

Запись на курсы по HTML, CSS, JavaScript, PHP, фреймворкам и CMS,
а также: помощь в поиске работы и заказов, стажировка на реальных проектах→

урок 18 из 107

[Верстка](#) [JavaScript](#) [PHP](#) [NodeJs](#) [Vue](#) [React](#) [Laravel](#) [WordPress](#) [AJAX](#) [Парсинг](#)[Бесплатные курсы по React для новичков. Начало 4-го ноября→](#) [Конкурс CSS картинок. Тема: Хэллоун. Призовой фонд: 100\\$. Подробности→](#)

Класс как набор методов

Часто классы используются просто как набор некоторых методов, сгруппированных вместе. В этом случае нам не нужно создавать много объектов этого класса, а достаточно всего одного.

Для примера давайте сделаем класс **ArraySumHelper**, который предоставит нам набор методов для работы с массивами. Каждый метод нашего класса будет принимать массив, что-то с ним делать и возвращать результат.

Пусть у нас будет следующий набор методов:

```
1  <?php
2  class ArraySumHelper
3  {
4      // Сумма элементов массива:
5      public function getSum1($arr)
6      {
7
8      }
9
10     // Сумма квадратов элементов массива:
11     public function getSum2($arr)
12     {
13
14     }
15
16     // Сумма кубов элементов массива:
17     public function getSum3($arr)
18     {
19
20     }
21
22     // Сумма четвертых степеней элементов массива:
23     public function getSum4($arr)
24     {
25
26     }
27 }
28 ?>
```

Давайте посмотрим, как мы будем пользоваться нашим классом:

```
1  <?php
2  $arraySumHelper = new ArraySumHelper; // создаем объект
3
4  $arr = [1, 2, 3];
5  echo $arraySumHelper->getSum1($arr); //
   найдем сумму элементов
6  echo $arraySumHelper->getSum2($arr); //
   найдем сумму квадратов элементов
7  echo $arraySumHelper->getSum3($arr); //
   найдем сумму кубов элементов
8  echo $arraySumHelper->getSum4($arr); //
   найдем сумму 4-тых степеней элементов
9  ?>
```

Вот еще пример - найдем сумму квадратов элементов массива и сумму кубов и сложим результат вместе:



```
1  <?php
2  $arraySumHelper = new ArraySumHelper;
3
4  $arr = [1, 2, 3];
5  echo $arraySumHelper->getSum2($arr) + $arraySumHelper->getSum3($arr);
6  ?>
```

То есть фактически мы получаем набор функций, просто сгруппированных в одном классе.

Однако, в отличие от обычного набора функций, мы можем пользоваться преимуществами ООП - например, делать вспомогательные методы приватными, чтобы они не были доступны извне класса.

Давайте приступим к написанию кода нашего класса.

Если обдумать реализацию методов, то становится очевидно, что они будут фактически одинаковыми, отличие будет только в степени, в которую будут возводиться элементы нашего массива. Вот код, иллюстрирующий это:

```
1  <?php
2  class ArraySumHelper
3  {
4      public function getSum1($arr)
5      {
6          $sum = 0;
7
8          foreach ($arr as $elem) {
9              $sum += $elem; // первая степень элемента - э
10             то сам элемент
11         }
12
13         return $sum;
14     }
15
16     public function getSum2($arr)
17     {
18         $sum = 0;
19
20         foreach ($arr as $elem) {
21             $sum += pow($elem, 2); // возведем во
22             вторую степень
23         }
24
25         return $sum;
26     }
27
28     public function getSum3($arr)
29     {
30         $sum = 0;
31
32         foreach ($arr as $elem) {
33             $sum += pow($elem, 3); // возведем в т
34             ретью степень
35         }
36
37         return $sum;
38     }
39
40     public function getSum4($arr)
41     {
42         $sum = 0;
43
44         foreach ($arr as $elem) {
45             $sum += pow($elem, 4); // возведем в ч
46             етвертую степень
47         }
48
49         return $sum;
50     }
51 }
```



```
    }  
48 | ?>
```

Вместо того, чтобы реализовывать каждый метод заново, давайте лучше сделаем вспомогательный приватный метод **getSum**, который параметрами будет принимать массив и степень и возвращать сумму степеней элементов массива:

```
1 | <?php  
2 |     private function getSum($arr, $power) {  
3 |         $sum = 0;  
4 |  
5 |         foreach ($arr as $elem) {  
6 |             $sum += pow($elem, $power);  
7 |         }  
8 |  
9 |         return $sum;  
10 |     }  
11 | ?>
```

Итак, давайте поменяем методы нашего класса с использованием нового метода **getSum**:

```
1 | <?php  
2 | class ArraySumHelper  
3 | {  
4 |     public function getSum1($arr)  
5 |     {  
6 |         return $this->getSum($arr, 1);  
7 |     }  
8 |  
9 |     public function getSum2($arr)  
10 |    {  
11 |        return $this->getSum($arr, 2);  
12 |    }  
13 |  
14 |    public function getSum3($arr)  
15 |    {  
16 |        return $this->getSum($arr, 3);  
17 |    }  
18 |  
19 |    public function getSum4($arr)  
20 |    {  
21 |        return $this->getSum($arr, 4);  
22 |    }  
23 |  
24 |    private function getSum($arr, $power) {  
25 |        $sum = 0;  
26 |  
27 |        foreach ($arr as $elem) {  
28 |            $sum += pow($elem, $power);  
29 |        }  
30 |  
31 |        return $sum;  
32 |    }  
33 | }  
34 | ?>
```

Наш класс **ArraySumHelper** больше учебный, чем реальный, но тут вам важно понять принцип - то, что часто некоторый класс может использоваться просто как набор методов и при этом создается только один объект класса. В дальнейшем мы будем разбирать более жизненные (но и более сложные) примеры.

Задача 18.1

Напишите реализацию методов класса **ArrayAvgHelper**, заготовки методов которого расположены ниже:

```
1 | <?php  
2 | class ArraySumHelper  
3 | {  
4 |     /*
```

```
5      Находит сумму первых
6      степеней элементов массива:
7      */
8      public function getAvg1($arr)
9      {
10
11     }
12
13     /*
14     Находит сумму вторых степеней
15     элементов массива и извлекает
16     из нее квадратный корень:
17     */
18     public function getAvg2($arr)
19     {
20
21     }
22
23     /*
24     Находит сумму третьих степеней
25     элементов массива и извлекает
26     из нее кубический корень:
27     */
28     public function getAvg3($arr)
29     {
30
31     }
32
33     /*
34     Находит сумму четвертых степеней
35     элементов массива и извлекает
36     из нее корень четвертой степени:
37     */
38     public function getAvg4($arr)
39     {
40
41     }
42
43     /*
44     Вспомогательный метод, который параметром
45     принимает массив и степень и возвращает
46     сумму степеней элементов массива:
47     */
48     private function getSum($arr, $power)
49     {
50
51     }
52
53     /*
54     Вспомогательный метод, который параметром
55     принимает целое число и степень и возвращает
56     корень заданной степени из числа:
57     */
58     private function calcSqrt($num, $power)
59     {
60
61     }
62 }
63 ?>
```

Математическая подсказка: корень первой степени - это само число. То есть **calcSqrt(число, 1)** должно просто вернуть само число.

А корень любой степени можно найти с помощью функции **pow**, просто параметром передав ей дробь. Например, **pow(число, 1/3)** - так найдем корень третьей степени.



