления статических методов используется ключевое слово `static`. Внутри статических методов переменная `$this` не определена. Пример:

```
class HTMLStuff {
 static function startTable() {
 echo "<table border=\"1\">\n";
 }

 static function endTable() {
 echo "</table>\n";
 }
}

HTMLStuff::startTable();
 // вывод строк и столбцов таблицы HTML
HTMLStuff::endTable();
```

Метод, объявленный с ключевым словом `final`, не может переопределяться в подклассах. Пример:

```
class Person {
 public $name;

 final function getName() {
 return $this->name;
 }
}

class Child extends Person {
 // недопустимый синтаксис
 function getName() {
 // ...
 }
}
```

Используя модификаторы доступа, можно изменять видимость методов. Методы, которые должны быть доступны за пределами методов объекта, объявляются открытыми (`public`), а методы экземпляра, предназначенные для вызова только из методов того же класса, должны объявляться приватными (`private`). Наконец, методы, объявленные защищенными (`protected`), могут вызываться только из методов класса данного объекта и методов подклассов этого класса.

Указывать видимость методов класса необязательно. Если видимость не указана, по умолчанию метод является открытым. Например, определение может выглядеть так:

```
class Person {
 public $age;

 public function __construct() {
 $this->age = 0;
 }

 public function incrementAge() {
 $this->age += 1;
 $this->ageChanged();
 }

 protected function decrementAge() {
 $this->age -= 1;
 $this->ageChanged();
 }

 private function ageChanged() {
 echo "Age changed to {$this->age}";
 }
}

class SupernaturalPerson extends Person {
 public function incrementAge() {
 // уменьшение возраста
```

```
 $this->decrementAge();
 }
}

$person = new Person;
$person->incrementAge();
$person->decrementAge(); // недопустимо
$person->ageChanged(); // также недопустимо
$person = new SupernaturalPerson;
$person->incrementAge(); // скрытый вызов decrementAge
```

При объявлении методов объекта можно использовать рекомендации типов (глава 3):

```
class Person {
 function takeJob(Job $job) {
 echo "Now employed as a {$job->title}\n";
 }
}
```

Если метод возвращает значение, для объявления типа возвращаемого значения можно воспользоваться механизмом рекомендаций:

```
class Person {
 function bestJob(): Job {
 $job = Job("PHP developer");
 return $job;
 }
}
```

Объявление свойств

В предшествующем определении класса `Person` свойство `$name` объявляется явно. Объявлять свойства необязательно — это всего лишь любезность по отношению к тому, кто будет заниматься сопровождением вашей программы. Хороший стиль PHP рекомендует объявлять свойства объектов, но вы можете добавить новые свойства в любой момент.

Версия класса `Person` с необъявленным свойством `$name`:

```
class Person {
 function getName() {
 return $this->name;
 }
 function setName($newName) {
 $this->name = $newName;
 }
}
```

Свойствам можно присвоить значения по умолчанию, но эти значения должны быть простыми константами:

```
public $name = "J Doe"; // работает
public $age = 0; // работает
public $day = 60 * 60 * hoursInDay(); // не работает
```

Модификаторы доступа позволяют изменить видимость свойств. Свойства, доступные вне области видимости объекта, должны объявляться открытыми (`public`), а свойства экземпляра, которые должны быть доступны только для методов того же класса, объявляются приватными (`private`). Наконец, свойства, объявленные защищенными (`protected`), доступны только для методов класса объекта и для методов подклассов этого класса. Пример объявления класса:

```
class Person {
 protected $rowId = 0;
 public $username = 'Anyone can see me';
 private $hidden = true;
}
```

Кроме свойств экземпляров, PHP также позволяет определять статические свойства — переменные класса объекта, к которым можно обращаться с указанием имени класса. Пример:

```
class Person {
 static $global = 23;
}

$localCopy = Person::$global;
```

Внутри экземпляра класса объекта также можно обращаться к статическим свойствам по ключевому слову `self` — например, `self::$global;`.

Если вы обращаетесь к несуществующему свойству объекта, а в классе объекта определен метод `__get()` или `__set()`, этот метод может получить или задать значение этого свойства.

Например, вы объявляете класс для представления данных, полученных из БД, но не хотите загружать большие значения данных, такие как BLOB (если они не были затребованы явно). Конечно, в одной из возможных реализаций для свойства будут созданы методы доступа, которые читают и записывают данные каждый раз, когда они будут затребованы. В другом возможном решении могут использоваться перегружающие методы:

```
class Person {
 public function __get($property) {
 if ($property === 'biography') {
 $biography = "long text here..."; // читается из БД
```

```
 return $biography;
 }
 }

 public function __set($property, $value) {
 if ($property === 'biography') {
 // запись значения в БД
 }
 }
}
```

Объявление констант

Как и в случае с глобальными константами, значения которых присваиваются функцией `define()`, PHP предоставляет возможность присваивания констант внутри классов. К константам, как и к статическим свойствам, можно обращаться напрямую из класса или из методов объектов с использованием синтаксиса `self`. После того как константа будет определена, ее значение изменяться не может:

```
class PaymentMethod {
 public const TYPE_CREDITCARD = 0;
 public const TYPE_CASH = 1;
}

echo PaymentMethod::TYPE_CREDITCARD;
0
```

По аналогии с глобальными константами константы классов принято записывать в верхнем регистре.

Видимость констант классов можно изменять при помощи модификаторов доступа. Константы класса, которые должны быть доступны за пределами методов объекта, должны объявляться открытыми (`public`), а те константы класса, которые должны быть доступны только для методов того же класса, объявляются приватными (`private`). Наконец, константы, объявленные защищенными (`protected`), доступны только для методов класса объекта и для методов подклассов этого класса. Определение видимости констант класса не является обязательным — если видимость не указана, метод считается открытым (`public`). Определение может выглядеть так:

```
class Person {
 protected const PROTECTED_CONST = false;
 public const DEFAULT_USERNAME = "<unknown>";
 private INTERNAL_KEY = "ABC1234";
}
```