**Теория**

**Определение класса**

Класс - это шаблон кода, который используется для создания объектов. Класс определяется с помощью ключевого слова class*,* после которого указывается произвольное имя класса. В имени класса может использоваться любое сочетание букв и цифр, но они не должны начинаться с цифры. Код, связанный с классом должен быть заключен в фигурные скобки, которые указываются после имени. Определение класса описывает, какие элементы будут содержаться в каждом новом экземпляре этого класса. На основе полученных данных давайте посмотрим синтаксис определения класса на примере:

|  |  |
| --- | --- |
| 1234567 | <?php   class first {    // Тело класса  } ?> |

Класс firstиз приведенного примера - уже полноправный класс, хотя пока и не слишком полезный. Но тем не менее мы сделали нечто очень важное. Мы определили тип, т.е. создали категорию данных, которые мы можем использовать в своих сценариях. Важность этого станет для вас очевидной по мере дальнейшего чтения.

**Создание объекта**

Так как класс - это шаблон для создания объектов, следовательно, объект - это данные, которые создаются и структурируются в соответствии с шаблоном, определенным в классе. Объект также называют экземпляром класса, тип которого определяется классом. Для создания нового экземпляра класса нам понадобится оператор new. Он используется совместно с именем класса следующим образом:

|  |  |
| --- | --- |
| 12345678 | <?php     // Ключевое слово new сообщает интерпретатору PHP о необходимости  // создать новый экземпляр класса first  $obj1 = new first();  $obj2 = new first(); ?> |

После оператора new указывается имя класса на основе которого будет создан объект. Оператор new создает экземпляр класса и возвращает ссылку на вновь созданный объект. Эта ссылка сохраняется в переменной соответствующего типа. В результате выполнения этого кода будет создано два объект типа first. Хотя функционально они идентичны (т.е. пусты) $obj1 и $obj2 - это два разных объекта одного типа, созданных с помощью одного класса.

Если вам все еще не понятно, давайте приведем аналогию из реальной жизни. Представьте, что класс - это форма для отливки, с помощью которой изготавливаются пластмассовые машинки. Объекты - это и есть машинки. Тип создаваемых объектов определяется формой отливки. Машинки выглядят одинаковыми во всех отношениях, но все-таки это разные предметы. Другими словами, это разные экземпляры одного и того же типа.

Давайте сделаем эти объекты немного интереснее, изменив класс first, добавив в него специальные поля данных, называемые свойства.

**Определение свойств**

В классе можно определить переменные. Переменные, которые определены в классе называются свойствами (или полями данных). Они определяются с одним из ключевых слов protected, public или private, характеризующих управление доступом. Эти ключевые слова мы рассмотрим подробно в следующей главе. А сейчас давайте определим некоторые свойства с помощью ключевого слова public:

|  |  |
| --- | --- |
| 12345678 | <?php     class first {    public $num = 0;    public $str = 'some text';  } ?> |

Как видите, мы определили два свойства, присвоив каждому из них значение. Теперь любые объекты, которые мы будем создавать с помощью класса first, будут иметь два свойства с указанными значениями.

|  |
| --- |
| **Примечание:** значения инициализирующие свойства должны быть литералами (константными значениями), инициализировать свойства в классе не обязательно (если значение не указано, по умолчанию это будет NULL). |

К свойствам объекта можно обращаться с помощь символов '->', указав объект и имя свойства. Поскольку свойства объектов были определены как public, мы можем считывать их значения, а также присваивать им новые значения, заменяя тем самым начальные значения, определенные в классе:

|  |  |
| --- | --- |
| 12345678910111213141516 | <?php     class first {    public $num = 0;    public $str = 'some text';  }     $obj = new first();     echo $obj->str;     // присваиваем свойству объекта новое значение  $obj->str = 'новая строка';  echo "<br>$obj->str"; ?> |

На самом деле в PHP необязательно объявлять все свойства в классе. Свойства можно добавлять к объекту динамически:

|  |  |
| --- | --- |
| 12345678910111213 | <?php     class first {    public $str = 'some text';  }     $obj = new first();     // добавляем объекту новое свойство  $obj->newprop = 'новое свойство';  echo $obj->newprop; ?> |

Нужно отметить, что этот способ присваивания свойств объектам считается дурным тоном в объектно-ориентированном программировании и почти никогда не используется.

**Работа с методами**

Методы - это обычные функции, которые определяются внутри класса, они позволяют объектам выполнять различные задачи. Объявление метода напоминает определение обычной функции, за исключением предваряемого одного из ключевых слов protected, public или private. Если в определении метода вы опустите ключевое слово, определяющее видимость, то метод будет объявлен неявно как public. К методам объекта можно обращаться с помощь символов '->', указав объект и имя метода. При вызове метода, так же как и при вызове функции нужно использовать круглые скобки.

|  |  |
| --- | --- |
| 1234567891011121314151617 | <?php     class first {    public $str = 'some text';         // определение метода    function getstr() {      echo $this->str;    }  }     $obj = new first();     // вызов метода объекта  $obj->getstr(); ?> |

Мы добавили метод getstr() к классу first. Обратите внимание на то, что при определении метода мы не использовали ключевое слово, определяющее область видимости. Это означает, что метод getstr() относится к типу public и его можно вызвать за пределами класса.

В определении метода мы воспользовались специальной псевдопеременной $this. Она используется для обращения к методам или свойствам внутри класса и имеет следующий синтаксис:

|  |  |
| --- | --- |
| 1 | $this->имя переменной или метода  |

Значением переменной $this является ссылка на текущий объект. Чтобы стало понятнее посмотрите на следующий пример:

|  |  |
| --- | --- |
| 1234567891011121314151617181920212223 | class first {    public $str = 'some text';         // при определении метода в классе, переменная $this не имеет никакого значения    function getstr() {      echo $this->str;    }  }     // создаем объект  $obj = new first();     // созданный нами объект имеет свойство и метод  // теперь в методе объекта переменная $this имеет  // ссылку на текущий объект, а именно на $obj      // т.е. если в методе заменить $this текущим экземпляром объекта  $this->str;     // это будет выглядеть как простое   // обращение к свойству текущего объекта  $obj->str;  |

Переменной $this нельзя ничего присваивать. Помните, что переменная $this всегда ссылается на текущий объект.

**Специальный метод – конструктор**

У класса может быть определен специальный метод - конструктор, который вызывается каждый раз при создании нового экземпляра класса (объекта) с целью инициализировать его, например установить значения свойств. Конструктор, как и любой другой метод может иметь параметры. Чтобы определить метод в качестве конструктора его необходимо назвать \_\_construct(). Обратите внимание на то, что имя метода должно начинаться с двух символов подчеркивания. Посмотрим, как это работает:

|  |  |
| --- | --- |
| 1234567891011121314151617181920212223242526 | <?php     class first {    // определяем два свойства    public $num1 = 0;    public $num2 = 0;         // определяем конструктор класса    function \_\_construct($num1, $num2) {      $this->num1 = $num1;      $this->num2 = $num2;    }         // метод, который складывает два числа    function summa() {      return $this->num1 + $this->num2;    }  }     // создаем объект и передаем два аргумента  $obj = new first(15, 35);     // вызываем метод и сразу выводим результат его работы  echo $obj->summa(); ?> |

Метод \_\_construct вызывается, когда создается объект с помощью оператора new. Указанные в скобках аргументы передаются конструктору. В методе конструктора используется псевдопеременная $this для присвоения значений соответствующим свойствам создаваемого объекта.

**Примечание:** если конструктор не имеет параметров и при создании новых экземпляров класса не передаются никакие аргументы, круглые скобки ()после имени класса можно опустить:

|  |  |
| --- | --- |
| 1 | $obj = new first; |

**Статические свойства и методы классов**

В объектной модели PHP существует возможность задавать свойства и методы не только для объектов — экземпляров класса, но и для класса в целом. Для этого тоже служит ключевое слово static:

class A {

 public static $x = 'foo';

 public static function test() {

 return 42;

 }

}

echo A::$x; *// 'foo'*

echo A::test(); *// 42*

Для доступа к таким свойствам и методам используются конструкции с двойным двоеточием («Paamayim Nekudotayim»), такие как ИМЯ\_КЛАССА::$имяПеременной и ИМЯ\_КЛАССА:: имяМетода().
Само собой разумеется, что у статических свойств и статических методов есть свои особенности и свои «подводные камни», которые нужно знать.

**Особенность первая, банальная — нет $this.**

Собственно это проистекает из самого определения статического метода — поскольку он связан с классом, а не объектом, в нём недоступна псевдопеременная $this, указывающая в динамических методах на текущий объект. Что совершенно логично.
Однако, нужно знать, что в отличие от других языков, PHP не определяет ситуацию «в статическом методе написано $this» на этапе парсинга или компиляции. Подобная ошибка может возникнуть только в рантайме, если вы попытаетесь выполнить код с $this внутри статического метода.

Код типа такого:

class A {

 public $id = 42;

 static public function foo() {

 echo $this->id;

 }

}

не приведет ни к каким ошибкам, до тех пор, пока вы не попытаетесь использовать метод foo() неподобающим образом:

$a = new A;

$a->foo();

(и сразу получите «Fatal error: Using $this when not in object context»).

**Особенность вторая — static не аксиома!**

class A {

 static public function foo() {

 echo 42;

 }

}

$a = new A;

$a->foo();

Вот так, да. Статический метод, если он не содержит в коде $this, вполне можно вызывать в динамическом контексте, как метод объекта.
Обратное не совсем верно:

class A {

 public function foo() {

 echo 42;

 }

}

A::foo();

Динамический метод, не использующий $this, можно выполнять в статическом контексте. Однако вы получите предупреждение «Non-static method A::foo() should not be called statically» уровня E\_STRICT. Тут решать вам — или строго следовать стандартам кода, или подавлять предупреждения. Первое, разумеется, предпочтительнее.

И кстати, всё написанное выше относится только к методам. Использование статического свойства через "->" невозможно и ведет к фатальной ошибке.

**Значение третье, кажущееся самым сложным — позднее статическое связывание.**
Разработчики языка PHP не остановились на двух значениях ключевого слова «static» и в версии 5.3 добавили еще одну «фичу» языка, которая реализована тем же самым словом! Она называется «позднее статическое связывание» или LSB (Late Static Binding).

Понять суть LSB проще всего на несложных примерах:

class Model {

 public static $table = 'table';

 public static function getTable() {

 return self::$table;

 }

}

echo Model::getTable(); *// 'table'*

Ключевое слово self в PHP всегда значит «имя класса, где это слово написано». В данном случае self заменяется на класс Model, а self::$table — на Model::$table.
Такая языковая возможность называется «ранним статическим связыванием». Почему ранним? Потому что связывание self и конкретного имени класса происходит не в рантайме, а на более ранних этапах — парсинга и компиляции кода. Ну а «статическое» — потому что речь идет о статических свойствах и методах.

Немного изменим наш код:

class Model {

 public static $table = 'table';

 public static function getTable() {

 return self::$table;

 }

}

class User extends Model {

 public static $table = 'users';

}

echo User::getTable(); *// 'table'*

Теперь вы понимаете, почему PHP ведёт себя в этой ситуации неинтуитивно. self был связан с классом Model тогда, когда о классе User еще ничего не было известно, поэтому и указывает на Model.
Как быть?
Для решения этой дилеммы был придуман механизм связывания «позднего», на этапе рантайма. Работает он очень просто — достаточно вместо слова «self» написать «static» и связь будет установлена с тем классом, который вызывает данный код, а не с тем, где он написан:

class Model {

 public static $table = 'table';

 public static function getTable() {

 return static::$table;

 }

}

class User extends Model {

 public static $table = 'users';

}

echo User::getTable(); *// 'users'*

Это и есть загадочное «позднее статическое связывание».

Нужно отметить, что для большего удобства в PHP кроме слова «static» есть еще специальная функция get\_called\_class(), которая сообщит вам — в контексте какого класса в данный момент работает ваш код.