Класс Link в ООП на PHP

Давайте сделаем класс, который будет создавать HTML ссылку. Назовем его **Link**.

Вот так мы будем пользоваться нашим классом:

```php
<?php
  // Выведет <a href="/test.html">link</a>:
  echo (new Link())->setAttr('href', '/test.html')->
    setText('link')->show();
?>
```

Реализация:

```php
<?php
  class Link extends Tag
  {
    public function __construct()
    {
      parent::__construct('a');
    }
  }
?>
```

Сделаем так, чтобы даже если атрибут **href** не задан, то по умолчанию он становился пустыми кавычками:

```php
<?php
  class Link extends Tag
  {
    public function __construct()
    {
      parent::__construct('a');
      $this->setAttr('href', '');
    }
  }
?>
```

Проверим:

```php
<?php
  // Выведет <a href="">index</a>:
  echo (new Link())->setText('index')->show();
?>
```

Задача 69.1

Самостоятельно реализуйте описанный класс **Link**.

Задача 69.2

С помощью этого класса создайте меню из 5 ссылок. Пусть первая ссылка ведет на страницу **/1.php**, вторая - на страницу **/2.php** и так далее.

Задача 69.3

Разместите созданную менюшку в отдельном файле, например, в **menu.php**.

Задача 69.4

Создайте страницы, на которые ведут ссылки вашей менюшки. Добавьте в ним какой-нибудь текст.

Задача 69.5

Подключите инклудом к тексту каждой страницы вашу менюшку из файла. Убедитесь, что ссылки из этой менюшки будут работать корректно.

Активация ссылок

После выполнения задач у вас должна получится менюшка. Давайте сделаем так, чтобы в этой менюшке выделялась каким-то образом та ссылка, на странице которой мы находимся.

Такая ссылка обычно называется *активной* и ее выделение происходит путем добавления ей CSS класса **active** (общепринятое название).

Добавленный к ссылке класс **active** каким-то образом выделяет ее - подчеркивает, красит в красный цвет и тому подобное: все это регулируется CSS стилями для этого класса.

Итак, давайте сделаем так, чтобы ссылки автоматически активировались (добавляли себе CSS класс **active**), если их **href** совпадает с урлом сайта.

Напоминаю, что **URL** сайта можно достать с помощью *$_SERVER['REQUEST_URI']* (точнее это будет **URI**, то есть адрес страницы без доменного имени, но нам тут именно так и надо).

Чтобы прочитать **href** нашей ссылки, используем геттер **getAttr**, унаследованный от родительского класса **Tag**. Вот так: *$this->getAttr('href')*.

Чтобы добавить нашей ссылке CSS класс **active**, используем метод **addClass**, также унаследованный от родителя. Вот так: *$this->addClass('active')*.

Соберем все вместе и напишем вспомогательный метод **activateSelf**, который будет проверять, совпадает ли **href** ссылки и **URI**, и активировать ее, если это так:

```
1  <?php
2    private function activateSelf()
3    {
4      if ($this->getAttr('href') == $_SERVER['RE
         QUEST_URI']) {
5        $this->addClass('active');
6      }
7    }
8  ?>
```

Осталось придумать в каком месте вызывать созданный нами метод. В конструкторе класса **Link** этого делать нельзя, так как в момент вызова конструктора **href** ссылки еще не задан (конструктор же вызывается в самом начале, а потом методы цепочки, в том числе **setAttr**, который и задает **href** ссылки).

После таких рассуждений становится очевидным, что метод **activateSelf** следует вызвать в момент вывода ссылки на экран, то есть в методе **show**, с помощью которого скорее всего и будет формироваться ссылка.

Однако, представляется возможным то, что при использовании нашего класса кто-то будет применять метод **open** и метод **close** отдельно.

Хотя описанное выше и маловероятно, тем не менее вызовем метод **activateSelf** в методе **open**, переопределив тем самым метод родителя:

```php
<?php
  class Link extends Tag
  {
    public function __construct()
    {
      $this->setAttr('href', '');
      parent::__construct('a');
    }

    // Переопределяем метод родителя:
    public function open()
    {
      $this->activateSelf(); // вызываем активацию
      return parent::open(); // вызываем метод родителя
     }

    private function activateSelf()
    {
      if ($this->getAttr('href') == $_SERVER['REQUEST_URI']) {
        $this->addClass('active');
      }
    }
  }
?>
```

Так как метод **show** использует внутри себя метод **open**, то изменения для метода **show** произойдут автоматически.

Можем теперь проверить работу нашего класса:

```php
<?php
  echo (new Link)->setAttr('href', '/index.php')->setText('index')->show();

  /*
    Если URL страницы не /index.php,
    то результат выполнения кода выведет
    <a href="/index.php">index</a>

    Если URL страницы /index.php,
    то результат выполнения кода выведет
    <a href="/index.php" class="active">index</a>
   * /
?>
```

Итак, теперь ссылки активируют сами себя. Это реально круто! При этом нам понадобилось совсем мало кода, чтобы реализовать такое поведение. Все потому, что у нас есть базовый класс **Tag**, который прячет внутри себя много универсального кода для манипуляций с тегами.

Реализуя новые классы на основе класса **Tag** мы не держим в голове детали реализации этого класса **Tag**. И вообще не видим код этого класса - он где-то в другом файле (если, конечно же, вы его туда вынесли) и не мешает нам работать. Мы просто знаем, какие методы предоставляет этот класс своим потомкам - и пользуемся ими.

Поэтому классы-потомки и получаются такими маленькими и изящными.

На самом деле наш код класса **Link** еще более крут, чем кажется. Дело в том, что наши ссылки могут иметь и другие - постоянные - классы. При этом наша активация никак не будет мешать этим классам - они будут оставаться, просто к ним будет добавляться еще и класс **active**.

Все потому, что так работает метод **addClass** - он добавляет новый класс к уже существующим классам.

Вот пример:

```php
<?php
  echo (new Link)
    ->setAttr('href', '/index.php')
    ->setAttr('class', 'link1 link2') // добавляем ссылке классы
     ->setText('index')
    ->show();

  /*
    Результат выполнения кода выведет
    <a href="/index.html" class="link1 link2 active">index</a>
  * /
?>
```

Задача 69.6

Добавьте в ваш класс **Link** активацию ссылок.

Задача 69.7

Проверьте работу активации ссылок на менюшке, которую вы создали в предыдущих задачах. Характерно, что правки в саму менюшку вносить не надо - создание ссылок никак не поменялось, просто ссылки теперь активируют сами себя. Поперходите по ссылкам меню и убедитесь в том, что соответствующие ссылки активируются.

Задача 69.8

Не очень хорошо то, что название класса **active** жестко зашито в коде (вдруг мы захотим поменять его на другое). Вынесите его в константу класса (константу используем для того, чтобы в процессе работы скрипта случайно не изменить наш CSS класс).