# Класс Image

В предыдущих уроках мы с вами реализовали класс **Tag** для работы с тегами.

Пусть с помощью этого класса мы хотим сделать HTML картинку:

```php
<?php
  $image = new Tag('img');
  echo $image->setAttr('src', 'img.png')->
    open(); // <img src="img.png">
?>
```

Давайте сделаем работу с картинками более удобной. Не будем использовать класс **Tag**, а создадим специальный класс для работы с картинками, назвав его **Image**.

С использованием нового класса мы могли бы переписать код вот так:

```php
<?php
  $image = new Image();
  echo $image->setAttr('src', 'img.png')->
    open(); // <img src="img.png">
?>
```

Пока особой разницы, как вы видите, нет. У нового класса, однако, есть преимущество - мы можем сделать вещи, характерные именно для тега **img**.

К примеру, в теге **img** атрибут **src** является обязательным. Было бы удобно, чтобы новый класс работал так: если атрибут **src** не задан через **setAttr**, то он все равно будет созданном теге, но с пустым значением:

```php
<?php
  $image = new Image();
  echo $image->open(); // <img src="">
?>
```

Кроме того, было бы неплохо сделать то же самое и для атрибута **alt** (важен для SEO, желательно, чтобы всегда был):

```php
<?php
  $image = new Image();
  echo $image->open(); // <img src="" alt="">
    - alt тоже будет
?>
```

# Реализация класса Image

Очевидно, что класс **Image** - это тот же класс **Tag**, но с некоторыми дополнениями.

Логично в этом случае не создавать этот класс с нуля, а унаследовать его от класса **Tag**:

```php
<?php
  class Image extends Tag
  {

  }
?>
```

Как мы описали выше, класс **Image** отличается от класса **Tag** тем, что в **Image** по умолчанию задаются атрибуты **src** и **alt**.

Кроме того, обратите внимание на то, как мы вызываем конструкторы классов:

```php
<?php
  $image = new Tag('img');
  $image = new Image();
?>
```

Как вы видите, класс **Tag** ожидает первым параметром имя тега, а класс **Image** вызывается без параметра - имя тега в данном случае нет необходимости задавать, так как наш класс всегда делает один и тот же тег **img**.

Для того, чтобы реализовать такое поведение, класс **Image** должен переопределить конструктор родителя:

```php
<?php
  class Image extends Tag
  {
    public function __construct()
    {
      // Вызовем конструктор родителя, передав имя
         тега:
      parent::__construct('img');
    }
  }
?>
```

Давайте в этом же конструкторе зададим атрибуты **src** и **alt**:

```php
<?php
  class Image extends Tag
  {
    public function __construct()
    {
      $this->setAttr('src', ''); // установим атр
        ибут src
      $this->setAttr('alt', ''); // установим атр
        ибут alt

      parent::__construct('img'); // вызовем кон
        структор родителя
    }
  }
?>
```

Установку атрибутов можно упростить и выполнить в виде цепочки:

```php
<?php
  class Image extends Tag
  {
    public function __construct()
    {
      $this->setAttr('src', '')->setAttr('alt
        ', '');
      parent::__construct('img');
    }
  }
?>
```

При использовании нашего класса в дальнейшем мы можем затереть эти атрибуты с помощью **setAttr** на свое значение, а можем не затереть - тогда они так и останутся со значением '', смотрите пример:

```php
<?php
  // Затрем src, но не alt:
  echo (new Image())->setAttr('src', 'img.png')->
    open(); // <img src="img.png" alt="">
?>
```

## Задача 68.1

Самостоятельно напишите реализацию описанного класса **Image**.

## Задача 68.2

Используя созданный вами класс выведите на экран какую-нибудь картинку.

## Задача 68.3

Установите созданной вами картинке атрибут **width** в значение **300**, а атрибут **height** - в значение **200**.

# Откажемся от open

В классе **Tag** у нас есть два метода, которые используются для завершения цепочки и вывода тега на экран: это метод **open** и метод **show**.

Почему у нас два метода: потому что класс **Tag** универсальный и предполагает использование как для тегов, не требующих закрытия, так и для парных тегов.

Очевидно, что при использовании класса **Image** мы всегда будем завершать цепочку методом **open**.

Раз так, то давайте сделаем использование метода **open** не обязательным. То есть вместо этого:

```php
<?php
  echo (new Image())->setAttr('src', 'img.png')->
    open();
?>
```

Мы будем писать вот так:

```php
<?php
  echo (new Image())->setAttr('src', 'img.png');
?>
```

Используем для этого магический метод **__toString**:

```php
<?php
  class Image extends Tag
  {
    public function __construct()
    {
      $this->setAttr('src', '')->setAttr('alt
        ', '');
      parent::__construct('img');
    }

    public function __toString()
    {
      return parent::open(); // вызываем метод род
        ителя
     }
  }
?>
```

Теперь получается, что при попытке преобразования объекта в строку, например, при выводе его через **echo**, автоматически будет вызываться метод **__toString**, внутри которого будет вызываться метод **open**.

Напоминаю, что если не выводить объект на экран, а, например, записать в переменную, то в эту переменную попадет объект, а не его строковое представление:

```php
<?php
  // В переменную $image запишется объект:
  $image = (new Image())->setAttr('src', 'im
    g.png');

  $image->setAttr('width', '200'); // вызовем еще
     метод
  e cho $image; // тут сработает __toString
?>
```

## Задача 68.4

Самостоятельно напишите реализацию метода **__toString**.