



---

Создание web–приложений,  
исполняемых на стороне сервера при помощи  
языка программирования PHP  
и технологии AJAX

# Урок №4

## Основы ООП

### Содержание

<b>Введение .....</b>	<b>4</b>
<b>Инкапсуляция .....</b>	<b>6</b>
<b>Наследование .....</b>	<b>11</b>
Абстрактный класс .....	14
Интерфейс .....	21
Трейты .....	24
Несколько конструкторов в классе .....	30
<b>AJAX .....</b>	<b>34</b>
Связанные списки AJAX .....	45
<b>Сериализация .....</b>	<b>50</b>
<b>JSON .....</b>	<b>53</b>

<b>Magic методы</b> .....	<b>56</b>
<b>PDO (PHP Data Object)</b> .....	<b>63</b>
Prepared Statements .....	66
<b>Разработка сайта</b> .....	<b>72</b>
Создание базы данных .....	73
Класс Customer .....	77
Форма регистрации .....	83
Класс Item .....	86
Форма добавления товара .....	89
Страница выбора товара .....	91
Страница отображения корзины .....	101
Итоги .....	108
<b>Домашнее задание</b> .....	<b>111</b>

# Введение

---

Нам предстоит ознакомиться с тем, как в языке РНР надо использовать принципы ООП при создании веб-приложений. Отметим для себя, что реализация ООП в РНР была существенно изменена, начиная с версии РНР 5.0. Все, о чем мы будем говорить, относится именно к этой обновленной реализации ООП.

Сначала вспомним несколько важных понятий. Мы будем оперировать все теми же, привычными для вас терминами: класс, объект, свойства, методы и т.п. Все эти термины будут иметь тот же привычный смысл, но иногда с некоторой поправкой на специфику РНР. Рассмотрим все три основы ООП – инкапсуляцию, наследование и полиморфизм.

Основой инкапсуляции есть класс. Чем в РНР является класс? Если бы мы говорили о С# или Java, я бы в первую очередь сказал, что класс – это пользовательский тип данных. Для РНР такое определение класса будет не совсем точным, поскольку РНР не является строго типизированным языком. В РНР класс – это структура данных, способ организации пользовательских данных, тех данных, с которыми надо иметь дело в приложении. Класс может включать в себя переменные, которые так же, как и всегда, называются свойствами, и функции, которые называются методами. Все члены РНР класса делятся по уровню доступа на `public`, `protected` и `private`. Причем возле свойств этот спецификатор надо указывать всегда, а возле методов можно не указывать. Методы

ды по умолчанию – `public`. Методы бывают как объектные, так и `static`.

Если мы не указываем в классе конструктор явно, то применяется конструктор по умолчанию, создаваемый РНР. Конечно же, мы можем создать в классе свой конструктор. Однако конструктор в РНР классе может быть только один! РНР не поддерживает перегрузки функций, к которой вы привыкли в С# и Java. В РНР перегрузки функций нет совсем, хотя многие РНР-разработчики могут обидеться за такое утверждение. Они пользуются термином «перегрузка», но это нечто совсем другое, о чем мы поговорим позже. Пока запомните: конструктор в классе только один. Однако если вам понадобится создавать объекты своего класса несколькими разными способами, не расстраивайтесь – решение есть. О нем мы тоже поговорим позже.

Конечно же в РНР поддерживается наследование. Множественного наследования нет, как и в С# и в Java. Однако есть абстрактные классы и интерфейсы. И еще кое-что интересное, называемое трейтами.

Изучение ООП в РНР открывает перед программистом новые горизонты. И это не гипербола и не преувеличение. ООП – это ключ к пониманию Ajax, к использованию новых механизмов для работы с БД, таких как `mysqli` и `PDO`. Наконец, это средство к изучению нового способа работы с РНР – фреймворков, которые практически все используют ООП идеологию.

На этом мотивирование завершаем и приступаем к работе.

# Инкапсуляция

Создайте в папке test файл с именем, например, oop1.php и занесите в него такой код:

```
<?php
class Point
{
    public $x;
    public $y;
}
$p=new Point();
?>
```

Этот код создает класс с именем Point и с двумя открытыми свойствами \$x и \$y. После определения класса написана строка кода, создающая переменную \$p типа этого класса, т.е. – объект. Попросите Apache прислать вам в браузер этот файл, для чего укажите в адресной строке такой адрес: <http://localhost/test/oop1.php>.

Вы не увидите на странице ничего. В том числе и сообщений о каких-то ошибках. Это вселяет надежду, что объект класса Point мы создали. Можно ли этот объект увидеть? Попробуйте для этого проверенную конструкцию echo. Вы получите сообщение о том, что у echo не получилось преобразовать объект класса Point в строку. Запомните это: echo не умеет выводить описание объектов. Точнее говоря, echo не умеет это делать без некоторой предварительной подготовки. Скоро вы научитесь выводить описания своих объектов с помощью echo. А сейчас идем дальше. Попробуйте вывести описание на-

шего объекта с помощью функции `var_dump()`. В этот раз все получится. У меня это выглядит так:

```
object(Point)#1 (4) { ["x"]=> NULL ["y"]=> NULL }
```

Мы видим нечто, похожее на ассоциативный массив, в котором имена ключей совпадают с именами свойств нашего класса, а значения пока пусты. Давайте инициализируем свойства созданного объекта. Добавьте в наш файл такие строки:

```
<?php
class Point
{
    public $x;
    public $y;
}
$p=new Point();
$p->x=100;
$p->y=200;
var_dump($p);
?>
```

Активируйте наш файл снова, и вы увидите указанные значения в описании объекта. Теперь добавим в класс собственный метод для вывода описания объекта:

```
<?php
class Point
{
    public $x;
    public $y;
    function Show()
    {
        echo 'Vertex: ('.$this->x.', '.$this->y.' )';
    }
}
```

```
}  
$p=new Point ();  
$p->x=100;  
$p->y=200;  
$p->Show() ;  
?>
```

Мы не указываем спецификатор доступа возле созданного метода, по умолчанию метод будет public. Обратите внимание, как мы обращаемся к свойствам класса внутри класса – обязательно через указатель `this`. Этот указатель содержит адрес текущего объекта и используется только внутри определения класса для ссылки на члены класса. Правда, этот указатель нельзя использовать внутри статических методов. Отметим, что при обращении к свойству через `this` символ '\$' переходит к указателю `this`, а возле имени свойства, после селектора, символ '\$' не указывается. И посмотрите, как мы вызываем созданный метод от имени нашего объекта `$p`.

Конечно, инициализировать свойства объекта так, как мы сделали в этом примере, неудобно. Лучше это делать в конструкторе. Добавим в наш класс конструктор с параметрами. Здесь мы встречаем одну особенность реализации ООП в PHP. Вы уже привыкли, что имя конструктора совпадает с именем класса. В PHP это тоже так. Было. До версии PHP 5.0. Начиная с версии PHP 5.0, конструктор в любом классе PHP имеет имя `__construct()`. В начале имени стоят два символа подчеркивания. Запомните, многие системные конструкции в PHP имеют имена, начинающиеся с одного или двух символов `'_'`. Поэтому нам не стоит так называть соб-



ственные переменные, массивы, классы и т.п. Добавим конструктор с двумя параметрами в наш класс:

```
<?php
class Point
{
    private $x;
    private $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
    function Show()
    {
        echo 'Vertex: ('.$this->x.','.$this->y.')
<br/>';
    }
}
$p=new Point (100,200);
$p->Show();
var_dump($
```

Понятно, что теперь строка создания объекта отличается от предыдущего варианта. Когда мы можем инициализировать свойства с помощью конструктора, то, в целях безопасности, их лучше сделать `private`. Как и всегда, для обслуживания `private` свойства можно создавать `public` сеттеры и геттеры.

Отметьте еще одну особенность: если в вашем классе нет явного конструктора, то вы можете создавать объекты такого класса, не указывая скобки после имени класса. Создайте еще один класс без конструктора, создайте его объект таким способом и посмотрите на результат:

```
<?php
class Rectangle
{
    public $top;
    public $left;
    public $width;
    public $height;
    function Show()
    {
        echo 'Vertex: ('. $this->top. ', '. $this->
left. ') width: '.
        $this->width. ' height: '. $this->
height. '<br/>';
    }
}
$r=new Rectangle;
$r->Show();
var_dump($r);
```

Вы видите результаты вывода метода Show() и var\_dump(), которые подтверждают, что объект был создан, но его свойства, конечно же, пока не инициализированы.

# Наследование

---

Наследование – это способ создания нового класса не с «чистого листа», а на основе существующего другого класса. Существующий класс, используемый для создания нового, называется родительским классом или суперклассом. Новый создаваемый класс называется дочерним или производным классом. Производный класс получает в наследство от своего родительского класса всю функциональность – свойства и методы, но не получает конструктор. К тому же, если в родительском классе есть `private` свойства, то они передаются в производный класс, но остаются там недоступными ни для чтения, ни для записи. Свойства родительского класса удобно описывать спецификатором доступа `protected`. Это делает свойства закрытыми для всех, кроме потомков класса.

Для обозначения наследования классу в PHP используется спецификатор `extends`. Давайте создадим новый класс `Rectangle`, наследуя его от нашего класса `Point`.

При создании производного класса вы должны помнить о том, что конструктор родительского класса не передается в производный класс. Всегда помните о таком правиле: если у родительского класса есть конструктор с параметрами, то в производном классе надо тоже создать конструктор с параметрами, и число параметров в этом конструкторе должно быть суммой количества

параметров родительского класса и числа параметров, необходимых для производного класса. Кроме этого, в теле конструктора производного класса надо вызвать конструктор родительского класса и передать ему необходимые параметры. Как это правило надо выполнить в нашем случае? Наш родительский класс Point требует для своего конструктора два параметра. В классе Rectangle есть два свойства, которые желательно инициализировать конструктором. Поэтому нам надо создать конструктор с четырьмя параметрами: два из них для родительского класса, два – для производного. В теле этого конструктора надо отдать два параметра конструктору родительского класса. Для доступа к членам родительского класса, не только конструктора, используется конструкция parent::.

Создайте новый файл с именем oop2.php и добавьте в него такой код:

```
<?php
class Point
{
    protected $x;
    protected $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
    function Show()
    {
        echo 'Vertex: ( '.$this->x.', '.$this->y.' )
<br/>';
    }
}
```

```

class Rectangle extends Point
{
    protected $width;
    protected $height;
    function __construct($x, $y, $w, $h)
    {
        parent::__construct($x, $y);
        $this->width=$w;
        $this->height=$h;
    }
}
$r=new Rectangle(100,100,200,150);
$r->Show();
var_dump($r);
?>

```

Активировав этот файл, вы получите такой вывод:

```

Vertex: (100,100)
object(Rectangle)#1 (4) { ["width":protected]=> int(200)
["height":protected]=> int(150) ["x":protected]=> int(100)
["y":protected]=> int(100) }

```

О чем говорит этот результат? Во-первых, объект класса Rectangle создан, это подтверждает вывод `var_dump()`. Во-вторых, наследование работает, так как мы вызвали метод `Show()` от объекта класса Rectangle, в котором этот метод мы не создавали, т.е. метод `Show()` мы получили наследованием. В-третьих, работа метода `Show()` может нас не устроить, так как этот метод не показывает значения свойств `$width` и `$height`. Мы можем переопределить метод `Show()` в классе Rectangle, чтобы он выводил и значения свойств `$width` и `$height`.

Например, так:

```
function Show()
{
    echo 'Vertex: ('. $this->top. ', '. $this->left. ')
                                width: '.
    $this->width. ' height: '. $this->
                                height. ' <br/>';
}
```

Вы должны помнить, что при переопределении метода нельзя изменять его сигнатуру.

Чтобы еще раз обратить внимание на применение конструкции `parent::`, продемонстрируем другой вариант переопределения метода `Show()`:

```
function Show()
{
    parent::Show();
    echo 'Width: '. $this->width. '
    height: '. $this->height. ' <br/>';
}
```

Здесь мы сначала вызываем родительский вариант метода `Show()`, который выводит в браузер описание вершины, а затем добавляем вывод ширины и высоты прямоугольника.

## Абстрактный класс

Предположим, что нам необходимо вычислять площади и периметры созданных прямоугольников. Для этого, мы можем добавить в класс `Rectangle` методы `Area()` и `Perimeter()`. А что если нам понадобится еще иметь дело с классом `Circle` (окружность), тоже произ-

водным от класса Point? И нас тоже будут интересовать площади и периметры созданных окружностей?

Я понимаю, что данный пример немного далек от веба. Вряд ли вам придется писать сайты, вычисляющие площади и периметры. Однако этот пример очень нагляден и показателен для объяснения того, о чем я хочу рассказать. Поэтому уделим пять минут созданию великого Евклида – геометрии.

Создайте файл oop3.php и занесите в него такой код – нашу исходную иерархию классов:

```
<?php
class Point
{
    protected $x;
    protected $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
    function Show()
    {
        echo 'Vertex: ('. $this->x. ', '. $this->y. ')
<br/>';
    }
}
class Rectangle extends Point
{
    protected $width;
    protected $height;
    function __construct($x, $y, $w, $h)
    {
        parent::__construct($x, $y);
        $this->width=$w;
        $this->height=$h;
    }
}
```

```

function Show()
{
    echo 'Vertex: ('. $this->x. ', '. $this->y. ')
                                width: '.
    $this->width. ' height: '. $this->
                                height. '<br/>';
}
}

class Circle extends Point
{
    protected $radius;
    function __construct($x, $y, $r)
    {
        parent::__construct($x, $y);
        $this->radius=$r;
    }
    function Show()
    {
        echo 'Vertex: ('. $this->x. ', '. $this->y. ')
                                radius: '.
        $this->radius. '<br/>';
    }
}
$r=new Rectangle(100,100,200,150);
$r->Show();
$c=new Circle(200,200,100);
$c->Show();
?>

```

К созданному ранее классу `Rectangle` мы добавили новый класс `Circle`, также производный от `Point`, и в обоих классах переопределили родительский метод `Show()`. Теперь вспоминаем, что нам надо вычислять площади и периметры, и добавляем соответствующие методы в классы `Rectangle` и `Circle`. Поскольку оба эти класса имеют общего родителя, правильно будет добавить эти



методы в родительский класс `Point`, позволив производным классам переопределить их. Так и поступим.

Правда, есть одна проблема: я знаю, как вычислять площадь и периметр для прямоугольника и окружности, но не знаю, как это делать для точки. Это не тривиальный вопрос, на самом деле. Дело в том, что у точки нет площади и периметра. Они не равны нулю – их просто нет по определению. Но мы очень часто решаем возникающие проблемы в некотором приближении, используя не общее решение, а частное, но подходящее для нашего случая. Так мы и поступим, положив, что площадь и периметр нашей точки равны нулю. Мы не зря уделили так много слов этой проблеме. Скоро увидите, почему.

В класс `Point` добавляем два таких метода:

```
function Area()  
{  
    return 0;  
}  
function Perimeter()  
{  
    return 0;  
}
```

В класс `Rectangle` добавляем два таких метода:

```
function Area()  
{  
    return $this->width * $this->height;  
}  
function Perimeter()  
{  
    return ($this->width + $this->height)*2;  
}
```

В класс Circle добавляем два таких метода:

```
function Area()
{
    return $this->radius * $this->radius * 3.1415;
}
function Perimeter()
{
    return $this->radius * 3.1415 * 2;
}
```

Что нам дает тот факт, что наши классы связаны наследованием? Это позволяет нам писать единообразный код для обработки объектов таких классов. Добавьте после наших трех классов такую обработку:

```
$figures=array();
$figures[]=new Rectangle(100,200, 100,100);
$figures[]=new Circle(200,200, 100);
$figures[]=new Point(100,100);
$figures[]=new Circle(300,200, 100);
$totalArea=0;
$totalPerimeter=0;
foreach($figures as $f)
{
    $totalArea += $f->Area();
    $totalPerimeter += $f->Perimeter();
}
echo 'Total area is:'. $totalArea. '<br/>';
echo 'Total perimeter is:'. $totalPerimeter. '<br/>';
```

Мы собрали в одном массиве произвольный набор объектов трех разных классов. А затем вычислили суммарные площади и периметры всех этих фигур. Кстати, у меня получился такой результат:

Total area is:72830

Total perimeter is:1656.6

А вы запомните свои результаты. Нам это скоро понадобится.

Что в этом коде ценного? То, что объекты трех разных классов мы обрабатываем в одном единственном цикле. Это возможно потому, что классы связаны наследованием. Если бы не наследование, нам надо было бы использовать три разных цикла: по одному для объектов каждого класса. Можно сказать, что задача решена? В некотором приближении, да.

Я хочу вернуться к реализации методов `Area()` и `Perimeter()` в классе `Point`. Дело в том, что не во всякой конкретной ситуации можно будет так легко поступить с реализацией тех методов в родительском классе, которые, вообще говоря, в этом классе реализации не имеют и предназначены для того, чтобы их переопределяли классы-потомки. Ну нет площади и периметра у точки! Да они нам у точки и не нужны. Нам важно получить удобный механизм вычисления для объектов-потомков. Как поступить в том случае, когда `return 0` не подойдет?

В таком случае эти методы не надо реализовывать вовсе! В классе можно описать метод без реализации. Такой метод называется абстрактным. Дальше работает такое правило: если в классе есть хотя бы один абстрактный метод, то класс становится абстрактным. В абстрактном классе могут быть обычные методы, свойства, конструктор – все, как в обычном классе. Плюс один или несколько методов без реализации – абстракт-

ных методов. Создайте копию файла oop3.php, назовите ее abstract.php и перепишите в ней класс Point как абстрактный:

```
abstract class Point
{
    protected $x;
    protected $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
    function Show()
    {
        echo 'Vertex: ('. $this->x. ', '. $this->y. ')
                                     <br/>';
    }
    abstract function Area();
    abstract function Perimeter();
}
```

Активируйте файл abstract.php и получите сообщение об ошибке: Cannot instantiate abstract class Point. За все надо платить. Плата за использование абстрактного класса заключается в том, что мы не можем создавать объекты абстрактного класса. Поэтому закомментируйте добавление в массив \$figures всех объектов Point и активируйте файл abstract.php снова. В это раз все в порядке. Вы должны получить такой же результат, как раньше. И это естественно, поскольку точки не вносили вклад в суммарную площадь и в суммарный периметр.

Итак, повторим.

Что такое абстрактный класс? Это класс с одним или несколькими абстрактными методами.

Что такое абстрактный метод? Это метод без реализации.

Зачем нужны абстрактные методы? Чтобы классы наследники реализовывали их по-своему.

Могут ли в абстрактном классе находиться неабстрактные члены? Да, без ограничений.

Зачем нужны абстрактные классы? Чтобы служить в качестве базовых типов для создания классов, связанных наследованием.

Может ли класс-наследник реализовать не все абстрактные методы своего базового класса? Да, может, но при этом сам станет абстрактным.

## Интерфейс

Наряду с абстрактными классами в РНР есть еще один замечательный кандидат на роль базового типа – это интерфейс. Интерфейс традиционно является контейнером методов без реализации, т.е. по сути – абстрактных методов, хотя спецификатор `abstract` в интерфейсе никогда не используется. Кроме таких методов в интерфейсе, в отличие от абстрактного класса, могут находиться еще только константы. Никаких свойств и методов с реализацией в интерфейсе быть не может. Еще одно отличие интерфейса как базового типа состоит в том, что один класс может сразу наследовать многим интерфейсам. В то время как базовый абстрактный класс может быть только один. Эта особенность интерфейсов позволяет компенсировать недостатки отсутствия множественного наследования.

Поскольку мы пришли к выводу, что присутствие

методов `Area()` и `Perimeter()` в классе `Point` неуместно, заберем их из класса вообще. Вынесем эти методы в интерфейс и посмотрим, что нам это даст. Создайте копию из файла `abstract.php` и назовите новый файл `interface.php`. Занесите в него такой код:

```
<?php
interface Geometry
{
    const Pi=3.1415926;
    function Area();
    function Perimeter();
}

class Point
{
    protected $x;
    protected $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
    function Show()
    {
        echo 'Vertex: ('. $this->x. ', '. $this->y. ')
        <br/>';
    }
}

class Rectangle extends Point implements Geometry
{
    protected $width;
    protected $height;
    function __construct($x, $y, $w, $h)
    {
        parent::__construct($x, $y);
        $this->width=$w;
        $this->height=$h;
    }
}
```

```

function Show()
{
    echo 'Vertex: ('. $this->x. ', '. $this->y. ')
                                width: '.
    $this->width. ' height: '. $this->
                                height. '<br/>';
}
function Area()
{
    return $this->width * $this->height;
}
function Perimeter()
{
    return ($this->width + $this->height)*2;
}
}

class Circle extends Point implements Geometry
{
    protected $radius;
    function __construct($x, $y, $r)
    {
        parent::__construct($x, $y);
        $this->radius=$r;
    }
    function Show()
    {
        echo 'Vertex: ('. $this->x. ', '. $this->y. ')
                                radius: '.
        $this->radius. '<br/>';
    }
    function Area()
    {
        return $this->radius * $this->radius *
                                Geometry::Pi;
    }
    function Perimeter()
    {
        return $this->radius * Geometry::Pi * 2;
    }
}

```

```

$figures=array();
$figures[]=new Rectangle(100,200, 100,100);
$figures[]=new Circle(200,200, 100);
//$figures[]=new Point(100,100);
$figures[]=new Circle(300,200, 100);
$totalArea=0;
$totalPerimeter=0;
foreach($figures as $f)
{
    $totalArea += $f->Area();
    $totalPerimeter += $f->Perimeter();
}
echo 'Total area is:'. $totalArea. '<br/>';
echo 'Total perimeter is:'. $totalPerimeter. '<br/>';
?>

```

Как видите, наш пример работает, как и ранее. Обратите внимание, что `Point` уже не абстрактный. Значит, интерфейс вполне годится на роль базового типа.

Понятно, у вас возникает вопрос, что выбрать в качестве родоначальника иерархии: абстрактный класс или интерфейс? Я бы советовал рассуждать таким образом. Если в базовом типе надо иметь не только методы, но и свойства, тогда выбирайте абстрактный класс. Если вы хотите сделать свою иерархию классов более гибкой и масштабируемой, лучше будет разнести наследуемые методы по разным интерфейсам. Запомните: если ваш класс наследует интерфейсу, то он обязан реализовать все методы, объявленные в интерфейсе.

## Трейты

В версии PHP 5.4 появилось еще одно интересное дополнение, предназначенное для создания эффектив-



ного наследования, называемое трейтами (trait). Трейты предназначены компенсировать проблемы, вызванные отсутствием множественного наследования. Посмотрим на наш код, в котором мы заносим объекты в массив `$figures`:

```
$figures=array();  
$figures[]=new Rectangle(100,200, 100,100);  
$figures[]=new Circle(200,200, 100);  
//$figures[]=new Point(100,100);  
$figures[]=new Circle(300,200, 100);
```

Этот код написан не в ООП стиле. Инкапсуляция требует, чтобы объект каждого класса был максимально автономным, т.е. весь обслуживающий код содержал в себе. В приведенном же скрипте мы выполняем добавление объектов в массив как бы «извне». Как исправить эту ситуацию? Добавить в каждый класс метод, позволяющий добавлять объект в некую коллекцию. Такое решение будет дублированием кода, что не есть хорошо. Добавить такой метод в интерфейс и реализовать в каждом классе-наследнике интерфейса? Снова дублирование при реализации. Вот здесь полезными могут оказаться трейты. В приведенном коде не зря присутствует закомментированная строка добавления объекта `Point` – трейты помогут решить и этот вопрос. Создайте новый файл, назовите его `trait.php`, скопируйте в него код из `interface.php` и внесите такие изменения:

```

<?php
// Trait
trait Collect
{
    public function Add(&$arr)
    {
        $arr[]=$$this;
    }
}

interface Geometry
{
    //copy from interface.php
}

class Point
{
    //copy from interface.php
}

class Rectangle extends Point implements Geometry
{
    use Collect;
    //copy from interface.php
}

class Circle extends Point implements Geometry
{
    use Collect;
    //copy from interface.php
}

$figures=array();
$o1=new Rectangle(100,200, 100,100);
$o2=new Circle(200,200, 100);
$o3=new Point(100,100);
$o4=new Circle(300,200, 100);
$o1->Add($figures) ;
$o2->Add($figures) ;
$o4->Add($figures) ;

```

```

$totalArea=0;
$totalPerimeter=0;
foreach($figures as $f)
{
    $totalArea += $f->Area();
    $totalPerimeter += $f->Perimeter();
}
echo 'Total area is:'. $totalArea. '<br/>';
echo 'Total perimeter is:'. $totalPerimeter. '<br/>';
?>

```

В самом начале файла мы добавили определение трейта с именем Collec. Как видите, в нем содержится вполне определенный метод. Об этом методе поговорим немного позже, а сейчас перечислим, что может содержаться в трейте.

В трейте может содержаться определенный метод, как в нашем примере.

В трейте может содержаться абстрактный метод, и тогда класс, использующий такой трейт, должен этот метод реализовать.

В трейте может содержаться свойство – тогда класс, использующий такой трейт, не может у себя объявлять свойство с таким же именем, но сможет использовать свойство трейта.

В трейте могут содержаться статические свойства и методы.

В трейте можно использовать указатель `this` (как в нашем примере), и этот указатель будет относиться к тому объекту, который использует трейт.

Теперь рассмотрим наш метод. В него по ссылке передается параметр `$arr`. Поскольку передача параметров в PHP по умолчанию выполняется по значению, мы эту

ситуацию изменяем, указав `&`. Мы хотим добавлять объекты, вызвавшие этот метод, во внешний массив. Обратите внимание, мы используем запись `$this`, т.е. забираем «весь объект». Но при необходимости, можно было бы обращаться к какому-либо одному свойству, например `$this->width`.

С помощью инструкции `use Collect` мы вставляем ссылку на трейт в классы `Rectangle` и `Circle`. Это приводит к тому, что любой объект этих классов сможет вызывать метод `Add()` как свой собственный объектный метод. Примеры таких вызовов вы видите внизу:

```
$o1->Add($figures);  
$o2->Add($figures);  
$o4->Add($figures);
```

Я пока деликатно обхожу стороной класс `Point`. Скоро увидите, почему.

Активируйте наш файл `trait.php`, указав в браузере такой путь:

<http://localhost/test/trait.php>

Вы получите сообщение об ошибке. Почему возникла эта ошибка? В начале урока написано, что трейты появились в PHP, начиная с версии PHP 5.4. А мы работаем в версии PHP 5.3. Но нам повезло. Наша среда `OpenServer` позволяет изменить версию используемого PHP буквально «на лету». Правда, отметьте, что иногда изменение версии PHP требует и изменения версии `Apache`. Но не в нашем случае. Нам достаточно изменить только версию PHP 5.3 на 5.4 и оставить пока прежнюю версию `Apache` 2.2. Это можно сделать из меню `Settings`

во вкладке Modules. Измените версию PHP и снова активируйте наш файл. Вы увидите знакомый результат:

Total area is:72831.852

Total perimeter is:1656.63704

Обсудим, что нам дало использование трейтов. Говорят, что трейты реализуют «горизонтальное наследование». В этом термине что-то есть. Мы можем спокойно добавлять трейты любому классу, не тревожа его «родственников» по наследованию. При этом мы избегаем дублирования кода в классах. Наш метод Add() определен единожды, в трейте, и не дублируется в классах.

Поговорим теперь о классе Point. Он уже не абстрактный, и мы можем создавать его объекты. Предположим, что нам нужны объекты этого класса. Но мы же не можем добавлять их в массив \$figures, так как в классе Point нет способа вычисления площади и периметра. Сейчас мы просто обходим объект \$o3 (точку) и не добавляем его в массив:

```
$o1->Add($figures);  
$o2->Add($figures);  
$o4->Add($figures);
```

Но это же никуда не годится. А если объекты создаются динамически, и мы не знаем кто из них кто? Как поступить в такой ситуации? Добавим в метод Add() простую проверку:

```

trait Collect
{
    public function Add(&$arr)
    {
        if(!$this instanceof Geometry) return
                                   false;

        $arr[]=$this;
    }
}

```

Конструкция `if(!$this instanceof Geometry)` позволяет определить, является ли текущий объект наследником `Geometry`, другими словами, есть ли у него методы вычисления площади и периметра. Если нет – такой объект просто не добавляется в массив.

Теперь разрешим объектам класса `Point` использовать трейт, добавив в этот класс `use Collect`. Что нам это дало? Нам не надо проверять, кто есть точка, а кто не точка, и мы вызываем метод `Add()` для всех объектов:

```

$o1->Add($figures);
$o2->Add($figures);
$o3->Add($figures);
$o4->Add($figures);

```

Код обработки единообразный для всех. Оставьте активной версию РНР 5.4, и двигаемся дальше.

## Несколько конструкторов в классе

Мы уже говорили о том, что в одном классе РНР не может быть более одного конструктора. Как можно поступить в том случае, если мы хотим иметь возможность создавать объекты какого-либо класса несколькими

разными способами? Решений может быть несколько. Я предлагаю вам познакомиться с одним из них. Рассмотрим для примера класс `Rectangle`, но сейчас он не будет связан наследованием с какими-либо классами:

```
class Rectangle
{
    protected $x;
    protected $y;
    protected $width;
    protected $height;
    function __construct($x, $y, $w, $h)
    {
        $this->x=$x;
        $this->y=$y;
        $this->width=$w;
        $this->height=$h;
    }
    function Show()
    {
        echo 'Vertex: ('. $this->x. ', '. $this->y. ')
                                width: '.
        $this->width. ' height: '. $this->
                                height. '<br/>';
    }
}
```

Такой конструктор позволяет создавать объекты этого класса, принимая координаты вершины, ширину и высоту прямоугольника. Предположим, нам надо иметь возможность создавать объекты этого класса, указывая координаты левой верхней и нижней правой вершин. Как это можно сделать? Очень часто для этой цели в классе используют статические методы, создающие и возвращающие объекты текущего класса.

Например, так:

```
public static function createInstance1( $x1, $y1,
                                       $x2, $y2 )
{
    $instance = new self();
    $instance->x=$x1;
    $instance->y=$y1;
    $instance->width=($x2-$x1);
    $instance->height=($y2-$y1);
    return $instance;
}
```

Или же предположим, что вам часто надо создавать прямоугольники, расположенные в точке (0,0) и вы хотите сэкономить на параметрах:

```
public static function createInstance2($w, $h )
{
    $instance = new self();
    $instance->x=0;
    $instance->y=0;
    $instance->width=$w;
    $instance->height=$h;
    return $instance;
}
```

Обратите внимание на указатель **self**, используемый в созданных статических методах. Это – некий аналог указателя **this**, но только для статических методов. Ведь в статическом методе нельзя обращаться к нестатическим свойствам класса, доступ к которым предоставляет **this**. Однако в статическом методе можно обращаться к статическим свойствам класса и именно для этого и можно использовать указатель **self**. Мы используем его



для создания нового объекта текущего класса внутри статического метода.

Я думаю, принцип понятен. Дальше должна работать ваша креативность. Приведенное решение не является самым лучшим для любой ситуации. Вы можете поискать другие решения или придумать собственные.

К этому моменту вы уже понимаете, как происходит взаимодействие браузера и веб-сервера. В ответ на каждый Request браузера веб-сервер присылает свой Response. Когда браузер получает Response, он загружает в свою клиентскую область разметку, присланную в этом Response. Предыдущее изображение в клиентской области браузера затирается. Кроме этого, если Response от веб-сервера по какой-либо причине задерживается, браузер может оказаться заблокированным.

В таком сценарии есть два неприятных момента: полное обновление клиентской области и блокировка браузера.

Очень часто не требуется полностью перерисовывать текущее содержимое клиентской области браузера. Достаточно изменить только определенную часть окна. Но классическая схема работы не способна обеспечить такой сценарий. Поэтому страница перерисовывается полностью, что приводит к пересылке неизменяемых данных и увеличивает трафик.

Почему плоха блокировка браузера, объяснять не надо.

Технология AJAX позволяет избавиться от этих проблем. Она позволяет передавать на веб-сервер асинхронные запросы, избегая блокировки браузера, и обновлять только определенные части страницы, избегая пересылки всей страницы, и уменьшая, таким образом, трафик. Термин AJAX расшифровывается, как «Асин-

хронный Javascript и XML» (Asynchronous Javascript And XML). Об AJAX сложилось мнение, как о чем-то полезном, но непонятном, и часто его используют «вслепую». Я попробую убедить вас в том, что эта технология очень проста.

Как понятно по названию, AJAX происходит из Javascript. В Javascript создан замечательный объект с некоторыми свойствами и методами. Для удобства назовем этот объект **ao** (AJAX Object).

Сначала рассмотрим два основных метода этого объекта, не рассматривая их параметры и способы использования.

- **ao.Open()** – этот метод умеет создавать асинхронные запросы к веб-серверу. Обратите внимание, метод **Open()** создает запрос, готовит все необходимые данные, но не отправляет этот запрос;
- **ao.Send()** – этот метод умеет отправлять асинхронные запросы веб-серверу, созданные методом **Open()**.  
Теперь поговорим о свойствах.
- **ao.readyState** – это целочисленное свойство, которое описывает текущее состояние объекта **ao**. Оно может принимать такие значения:
  - 0 – объект еще не инициализирован;
  - 1 – объект уже вызвал метод **Open()**, но еще не вызвал метод **Send()**;
  - 2 – объект уже вызвал метод **Send()**, но ответ от веб-сервера еще не прибыл;
  - 3 – ответ от веб-сервера уже прибыл, но присланные данные еще не готовы к использованию;
  - 4 – присланные данные уже готовы к использованию.

Обратите внимание, что значения этого свойства изменяются автоматически, в зависимости от происходящих событий, связанных с обработкой асинхронного запроса. Например, мы вызвали метод `Send()`, т.е. наш асинхронный запрос полетел к веб-серверу, и это свойство стало равно 2. Получили ответный `Response` – и значение свойства изменилось на 3. Перейдем к рассмотрению следующих свойств.

- **`ao.status`** – позволяет понять, что прибыло в ответном `Response` от веб-сервера. Это свойство хранит код завершения `Request`. Если это свойство равно 200, значит, запрос успешно обработан.
- **`ao.responseText`** – именно в это свойство заносится полученный от веб-сервера HTML ответ. Значение свойства `ao.readyState` изменяется со значения 3 на 4 именно после того, как в свойство `ao.responseText` будут занесены полученные данные.
- **`ao.responseXML`** – вы можете написать асинхронный запрос, в котором будете просить прислать вам данные не в виде HTML, а в виде XML. Такие данные будут заноситься в свойство `ao.responseXML`, в то время как обычные HTML данные будут заноситься в свойство `ao.responseText`.
- **`ao.onreadystatechange`** – в это свойство надо занести адрес функции, которая будет обрабатывать полученные от веб-сервера данные. Функция заберет данные из `ao.responseXML` или из `ao.responseText` и сделает с ними то, что нам надо.

Обратите внимание, все эти действия происходят автоматически и асинхронно. Нам надо только начать цепочку событий, вызвав методы `Open()` и `Send()`, а все последующее будет выполняться автоматически.

Тип этого замечательного объекта – `XMLHttpRequest`. Сегодня он доступен практически во всех браузерах, правда, инициализировать его надо в разных браузерах по-разному.

Давайте теперь применим всю эту теоретическую информацию на практике. Сначала рассмотрим простой пример, а потом применим AJAX в ходе создания нашего нового сайта. Кроме того, мы планировали переделать работу со связанными списками на странице `tours.php` нашего туристического сайта.

Каким будет наш простой пример? Мы создадим страницу с текстовым полем, в которое пользователь сможет вводить имя. По мере ввода имени, после каждого события `oninput`, мы будем асинхронно отправлять введенный текст на сервер. На сервере в массиве будет храниться какой-то список имен. Если введенное пользователем имя совпадет с какими-то именами в массиве, сервер пришлет список имен, совпадающих с введенным значением. Это будет что-то вроде всплывающей подсказки.

Для нашего примера мы будем использовать три файла:

- `index.html` с текстовым полем;
- `script.js` с ajax кодом;
- `handler.php`, как обработчик ajax запроса.

Создайте папку `ajax1` и в ней указанные три файла.

В index.html вставьте такую разметку:

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ajax example</title>
    <script type="text/javascript" src="script.
                                   js"></script>
</head>
<body>
<input type="text" id='usertext' oninput="Process()" >
<div id="result"></div>
</body>
</html>
```

Обратите внимание, что в этом файле нет ни формы, ни кнопок submit. AJAX не нуждается в формах и таких кнопках для отправки своих асинхронных запросов. В этом файле мы присвоили id тем элементам, к которым будем обращаться из Javascript. Еще мы указали обработчик oninput, по которому будем запускать наш AJAX запрос. Вместо oninput можно использовать любое другое событие, просто это событие подходит нам по логике.

В script.js вставьте такой код:

```
var ao = createAjaxObject(); //ao is global variable

function createAjaxObject()
{
    var ao=null;
    try
    {
        ao = new XMLHttpRequest(); //for modern
browsers
    }
}
```

```

        catch(e)
        {
            try{ //for new IE
                ao = new ActiveXObject("Msxml2.
                                        XMLHTTP");
            }
            catch(e)
            {
                try{ //for old browsers
                    ao = new
                    ActiveXObject("Microsoft.XMLHTTP");
                }
                catch(e){
                    alert("AJAX is not supported
                            by your browser!")
                    return false;
                }
            }
        }
        return ao;
    }

function Process()
{
    if(ao.readyState == 4 || ao.readyState == 0)
    {
        name = document.
        getElementById("usertext").value;
        ao.open("GET", "handler.php?name="+name,
                true);
        ao.onreadystatechange = getData;
        ao.send(null);
    }
}

function getData()
{
    if(ao.readyState == 4 && ao.status == 200)
    {
        resp = ao.responseText;
        document.getElementById("result").i
        nnerHTML = resp;
    }
}

```

Функцию, которая пытается кроссбраузерно создать AJAX объект, я назвал `CreateAjaxObject()`. Она пытается по очереди создать этот объект сначала для современных браузеров, потом для новых версий IE, потом для старых браузеров. Логика работы этой функции понятна. Мы создаем глобальную переменную `ao`, чтобы она была доступна в других функциях.

Все начинается с функции `Process()` – именно в ней готовится асинхронный запрос в методе `Open(«GET», «handler.php?name=»+name, true)`.

Первый параметр говорит о том, что для передачи данных мы хотим использовать метод «GET». Скоро рассмотрим, как использовать «POST».

Во втором параметре мы указываем, какой файл на сервере должен принять данные и обработать наш AJAX запрос. Мы указали обработчик `handler.php?name=»+name` и передали ему в адресной строке данные под именем `name`. Эти данные представляют собой то, что пользователь ввел в текстовое поле с `id='usertext'`. Мы извлекаем их в предыдущей строке: `name = document.getElementById(«usertext»).value`.

В третьем параметре мы передаем булевское значение `true`. Значит, мы хотим, чтобы наш AJAX запрос выполнялся асинхронно. Если бы мы передали здесь значение `false`, запрос был бы синхронным.

Обратите внимание, что в свойство `onreadystatechange` мы заносим адрес функции: `ao.onreadystatechange = getData;`

Скобки после имени функции не указаны! Т.е. `onreadystatechange` – это делегат. Указанная здесь функция `getData()` будет вызвана автоматически, когда дан-



ные из Response будут занесены в `ao.responseText`. Эта функция просто заберет присланные данные и вставит в заготовленный `div` с `id='result'`.

Кстати, свойство `onreadystatechange` можно инициализировать анонимной функцией:

```
ao.onreadystatechange = function()
{
  if(ao.readyState == 4 && ao.status == 200)
  {
    resp = ao.responseText;
    document.getElementById("result").
innerHTML = resp;
  }
}
```

Когда все необходимые свойства объекта `ao` инициализированы и запрос подготовлен в методе `Open()`, вызывается метод `Send()`, и объект `ao` начинает следить за судьбой отправленного запроса, проверяя свое свойство `ao.readyState`. Отметьте, что параметр `null` в методе `Send()` необходим. Если его не указать, то в Mozilla Firefox метод `Send()` работать правильно не будет.

Обратите внимание на используемые проверки:

- `if(ao.readyState == 4 || ao.readyState == 0)` – если запрос еще не подготовлен или уже завершен и можно готовить следующий запрос;
- `if(ao.readyState == 4 && ao.status == 200)` – если в `ao.responseText` уже занесены полученные от сервера данные и при этом сервер прислал код успешного завершения 200.

Остался обработчик `handler.php`:

```

<?php
    $name = $_GET['name'];
    $names = array("mary", "maria", "marcello",
                  "mark", "max", "mike");
    $name = strtolower($name);

    $response="";
    foreach($names as $n)
    {
        if(substr($n,0,strlen($name)) === $name)
        {
            $response .= $n."<br/>";
        }
    }
    echo $response;
?>

```


Получаем пользовательский ввод и заносим в переменную \$name. Создаем массив с именами. Я намеренно ввел имена, начинающиеся на "m", чтобы результат был красноречивее. Чтобы избежать проблем и несовпадением регистров, перевожу полученные данные в нижний регистр, как у меня в массиве. Перебираю в цикле все имена из своего массива и те из них, начало которых совпадает с введенным пользователем значением, заносю в переменную \$response. Эту переменную возвращаю в браузер как итог работы запроса. Активируйте наш пример, введя адрес <http://localhost/ajax1/index.html>.

У меня работа запроса выглядит так:



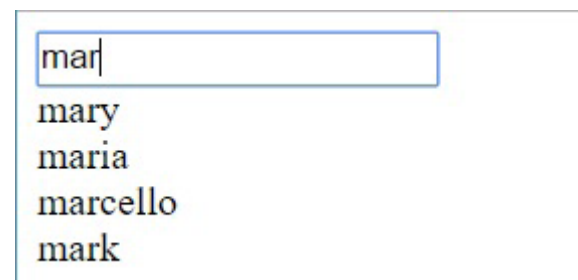
A screenshot of a web interface showing a search input field with the letter 'm' entered. Below the input field, a list of suggestions is displayed: 'mary', 'maria', 'marcello', 'mark', 'max', 'mike', and 'monica'. The input field has a blue border and a cursor at the end of the text.

Рис. 1. Введена строка "m"



A screenshot of a web interface showing a search input field with the letters 'ma' entered. Below the input field, a list of suggestions is displayed: 'mary', 'maria', 'marcello', 'mark', and 'max'. The input field has a blue border and a cursor at the end of the text.

Рис. 2. Введена строка "ma"



A screenshot of a web interface showing a search input field with the letters 'mar' entered. Below the input field, a list of suggestions is displayed: 'mary', 'maria', 'marcello', and 'mark'. The input field has a blue border and a cursor at the end of the text.

Рис. 3. Введена строка "mar"

Обратите внимание, что при работе этого примера у нас вся страница в браузере не обновляется, обновляется только div с идентификатором result. Кроме этого, вы можете продолжать ввод, пока запрос обрабатывается, поскольку браузер не блокируется.

Рассмотрим, как работать с AJAX запросами по методу POST. Для этого надо привести функцию Process() к такому виду:

```
function Process()
{
    if(ao.readyState == 4 || ao.readyState == 0)
    {
        name = document.
            getElementById("usertext").value;
        ao.open("POST", "handler.php", true);
        ao.setRequestHeader("Content-type",
            "application/x-www-form-urlencoded");
        ao.onreadystatechange = getData;
        ao.send("name="+name);
    }
}
```

Во-первых, здесь изменен первый параметр метода Open() с GET на POST.

Во-вторых, данные обработчику больше не передаются в адресной строке с помощью «?».

В-третьих, добавлен вызов метода setRequestHeader(), устанавливающего необходимый тип данных.

В-четвертых, данные в виде key=value передаются в методе Send().

Кроме этого, в handler.php данные теперь надо извлекать не из массива \$\_GET, а из массива \$\_POST: \$name = \$\_POST['name'].

Внесите изменения и проверьте работу примера. Все работает как и раньше, но сейчас данные передаются с помощью метода POST, что более предпочтительно с точки зрения объема передаваемых данных.

## Связанные списки AJAX

Сейчас мы вернемся к нашему предыдущему сайту для турагентства и изменим страницу выбора тура. Мы организуем выбор страны и города с помощью технологии AJAX. Страница будет работать таким образом: пользователь выбирает из списка страну, и рядом появляется список для выбора города в этой стране; пользователь выбирает город, и рядом появляется таблица с перечнем отелей в этом городе. Список городов и список отелей будут создаваться AJAX запросами. Для работы с БД будем использовать функции `mysql_`, поскольку наш сайт был сделан с применением этого подхода.

Разберем создание этой страницы подробно. Для начала вставим в страницу `tours.php` такой код:

```
<?php
connect();
echo '<div class="form-inline">';
echo '<select name="countryid" id="countryid"
onchange="showCities(this.value)">';
echo '<option value="0">select country</option>';
$res=mysql_query('select * from countries');
while ($row=mysql_fetch_array($res,MYSQL_NUM))
{
    echo '<option value="'. $row[0].'">'. $row[1]. '
                                     </option>';
}
echo '</select>';
```

```

//list of cities
echo '<select name="cityid" id="citylist"
      onchange="showHotels(this.value)"></select>';
echo '</div>';
//list of hotels
echo '<div id="h"></div>';
//javascript functions
?>
<script>
function showCities(countryid)
{
    if(countryid=="0"){
        document.getElementById('citylist').
            innerHTML="";
    }
    //creating AJAX object
    if(window.XMLHttpRequest){
        ao=new XMLHttpRequest();
    }
    else{
        ao=new ActiveXObject('Microsoft.
                               XMLHTTP');
    }
    //creating callback function accepting result
    ao.onreadystatechange=function(){
        if(ao.readyState==4 && ao.status==200)
        {
            document.getElementById('citylist').
                innerHTML =
                ao.responseText;
        }
    }
    //creating and sending AJAX request
    ao.open('GET',"pages/ajax1.php?cid="+countryid,
true);
    ao.send(null);
}

function showHotels(cityid)
{

```

```

var h=document.getElementById('h');
if(cityid=="0"){
    h.innerHTML="";
}
//creating AJAX object
if(window.XMLHttpRequest){
    ao=new XMLHttpRequest();
}
else{
    ao=new ActiveXObject('Microsoft.
                                XMLHTTP');
}
//creating callback function accepting result
ao.onreadystatechange=function(){
    if(ao.readyState==4 && ao.status==200){
        h.innerHTML=ao.responseText;
    }
}
//creating and sending AJAX request
ao.open("POST","pages/ajax2.php",true);
ao.setRequestHeader("Content-
    Type","application/x-www-form-urlencoded");
ao.send("cid="+cityid);
}
</script>

```

Вы видите, что список стран создается обычным способом, путем выполнения запроса 'select \* from countries'. Однако этому элементу select мы добавляем обработчик события onchange. Этот обработчик вызывает javascript функцию showCities(this.value), в которую передает выбранное в списке значение. Не забывайте, что выбранным значением в нашем списке будет идентификатор выбранной страны, а не ее название. Переходим к коду этой функции и смотрим, что она делает.

Сначала выполняется проверка полученного идентификатора. Потом выполняется создание AJAX объекта.

Когда объект создан, в его свойство `onreadystatechange` заносится анонимная `callback` функция, которая будет автоматически вызвана при получении данных от AJAX запроса. Эта функция вставит полученные от запроса данные в элемент с идентификатором `citylist`. Этот элемент и есть список городов. Он создан на странице без собственных элементов `option` именно потому, что эти `option` должны «прилетать» в него от AJAX запроса. Как вы можете видеть, у этого элемента указан собственный обработчик события `onchange` – javascript функция `showHotels(this.value)`. Эта функция подобна предыдущей функции `showCities()`. Она получает результат выполнения своего AJAX запроса и заносит его в элемент с идентификатором `'h'`. С этой страницей разобрались. Теперь осталось рассмотреть два обработчика для AJAX запросов, в коде они называются `ajax1.php` и `ajax2.php`.

Обработчик первого AJAX запроса должен подготовить список `option` из городов выбранной страны и при-слать его в элемент `citylist`. Этот обработчик выглядит так:

```
<?php
include_once('functions.php');
connect();
$cid=$_GET['cid'];
$sel='select * from cities where countryid='.$cid;
$res=mysql_query($sel);
echo '<option value="0">select city</option>';
while ($row=mysql_fetch_array($res,MYSQL_NUM)) {
    echo '<option value="'.$row[0].'">'.$row[1]. '</
                                options>';
}
mysql_free_result($res);
```



Здесь из таблицы `Cities` выбираются города для страны, идентификатор которой был прислан AJAX запросом, и из этих городов создается набор `option`.

Обработчик второго запроса `ajax2.php` выполняет похожую работу, с тем отличием, что он готовит таблицу из списка полученных отелей для выбранного города. Вот его код:

```
<?php
include_once('functions.php');
connect();
$cid=$_POST['cid'];
$sel='select id,hotel,stars,cost from hotels where
cityid='.$cid;
$res=mysql_query($sel);
echo '<table class="table table-stripped"
id="table1"><thead><tr><th>Hotel</th><th>Cost</
th><th>Stars</th>
<th>Details</th></thead><tbody>';
while ($row=mysql_fetch_array($res,MYSQL_NUM)) {
    echo '<tr><td>'.$row[1]. '</
        td><td>'.$row[3]. '$</td><td>'.$row[2];
    echo '<td><a href="pages/info.
        php?hotel='.$row[0].'" target="_blank"
        class="btn btn-default btn-xs">details</a></
        td></tr>';
}
echo '</tbody></table>';
mysql_free_result($res);
```

Переименуйте существующую страницу `tours.php`, например в `tours_old.php`. Вставьте в папку `pages` созданный файл `tours.php` и туда же добавьте два обработчика `ajax1.php` и `ajax2.php`. Проверьте, как все это работает.

# Сериализация

Очень часто приложению надо передавать свои объекты куда-нибудь или же сохранять их на диске или в памяти. Для этих целей полезно использовать сериализацию. Сериализация в PHP – это преобразование объекта какого-либо класса в строку. Выполняется сериализация с помощью функции `serialize()`, которая принимает сериализуемый объект и возвращает его специальное строковое представление. Для обратного преобразования этой строки в объект используется функция `unserialize()`. Рассмотрим на примере, как все это работает. Создайте в папке `test` два файла с именами `serialize.php` и `unserialize.php`. В `serialize.php` занесите такой код:

```
<?php
class Point
{
    private $x;
    private $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
    function Show()
    {
        echo 'Vertex: ( '.$this->x.', '.$this->y.' )
        <br/>';
    }
}
```

```

$p=new Point (100,200);
$p->Show();
$strpoint=serialize($p);
echo $strpoint;
file_put_contents('point.txt',$strpoint);
?>

```

Мы создаем объект нашего класса Point. Вызываем для созданного объекта метод Show(). Затем выполняем сериализацию этого объекта в строку \$strpoint и выводим эту строку в браузер. После этого записываем строку с сериализованным объектом в файл с именем point.txt. Активируйте файл serialize.php, и вы получите в браузере такой вывод:

```

Vertex: (100,200)
O:5:"Point":2:{s:8:"Pointx";i:100;s:8:"Pointy";i:200;}

```

Первая строка – это результат работы метода Show(). Вторая – содержимое строки \$strpoint, выведенное конструкцией echo. Кроме этого, в папке test будет создан файл point.txt с записанным сериализованным объектом. Можете заглянуть в этот файл J

Теперь мы попробуем восстановить объект из файла point.txt. Сделаем это во втором нашем файле – unserialize.php. Имейте в виду, что там, где вы планируете извлекать сериализованный объект какого-либо класса из строки, созданной функцией serialize(), должно присутствовать определение этого класса. Занесите в файл unserialize.php такой код:

```

class Point
{
    private $x;
    private $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
    function Show()
    {
        echo 'Vertex: ( '.$this->x.', '.$this->y.' )'
            .<br/>';
    }
}
$strpoint=file_get_contents('point.txt');
echo $strpoint.'<br/>';
$p=unserialize($strpoint);
$p->Show();

```

Добавляем определение класса Point, объект которого собираемся восстановить. Читаем содержимое файла с сериализованным объектом в переменную \$strpoint. Выводим прочитанную строку в браузер. Теперь отдаем эту строку функции unserialize() и забираем полученный объект. Чтобы убедиться, что мы получили из строки именно объект класса Point, пытаемся вызвать метод Show(). Все работает.

Как видите, в сериализации и десериализации все просто. Более подробно о сериализации можно посмотреть здесь:

<http://php.net/manual/en/language.oop5.serialization.php>.

# JSON

Существует еще один популярный способ упаковывать объекты для транспортировки и хранения. Это использование формата JSON. Вы должны помнить, что JSON – это представление объекта в виде ассоциативного массива, разработанное в Javascript. Однако это представление оказалось таким удачным, что вышло за пределы использования в Javascript и на сегодня стало универсальным кроссплатформенным форматом представления объектов. Почти все языки программирования (C++, C#, Java и др.) содержат инструменты для обработки JSON. Особенно популярен этот формат в вебе, где он успешно конкурирует с XML. На сегодня существует огромное количество различных онлайн сервисов, которые возвращают результаты своей работы именно в JSON. Понятно, что PHP тоже очень продуктивно использует этот формат. Рассмотрим пример.

Создайте в папке test два файла с именами encodejson.php и decodejson.php. Вставьте в encodejson.php такой код:

```
<?php
class Point
{
    public $x;
    public $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
}
```

```

function Show()
{
    echo 'Vertex: ('. $this->x. ', '. $this->y. ')
                                <br/>';
}
}
$p=new Point (100,200);
$p->Show();
$jsonpoint=json_encode($p);
echo $jsonpoint.'<br/>';
file_put_contents('jsonpoint.txt',$strpoint);
?>

```

Обратите внимание, что для преобразования в JSON свойства должны быть public. Активируйте этот файл, и вы получите в браузере такой вывод:

```

Vertex: (100,200)
{«x»:100,«y»:200}

```

В папке test будет создан файл с именем jsonpoint.txt, в котором будет записано JSON представление нашего объекта.

Теперь выполним обратное преобразование. Добавьте в файл decodejson.php такой код:

```

<?php
$jsonpoint=file_get_contents('jsonpoint.txt');
$obj=json_decode($jsonpoint);
var_dump($obj);
?>

```

Активируйте этот файл, и вы получите в браузере такой вывод:

```

string(23) «»{«x»:100,«y»:200}«»

```

В результате выполнения этого преобразования вы не получили объект класса `Point`, вы получили строковый объект со свойствами, имена и значения которых соответствуют обработанному объекту класса `Point`.

Обратите внимание на два момента, важные для правильного использования JSON:

для извлечения объекта из JSON представления вам не надо иметь в своем распоряжении определения исходного класса;

хотя Javascript может работать и с одинарными и с двойными кавычками, JSON требует использования только двойных кавычек.

Более подробно об использовании функций для обработки JSON можно прочесть здесь:

<http://php.net/manual/en/ref.json.php>.

# Magic методы

В состав PHP входит ряд методов, называемых magic методами. Их объединяет то, что их имена начинаются в двух символов подчеркивания и еще у них общий стиль поведения при вызове. Давайте познакомимся с этими методами поближе. Создайте в папке test файл magic.php и добавьте в него наш класс Point из первого примера и такой код:

```
<?php
class Point
{
    private $x;
    private $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
}
$p=new Point(100,200);
echo $p;
```

Активируйте этот файл и получите сообщение об ошибке. Вы должны помнить, что конструкция echo не умеет самостоятельно преобразовывать объекты наших классов к строковому представлению, чтобы выводить это представление в браузер. Вот в этой ситуации нам придет на помощь magic метод с именем \_\_toString(). Рассматривайте этот метод как обработчик события, которое возникает всякий раз, когда echo пытается пре-



образовать какой-либо объект в строку. Добавьте в наш класс такой метод:

```
public function __toString()  
{  
    return "(".$this->x." : ". $this->y.")";  
}
```

Активируйте файл `magic.php` снова и убедитесь, что в этот раз ошибки нет. Что происходит при наличии метода `__toString()`? Когда мы просим конструкцию `echo` вывести в браузер описание объекта какого-либо класса, возникает событие при котором активируется `magic` метод `__toString()`, если он есть в классе, конечно. Этот метод должен возвращать строку, которая и будет передаваться конструкции `echo` для вывода в браузер. Вообще говоря, в этой строке может быть все, что угодно. Но мы с вами занесли в эту строку описание нашего объекта. Вы тоже подумали о C# и Java?

Я не думаю, что вы найдете где-либо сравнение `magic` методов с обработчиками событий. Я тоже не видел такого сравнения, но мне поведение `magic` методов напоминает именно поведение обработчиков событий. Поэтому я делюсь с вами этой аналогией, предупреждая, что в официальном описании `magic` методов этого нет.

Между прочим, конструктор любого класса тоже является `magic` методом. Он активируется при наступлении события «создание нового объекта».

Рассмотрим другие `magic` методы. Добавьте в файл `magic.php` такую строку обработки:

```

<?php
class Point
{
    private $x;
    private $y;
    function __construct($x, $y)
    {
        $this->x=$x;
        $this->y=$y;
    }
    public function __toString()
    {
        return "(".$this->x." : ". $this->y.")";
    }
}
$p=new Point(100,200);
echo $p.'<br/>';
echo $this->xCoord;

```

Вы не ошиблись. Я хочу вывести в браузер значение свойства, которого в классе не существует! К тому же, свойства обычно `private`. Добавьте в наш класс еще один `magic` метод:

```

public function __get($index)
{
    if($index=="xCoord")    return $this->x;
}

```

Снова активируйте файл и убедитесь в том, что сейчас никакой ошибки нет и вы увидели в браузере значение свойства `$x`. Как это все работает? Метод `__get()` вызывается когда идет обращение к несуществующему свойству класса. Можете опять думать об этом методе, как об обработчике события «обращение к несуществу-

ющему свойству». Этот метод при вызове получает имя этого несуществующего свойства и дальше может выполнять любые действия. В нашем случае этот метод возвращает пользователю значение свойства `$x`, если пользователь попросил вывести значение свойства (несуществующего) с именем `xCoord`. Вот такой вот волшебный геттер. Там, где есть геттер, должен быть и сеттер.

Добавьте в наш класс еще один magic метод:

```
public function __set($property, $value)
{
    if ($property == "xCoord" )
    {
        $this->$x = $value;
    }
    else if ($property == "yCoord" )
    {
        $this->$y = $value;
    }
}
```

Вы уже догадались, что этот magic метод будет вызываться при попытке изменить значение несуществующего свойства. Если при этом имя несуществующего свойства будет «`xCoord`» или «`yCoord`», выполниться изменение значения свойства `$x` или `$y`.

Еще раз повторяю, что логика работы magic методов может быть самой разной. Мы могли бы написать эти методы так, чтобы они работали только с существующими свойствами. При этом один magic сеттер и один magic геттер позволяли бы работать с любым числом private свойств.

Например, так:

```
public function __get($property)
{
    if (property_exists($this, $property))
    {
        return $this->$property;
    }
}

public function __set($property, $value)
{
    if (property_exists($this, $property))
    {
        $this->$property = $value;
    }
}
```

Magic методы – это просто интересный инструмент, а как им пользоваться, вы должны решать самостоятельно в каждом конкретном случае.

Существует еще ряд magic методов. Предположим, что вы работаете с объектом какого-либо класса и вам надо узнать, есть ли у этого объекта private свойство с именем, например «width». Добавьте в наш класс еще один magic метод:

```
public function __isset($property)
{
    return isset($this->$property);
}
```

И такую строку обработки для его проверки:

```
if(isset($p->width))
    echo "Exists<br/>";
else
    echo "Not exists<br/>";
```

Рассмотрим еще пару magic методов, а об остальных вы почитаете самостоятельно.

Мы только что познакомились с сериализацией. Давайте предположим, что нам надо сериализовать объекты нашего класса Point так, чтобы свойство \$y не сериализировалось. Решить такую задачу поможет magic метод \_\_sleep(). Этот метод возвращает массив, в который включает имена свойств, которые надо сериализировать.

```
public function __sleep()  
{  
    return array('x');  
}
```

В нашем классе всего два свойства, поэтому массив будет совсем коротким. Этот метод приведет к тому, что при сериализации объекта будет сохранено только свойство \$x. Это можно проверить такой обработкой:

```
$str=serialize($p);  
echo $str;
```

В результирующей строке \$str свойства \$y не будет.

При попытке вызвать недоступный или несуществующий метод будет активироваться magic метод \_\_call(). Этому методу при вызове передаются два параметра: имя вызываемого метода и массив параметров, для вызываемого метода. Добавьте в наш класс еще один magic метод:

```
public function __call($name, $arguments)
{
    echo "Trying to call method '$name'
with parameters "
    . implode(', ', $arguments). "\n";
}
```

И добавьте такую строку вызова:

```
$p->Ufo(1,2,3);
```

Вы получите в браузере такой вывод:

Trying to call method 'Ufo' with parameters 1, 2, 3

Более подробно о magic методах можно прочесть  
здесь:

<http://php.net/manual/en/language.oop5.magic.php>

# PDO (PHP Data Object)

После нашего знакомства с трейтами мы работаем с версией PHP 5.4. В этой версии, помимо добавления трейтов, произошло еще одно существенное изменение – были объявлены устаревшими функции `mysql_`. Поэтому нам надо познакомиться с другими способами работы с БД, которые предлагает PHP.

Сейчас мы рассмотрим работу с объектом PDO, который предоставляет замечательный функционал для работы с БД. В отличие от функций `mysql_`, позволяющих работать только с MySQL, PDO поддерживает работу со многими СУБД, для которых существуют драйвера. Узнать, какие драйвера доступны, можно, выполнив такой вызов:

```
print_r(PDO::getAvailableDrivers());
```

Перейдем к знакомству с объектом PDO. У нас есть БД, созданная для сайта турагентства. Будем использовать ее в нашем примере. Начнем с подключения к СУБД. У PDO существует знакомое вам понятие «строка подключения». Это строковая конструкция, в которой, среди других атрибутов, указывается тип СУБД, к которой мы хотим подключиться. Строка подключения указывается первым параметром конструктору PDO. Создайте в папке `test` файл с именем `pdo1.php`. В этом файле мы создадим функцию для подключения к нашей БД и рассмотрим основные принципы выполнения запросов с использованием PDO.

Как и в предыдущем случае, создадим для подключения к СУБД функцию с именем `connect()` с параметрами, имеющими значения по умолчанию. Назначение этой функции – создать и вернуть объект PDO для работы с нашей БД. Именно от имени этого объекта мы будем вызывать необходимые методы для выполнения запросов к БД. Объект PDO создается вызовом конструктора. Мы передадим в этот конструктор строку подключения, имя авторизованного пользователя, пароль этого пользователя и массив с некоторыми настройками. Создавать объект будем в `try-catch` блоке и, в случае исключительной ситуации, будем возвращать `false`. Затем в качестве демонстрации выполним какой-нибудь запрос к нашей БД и выведем его результаты в браузер.

Вставьте в файл `pdo1.php` такой код. Обратите внимание, что СУБД размещается с номером порта 3307.

```
<?php
function connect(
    $host="localhost:3307",
    $user="root",
    $pass="123456",
    $dbname="travels")
{
    $cs='mysql:host='.$host.';dbname='.$dbname.';charset=utf8;';
    $options=array(
        PDO::ATTR_ERRMODE=>PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE=>PDO::FETCH_ASSOC,
        PDO::MYSQL_ATTR_INIT_COMMAND=>'SET NAMES UTF8'
    );
}
```



```

        try
        {
            $pdo=new
            PDO($cs,$user,$pass,$options);
            return $pdo;
        }
        catch(PDOException $e)
        {
            echo $e->getMessage();
            return false;
        }
    }

    $pdo=connect();
    $list=$pdo->query('select * from countries');
    while ($row=$list->fetch())
    {
        echo $row['id'].'
                '.$row['country'].'<br/>';
    }
    ?>

```

Активируйте этот файл, и вы получите в браузере такой вывод – результат выполнения запроса SELECT к таблице Countries:

7	Argentina
8	Belgium
9	Canada
10	United Kingdom

Обсудим приведенный код. При возникновении ошибок PDO выбрасывает исключения, поэтому мы создаем объект в try-catch блоке. Что означают значения элементов в массиве \$options, который мы передаем в конструктор?

- PDO::ATTR\_ERRMODE=>PDO::ERRMODE\_

EXCEPTION – при возникновении какой-либо ошибки сразу выбрасывать исключение;

- PDO::ATTR\_DEFAULT\_FETCH\_MODE=>PDO::FETCH\_ASSOC – по умолчанию возвращать данные из ресурсов в виде ассоциативного массива;
- PDO::MYSQL\_ATTR\_INIT\_COMMAND=>'SET NAMES UTF8' – активировать использование в БД кодировки UTF8.

Обратите внимание, что наша функция connect() создает и возвращает объект PDO, который мы сохраняем в переменной \$pdo и используем в дальнейшем. Для выполнения запроса 'select \* from countries' мы применяем метод query() объекта PDO и заносим полученный ресурс в переменную \$list. Затем, для извлечения строк из этого ресурса, мы используем в цикле метод fetch(). В этом же цикле выводим полученные данные.

## Prepared Statements

Использование метода query() объекта PDO не является лучшим способом выполнения запросов. Намного более эффективным способом работы с объектом PDO является использование подготовленных запросов (Prepared Statements). Prepared Statements – это параметризованные, многократно используемые, компилируемые запросы. Эти запросы выполняются быстрее классических и к тому же обеспечивают встроенную защиту от SQL-инъекций. Параметризация подготовленных запросов дает еще одно преимущество – избавляет программиста от рутинной работы по вставке переменных в тело запроса.

Рассмотрим примеры выполнения подготовленных

запросов с разными способами параметризации. Работа с такими запросами состоит из нескольких этапов. Сначала создается сам запрос. Затем, если необходимо, для подготовленного запроса создаются параметры. Затем запрос выполняется.

Создайте в папке test файл с именем pdo2.php и добавьте в него такой код:

```
<?php
include_once('pdo1.php');
$pdo=connect();
$name1="Brasil";
$name2="USA";
$name3="Ukraine";
//first way of using prepared statements - no
protection from SQL-injection
//such usage should be avoided
$ps1 = $pdo->prepare("INSERT INTO Countries
(country) values
( '".$name1."' )");
//executing the first prepared statements
$ps1->execute();
//second way of using prepared statements with
named placeholders
//this statement provides protection from SQL-
injection
$ps2 = $pdo->prepare("INSERT INTO Countries
(country) values
(?)");
//inserting parameter for the second prepared
statements
$ps2->bindParam(1, $name2);
//executing the second prepared statements
$ps2->execute();
```

```

    //third way of using prepared statements with named
    placeholders
    //this statement provides protection from SQL-
    injection
    $ps3 = $pdo->prepare("INSERT INTO Countries
    (country) values
        ( :country )");
    //inserting parameter and executing the third
    prepared statements
    $ps3->execute(array("country" => $cname3));

    //SELECT in prepared statements
    $ps4 = $pdo->prepare("SELECT * FROM Countries");
    $list=$ps4->execute();
    $ps4 ->setFetchMode(PDO::FETCH_NUM);
    while ($row=$list->fetch())
    {
        echo $row[0].' ' . $row[1]. '<br/>';
    }
    ?>

```

Обсудим приведенный код. Значения для параметров подготовленного запроса можно передавать сразу в теле запроса в переменной, как это сделано в запросе \$ps1. Однако при таком применении никакой защиты от SQL-инъекций не выполняется. И если в переменной \$cname1 будет какой-либо вредоносный скрипт – он попадет в таблицу БД.

Намного более предпочтительно использовать в теле подготовленного запроса специальные плейсхолдеры для значений. Такие плейсхолдеры могут представлять собой просто символы «?» для каждого требуемого параметра, как в нашем запросе \$ps2. Значения для таких параметров, называемых неименованными, можно передавать двумя способами.

### 1. С помощью метода `bindParam()`:

```
$ps2->bindParam(1, $cname2);
```

где первый параметр – индекс параметра (начинается с 1!), а второй – значение.

### 2. С помощью массива со значениями, передаваемого методу `execute()`:

```
$ps2->execute(array($cname2));
```

Плейсхолдеры могут быть также именованными, как в нашем запросе `$ps3`. Значения для таких плейсхолдеров можно передавать также двумя способами.

### 1. С помощью метода `bindParam()`:

```
$ps2->bindParam(":countryname", $cname3);
```

где первый параметр – имя плейсхолдера, а второй – значение.

### 2. С помощью ассоциативного массива со значениями, передаваемого методу `execute()`, имена ключей в таком массиве должны совпадать с именами плейсхолдеров:

```
$ps2->execute(array("countryname" => $cname3));
```

Во всех рассмотренных случаях количество плейсхолдеров для параметров может быть произвольным. Немного позже рассмотрим еще одно преимущество Prepared Statements при работе с объектами.

Prepared Statements также можно использовать для выполнения запросов, возвращающих ресурс. Извлекать данные из ресурсов можно методом `fetch()`. Данные будут извлекаться в виде, который указан при соз-

дании объекта \$pdo. В нашем случае в массиве \$options мы указали, что хотим извлекать данные в виде ассоциативного массива. Однако перед вызовом метода fetch() такую установку можно изменить, вызвав метод setFetchMode():

```
$ps4 = $pdo->prepare("SELECT * FROM Countries");
$list=$ps4->execute();
$ps4 ->setFetchMode(PDO::FETCH_NUM);
```

После такого вызова метод fetch() будет возвращать индексный массив, не ассоциативный.

Еще есть режим вывода PDO::FETCH\_OBJ, при котором метод fetch() будет возвращать объект.

```
$ps4 ->setFetchMode(PDO::FETCH_OBJ);
while ($row=$list->fetch())
{
    echo $row->id.' ' . $row->countryname.'<br/>';
}
```

Необходимо отметить еще ряд полезных возможностей PDO. Сталкивались ли вы при работе с БД с такой ситуацией, когда вам надо узнать индекс последней добавленной в БД записи? У объекта PDO на этот случай есть замечательный метод lastInsertId():

```
$ps2 = $pdo->prepare
("INSERT INTO Countries (countryname) values (?)");
$ps2->execute(array($cname2));
$id=$pdo->lastInsertId();
```

Существует упрощенный метод exes(), предназначенный для выполнения запросов, не возвращающих

ничего, кроме количества обработанных строк (INSERT, DELETE, UPDATE):

```
$pdo->exec('DELETE FROM Images WHERE hotelid=1');
```

Если вы по какой-то причине не пользуетесь подготовленными запросами, то можете выполнять защиту от SQL-инъекций, обрабатывая переменные, передаваемые в запрос, методом `quote()`:

```
$cname1= $pdo->quote($cname1);
$ps1 = $pdo->prepare("INSERT INTO Countries (country)
values
    ( '". $cname1. "' )");
```

Если вам надо узнать, сколько строк вернул запрос, можете использовать метод `rowCount()`:

```
$ps4 = $pdo->prepare("SELECT * FROM Countries");
$list=$ps4->execute();
$rows = $ps4->rowCount();
```

В результате выполнения скрипта в файле `pdo2.php` вы увидите в браузере обновленный список стран:

7	Argentina
8	Belgium
11	Brasil
9	Canada
13	Ukraine
10	United Kingdom
12	USA

# Разработка сайта

---

Приведенных сведений об ООП достаточно для того, чтобы приступить к разработке следующего сайта. В этот раз мы создадим интернет-магазин, использующий БД и построенный по принципам ООП.

Наш сайт будет состоять из таких страниц:

- **Catalog** – на этой странице пользователь будет видеть отображение товаров, здесь же будет присутствовать механизм фильтрации, позволяющий пользователю указать критерии выбора товаров. На этой же странице он сможет отбирать товары в свою корзину.
- **Cart** – корзина пользователя, в которой он сможет редактировать список отобранных товаров и оформлять заказ на покупку.
- **Registration** – страница регистрации.
- **AdminForms** – пункт меню администратора сайта.
- **Reports** – отчеты о работе интернет-магазина.

Данные нашего сайта будут снова храниться в базе данных СУБД MySQL. Представление данных в приложении будет организовано в виде классов. Для каждого класса данных в приложении будет своя таблица в БД, таким образом каждый объект класса данных будет представлен строкой в соответствующей таблице. Инструменты для работы с СУБД мы тоже оформим в виде класса, в котором применим статические методы.

Создайте для нового сайта папку с именем Site3 и в ней создайте вложенные папки и файлы по образцу на-



ших предыдущих сайтов. Во вложенной папке pages создайте файлы menu.php, catalog.php, cart.php, registration.php и admin.php. К этим файлам добавьте файл classes.php – в нем будут храниться наши классы. В файле index.php создайте подключение и активацию пунктов меню, как мы делали это ранее.

## Создание базы данных

Опишем таблицы, которые будут входить в состав БД нашего сайта.

- Таблицы Roles и Customers для обработки зарегистрированных пользователей. У нас снова будет две роли: Admin и Customer.
- Таблицы Categories и Items будут содержать информацию о группах товаров и самих товарах. У нас будет классификатор товаров, условно называемый категориями.
- Таблица Sales будет хранить информацию о продажах.
- Традиционная таблица Images позволит хранить произвольное количество изображений каждого товара.

Чтобы создать все эти таблицы, мы снова напишем скрипт createdb.php. А вот подключение к СУБД мы оформим не так, как в предыдущем сайте. Раньше у нас был файл functions.php, в котором мы собирали необходимые нам функции. Сейчас мы создадим специальный класс Tools, в котором будем хранить необходимый для сайта функционал, в частности – метод connect() для подключения к СУБД. Откройте файл classes.php и добавьте в него такой код:

```

<?php
class Tools
{
    static function connect(
        $host="localhost:3307",
        $user="root",
        $pass="123456",
        $dbname="shop")
    {
        $cs='mysql:host='.$host.';dbname='.$dbname.';charset=utf8;';

        $options=array(
            PDO::ATTR_ERRMODE=>PDO::ERRMODE_
                EXCEPTION,
            PDO::ATTR_DEFAULT_FETCH_MODE=>PDO::FETCH_
                ASSOC,
            PDO::MYSQL_ATTR_INIT_COMMAND=>'SET NAMES
                UTF8'
        );
        try {
            $pdo=new
                PDO($cs,$user,$pass,$options);
            return $pdo;
        }
        catch(PDOException $e)
        {
            echo $e->getMessage();
            return false;
        }
    }
}

```

В этом файле мы разместили класс с именем Tools, в котором будем хранить служебные инструменты для работы с БД. Сейчас в этом классе описан один метод с именем connect(). Вы, конечно же, узнали нашу функцию для подключения к БД. Обратите внимание на спецификатор static, указанный перед этим методом. Да, PHP

поддерживает в классах статические методы и свойства. Вызывать этот метод мы будем от имени класса, таким образом: `Tools::connect()`. Имея средство подключения к БД, мы можем создать файл со скриптом, в котором будут создаваться необходимые нам таблицы. Создайте в папке `pages` еще один файл с уже привычным именем `createdb.php`. Как и раньше, запустите `PhpMyAdmin` и создайте базу данных с именем `shop`. Теперь откройте в блокноте файл `createdb.php` и вставьте в него такой код:

```
<?
include_once('classes.php');
$pdo=Tools::connect();

$role='create table Roles(id int not null auto_
                                increment primary key,
                                role varchar(32)not null unique)default
                                charset="utf8"';

$customer='create table Customers(id int not null
                                auto_increment primary key,
                                login varchar(32)not null unique,
                                pass varchar(128)not null,
                                roleid int,
                                foreign key(roleid) references Roles(id) on
                                                update cascade,
                                discount int,
                                total int,
                                imagepath varchar(255))default charset="utf8"';

$cat='create table Categories(id int not null auto_
                                increment primary key,
                                category varchar(64)not null unique)
                                default charset="utf8"';

$sub='create table SubCategories(id int not null
                                auto_increment primary key,
                                sucategory varchar(64)not null unique,
```

```

        catid int,
        foreign key(catid) references Categories(id) on
                                update cascade)
        default charset="utf8";

$item='create table Items(id int not null auto_
                                increment primary key,
        itemname varchar(128)not null,
        catid int,
        foreign key(catid) references Categories(id) on
                                update cascade,
        pricein int not null,
        pricesale int not null,
        info varchar(256) not null,
        rate double,
        imagepath varchar(256) not null,
        action int)
        default charset="utf8";

$images='create table Images(id int not null auto_
                                increment primary key,
        itemid int,foreign key(itemid) references
Items(id) on delete cascade,
        imagepath varchar(255))default charset="utf8";

$sale='create table Sales(id int not null auto_
                                increment primary key,
        customername varchar(32),
        itemname varchar(128),
        pricein int,
        pricesale int,
        datesale date)default charset="utf8";

$pdo->exec($role);
$pdo->exec($customer);
$pdo->exec($cat);
$pdo->exec($sub);
$pdo->exec($item);
$pdo->exec($images);
$pdo->exec($sale);

```

Активируйте этот файл, и вы получите в БД shop указанные таблицы. В этом скрипте вам все уже знакомо. Обратите внимание на вызов статического метода `connect()` и на способ выполнения запросов `CREATE TABLE` с помощью метода `exec()` объекта PDO.

## Класс Customer

Мы уже говорили, что сущности, с которыми нам надо работать на нашем сайте, такие как пользователи, товары, заказы и прочие, будут представлять собой объекты соответствующих классов. Это обычная практика – представлять обрабатываемые данные в виде объектов классов, а хранить их в таблицах БД. Для каждого класса данных в приложении должна быть соответствующая таблица в БД. Крайне важно сделать так, чтобы имена полей в таблицах совпадали с названиями свойств в классах. При этом запись объектов в таблицы и считывание строк таблицы в объекты будет выполняться намного эффективнее.

Вы уже знакомы с термином ORM (объектно-реляционное отображение) и знаете, что существуют специальные инструменты, реализующие соответствие «таблица – класс», например Entity Framework или Hibernate. Мы с вами создадим подобное соответствие для нашего сайта самостоятельно.

Рассмотрим подробно процесс создания такого соответствия для таблицы Customers и класса Customer. Таблица у нас уже есть, поэтому перейдем к созданию класса. Его надо создать таким образом, чтобы в нем были свойства, соответствующие таблице с такими же именами и типами, как поля в таблице.

Добавляем в файл `classes.php` такой код:

```
class Customer
{
    protected $id;           //user id
    protected $login;
    protected $pass;
    protected $roleid;
    protected $discount; //customer's personal discount
    protected $total;    //total ammount of purchases
    protected $imagepath; //path to the image
}
```

Поговорим о некоторых свойствах. Мы добавим в этот класс конструктор и будем создавать объекты с его помощью. Возникает вопрос, как надо поступать со свойством `$id` при создании объекта в конструкторе? Надо ли это свойство инициализировать, и если надо, то чем? Дело в том, что при создании объекта у него еще нет никакого идентификатора. Эта характеристика у объекта появится только тогда, когда мы добавим объект в таблицу, и СУБД присвоит ему идентификатор. Это приводит к тому, что когда мы создаем объект в первый раз, мы не должны указывать значение для идентификатора, а когда мы будем считывать объект из таблицы – необходимо будет заносить в свойство `$id` значение, прочитанное из БД.

Конечно же, эту проблему можно решить разными способами. Мы поступим так. Создадим у конструктора параметр для свойства `$id` со значением по умолчанию, равным нулю. А при создании объекта на основании прочитанной из БД информации будем передавать в этот параметр значение прочитанного идентификатора.

Добавим в наш класс такой конструктор:

```
class Customer
{
    protected $id;           //user id
    protected $login;
    protected $pass;
    protected $roleid;
    protected $discount; //customer's personal discount
    protected $total;     //total ammount of purchases
    protected $imagepath; //path to the image
    function __construct($login,$pass,$imagepath,$id=0)
    {
        $this->login=$login;
        $this->pass =$pass;
        $this->imagepath =$imagepath;
        $this->id =$id;
        $this->total =0;
        $this->discount =0;
        $this->roleid =2;
    }
}
```

Этот код понятен без объяснений, единственное, на что надо обратить внимание – это то, что в таблице Roles уже должна существовать роль Customer с идентификатором «2».

Теперь переходим к реализации для этого класса простого ORM отображения. Создадим два метода. Один будет называться intoDb() и будет выполнять запись текущего объекта в таблицу БД. Второй метод назовем fromDb(), он будет выполнять обратную задачу – читать строку из таблицы по заданному идентификатору и возвращать объект класса Customer, соответству-

ющий прочитанной строке. Метод `fromDb()` удобно сделать статическим. В этих методах вы увидите примеры очень полезных возможностей PDO. Сначала рассмотрим метод `intoDb()`.

```
function intoDb()
{
    try{
        $pdo=Tools::connect();
        $ps=$pdo->prepare("INSERT INTO
                                Customers
                                (login,pass,roleid,discount,total,i
                                magepath)
                                VALUES (:login,:pass,:roleid,:
                                discount,:total,:imagepath)");
        $ar=(array)$this;
        array_shift($ar);
        $ps->execute($ar);
    }
    catch(PDOException $e)
    {
        $err=$e->getMessage();
        if(substr($err,0,strlen($err,":"))==
        'SQLSTATE[23000]:
        Integrity constraint violation')
            return 1062;
        else
            return $e->getMessage();
    }
}
```

Для добавления объекта в БД мы используем подготовленный запрос с именованными параметрами. Сразу бросается в глаза, что имена параметров совпадают



с именами свойств класса. Обратите внимание на строку `$ar=(array)$this`. В ней происходит преобразование объекта класса в массив. Мы планируем передать этот массив методу `execute()` для добавления в БД. Однако в запросе `INSERT` мы не должны передавать значение для поля таблицы `id`, ведь оно определяется сервером. Чтобы убрать из массива первый элемент, содержащий значение свойства `$id`, мы используем функцию `array_shift()`. И уже такой преобразованный массив `$ar` передаем для вставки в таблицу БД. Согласитесь, преобразование текущего объекта в массив и передача его в метод `execute()` здорово сокращают код записи объекта в таблицу. Но за это надо платить – свойства класса `Customer` сейчас сделаны `public`, чтобы PDO имел к ним доступ.

Обратите внимание, что в `catch` блоке мы особым образом обрабатываем ошибку, возникающую при нарушении уникальности логина пользователя. В этом случае PDO возвращает строку `'SQLSTATE[23000]: Integrity constraint violation'`, и мы проверяем наличие этой строки. Если получаем именно эту ошибку, то метод `register()` возвращает значение 1062. В случае возникновения любых других ошибок метод `register()` возвращает их описание. Значение 1062 не имеет какого-либо особого смысла, оно выбрано из тех сообщений, что код ошибки MySQL при нарушении уникальности поля именно 1062. Поскольку метод `intoDb()` будет вызываться в методе `register()`, то именно в методе `register()` мы будем получать и обрабатывать эти возвращаемые значения.

Теперь рассмотрим метод `fromDb()`.

```

static function fromDb($id)
{
    $customer=null;
    try{
        $pdo=Tools::connect();
        $ps=$pdo->prepare(("SELECT * FROM
                                Customers
                                WHERE id=?"));
        $res=$ps ->execute(array($id));
        $row=$res->fetch();
        $customer=new
        Customer($row['login'],
        $row['pass'], $row['imagepath'],
        $row['id']);
        return $customer;
    }
    catch(PDOException $e)
    {
        echo $e->getMessage();
        return false;
    }
}

```

Поскольку этот метод должен возвращать нам объект класса Customer с указанным значением id, было бы нелогичным делать этот метод объектным. Ведь в таком случае нам надо было бы создать некий объект класса Customer, чтобы с его помощью вызвать метод fromDb(), который должен вернуть нам объект класса Customer. Поэтому мы сделали этот метод статическим (static). Создаем подготовленный запрос SELECT с неименованным плейсхолдером для параметра. Этим запросом читаем строку из таблицы по указанному id. Результат запроса извлекаем из ресурса без цикла, поскольку бо-

лее одной записи возвращено быть не может. Создаем конструктором объект класса Customer из прочитанных значений и возвращаем этот объект.

Вот это и есть наше объектно-реляционное отображение для класса Customer. Такие же методы fromDb() и intoDb() надо будет создать для всех наших классов. Теперь продемонстрируем использование созданного функционала на примере обработки регистрации пользователей.

## Форма регистрации

Откройте в блокноте страницу register.php и вставьте туда форму регистрации и ее обработчик. Можно использовать страницу из нашего предыдущего сайта.

```
<h3>Registration Form</h3>
<?php
if(!isset($_POST['regbtn']))
{
?>
<form action="index.php?page=3" method="post"
    enctype="multipart/form-data" >
    <div class="form-group">
        <label for="login">Login:</label>
        <input type="text" class="form-control"
            name="login">
    </div>
    <div class="form-group">
        <label for="pass1">Password:</label>
        <input type="password" class="form-control"
            name="pass1">
    </div>
    <div class="form-group">
        <label for="pass2">Confirm Password:</label>
        <input type="password" class="form-control"
            name="pass2">
```

```

        </div>
        <div class="form-group">
        <label for="imagepath">Select image:</label>
        <input type="file" class="form-control"
                                name="imagepath">
        </div>
        <button type="submit" class="btn btn-primary"
                name="regbtn">Register</button>
    </form>
    <?php
    }
    else
    {
        //upload processing
        if(is_uploaded_file($_FILES['imagepath']['tmp_
                                name']))
        {
            $path="images/".$_FILES['imagepath']
                                ['name'];
            move_uploaded_file($_FILES['imagepath']
                                ['tmp_name'], $path);
        }
        //customer registration
        if(Tools::register($_POST['login'],
                                $_OST['pass1'],$path))
        {
            echo "<h3/><span style='color:green;'>
                    New User Added!</span><h3/>";
        }
    }
    ?>

```

Обратите внимание, как вызывается функция `register()` – как статический метод класса `Tools`. Добавьте в файл `classes.php` в класс `Tools` после метода `connect()` такой статический метод:

```

static function register($name,$pass,$imagepath)
{
    $name=trim($name);
    $pass=trim($pass);
    $imagepath =trim($imagepath);
    if ($name==" " || $pass==" " )
    {
        echo "<h3/><span style='color:red;*>
        Fill All Required Fields!</span><h3/>";
        return false;
    }
    if (strlen($name)<3 || strlen($name)>30 ||
    strlen($pass)<3 ||
    strlen($pass)>30)
    {
        echo "<h3/><span style='color:red;*>
        Values Length Must Be Between 3 And 30!</
        span><h3/>";

        return false;
    }

    Tools::connect();
    $customer=new Customer($name,$pass,
        $imagepath);
    $err=$customer->intoDb();
    if ($err)
    {
        if($err==1062)
            echo "<h3/><span
            style='color:red;*>
            This Login Is Already Taken!</
            span><h3/>";
        else
            echo "<h3/><span
            style='color:red;*>
            Error code:". $err."!</span><h3/>";
        return false;
    }
    return true;
}

```

Теперь механизм регистрации пользователей нашего сайта готов. Проверьте его в действии. Обратите внимание, чтобы выполнялся `upload`, т.е. чтобы в папку `images` нашего сайта выгружались картинки, выбранные при заполнении формы регистрации, а в поле `imagepath` таблицы `Customer` заносился путь к выбранной картинке.

## Класс Item

Простой по своей структуре класс `Customer` решает относительно простые задачи на нашем сайте. Поэтому он не может в полной мере продемонстрировать преимущества использования ООП. Давайте сейчас рассмотрим создание и использование нашего главного класса – `Item`. Этот класс описывает товар.

Какой функционал должен предоставлять этот класс? Конечно же, возможность записать объект в таблицу `Items` и получить объект из строки этой таблицы. Это будут выполнять методы `intoDb()` и `fromDb()`, аналогичные методам класса `Customer`. Кроме этого класс `Item` должен предоставлять возможность:

- красиво выводить описание товара на страницу пользователя;
- выполнять добавление товара в корзину;
- выполнять продажу товара;
- применять к товару скидки и другие изменения.

Все эти действия мы реализуем в виде методов класса `Item`. Работу с классом `Item` выполним вместе, подробно обсуждая все, что будем делать. Затем вам надо будет самостоятельно выполнить аналогичную работу для других классов. Все наши классы мы будем созда-

вать в файле `classes.php`, поэтому добавляем туда описание класса `Item`.

```
class Item
{
    public $id, $itemname, $catid, $pricein,
           $pricesale, $info, $rate,
           $imagepath, $action;

    function __construct($itemname, $catid,
                        $pricein, $pricesale, $info,
                        $imagepath, $rate=0, $action=0, $id=0)
    {
        $this->id=$id;
        $this->itemname=$itemname;
        $this->catid=$catid;
        $this->pricein=$pricein;
        $this->pricesale=$pricesale;
        $this->info=$info;
        $this->rate=$rate;
        $this->imagepath=$imagepath;
        $this->action=$action;
    }

    function intoDb()
    {
        try{
            $pdo=Tools::connect();
            $ps=$pdo->prepare("INSERT INTO
                                Items
                                (itemname, catid, pricein,
                                 pricesale, info, rate,
                                 imagepath, action)
                                VALUES (:itemname, :catid,
                                         :pricein, :pricesale, :info,
                                         :rate, :imagepath,
                                         :action)");
            $ar=(array)$this;
```

```

        array_shift($ar);
        $ps->execute($ar);
    }
    catch(PDOException $e)
    {
        return $e->getMessage();
    }
}

static function fromDb($id)
{
    $customer=null;
    try{
        $pdo=Tools::connect();
        $ps=$pdo->prepare(("SELECT * FROM
                                Items WHERE
                                id=?"));
        $res=$ps->execute(array($id));
        $row=$res->fetch();
        $customer=new
        Item($row['itemname'], $row['catid'],
            $row['pricein'],
            $row['pricesale'], $row['info'],
            $row['imagepath'],
            $row['rate'],
            $row['action'],$row['id']);
        return $customer;
    }
    catch(PDOException $e)
    {
        echo $e->getMessage();
        return false;
    }
}
}

```



На данный момент в классе описаны только конструктор и методы `intoDb()` и `fromDb()`. Однако уже сейчас мы можем создать механизм добавления новых товаров в БД. А когда в нашей таблице `Items` появятся товары, мы продолжим добавление функционала к классу `Item`.

## Форма добавления товара

Форма добавления товаров будет располагаться на странице `admin.php`:

```
<?php
if(!isset($_POST['addbtn']))
{
?>
<form action="index.php?page=4" method="post"
enctype="multipart/form-data" >
<label for="catid">Category:</label>
        <select class="" name="catid">
<?php
    $pdo=Tools::connect();
    $list=$pdo->query("SELECT * FROM Categories");
    while ($row=$list->fetch())
    {
        echo '<option value="'. $row['id']. '">'. $
            row['category'].
            '</option>';
    }
?>
</select>
<div class="form-group">
    <label for="name">Name:</label>
    <input type="text" class="" name="name">
</div>
<div class="form-group">
    <label for="pricein">Incoming Price and Sale
                                Price:</label>
```

```

        <div>
            <input type="number" class=""
                name="pricein">
            <input type="number" class=""
                name="pricesale">
        </div>
    </div>

    <div class="form-group">
        <label for="info">Description:</label>
        <div><textarea class="" name="info"></
            textarea></div>
    </div>
    <div class="form-group">
        <label for="imagepath">Select image:</label>
        <input type="file" class="" name="imagepath">
    </div>

        <button type="submit" class="btn
            btn-primary"
            name="addbtn">Register</button>
    </form>
<?php
}
else
{
    if(is_uploaded_file($_FILES['imagepath']['tmp_
        name']))
    {
        $path="images/".$_FILES['imagepath']
            ['name'];
        move_uploaded_file($_FILES['imagepath']
            ['tmp_name'], $path);
    }
    $catid=$_POST['catid'];
    $pricein=$_POST['pricein'];
    $pricesale=$_POST['pricesale'];
    $name=trim(htmlspecialchars($_POST['name']));
    $info=trim(htmlspecialchars($_
        POST['info']));

```

```
$item=new Item($name,$catid,$pricein,$pricesale  
                , $info, $path) ;  
$item->intoDb() ;  
}  
?>
```

Здесь вы видите уже привычный для вас механизм использования формы и ее обработчика в одном файле. В этой форме используется загрузка изображения для главной картинки товара, затем на основе введенных в форму данных создается объект класса `Item`, от имени которого вызывается метод `intoDb()`.

Используя созданную форму, занесите в БД несколько товаров, чтобы мы могли перейти к разработке страницы `Catalog`, отображающей список товаров и позволяющей выбирать товары в корзину.

## Страница выбора товара

Эта страница предоставляет пользователю возможность просматривать товары и отбирать понравившиеся в корзину. При первом обращении к странице она отображает все товары. Затем у пользователя есть возможность выбрать в выпадающем списке интересующую его категорию товара, и на странице будут отображены только товары указанной категории. Такой выбор товаров по категориям будет выполняться с помощью `AJAX`.

Каждый товар будет отображаться в отдельном контейнере и предоставлять пользователю основную информацию о товаре и главную картинку товара. Имя товара в этом контейнере будет кликабельным, и при кли-

ке будет отображать в соседней вкладке полное описание товара и галерею из всех его изображений. Кроме того, в указанном контейнере будет кнопка, позволяющая отобрать товар в корзину. Такой контейнер будет создаваться с помощью метода `Draw()`, который мы добавим в класс `Item`.

Корзина будет создана в куки-файлах. Она будет позволять пользователю редактировать список отобранных товаров и выполнять покупку.

Создание этой страницы продемонстрирует эффективность ООП подхода при создании приложения. Благодаря созданным классам и их функционалу код страницы будет небольшим по объему, но при этом будет выполнять много сложной работы. Основной алгоритм состоит из трех пунктов.

1. Получаем список всех товаров из таблицы `Items`.
2. В цикле для каждого элемента списка вызываем метод `Draw()`.
3. Если в выпадающем списке выбрана категория товара, получаем список товаров только для выбранной категории и переходим к пункту 2.

Чтобы реализовать этот сценарий, нам надо добавить в класс `Item` статический метод, возвращающий список товаров из таблицы `Items`. Назовем этот метод `GetItems($catid=0)`. Добавим ему параметр со значением по умолчанию. Если значение этого параметра будет равно 0, метод будет возвращать все товары, независимо от категории, если же этот параметр будет содержать идентификатор категории, то метод будет возвращать только товары указанной категории. Кроме этого метода

добавим в класс `Item` упоминавшийся метод `Draw()`, который будет отображать товар в красивом контейнере.

Добавим в класс `Item` эти два метода:

```
static function GetItems($catid=0)
{
    $ps=null;
    $items=null;
    try{
        $pdo=Tools::connect();
        if($catid == 0)
        {
            $ps=$pdo->prepare('select * from
                                items');

            $ps->execute();
        }
        else
        {
            $ps=$pdo->prepare
('select * from items where categoryid=?');
            $ps->execute(array($catid));
        }
        while ($row=$ps->fetch())
        {
            $item=new Item($row['itemname'],
                           $row['catid'],
                           $row['pricein'],
                           $row['pricesale'], $row['info'],
                           $row['imagepath'], $row['rate'],
                           $row['action'],$row['id']);
            $items[]=$item;
        }
        return $items;
    }
    catch(PDOException $e)
    {
        echo $e->getMessage();
        return false;
    }
}
```

```

function Draw()
{
    echo "<div class='col-sm-3 col-md-3 col-lg-3
        container'
        style='height:350px;margin:2px;'>";
    //itemInfo.php contains detailed info about product
    echo "<div class='row' style='margin-top:2px;
        background-color:#ffd2aa;'>";
    echo "<a href='pages/itemInfo.php?name=".$this->id."'
        class='pull-left' style='margin-left:10px;'
        target='_blank'>";
    echo $this->itemname;
    echo "</a>";
    echo "<span class='pull-right' style='margin-
        right:10px;'>";

    echo $this->rate."&nbsp;rate";
    echo "</span>";
    echo "</div>";
    echo "<div style='height:100px;margin-top:2px;'
        class='row'>";
    echo "<img src='".$this->imagepath."'
        height='100px' />";
    echo "<span class='pull-right' style='margin-
        left:10px;color:red;
        font-size:16pt;'>";
    echo "$&nbsp;".$this->pricesale;
    echo "</span>";
    echo "</div>";
    echo "<div class='row' style='margin-
        top:10px;'>";
    echo "<p class='text-left col-xs-12'
        style='background-color:lightblue;
        overflow:auto;height:60px;'>";
    echo $this->info;
    echo "</p>";
    echo "</div>";
    echo "<div class='row' style='margin-
        top:2px;'>";
    echo "</div>";
}

```

```

echo "<div class='row' style='margin-
      top:2px;'>";
//creating cookies for the cart
//will be explained later
$ruser='';
if(!isset($_SESSION['reg']) || $_SESSION['reg']
                                     =='')
{
    $ruser="cart_". $this->id;
}
else
{
    $ruser=$_SESSION['reg']."_". $this->id;
}
echo "<button class='btn btn-success col-xs-
      offset-1 col-xs-10'
onclick=createCookie('".$ruser."', '". $this->id."')>
      Add To My Cart</button>";
echo "</div>";
}

```

Понятно, что приведенный вариант метода Draw() является просто схематичным решением, и каждый из вас должен создать этот метод по-своему.

Теперь откройте в блокноте файл catalog.php и добавьте туда такой код:

```

<form action="index.php?page=1" method="post">
<div class="row" style="margin-right:10px;">
    <select class="pull-right" name="catid">
        <option value="0">Select category...</option>
<?php
$pdo=Tools::connect();
$ps = $pdo->prepare("SELECT * FROM Categories");
$ps->execute();
while ($row=$ps->fetch())
{
    echo '<option value="'. $row['id']. '">'. $row
        ['category']. '</option>';
}
?>
</select>
</div>
<?php

echo '<div class="row" style="margin-right:10px;">';
$items=Item::GetItems();
foreach($items as $item)
{
    $item->Draw();
}
echo '</div>';
?>
</form>

```

У меня созданная страница выглядит таким образом:



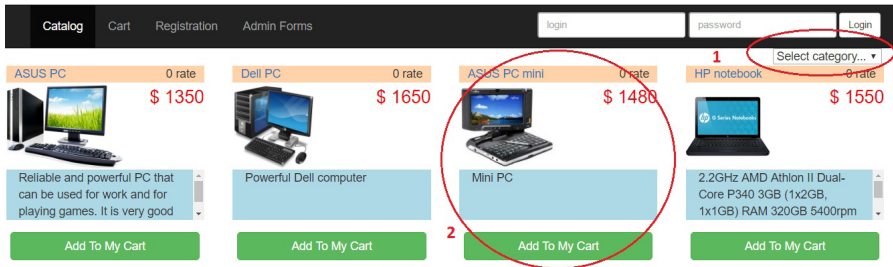


Рис. 4. Страница выбора товаров всех категорий

Под индексом 1 на этом рисунке указан список для выбора категории, под индексом 2 – результат вывода метода Draw() для одного товара.

Теперь надо сделать так, чтобы при выборе категории товара в выпадающем списке обновлялся список товаров на странице. Сделаем это с помощью AJAX. Сделать это несложно. Надо добавить обработчик события onchange для списка выбора категорий и небольшой код на Javascript, в котором будет создаваться AJAX запрос. Кроме этого надо будет добавить в папку pages еще один файл – обработчик AJAX запроса.

Сначала изменим список выбора категорий, добавив туда обработчик:

```
<select class="pull-right" name="catid"
        onchange="getItemsCat(this.value)">
```

Далее добавим атрибут id элементу div, в котором выполняется цикл для вызовов метода Darw(), чтобы по этому идентификатору можно было вставлять результат, присланный AJAX запросом:

```
echo '<div class="row" id="result" style="margin-right:10px;" >';
```

Теперь добавим внизу страницы catalog.php требуемый код на Javascript:

```
<script>
function getItemCat(cat)
{
    if (cat=="")
    {
        document.getElementById('result').
            innerHTML="";
    }
    //creating AJAX object
    if (window.XMLHttpRequest)
        ao=new XMLHttpRequest();
    else
        ao=new ActiveXObject('Microsoft.XMLHTTP');
    //anonymous function for result processing
    ao.onreadystatechange=function()
    {
        if (ao.readyState==4 && ao.status==200)
        {
            document.getElementById('result').
                innerHTML = ao.responseText;
        }
    }
    //preparing post AJAX request
    ao.open('post', 'pages/lists.php', true);
    ao.setRequestHeader("Content-type",
        "application/x-www-form-urlencoded");
    ao.send("cat="+cat);
}
</script>
```

Вы можете заметить, что в этом примере создание объекта AJAX отличается от того, что мы делали в предыдущем примере. Там мы использовали трехступенчатый механизм создания объекта. Здесь же применяем двухступенчатый. Это объясняется тем, что на сегодня

уже практически не осталось старых версий браузеров, на которые рассчитана третья ветка поиска в нашем первом примере. Поэтому сегодня всегда ограничиваются таким способом создания объекта AJAX, как в этом примере.

Наконец, добавим в папку pages еще один файл – обработчик AJAX запроса с именем lists.php:

```
<?php
include_once('classes.php');
$cat=$_POST['cat'];
$pdo=Tools::connect();
//calling GetItems() method with parameter
$items=Item::GetItems($cat);
if($items==null)exit();
//drawing selected items
foreach($items as $item)
{
    $item->Draw();
}
?>
```

Поскольку список товаров может быть большим, мы используем POST AJAX запрос, для которого размер передаваемых данных не столь критичен, как для GET метода. Теперь можно проверить, как работает наш механизм отбора товаров по категориям:

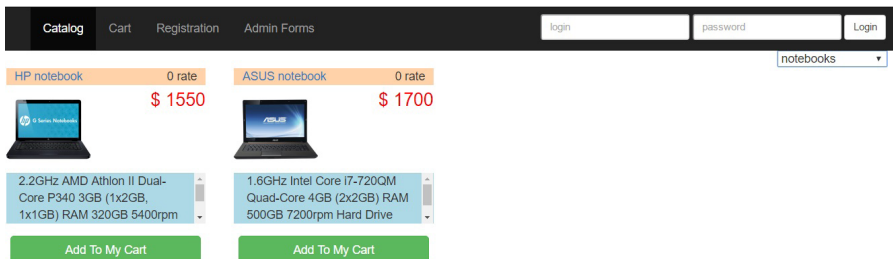


Рис. 5. Страница выбора товаров всех категорий

Перейдем к реализации корзины. Для каждого товара, отправляемого в корзину, мы будем создавать куки файл, в который будем записывать идентификатор отобранного товара. Имя куки файла будет представлять собой конкатенацию имени пользователя и идентификатора – так мы добьемся уникальности имени. Если же покупку будет совершать незарегистрированный пользователь, то в качестве имени пользователя будем использовать какое-либо предопределенное значение и ограничим время существования такого куки файла временем существования сессии. А зарегистрированные пользователи смогут хранить свою корзину, например, семь дней.

Мы с вами еще не сделали механизм аутентификации и авторизации для нашего сайта. Вы должны будете сделать это самостоятельно, используя в качестве подсказки решение из предыдущего сайта. Вам надо будет добавить в класс Customer метод `login()`, который будет проверять входящего пользователя. При удачной авторизации в сессию должна заноситься переменная с индексом `reg`, в которой будет храниться имя вошедшего пользователя (`$_SESSION['reg']=$name`).

Вы помните, что куки создаются вызовом функции `setcookie()`. Однако эта функция очень чувствительна к контексту вызова, и ее не всегда можно вызвать в требуемый момент. Она должна вызываться до того, как в браузер был направлен какой-либо вывод. В нашем случае вызов `setcookie()` приведет к ошибке. Поэтому мы будем создавать куки для корзины с помощью Javascript.

Добавим в файл `catalog.php` в раздел Javascript к уже

имеющейся там функции `getItemsCat()` еще одну функцию, создающую куки:

```
function createCookie(uname,id)
{
    var date = new Date(new Date().getTime() + 60 *
                        1000 * 30);
    document.cookie = uname+"="+id+"; path=/;
                        expires=" + date.
    toUTCString();
}
```

Перейдите на страницу отображения товаров и отправьте в корзину несколько товаров. Затем с помощью браузера проверьте, создались ли куки. У меня в браузере Chrome созданные куки выглядят так:

The screenshot shows a web application interface with a navigation bar (Catalog, Cart, Registration, Admin Forms) and a login section. Below the navigation bar, there is a product catalog with four items: ASUS PC (\$1350), Dell PC (\$1650), ASUS PC mini (\$1480), and HP notebook (\$1550). Each item has a description and a 'Select category...' dropdown. Below the catalog, there is a table showing the cookies set by the application. The table has columns for Name, Value, Domain, Path, Expires / Max-Age, Size, HTTP, Secure, and SameSite. The cookies listed are:

Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Secure	SameSite
ym_uid	142052231203168027	localhost	/	2016-12-05T15:03:50.000Z	9			
cart_1	1	localhost	/	2018-11-24T19:03:50.000Z	26			
cart_4	4	localhost	/	2016-12-04T19:14:08.000Z	7			

Рис. 6. Куки для корзины

## Страница отображения корзины

Теперь создадим механизм отображения корзины для пользователя, редактирования состава корзины и оформление заказа. Для этого мы переберем в цикле массив `$_COOKIE`, найдем в нем элементы с именами, совпадающими с именем текущего пользователя, и про-

читаем оттуда идентификатор товара. По идентификатору создадим объект товара и вызовем для него метод, рисующий отображение товара для корзины. Наш метод будет называться DrawForCart() и будет отображать товар для корзины в одну строку. Вот код этого метода:

```
function DrawForCart()
{
    echo "<div class='row' style='margin:2px;'>";
    echo "<img src='".$this->imagepath.'"
                                width='70px'
        class='col-sm-1 col-md-1 col-lg-1'>";
    echo "<span style='margin-
        right:10px;background-color:#ddeaaa;

color:blue;font-size:16pt' class='col-sm-3 col-
                                md-3 col-lg-3'>";
    echo $this->itemname;
    echo "</span>";

    echo "<span style='margin-
        left:10px;color:red;font-size:16pt;
        background-color:#ddeaaa;'
        class='col-sm-2 col-md-2 col-lg-2' >";
    echo "$&nbsp;".$this->pricesale;
    echo "</span>";
    $ruser='';
    if(!isset($_SESSION['reg']) || $_SESSION['reg']
                                == "")
    {
        $ruser="cart_".$this->id;
    }
    else
    {
        $ruser=$_SESSION['reg']."_".$this->id;
    }
}
```

```

        echo "<button class='btn btn-sm btn-danger'
            style='margin-left:10px;'
            onclick=eraseCookie('$ruser')>x</
            button>";

    echo "</div>";
}

```

Обратите внимание, что куки файл удаляется javascript функцией `eraseCookie()`. В эту функцию при вызове передается полное имя или часть имени куки файла. Так сделано потому, что эта функция применяется и для удаления одного куки при редактировании корзины, и при удалении всех куки при покупке товара. Теперь можно создать код для страницы `cart.php`:

```

<?php
echo '<form action="index.php?page=2"
method="post">';
//define the current user name
$ruser='';
if(!isset($_SESSION['reg']) || $_SESSION['reg'] == "")
{
    $ruser="cart";
}
else
{
    $ruser=$_SESSION['reg'];
}
//total cost of the cart
$total=0;
foreach($_COOKIE as $k => $v)
{
    $pos=strpos($k, "_");
    if(substr($k,0,$pos) == $ruser)
    {

```

```

        //get the item id
        $id=substr($k,$pos+1);
        //create the item object by id
        $item=Item::fromDb($id);
        //increase the total cost
        $total+=$item->pricesale;
        //draw the item
        $item->DrawForCart();
    }
}
echo '<hr/>';
echo "<span style='margin-left:100px;color:blue;font-size:16pt;
background-color:#fffff;' class='' >
Total cost is: </span><span style='margin-
left:10px;color:red;font-size:16pt;
background-color:#fffff;' class='' >".$total."</
span>";
echo '<button type="submit" class="btn btn-success"
name="suborder" style="margin-left:150px;"
onkeyup=eraseCookie("'" . $ruser . "' )>
Purchase order</button>';
echo '</form>';
if(isset($_POST['suborder']))
{
    foreach($_COOKIE as $k => $v)
    {
        $pos=strpos($k,"_");
        if(substr($k,0,$pos) == $ruser)
        {
            //get the item id
            $id=substr($k,$pos+1);
            //create the item object by id
            $item=Item::fromDb($id);
            //sale the item
            $item->Sale();
        }
    }
}
}

```



```

?>
<script>
//creating cookie with javascript
function createCookie(uname,id)
{
    var date = new Date(new Date().getTime() + 60 *
                                1000 * 30);
    document.cookie = uname+"="+id+"; path=/;
    expires=" +
    date.toUTCString();
}
//deleting cookie with javascript
function eraseCookie(uname)
{
    var theCookies = document.cookie.split(';');
    for (var i = 1 ; i <= theCookies.length; i++)
    {
        if(theCookies[i-1].indexOf(uname) === 1)
        {
            var theCookie=theCookies[i-1].
                split('=');
            var date = new Date(new
                Date().getTime()-60000);
            document.cookie =
theCookie[0]+"="+id+"; path=/;
expires=" + date.toUTCString();
        }
    }
}
</script>

```

И снова вы видите преимущества использования ООП, позволяющего вместить в небольшом количестве строк кода много различных действий. Вот так выглядит корзина у меня:

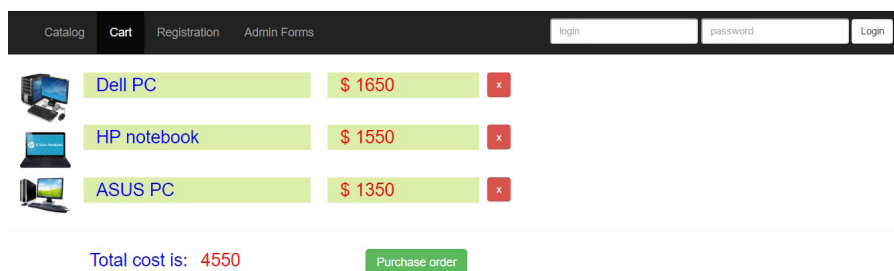


Рис. 7. Корзина

Красные кнопки в каждой строке предназначены для удаления одного товара из корзины. Зеленая кнопка предназначена для оформления заказа на покупку товаров, указанных в корзине. При нажатии на эту кнопку выбранные товары удаляются из таблицы Items, и информация о них заносится в таблицу Sales. Таким образом, в таблице Sales мы накапливаем информацию для формирования отчетов о финансовой деятельности нашего магазина. Используемый здесь метод Sale() может быть таким:

```
function Sale()
{
    try{
        $pdo=Tools::connect();
        $ruser='cart';
        if(isset($_SESSION['reg']) && $_SESSION['reg'] !='')
        {
            $ruser=$_SESSION['reg'];
        }

        //Increasing total field for Customer
        $sql = "UPDATE Customers SET total=total
+ ?
```

```

WHERE login = ?";

$ps = $pdo-
>prepare($sql);
    $ps->execute(array($this-
>pricesale,$ruser));
    //Inserting info about sold item into
    table Sales
$ins = "insert into Sales
(customername,itemname,pricein,pricesale,datesale)
values(?,?,?, ?, ?)";
$ps = $pdo->prepare($ins);
$ps->execute(array($ruser,$this-
    >itemname,
    $this->pricein,$this->pricesale,
    @date("Y/m/d H:i:s")));
    //deleting item from Items table
$del = "DELETE FROM Items WHERE id = ?";
$ps = $pdo->prepare($del);
$ps->execute(array($this->id));
}
catch(PDOException $e)
{
    echo $e->getMessage();
    return false;
}
}

```

Код обработки кнопок удаления товара из корзины и кнопки покупки не содержит ничего нового для вас. Для удаления и создания куки файлов мы используем Javascript. Обработка нажатия кнопки покупки также должна приводить к удалению куки файлов, из которых сформирована текущая покупка. Сумма совершенной покупки добавляется в таблицу Customers в поле total.

## Итоги

Подведем итоги рассмотренному процессу разработки сайта интернет-магазина. Мы разрабатывали этот сайт в стиле ООП, и сейчас надо поговорить об особенностях такой разработки.

Что прежде всего бросается в глаза? Кода было написано больше, чем в случае разработки предыдущего сайта для турагентства. Безусловно, можно говорить, что логика интернет-магазина несколько сложнее логики предыдущего сайта, но в целом функционал обоих сайтов приблизительно одинаков. И при этом для интернет-магазина кода написано больше. Это объясняется тем, что для этого сайта мы создавали классы. Посмотрите на объем файла `classes.php`, он довольно большой.

Получили ли мы какую-нибудь выгоду от того, что создали эти классы? Давайте разберемся с этим вопросом. По ходу изложения материала я обращал ваше внимание на то, каким небольшим по объему был код в некоторых случаях – например, на странице `catalog.php`. Дело в том, что значительная часть функционала вынесена в сами классы. Смотрите: чтобы красиво отобразить в браузере товар, нам достаточно вызвать метод `Draw()`. Чтобы выполнить продажу товара – достаточно вызвать метод `Sale()`. И так во многих других случаях.

Мы получили хороший повторно используемый код. Теперь нам будет намного проще изменять функционал сайта, если возникнет такая необходимость. Наш код хорошо масштабируемый.

Это, безусловно, преимущество, особенно важное в тех случаях, когда требования к создаваемому проек-

ту изменяются по ходу разработки. А это бывает очень часто.

А теперь предположите, что вам не надо было писать все классы «с нуля», что какие-то классы у вас уже были созданы для других проектов. А так обычно и бывает у опытных программистов, которые, создавая класс, стараются сделать его универсальным, а не подходящим для текущего конкретного случая. Если вам не совсем подходят ваши классы из других проектов, они могут подойти вам в качестве базовых типов для новых классов. Остальное поможет сделать наследование. А это тоже экономия кода и времени.

Как же ответить на вопрос, стоит ли писать в ООП стиле или можно обойтись без него? Рекомендация будет такой. Обойтись без ООП можно в случае создания очень небольших проектов с хорошо известной вам структурой и логикой. Если вы создаете сайт среднего размера и больше, если вы понимаете, что логику сайта, возможно, придется изменять, если вы как разработчик создаете новый для себя продукт, с которым еще не имели дела ранее – обязательно используйте ООП. Это единственная гарантия, что вы не утонете в огромном количестве кода, что вам не надо будет создавать много разных «подпорок», обеспечивающих адекватное поведение кода, что вам не придется переписывать свой проект несколько раз, потому что изменились требования или потому что вы сами что-то упустили из вида.

Ответственный разработчик всегда выберет разработку в стиле ООП. У вас сейчас может быть только один довод против использования ООП – вам это непривычно, и вам кажется, что ООП создает больше

проблем, чем решает. Это объясняется тем, что у вас мало практики. Поэтому – приобретайте эту практику. Используйте ООП. И наш урок поможет вам в этом: вы должны выполнить домашнее задание – конечно же, в стиле ООП.

# Домашнее задание

---

Уже традиционно в качестве домашнего задания вам предлагается завершить создание сайта. При выполнении этой работы каждый из вас должен придумать и реализовать свой собственный способ выполнения работы.

Что нам осталось выполнить?

- Механизм аутентификации и авторизации пользователей (обработчик для формы входа).
- На странице администратора добавить форму для работы с категориями товаров и форму добавления картинок к товару (сейчас у нас добавляется только одна, титульная картинка для каждого товара, которая хранится в таблице `Items`, однако еще надо сделать добавление картинок для товаров в таблицу `Images`).
- Создать страницу `iteminfo.php`, которая активируется при клике по названию товара на странице `Catalog`; на этой странице отображать полное описание товара с галереей.
- В качестве дополнительного задания можно сделать отчеты по результатам торговой деятельности магазина, основываясь на данных в таблице `Sales` (там есть данные о закупочной и продажной цене проданных товаров, о том, кто купил и когда и т.п.).



## Урок №4

# Основы ООП

© Александр Геворкян  
© Компьютерная Академия «Шаг»  
[www.itstep.org](http://www.itstep.org)

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объёме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объём и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.