

# **МДК 02.02. Web-программирование. Язык PHP**

## **Объекты и классы в PHP**

**Класс** – это тип объекта, включающий в себя набор переменных и функций для управления этими переменными.

Переменные называют **свойствами**, а функции – **методами**.

Для использования методов и свойств класса необходимо создать **экземпляр класса**.

## Описание класса

```
class Имя_класса  
{  
// свойства и методы класса  
}
```

## Создание свойства класса

```
class Имя_класса {  
<Область видимости> <Имя переменной>;  
}
```

Имена переменных указываются со знаком \$.

Имя класса чувствительно к регистру символов.

## Создание метода класса

```
class Имя_класса {  
    [<Область видимости>] function <Имя функции>  
    ([Параметры]) {  
        // Тело функции  
    }  
}
```

## Доступ к объектам

**Обращение к свойствам класса или другим функциям внутри функции используется указатель `$this`:**

```
class Имя_класса {  
    Область_видимости Имя_переменной;  
    [Область видимости] function Имя_функции  
    ([Параметры]) {  
        $this->Имя_переменной без знака $ = Значение;  
        $this->Имя_функции ();  
    }  
}
```

**Доступ к свойствам и методам объекта**

**\$объект->свойство**

**\$объект->метод()**

## Создание объекта

Для создания объекта определенного класса используется ключевое слово **new**:

объект = **new** Класс.

Вот два способа создания объектов:

1. \$object = **new** User;
2. \$temp = **new** User('name', 'password');

## Пример

```
<?php
// Создание нового класса Class1:
class Class1 {
public $name;      // свойство
function Getname() { // метод
    echo "<h3>John</h3>";
}
}
$object = new Class1; // Создание объекта класса Class1
$object->name = "Alex"; // Получение доступа к членам класса
echo $object->name; // Вывод 'Alex'

$object->Getname(); /* Доступ к методу класса. Вывод 'John'
                    заголовком 3 уровня*/
?>
```

## Конструктор и деструктор

Чтобы при создании класса присвоить начальные значения каким-либо переменным, необходимо создать метод, имеющий predetermined название `__construct()`. Такой метод называется **конструктором**.  
Конструктор автоматически вызывается сразу после создания объекта.

## Конструктор и деструктор

```
class <Имя класса> {  
    <Область видимости> <Имя переменной со знаком $>;  
    [<Область видимости>] function <Имя функции>  
    ([Параметры]) {  
        $this-><Имя переменной без знака $> = <Значение>;  
        $this-><Имя функции>();  
    }  
    public function __construct(<Параметр1>) {  
        $this-><Имя переменной без знака $> = <Параметр1>;  
    }  
}
```

Если конструктор вызывается при создании объекта, то перед уничтожением объекта автоматически вызывается метод, называемый **деструктором**.

Деструктор реализуется в виде predefined метода **`__destruct()`**.

## Пример

```
<?php
class Class1 {
public $var;
public function __construct($var) {
$this->var = $var;
echo 'Вызван конструктор<br>';
}
public function __destruct() {
echo 'Вызван деструктор';
}
public function f_get() {
return $this->var;
}
}
$obj = new Class1(5);
echo 'Значение свойства var равно ' . $obj->f_get() . '<br>';
echo 'Вывод перед удалением объекта<br>';
unset($obj);
?>
```

## Наследование

Позволяет одному объекту приобретать свойства другого объекта, а также иметь собственные.

```
class Class1 {  
    public function f_print() {  
        echo 'Метод f_print класса Class1<br>';  
    }  
    public function f_display() {  
        echo 'Метод f_display класса Class1<br>';  
    }  
}  
  
class Class2 extends Class1 {  
    public function f_new() {  
        echo 'Метод f_new класса Class2<br>';  
    }  
}
```

Ключевое слово **extends** указывает, что класс Class2 наследует все свойства и методы класса Class.

```
$obj = new Class2();
```

```
$obj->f_new();
```

```
$obj->f_print();
```

```
$obj->f_display();
```

Чтобы использовать метод, объявленный в родительском классе, следует вызвать его с помощью ключевого слова `parent`.

## Пример

```
class Class1 {  
    public function f_display() {  
        echo 'Метод f_display класса Class1<br>';  
    }  
}  
  
class Class2 extends Class1 {  
    public function f_display() {  
        parent::f_display();  
        echo 'Привет';  
    }  
}  
  
$obj = new Class2();  
$obj->f_display();
```

**Конструктор и деструктор в родительском классе автоматически не вызываются. Для их вызова также необходимо использовать ключевое слово `parent`.**

## Запрещение переопределение метода

Используется ключевое слово **final**:

```
class Class1 {  
    final public function f_display() {  
        echo 'Метод f_display класса Class1<br>';  
    }  
}  
  
class Class2 extends Class1 {  
    public function f_display($msg) {  
        echo $msg;  
    }  
}  
  
$obj = new Class2();
```

## Статические свойства и методы

Внутри класса можно создать свойство или метод, которые будут доступны без создания экземпляра класса.

Для этого перед определением свойства или метода следует указать ключевое слово **static**.

### Пример

```
public static $var = 5;  
public static function f_print() {  
// Тело функции  
}
```

**Доступ к статическому свойству вне класса:**

```
echo <Название класса>::$var;
```

**Вызов статического метода без создания класса:**

```
<Название класса>::<Название метода>(<Параметры>);
```

## Объявление констант внутри класса

Константу внутри класса можно объявить с помощью ключевого слова **const**:

```
class <Имя класса> {  
    const <Имя константы без $> = <Значение>;  
    // Описание свойств и методов класса  
}
```

**Доступ к константе вне класса :**

```
<Имя класса без $>::<Имя константы без $>
```

Внутри класса к константе можно также обратиться с помощью ключевого слова **self**:

`self::<Имя константы без $>`

## Пример

```
class CMyClass {  
    const myconst = 10;  
    public $myvar;  
    public function __construct($i) {  
        $this->myvar = $i;  
    }  
    public function f_Sum1($x) {  
        return ($x + self::myconst);  
    }  
}  
  
$obj = new CMyClass(20);  
echo $obj->f_Sum1(5), '<br>';  
echo CMyClass::myconst;
```

## Область видимости

- ✓ **public** Свойства с этой областью видимости получают по умолчанию при объявлении переменной с помощью ключевых слов `var` или `public` или когда переменная объявляется неявно при первом же ее использовании. Методы считаются открытыми по умолчанию.
- ✓ **protected** На свойства и методы с этой областью видимости можно ссылаться только через принадлежащие объектам методы класса и такие же методы любых подклассов.
- ✓ **private** К представителям класса с этой областью видимости можно обращаться через методы этого же класса, но не через методы его подклассов.

## Область видимости :

- ✓ открытую (public) область видимости следует применять, когда к представителю класса нужен доступ из внешнего кода и когда расширенные классы должны его наследовать;
- ✓ защищенную (protected) область видимости необходимо использовать, когда к представителю класса не должно быть доступа из внешнего кода, но расширенные классы все же должны его наследовать;
- ✓ закрытую (private) область видимости следует применять, когда к представителю класса не должно быть доступа из внешнего кода и когда расширенные классы не должны его наследовать.

## Пример

```
class Person {  
    public $name;    // Переменная доступна везде  
    protected $age; // Доступна в классе и в производных классах  
    private $salary; // Доступна только в этом классе  
    public function __construct() {  
        // ...  
    }  
    protected function set_age() {  
        // ...  
    }  
    private function set_salary() {  
        // ...  
    }  
}
```

## **Создание шаблона сайта при помощи класса**

При создании больших сайтов обычно страницу делят на три части – верхний колонтитул (заголовок), тело страницы и нижний колонтитул (футер, подвал).

Нижний колонтитул практически всегда одинаков для всех страниц, а вот верхние колонтитулы по определению не могут совпадать.

1. `header.php` – верхний колонтитул (пример 1);
2. `index.php` – основное содержание страницы (пример 2);
3. `footer.php` – нижний колонтитул (пример 3).

# Пример 1. Содержимое файла header.php

```
<?php
class Header {
private $title;
private $meta;
public function __construct($var1, $var2) {
$this->title = $var1;
$this->meta = $var2;
}
public function f_display() {
echo "<html><head>\n";
echo '<title>' . $this->title . "</title>\n";
echo '<meta name="description" content="';
echo $this->meta . "\">\n";
echo '<meta http-equiv="Content-Type" content="text/html; ';
echo "charset=windows-1251\">\n";
echo '</head>';
echo "<body>\n";
}
}
?>
```

## Пример 2. Содержимое файла index.php

```
<?php
require_once('header.php');
$title = 'Заголовок';
$meta = 'Описание';
$obj = new Header($title, $meta);
$obj->f_display();
echo '<div>Основное содержание страницы</div>';
require_once('footer.php');
?>
```

### Пример 3. Содержимое файла footer.php

```
</body>
```

```
</html>
```

Если открыть файл index.php в Web-браузере и отобразить исходный код, то мы увидим:

```
<html><head>
```

```
<title>Заголовок</title>
```

```
<meta name="description" content="Описание">
```

```
<meta http-equiv="Content-Type" content="text/html;  
charset=windows-1251">
```

```
</head><body>
```

```
<div>Основное содержание страницы</div></body>
```

```
</html>
```

**Меняя значения переменных \$title и \$meta, можно сделать уникальными заголовок и описание каждой страницы.**