Лабораторно упражнение № 8

Класове

1. Декларацията на клас

Декларацията на класа става с ключова дума **class** и описва атрибутите и методите на класа, както и областта им на достъп:

В PHP5 клас се дефинира със следния синтаксис:

class Име_на_класа{
<модификатор за достъп> $име_на_свойство; // var $име_свойство; PHP4
/*списък от свойства (променливи)*/
[<модификатор за достъп>] function име_метод(){
/* определение на метода */
}
/*списък от методи*/
}

Имената на свойствата на обектите на класа според синтаксиса на PHP4 се обявяват с помощта на ключовата дума **var (еквивалентен на public),** докато за PHP5 се изисква задаване на модификатор за достъп, а методите, прилагани към обектите от даден клас се описват чрез функции. В PHP4 всички членове на класа са достъпни.

1.1. Обръщение към текущия член на класа – чрез ключова дума **this**

Вътре в дефинирането на даден клас може да използваме ключова дума **this** за обръщение към текущия член на класа.

1.2.Модификатори за достъп:

- **public** – членът е достъпен както в класа, така и извън него;
- **protected** – съответния член е достъпен в класа, родителските методи и наследници;
- **private** – членът е достъпен само в класа;
- **final** – членът не може да се наследи.

Пример 1: Да се дефинира клас Статии - Article, с полета $Title - заглавие, $Author – автор и $Description - описание.

```
<?php
class Articles
{ // Създаване на клас Статия
  private $title;
  private $author;
  private $description;
} //край на дефиницията на клас Articles
?>
```

2. Деклариране на конструктор, деструктор:

В PHP5 се използва **function__construct([списък от параметри]),** докато в PHP4 – името на конструктора съвпада с името на класа. Конструкторите се

изпълняват при инициализация на обект! PHP не позволява деклариране на функции (в това число и конструктори) с еднакви имена в рамките на класа!

```
  function __construct ($t, $a, $d )
   {$this->title = $t;
    $this->author = $a;
    $this->description = $d;
   }
```

Пример 1 - продължение: За клас Article да се създаде конструктор и метод за извеждане на информацията за екземплярите на класа чрез функция show_ article ().

```
<?php
  class Articles { // Създаване на клас Статия
  private $title;
  private $author;
  private $description;
// конструктор, който присвоява стойности на
// атрибутите на класа
  function __construct ($t, $a, $d ){
    $this->title = $t;
    $this->author = $a;
    $this->description = $d;
   }
/*При изпълнение на методите на класа се създава една променлива с име $this, която представлява указател към текущия клас.  */
/* //За PHP4 - Декларация на конструктор
function Articles($t, $a, $d ){
     $this->title = $t;
     $this->author = $a;
     $this->description = $d;
   }
function __destruct ()
   {//Деструктор - служи за унищожаване на обекти.
   }
  //метод за извеждане на екземплярите на класа
  function show_article(){
    $art = $this->title ."<br>". $this->description.
"<br>Автор:".$this->author;
    echo $art;
  }
}
//извиква се конструктора
$newa=new Articles("My Php Project with calling constructor!!!","Ivan Ivanov", "This is an OOP exercice" );
$newa->show_article();
?>
```

Резултат:
My Php Project with calling constructor!!!
This is an OOP exercice
Автор: Ivan Ivanov

Променлива (обект) от тип клас се създава с помощта на оператора **new**. Извиква се конструктора на класа **__construct()**, който може да не бъде изрично дефиниран:
$newa=new Articles();

Създавайки обект, ние можем да използваме от него всички методи и да получим всички свойства, определени в описанието на класа. За това се използва такъв синтаксис:
$име_обекта->име_свойство или
$име_обекта->име_метод(списък аргументи).

Забележете, че пред названието на свойството или метода знак $ не се слага.

3. Статични атрибути и методи – това са атрибути и методи, които принадлежат на самия клас, а не на обекта. Създават се по време на компилирането на декларацията на класа, а не по време на създаването на обект на класа.

3.1. Деклариране - чрез ключовата дума **static.**
Достъпът е по **името на класа следвано от :: име на статичния член**.

3.2. Достъп до статични атрибути - чрез ключовата дума **self**
Вътре в статичните методи достъпът до статични атрибути се осъществява чрез **self::$NameOfClassAttribute.**

3.3. Деклариране на константи в класа – декларират се чрез ключовата дума **const**:
const Myage=25;

3.4. Достъп до константа в класа
Достъпът до константа в класа става както към статичен атрибут.
Достъп извън класа:
Пример:
echo 'my age is '.Articles**::**Myage.'**
**';

Пример 1 - продължение:

```
 <?php
  class Articles { // Създаване на клас Статия
  private $title;
  private $author;
  private $description;
  const  Myage=25;  // Деклариране на константи в класа
// конструктор PHP5, който присвоява стойности на
```

```
// атрибутите на класа

  function __construct ($t, $a, $d )
   {  $this->title = $t;
     $this->author = $a;
      $this->description = $d;
   }

/* //За PHP4 - Декларация на конструктор
function Articles($t, $a, $d )
   {  $this->title = $t;
     $this->author = $a;
     $this->description = $d;
   }

function __destruct ()
   { //Деструктор... }
 */
  //метод за извеждане на екземплярите на класа
  function show_article(){
   $art = $this->title . "<br>" . $this->description ."<br>Author: " .
$this->author;
    echo $art;
  }
public static $Task="During this year - 2017/2018 I have to
develop:";//статичен атрибут

public static function showCurrentTask()
{    echo self::$Task.'<br>';     }

}//end of class

// Извикване на статичния метод:
Articles::showCurrentTask();

//const  Myage=25;
//константа Myage декларирана в класа Articles
// достъп до нея извън класа става чрез:
echo 'my age is '.Articles::Myage.'<br>';

//извиква се конструктора
$newa=new Articles("My Php Project with calling
constructor!!!","Ivan Ivanov", "This is an OOP exercice" );
$newa->show_article();
?>
```

Резултат:

```
During this year - 2017/2018 I have to develop:
my age is 25
My Php Project with calling constructor!!!
This is an OOP exercice
Author: Ivan Ivanov
```

4. **Клониране на обекти** – чрез **clone()** се създава нов обект, докато чрез присвояване = на практика се създава нова променлива, която сочи същия обект.

```
<?php
class oblast{
public $grad;}
$oblast = new oblast();
$oblast->grad = 'Варна';
$oblast2 = clone $oblast;
echo $oblast->grad."<br>";
echo $oblast2->grad."<br>";
$oblast2->grad = 'София';
echo $oblast2->grad."<br>";
echo $oblast->grad;
 ?>
```

Резултат:

Варна
Варна
София
Варна

5. **Наследяване: extends**

В PHP наследяването се указва чрез extends. Атрибутите и методите могат да се декларират в родителския клас и да се наследят от наследниците.

Пример 2. Пример за наследяване: **extends**

```
<?php
class Person{
    private $EGN;
    public $name = "Harry";
    protected function getID(){
        return $this->EGN;
    }
  }
class Student extends Person
{   public $fnomer = "032784";
```

```
//Предефиниране на родителската функция
    public function getID(){
        return $this->fnomer;
    }
//достъп до public, protected атрибути на родителския клас
}
$Student1 = new Student();
echo "name: ".$ Student1 ->name; //извиква се от базовия
клас
echo "<br>fak. N:".$ Student1 ->getID(); //извиква се от
//собствения клас
//Достъп до атрибути на родителски клас и на класа
//наследник
//са възможни само, ако са с модификатор public
?>
```

Резултат:

name: Harry
fak. N:032784

!!!Един клас може да наследява в PHP само един родителски клас, тоест няма множествено наследяване.

Атрибутите и методите на родителския клас, които са декларирани като public и protected са достъпни в класовете наследници.

В класа наследник могат да бъдат предефинирани, тоест - декларирани със същите имена.

Достъп до атрибути на родителски клас и на класа наследник (отвън) са възможни само, ако тези атрибути са с модификатор public

6. Синоним parent::

Синоним parent – синоним на родителския клас. Посредством **parent::** методите на класа наследник могат да извършват обръщения към методите на родителския клас.

Пример 2 - Продължение:

```
<?php
class Person{
    public $EGN;
    public $name;
    protected function getID(){
        return $this->EGN;
    }
        function __construct($name, $EGN){
        $this->name = $name;
        $this->EGN = $EGN;
    }
}
```

```
class Student extends Person{
    public $fnomer;
//Предефиниране на родителската функция
        function __construct($fnomer,$name, $EGN){
         $this->fnomer=$EGN;
         parent::__construct($name, $EGN);
    }
    public function getID(){
         return $this->fnomer;
    }
//достъп до public, protected атрибути на родителския клас
}
$Student1 = new Student(" ","Kalin","555555");
echo $Student1 ->name.'<br>';
//извиква се атрибут от базовия клас

echo $Student1 ->getID(); //извиква се от собствения клас
//Достъп до атрибути на родителски клас и на класа
//наследник са възможни само, ако са с модификатор public
?>
```

Резултат:

```
Kalin
555555
```

7. Абстрактни методи и класове.

Абстрактни методи и класове – когато се реализира йерархия от класове е желателно някои методи да бъдат реализирани вътре в наследниците.

Пример 3. Да се създаде абстрактен клас Shape с абстрактен метод surface() и клас Rectangle, който го наследява, използвайки неговия абстрактен метод surface**().**

```
<?php
abstract class Shape
{    abstract protected function surface(); }
class Rectangle extends Shape
{public $a;
public $b;
function __construct($n,$m)
{ $this->a=$n;
$this->b=$m;}
public function surface()
{ return ($this->a*$this->b);}
} //end of Rectangle
$l=8; $w=4;
$class1 = new Rectangle($l,$w);
$class1->surface();
echo "Лице на правоъгълник с дължина $l и широчина $w е
=". $class1->surface()."\n";?>
```

Резултат:

Лице на правоъгълник с дължина 8 и широчина 4 е =32

8. Интерфейси

Наименованието произлиза от латинското inter-face (между – лице). Интерфейсите позволяват да се създаде код, специфициращ методите, който трябва да имплементира един клас наследяващ интерфейса.

Описват се чрез ключова дума interface, описанието им прилича на това на клас, но съдържат само декларации на методи - с метод на достъп public, без описание на имплементацията на методите.

8.1. **implements** оператор

За да се имплементира един интерфейс от даден клас, използваме **implements** оператор. Всички методи на интерфейса трябва задължително да се имплементират в класа. Класовете могат да имплементират повече от един интерфейс, като имплементираните интерфейси се описват един след друг, разделени със запетая.

Когато един клас наследява интерфейс, в него не се допуска да има дублиране на методите на интерфейса.

8.2. **extends** интерфейс

Интерфейсът може от своя страна да наследява (extends) интерфейс и то - повече от един интерфейси (докато за класа множествено наследяване не е възможно).

Даденият пример декларира един интерфейс Shape, съдържащ един метод surface().

Интерфейсът Shape се наследява от клас Rectangle, който имплементира методът surface().

Пример 4: Интерфейси

```
<?php
interface Shape{
     public function surface();
}
class Rectangle implements Shape
{//декларация на клас, който реализира интерфейса Shape
public $a;
public $b;
function __construct($a,$b){
              $this->a = $a;
              $this->b = $b;
}
 function surface(){
  return ($this->a*$this->b);
 }
}
$r = new Rectangle(6, 8); //обект на клас Rectangle
echo "Surface = ".$r->surface(); //Surface = 48
```

```
?>
```

Пример 5: Пример за клас, имплементиращ два интерфейса:

```
<?php
interface A { function a( ); }
interface B { function b( ); }
class C implements A, B {
    public function a( ) { echo "Hello a!!!"; }
    function b( ) { echo "Hello b!!!"; }
}
$r = new C( );      //обект на клас C
echo $r->a( );
echo $r->b( );
//Hello a!!!Hello b!!!
?>
```

9. Функции за класове и обекти

- call_user_method_array — Извиква потребителски метод, определен с масив от параметри [непрепоръчителна]
- call_user_method — Извиква потребителски метод на определен обект [непрепоръчителна]
- class_alias — Creates an alias for a class
- class_exists — Проверява дали даден клас е бил дефиниран
- get_called_class — the "Late Static Binding" class name
- get_class_methods — Връща имената на методите на клас
- get_class_vars — Връща свойствата по подразбиране на даден клас
- get_class — Връща името на класа на обект
- get_declared_classes — Връща масив с имената на дефинираните класове
- get_declared_interfaces — Връща масив с всички декларирани интерфейси
- get_object_vars — Връща свойствата на дадения обект
- get_parent_class — Връща името на родителския клас на обект или клас
- interface_exists — Проверява дали даден интерфейс е дефиниран
- is_a — Проверява дали обектът е от този клас или от някой от родителите му
- is_subclass_of — Проверява дали клас е родителски за даден обект
- method_exists — Проверява дали указаният метод на класа съществува
- property_exists — Проверява дали обектът или класът притежават дадено свойство

Повече информация на адрес: http://php.net/manual/bg/ref.classobj.php