Класс HtmlList в ООП на PHP

Сейчас мы с вами сделаем класс **HtmlList** для создания списков **ul** и **ol** (удачнее было бы назвать класс как **List**, но это слово является зарезервированным в PHP).

У этого класса будет метод **addItem** для добавления пунктов списка и метод **show** для вывода результата на экран.

Для самих пунктов списка, то есть для тегов **li**, также сделаем отдельный класс, назовем его **ListItem**.

Вот пример того, как мы будем пользоваться нашими классами:

```
<?php
  $list = new HtmlList('ul');

  echo $list
    ->addItem( (new ListItem())->setText('ite
      m1') )
     ->addItem( (new ListItem())->setText('ite
      m2') )
     ->addItem( (new ListItem())->setText('ite
      m3') )
     ->show();

  /*
    Результат выполнения кода выведет следующее:
    <ul>
      <li>item1</li>
      <li>item2</li>
      <li>item3</li>
    </ul>
  */
?>
```

Описав, как будут работать наши классы, давайте теперь приступим к их реализации.

Класс **ListItem** по сути этот тот же класс **Tag**. С той разницей, что конструктор класса **Tag** требует имя тега, а конструктор **ListItem** не требует параметров, так как всегда создает один и тот же тег **li**.

Поэтому для реализации класса **ListItem** достаточно просто просто наследовать от класса **Tag**, переопределив его конструктор:

```
<?php
  class ListItem extends Tag
  {
    public function __construct()
    {
      // Вызываем конструктор родителя, передав в к
        ачестви имени 'li':
      parent::__construct('li');
    }
  }
?>
```

Давайте теперь напишем реализацию класса **HtmlList**.

Данный класс также удобно унаследовать от **Tag**, расширив затем родителя нужными нам методами. Итак, наследуем:

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

  }
?>
```

Класс **HtmlList** предназначен как для создания тегов **ul**, так и для создания тегов **ol**. Тип тега будет передаваться параметром конструктора, поэтому наш класс **HtmlList** не будет переопределять конструктор родителя - этот конструктор как раз такой, как нам нужно.

Внимательный читатель может заметить, что на самом деле в конструктор класса **HtmlList** можно передать любое имя тега, не только **ul** или **ol**. Пока проигнорируем эту проблему, оставив контроль имени тега на программисте-пользователе нашего класса.

Реализуем метод **addItem** для добавления пунктов списка:

```php
<?php
  class HtmlList extends Tag
  {
    private $items = []; // массив для хранения лишек

    public function addItem($li)
    {
      $this->items[] = $li;
      return $this; // вернем $this для цепочки
    }
  }
?>
```

Давайте улучшим наш код, указав, что параметр нашего метода принимает только объекты класса **ListItem**:

```php
<?php
  class HtmlList extends Tag
  {
    private $items = [];

    public function addItem(ListItem $li)
    {
      $this->items[] = $li;
      return $this;
    }
  }
?>
```

Давайте теперь сделаем метод **show**. На самом деле наш класс **HtmlList** наследует от своего родителя такой метод - но этот наследуемый метод делает немного не то, что нам нужно.

Наследуемый метод **show** выводит открывающий тег, закрывающий, а между ними текст. Но в нашем случае в качестве текста будут выступать теги **li**.

Давайте в таком случае просто переопределим метод **show** родителя и напишем ему свою реализацию:

```php
<?php
  class HtmlList extends Tag
  {
    private $items = [];

    public function addItem(ListItem $li)
    {
      $this->items[] = $li;
      return $this;
    }

    // Переопределим метод родителя:
    public function show()
    {
      // тут будет наша реализация без вызова parent::show
    }

```

```
    }
?>
```

Пишем свою реализацию:

```php
<?php
  public function show()
  {
    $result = $this->open(); // открывающий тег

    // тут надо сформировать лишки и добавить в $result

    $result .= $this->close(); // закрывающий тег

    return $result;
  }
?>
```

Давайте сформируем лишки. Для этого запустим цикл **foreach** для массива **$this->items**:

```php
<?php
  public function show()
  {
    $result = $this->open();

    foreach ($this->items as $item) {
      $result .= 'тут нужно добавлять теги li';
    }

    $result .= $this->close();

    return $result;
  }
?>
```

В нашем цикле нужно в переменную **$result** записывать теги **li** в формате *
текст
*.

Здесь нам очень поможет то, что объекты класса **ListItem** являются наследниками класса **Tag**, а следовательно, имеют метод **show**, который и делает то, что нам нужно.

В нашем цикле **foreach** в переменную **$item** как раз-таки попадают объекты класса **ListItem**. Значит, просто будем вызывать у них метод **show** и наша задача будет решена:

```php
<?php
  public function show()
  {
    $result = $this->open();

    foreach ($this->items as $item) {
      $result .= $item->show(); // вызываем метод show
     }

    $result .= $this->close();

    return $result;
  }
?>
```

Добавим созданный метод **show** в наш класс **HtmlList**:

```php
<?php
  class HtmlList extends Tag
  {
    private $items = [];

    public function addItem(ListItem $li)
    {
```

```
      $this->items[] = $li;
      return $this;
    }

    public function show()
    {
      $result = $this->open();

      foreach ($this->items as $item) {
        $result .= $item->show();
      }

      $result .= $this->close();

      return $result;
    }
  }
?>
```

Давайте проверим работу нашего класса:

```
<?php
  $list = new HtmlList('ul');

  echo $list
    ->addItem((new ListItem())->setText('ite
      m1'))
     ->addItem((new ListItem())->setText('ite
      m2'))
     ->addItem((new ListItem())->setText('ite
      m3'))
     ->show();
?>
```

Результат выполнения кода выведет следующее (форматирование мое):

```
<ul>
  <li>item1</li>
  <li>item2</li>
  <li>item3</li>
</ul>
```

А теперь рассмотрим не очевидные на первый взгляд бонусы: так как и класс **HtmlList**, и класс **ListItem** наследуют от класса **Tag**, то автоматически получают все его методы, например, **setAttr**.

Это дает нам возможность задавать атрибуты создаваемых тегов. Смотрите пример:

```
<?php
  $list = new HtmlList('ul');

  echo $list->setAttr('class', 'eee')
    ->addItem((new ListItem())->setText('ite
      m1')->setAttr('class', 'first'))
     ->addItem((new ListItem())->setText('ite
      m2'))
     ->addItem((new ListItem())->setText('ite
      m3'))
     ->show();

  /*
    Результат выполнения кода выведет следующее:
    <ul class="eee">
      <li class="first">item1</li>
      <li>item2</li>
      <li>item3</li>
    </ul>

```

```
    */
18 ?>
```

Задача 70.1

Реализуйте самостоятельно описанные мною классы. Проверьте их работу.

Задача 70.2

Сделайте так, чтобы при преобразовании наших классов к строке, метод **show** не нужно было вызывать. Модифицируйте весь код в соответствии с этим. Не забудьте про вот это место метода **show** класса **HtmlList**:

```
1 <?php
2   foreach ($this->items as $item) {
3     $result .= $item->show(); // здесь тоже пре
        образование к строке
4   }
5 ?>
```

Задача 70.3

Сделайте классы **Ul** и **Ol**, которые будут наследовать от класса **HtmlList**. Эти классы должны будут создавать соответствующий тип списков. Пример:

```
1 <?php
2   $ul = new Ul; // сделаем список ul
3   $ol = new Ol; // сделаем список ol
4 ?>
```

С помощью созданных классов выведите следующие списки:

```
1 <ul>
2   <li>item1</li>
3   <li>item2</li>
4   <li>item3</li>
5 </ul>
```

```
1 <ol>
2   <li>item1</li>
3   <li>item2</li>
4   <li>item3</li>
5 </ol>
```