Класове в PHP

Виолета Божикова

PHP и ООП

- PHP е създаден като процедурно – ориентиран език. Елементите за ООП са въведени в PHP3.
- В PHP4 са запазени съвместимостите с PHP3, без да се разширят ООП възможностите.
- В PHP5, обектния модел е изцяло преработен:
  - ново название за конструкторите и поява на деструктори.
  - поява на модификатори за достъп до свойствата и методите на класа; статични (Static) свойства и методи; крайни свойства и методи (Final properties and methods)

PHP и ООП

- В PHP4 всички методи и променливи на класа са достъпни отвън, тоест - явяват винаги открити.
- В PHP5 променливите и методите могат да са както открити (достъпни отвсякъде), така и закрити (достъпни само в класа), както и защитени (достъпни вътре в класа и в неговите производни класове).
- Освен това PHP5 в се появи възможност, за създаване на интерфейси и абстрактни класове.
- Цял набор от “магически” методи (magical methods) и др.
- Като цяло, обектният модел на PHP5 значително се усъвършенства и много по-точно съответства на ОО парадигма.

Типът клас

- Класът е възникнал като разширение на типа структура, който от своя страна е разширение на типа масив. От тук следва, че класът е обобщен тип данни, който в PHP включва:
  - Променливи (наричани fields в другите езици). В PHP те се наричат още атрибути или свойства (properties) на класа;
  - функции, (наричани методи в другите езици), описват операции с вътрешните данни на класа;
  - манипулатори на събития (събитийни процедури) – специален тип променливи, на които се присвоява адреса на функции, дефинирани извън класа.

Декларацията на клас

- Декларацията на класа в PHP става с ключова дума class, следвана от името на класа и тяло, включващо описание на атрибутите и методите на класа, както и областта им на достъп.

Декларацията на клас

- В PHP5 клас се дефинира със следния синтаксис:

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


}
```

Декларацията на клас

- В PHP5 клас се дефинира със следния синтаксис:

```
class  Books {          /* Членове променливи*/
   private $price;    private $title;
                              /* Членове функции*/
   function setPrice($par)
         {   $this->price=$par;    }
   function getPrice()
         {   echo $this->price ."<br/>";    }
   function setTitle($par)
         {  $this->title=$par;   }
   function getTitle()
        {  echo $this->title ." <br/>";   }
 }
```

Декларацията на клас PHP4/PHP5

- Вътре в дефинирането на даден клас може да използваме ключова дума **this** за обръщение към текущ член на класа.
- **!!! Не може** дефиницията на PHP клас да се помества в множество файлове.
- !!! Също така **не може** дефиницията на клас да се помества в множество PHP блокове, освен ако прекъсването не е в декларацията на метод. Следният пример няма да работи коректно:

```
<?php
   class test {
   ?>
   <?php
      function test()
      {      print 'OK‘;    }
   }
   ?>
```

Декларацията на клас PHP4/PHP5

- Следното обаче е позволено:

```
<?php
    class test {
        function test() {
            ?>
            <?php
            print 'OK';
        }
    }
    ?>
```

Автоматично зареждане на обекти

- Много разработчици на PHP обектно-ориентирани приложения до PHP 5 създаваха по един PHP файл за всеки клас. След което, създаваха дълъг списък на include конструкции, за файловете , които трябва да се включат в началото на всеки скрипт (по един за всеки клас).
- В PHP 5 това вече не е необходимо. Можете да дефинирате функция **__autoload()**, която се извиква автоматично, в случай че се опитате да използвате клас/интерфейс, който не е дефиниран в текущия скрипт. Извикването на тази функция е последната възможност на скриптовата машина да зареди този клас преди да бъде генерирана фатална грешка.

```
<?php
function __autoload ($class_name)
{  require_once $class_name . '.php';}
/*класовете MyClass1 и MyClass2, които са съответно във
  файловете MyClass1.php и MyClass2.php  ще се заредят*/
$obj  = new MyClass1();
$obj2 = new MyClass2();
?>
```

Модификатори за достъп до членовете

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

Модификатори за достъп до членовете

- public е модификатор за достъп по подразбиране за функциите на класа, тоест може и да не се укаже;
- за променливите на класа се изисква задължително указване на модификатор за достъп (var – public).

Имената на класовете

- Няколко бележки и по повод на имената на класовете. Името на класа трябва да удовлетворява правилата на именоване на обекти в езика PHP, но има редица имена, които са резервирани от разработчиците за техни цели.
- За създаване на класове и функции не бива да използат такива имена.
- Освен това, резервирано е и име stdClass, понеже се използва вътре във виртуалната машина на PHP.
- С функция **get_declared_classes()** се разпечатват както резервираните, така и потребителсите класове:

```
<?php print_r(get_declared_classes());      ?>
```

Резултат:

```
Array ( [0] => stdClass
        [1] => Exception [2] ....
```

Декларация на клас, създаване на обект и достъп до елементите на класа: Демонстрационен пример

- Например, нека е нужно да създадем клас, описващ категория статия.
- Всяка статия има такива свойства като заглавие на статията, автор на статията и кратко съдържание.
- Следва описание на действията, които желаем да извършваме с нея: нека да трябва да задаваме стойности на изброените свойства, да изобразяваме статията в браузъра. Тогава определението на този клас може да изглежда по следния начин:

```
<?php
class Articles { // Създаване на клас Статия
  private $title; private $author; private $description;
// метод, който присвоява стойности на
// атрибутите на класа
  function make_article($t, $a, $d)
{$this->title = $t;  $this->author = $a;  $this->description =
    $d;}
//метод за извеждане на екземплярите на класа в
    браузъра
  function show_article()
{    $art = $this->title . "<br>" . $this->description .
      "<br>Автор: " . $this->author;
    echo $art;
  } } //край на дефиницията на клас Articles
```

//Следва дефинирането на обект:

```
$newa=new Articles();
//дефиниран е обект $newa от тип Articles()
$newa->make_article("My Php Project", "Ivan
   Ivanov", "This is an OOP exercice");
 $newa->show_article();
?>
```

Резултат от изпълнение на скрипта:

My Php Project

This is an OOP exercice

Автор: Ivan Ivanov

Създаване на променлива (обект) от тип клас

И така, както се вижда от примера:

- обект от тип клас се създава с помощта на оператора new: така се извиква конструктора на класа **__construct()**, който може и да не бъде изрично дефиниран:

$newa=new Articles();

- Така (чрез new) се създава обект в паметта и указателят към този обект се присвоява на променлива та $**newa**.
- Създавайки обекта, вече можем да използваме всички методи и свойства, определени в описанието на класа.
- използва се следния синтаксис:

$име_обект->име_свойство

или

$име_обект->име_метод(списък на аргументи).

!!!Пред названието на свойството или метода знак $ не се слага.

- **!!!Пред названието на свойството или метода знак $ не се слага.**
- Пример:

```
$newa=new Articles();
$newa->make_article("My Php Project", "Ivan Ivanov", "This is an OOP exercice");
$newa->show_article();
```

Деклариране на конструктор, деструктор:

- !!!В PHP5, за деклариране на конструктор се използва магическия метод __construct ():

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!

Деклариране на конструктор:

Вместо декларираната в примера функция make_article($t, $a, $d) бихме могли да дефинираме конструктор (PHP 5 стил):

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

Деклариране на конструктор:

Също, вместо декларираната в примера функция make_article($t, $a, $d) бихме могли да дефинираме конструктор (PHP 4 стил):

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

Деклариране на деструктор:

- function __destruct() служи за унищожаване на обекти.

В примера за клас Articles бихме могли да добавим и деструктор, по следния начин:

```
function __destruct()
  {    //Деструктор... }
```

Пример с деструктор:

```
...class Animal
{  public $name = "No-name animal";
   public function __construct($name)
    {  echo "<BR>1. I'm alive!";
       $this->name = $name;    }
    public function disp()
    { echo "<BR>2. I'm ".$this->name; }
    public function __destruct()
    { echo "<BR>3. I'm dead now :("; }
}
$animal = new Animal("Bob");
echo $animal->disp();
...
?> //извиква се деструктора
```

1. I'm alive!
2. I'm Bob
3. I'm dead now :(

Достъп до атрибути и методи на класа чрез променлива с име $this

- При изпълнение на конструктора на класа се създава една променлива с име $this, която представлява указател към текущия клас.

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

Статични атрибути и методи

- **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. Декларират се чрез **static**.
- Достъпът е **по името на класа** следвано от

:: **име на статичния член**.

Статични атрибути и методи

- **Пример**: да декларираме в класа Articles следните статични членове:

```
...
public static $Task="During this year … I have to
   develop:";                    //статичен атрибут
public static function showCurrentTask()
{
    echo self::$Task.'<br>';
}
}//end of class
```

Статични атрибути и методи - извикване

- Извикването, извън класа статичния метод showCurrentTask():

```
Articles::showCurrentTask();
```

Вътре в статичните методи достъпът до статични атрибути се осъществява чрез self::$NameOfClassAttribute, а за нашия пример:

```
...
echo self::$Task;
```

Целият код до момента е показан на следващия слайд:

```php
<?php
 class Artides { // Създаване на клас Статия
 private $title;
 private $author;
 private $description;
 const  Myage=25;  // Деклариране на константи
// конструктор - PHP5
 function __construct ($t, $a, $d )
  {    $this->title = $t;
       $this->author = $a;
       $this->description = $d;
  }
/*  PHP4 – Декларация на конструктор
function Artides($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 $Task="During this year a have to develop a thesis!";
 //статичен атрибут
public static function showCurrentTask()
{        echo self::$Task.'<br>';    }
}  //end of class
 // Извикване на статичния метод:
Articles::showCurrentTask();
//достъп до константа в класа
echo 'my age is '.Articles::Myage.'<br>';
//извикване на конструктора
$newa=new Articles("My Php Project with calling constructor!!!",
"Ivan Ivanov", "This is an OOP exercice" );
$newa->show_article();
$newb = clone $newa; //клониране на обекта newa
?>
```

Резултат:

During this year a have to develop a thesis!
my age is 25
My Php Project with calling constructor!!!
This is an OOP exercice
Author: Ivan Ivanov

Деклариране на константи в класа

- декларира се чрез ключовата дума const (вижда се в показания вече пример):

```
const  Myage=25;
```

//константа Myage декларирана в класа Articles

- достъп до нея извън класа става така:

```
echo 'my age is '.Articles::Myage.'<br>';
```

- Резултат:

```
my age is 25
```

Клониране на обекти

- **Клониране на обекти** – Клонирането създава пълно копие на обект, като се копират и стойностите на атрибутите в момента на клонирането, тоест промяната на оригинала няма да засегне копието.
- В примера имахме:

```
$newb = clone $newa;
//клониране на обект newa
```

- Възможно е също:

```
$newc = $newa;
// създаване на нов обект, който сочи същия обект.
```

Наследяване: extends

- Наследяването е възможност за създаване на клас въз основа на родителски клас.
- В PHP наследяването се указва чрез **extends**. Атрибутите и методите могат да се декларират в родителския клас и да се наследят от наследниците, като по този начин се намалява обема на програмата и създава ясна логическа връзка между класовете.
- Механизмът на наследяване – е както знаем важна част на всеки обектно-ориентиран подход. Да дадем един пример, който да обясни неговата същност.

Наследяване: пример

- Да допуснем, че имаме описание на **Човек**. Всеки от нас може да опише типа Човек по различен начин.
- Може да опишем човека и като **Програмист**: програмистът знае такива и такива езици за програмиране, такива и такива ОС, участвал е в такива и такива проекти. Но един Програмист, не престава да бъде Човек въобще, тоест той има всички свойства на един Човек: име, фамилия, местожителство и др.
- Ако тези разсъждения ги опишем с термините на ООП, то може да кажем, че ние сме описали два класа – клас Човек (**Person**) и неговият наследник - клас Програмист (**Programmer**), всеки със своите свойства и методи.

class Programmer extends Person

```php
<?php
class Person {
  private $first_name;
  private $last_name;

  function make_person($t,$a){
    $this->first_name = $t;
    $this->last_name = $a;
  }
  function show_person(){
  echo ("<h2>" . $this->first_name . " " .
  $this->last_name . "</h2>");
  }
}
class Programmer extends Person{
 public $langs = array ("Lisp");
 function set_lang($new_lang){
    // методът добавя еще
    // един език към списъка
    $this->langs[] = $new_lang;
  }
}
?>
```

```php
<?php
$progr = new Programmer;
$progr->set_lang("PHP");
   // методи, определени за клас
   // Programmer
print_r($progr->langs);
// методи, определени за класа Person
$progr->make_person("Ivan","Ivanov");
$progr->show_person();
?>
```

Резултат:
Array ([0] => Lisp [1] => PHP)
Ivan Ivanov

Още един елементарен пример:

```
<?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*/
}
$John = new Student();
echo $John->name;
//извиква се от базовия клас
echo $John->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:
Harry032784

Още за наследяването в PHP :

- Един клас може да наследява само един родителски клас, тоест няма множествено наследяване.
- Атрибутите и методите на родителския клас, които са декларирани като **public** и **protected** са достъпни за класовете - наследници. В класа наследник такива методи могат да бъдат предефинирани.
- Вижда се, че отвън, достъп до атрибутите на родителския клас и на класа наследник са възможни само, ако са с модификатор public.

Синоним parent

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

```
class Student extends Person
{...
 function __construct($fnomer,$name, $EGN)
 {...    parent::__construct($name, $EGN);}
...}
```

```php
<?php
class Person{
  public $EGN;
  public $name;
  protected function getID(){
      return $this->EGN;
  }
function __construct($name, $EGN =
123456)
{     $this->name = $name;
      $this->EGN = $EGN;
  }
}
class Student extends Person{
  public $fnomer = "032784";
//Предефиниране на родителската
функция
function __construct($fnomer,$name,
$EGN)
  {
      $this->fnomer=$EGN;
      parent::__construct($name, $EGN);
  }
  public function getID(){
      return $this->fnomer;
  }
//достъп до public, protected атрибути
на родителския клас
}
$John = new
Student("343434","Kalin","555555");
echo $John->name.'<br>'; //извиква се
атрибут от базовия клас
echo $John->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

```php
<?php
class Person{
  public $EGN;
  public $name;
  protected function getID(){
      return $this->EGN;
  }
function __construct($name, $EGN =
123456)
{     $this->name = $name;
      $this->EGN = $EGN;
  }
}
class Student extends Person{
  public $fnomer = "032784";
//Предефиниране на родителската
функция
function __construct($fnomer,$name,
$EGN)
  {
      $this->fnomer=$EGN;
      parent::__construct($name, $EGN);
  }
  public function getID(){
      return $this->fnomer;
  }
//достъп до public, protected атрибути
на родителския клас
}
$John = new
Student("343434","Kalin","555555");
echo $John->name.'<br>'; //извиква се
атрибут от базовия клас
echo $John->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

Ключова дума final

- В PHP 5 е въведена ключовата дума final, като използването и пред дефиницията на метод от родителски клас предотвратява възможността този метод да бъде дефиниран повторно в дъщерен клас.
- Ако даден клас е дефиниран като final, то той не може да бъде наследяван:

```
final class BaseClass {...   }

class ChildClass extends BaseClass {...}

// Fatal error: Class ChildClass may not inherit fr
   om final class (BaseClass)
```

Итериране на обекти

- PHP 5 предоставя възможност на обектите да бъдат дефинирани така, че да могат да бъдат итерирани извън класа, примерно с цикъл foreach.
- По подразбиране, извън класа, всички видими свойства на обектите ще бъдат изведени при итерирането.

Итериране на обекти

```
<?php
class MyClass
{   public $var1 = 'public променлива';
    public $var2 = 'public променлива';
    protected $var4 = 'protected променлива';
    private   $var5   = 'private променлива';
    function iterateVisible()
    { echo "Iterate Visible Fields:\n";
      foreach($this as $key => $value)
      { print "$key => $value\n“;      }  }
}
$class = new MyClass(); $class->iterateVisible();
echo "\nIterate Visible Fields out of the
    class:\n";
foreach($class as $key => $value) {    print
    "$key => $value\n“;}
?>
```

Iterate Visible Fields:
var1 => public променлива
var2 => public променлива
var4 => protected променлива
var5 => private променлива

Iterate Visible Fields out of the class:
var1 => public променлива
var2 => public променлива

Абстрактни методи и класове

- Абстрактни методи и класове: когато се реализира йерархия от класове, понякога е желателно, някои от описаните родителски методи да бъдат реализирани вътре в наследниците.

Абстрактни методи и класове - пример

```
<?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 "The surface of a rectangle with dimensions
$l and $w is ".$class1->surface() ."\n";
?>
```

The surface of a rectangle with dimensions 8 and 4 is 32

Интерфейси

- Интерфейсите позволяват да се създаде код, специфициращ методите, които 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 (в C#.Net не се допуска ключова дума public, но се подразбира public), без описание на имплементацията на методите.

Интерфейси

implements

- 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 описават един след друг, разделени със запетая.
- Когато един клас наследяване интерфейс, в него не се допуска да има дублиране на имената на методите на интерфейса.

Интерфейси

extends

- Интерфейсът може от своя страна да наследява (extends) интерфейс и то - повече от един интерфейси (докато за класа множествено наследяване не е възможно).
- Когато един интерфейс IBar extends Bar, не се допуска IBar да дублира методите на интерфейс Bar (такова в C#.Net се допуска).
- Даденият пример декларира един интерфейс Shape, съдържащ един метод surface().
- Интерфейсът Shape се наследява от клас Rectangle, който имплементира методът surface().

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
?>
```

Интерфейси

Ето един 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!!!
?>
```

Изключения

Какво е изключение?

- Едно изключение е логическа/системна грешка, която възниква по време на нормалното изпълнение на един скрипт.
- PHP5 дава възможност на програмистът да прихване възникнало изключение, да прекрати нормалното изпълнение на програмата и да изпълни специално написан код за обработка на това изключение.
- Моделът на обработка на изключенията тук е подобен на този в другите езици. Използва се структурата try…catch.

Изключения

Използване на try…catch блок: Пример:

```
try {
        check();
      }
     catch(Exception $e) {
        echo "Message : " . $e->getMessage();
        echo "Code : " . $e->getCode();
     }

function check()
{   if(some error condition)
        {
     throw new Exception("Error String",Error Code);
        }
}
```

Изключения

- В този пример, метод check() е критичният код, намиращ се в try {} блок.
- Под try{}блока има един (или повече) catch() {} блок, който изследва типа на възникналото изключение (параметърът му $e е от тип Exception). В тялото на този блок се поставя кодът който обработва възникналото изключение.
- Във function check(), не се обработва изключението, а само се прихваща и се хвърля (генерира) ново изключение (от тип Exception) използвайки ключова дума ‘throw’, което следва да се обработи от друг handler, в случая от catch блока на try…catch конструкцията. Операторът throw има следния синтаксис:

throw **new** Exception([string exception message,[int exception code]]);

Изключения

- Хвърленото изключение тип Exception приема два параметъра. Първият е string указващ съобщението за грешката (exception message), а вторият – целочислен код за грешката (exception code), който искате да присвоите на тази грешка.

Изключения - пример

```
<?php
function checkNum($number)
{ if($number > 3)
   { throw new Exception("Number is greater than 3","100");
   //ако подаденото число е по-голямо от 3 се хвърля изключение
    }
return true;
}
try
   {  checkNum(28);  //критичния код
    }
/*    catch the exception here    */
catch(Exception $e)
   {  // code to handle the Exception
   echo 'Catch exception here<br />';
   echo 'Message: '   .$e->getMessage().'<br />';
   echo 'Code: '   .$e->getCode().'<br />';
   }
?>
```

```
Catch exception here
Message: Number is greater than 3
Code: 100
```

Изключения

- На следващия слайд е показана анатомията на клас Exception.
- От кода на клас Exception се вижда, че с изключение на **function** __construct и **function** __toString(), никакви други методи на клас Exception не могат да бъдат припокривани.
- Следва един пример за създаване на собствено изключение. В примера се създава клас CustomerException, наследник на клас Exception, като се препокрива конструктора на клас Exception:

function __construct($message = null, $code = 0) …

Анатомия на PHP5 Exception class

```
class Exception {
    protected $message;
    protected $code;
    protected $file;
    protected $line;
    private $string;
    private $trace;
    public function __construct($message = null, $code = 0);
    public function __toString();
    final public function getCode();
    final public function getMessage();
    final public function getFile();
    final public function getLine();
    final public function getTrace();
    final public function getTraceAsString();
    final private __clone();
}
```

Наследяване на клас Exception

- Вие можете да създадете собствено изключение с име **CustomerException**, като наследите клас **Exception** и дефинирате собствен конструктор, който се обръща към базовия конструктор на клас **Exception**:

```
class CustomerException extends Exception
 {
      public function __construct($message, $code)
    {    $t_message = $message;
         parent::__construct($t_message, $code);
    }
}        …
```

Използване на собствено изключение

```
<?php
class CustomerException extends Exception
 {
   public function __construct($message, $code)
{    $t_message = $message;
     parent::__construct($t_message, $code);
   }
}

function testException()
{throw new CustomerException("Attention! CustomerException
has been raised",102);}

try {testException();}
catch(CustomerException $e) {
        echo "<br>Error Message : ". $e->getMessage();
        echo "<br>Error Code : ". $e->getCode();
}
?>
```

Изход: Error Message : Attention! CustomerException has been raised
Error Code : 102