Usuario:ManuelRomero/ProgramacionWeb/php/POO/introduccion

De WikiEducator
< Usuario:ManuelRomero | ProgramacionWeb/php | POO

Trabajo en proceso, espera cambios frecuentes. **Tu ayuda y retroalimentación son bienvenidos**.
Ver página de charlas (//es.wikieducator.org/index.php?title=Usuario_Discusi%C3%B3n:ManuelRomero/ProgramacionWeb/php/POO/introduccion&action=&lqt_method=talkpage_new_thread).

BLOQUE 2 PHP: **PROGRAMACIÓN ORIENTADO A OBJETOS**

¡Construyendo componentes!

PHP Como lenguaje orientado a objetos

Conceptos básicos | Ejercicios | Práctica | Volver

Archivo:DWES TituloTema6.png

Show presentation

Programación orientada a objetos

- En programación el paradigma imperativo está basado en funciones y datos,
- El paradigma orientado a objetos está basado en Objetos.
- Los **objetos** son el elemento básico y central de la programación orientada a objetos (OOP) o (POO)
- Podemos hablar de *universo de discurso* como el sistema que queremos automatizar por software
- Un Objeto es una entidad (concreta o abstracta) que presenta una identidad propia y un comportamiento o actividad en un entorno con discurso.

Definición

Objeto Cada elemento activo **que identificamos dentro de un determinado universo de discurso**

Ejemplo

En un banco hay cuentas bancarias (objeto)
Las cuentas bancarias se identifican con un número y un titular (nombre, apellido y dni) ***atributos***
Las cuentas se pueden dar de alta, de baja, hacer extracciones e ingresos y transferencias... ***métodos***

- Puede parecer una forma más complicada de programar, pero es una manera de dividir la naturaleza del problema que estamos estudia pueden interactúar entre ellas.
- Cada una de ellas va a tener una identidad propia asignando valores a sus atributos
- Cada una de ellas va a tener un comportamiento concreto que va a ser lo que sabe hacer para que los demás o el programa principal lo

Elementos en la programación orientada a objetos

- De lo dicho anteriormente deducimos que tenemos dos elementos:

1. Los ***atributos*** o características de la clase.
2. Los ***métodos*** o comportamiento de la clase .

Puntos clave

- Una ***clase*** es la estructura de un tipo concreto de objetos.
- Los ***objetos*** son elementos concretos en mi sistema.

Qué es una clase

- Una clase es la definición de un ***componte software***
- el cual tiene una identidad que va a quedar definida por los valores que tengan sus ***atributos***
- y un comportamiento o capacidad de realizar acciones que quedará definido por sus ***métodos***

}}

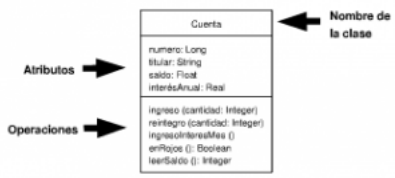
Elementos de la POO

- La estructura común (datos y comportamiento) de los objetos sirven especificar su composición
- Esta composición queda descrita y especificada en la clase.
- Una clase tiene dos elementos

```
Atributos ==> Son las características o datos de un objeto
                      Su valor nos da el estado de un objeto en un momento dado
```

Elementos de la POO

```
Comportamiento==> permiten modificar el estado de un objetos (métodos set)
                      Permiten saber cómo está el estado de un objeto (métodos get)
                      Permiten que un objeto haga cosas en el sistema (comunicación entre objetos)
                      Son las cosas que el objeto sabe hacer Servicios que ofrece
                      También lo son acciones internas para  facilitar el comportamiento del  objeto
                      (métodos privados)
```



OPP En php

- PHP no se diseñó como lenguaje orientado a objetos, por lo que muchas de las características de este paradigma se ha ido incorporando especialmente a partir de la versión 5.3.
- PHP Almacena el valor de un objeto como una referencia (dirección de memoria), no guarda el valor.
- Esto implica que si queremos pasar un objeto a través de la red, debemos serializarlo, para que viaje también el valor del mismo. Veren

Los elementos de una clase en php se identifican como

1. propiedades: son los atributos o características de la clase.
2. métodos: representas el comportamiento de la misma.

Definir una clase en php

```
class NombreClase{
    //propiedades
    //métodos
}
```

- ***NombreClase*** es un identificador válido con la siguiente expresión regular

```
^[a-zA-Z_][a-zA-Z0-9_]*$
```

- El nombre de las clases se recomienda que empiece por mayúsculas

- ***propiedades*** son las variables o constantes definidas dentro de la clase.
- ***métodos*** son las funciones que definirán el comportamiento de la clase.

Ejemplo

Vamos a crear una clase llamada fecha

- Atributos de la clase (dia, mes, year)

```php
class fecha{
//Atributos
 public $dia;
 public $mes;
 public $year;
 //....
}
```

Pilares básicos de la POO

Puntos clave

Encapsulación
Herecia
Polimorfismo
Abstraccion

Encapsulación: Acceso a los componentes

- A la hora de definir tanto las propiedades como los métodos, especificaremos el nivel de acceso que se tiene a ese elemento
- Es una buena práctica de programación no dejar acceso directo a los atributos de una clase, sino acceder a ellos a través de los método
- LA encapsulación es uno de los pilares de la programación orientada a objetos y me permite o restringe la visibilidad de sus componente

visibilidad

- Visibilidad o alcance de las propiedades no constantes y los métodos de las clases
- Implementa el principio de encapsulación.
- Permite especificar el nivel de acceso que se tienen sobre los elementos

Visibilidad

- Son tres los tipos de visibilidad que podemos especificar:

public
tipo de visibilidad asignada por defecto en caso de no especificar.
private
protected

public

- Este tipo de visibilidad es asignada por defecto
- Los elementos públicos pueden ser accesibles en cualquier momento del programa.
- Recordemos que para acceder a un elemento debemos especificar el objeto o clase del que queremos el elemento
- Si el elemento es estático o constante usaremos el operador ***::*** llamado operador de especificación ámbito #::
- Si el elemento es no estático accedemos a través del operador **->**
- Observa como accedo a los atributos el código siguiente

```php
<?php
class Fecha {

    private $dia;
    private $mes;
    private $year;

    const DIA = 30;

//.....
    public function validaDia() {
        if ($this->dia > $this::DIA)
```

```
            $error = true;
        return $error;
    }

}
```

Tip: $this es una seudovariable que representa el objeto actual

private

- Los elementos especificado con este modificador de acceso hace que su visibilidad se reduzca al interior de la clase, no pudiendo accede
- En OOP es una tendencia hacer todos los atributos privados y acceder a ellos por los métodos setter and getter.

```
<?php
class Usuario {
    private $usuario;
    private $password;

    public function __construct($usuario) {
        $this->password = 'passDefecto';
        $this->usuario = $usuario;
    }

    public function __destruct() {
        echo 'Me voy......';
    }
    public function getUsuario(){
        return $this->usuario;
    }
    public function getPass(){
        return $this->password;

    }
    public function setUsuario($user){
        this->usuario =$user;
    }
    public function setPass($pass){
        this->password =$pass;
    }
}

$usuario = new Usuario('manolo');
//....
$pass = $_POST['pass'];
//....
$usuario->setPass($pass);
//...
?>
```

protected

- Este tipo de visibilidad implica que los elementos así especificados solo son accesible por la propia clase y por las clases derivadas
- Para ello hay que ver la herencia que veremos más adelante

```
<?php
class persona{
    protected $nombre;
    protected $fNac;
    //....
}
//.....
class medico extends persona{
    private $numColegiado;

    public function __construct($nombre, $fechaNacimiento, $colegiado) {
        $this->nombre=$nombre;
        $this->fNac=$fechaNacimiento;
        $this->numColegiado=$colegiado;

    }
    public function visualiza(){
        echo "Medico $this->nombre";
    }
}
$medicoPueblo1= new medico("pedro", "1/1/1969","123456");
$medicoPueblo1->visualiza();

?>
```

Propiedades

- Al igual que en el código estructurado los valores que almaceno en memoria, las propiedades de los objetos pueden ser.

1. Variables
2. Constantes

Constantes

- Para definir constantes se usa la palabra reservada ***const***. Como ya sabemos este valor no puede ser modificado durante la ejecución.
- El identificador de las constantes no empieza por $.
- A una constante hay que asignarle un valor no pudiendo asignar expresiones.
- Todos los objetos de la misma clase comparte el valor de la constante. Por lo que se tomará como un valor estático.

Accediendo al valor de una constante

1. Dentro de la clase: usaremos el operador ***self*** junto con el ***operador de resolución de ámbito ::***
2. En el programa: identificando la clase, mediante el nombre de la clase, o bien la variable objeto de esa clase junto con el operador de re identificador de la constante.

- Vemos un ejemplo de su uso

```php
<?php
class Constantes{
      const K = 10;
      const IVA = 0.21;
      function getValores(){
              echo "Valor de la constante --".self::K."--<br/>";
              echo "Valor del producto de 235 euros base ".((self::IVA*235)+235);
      }
}

$a=new Constantes();
//Mostramos los valores de las constantes
$a->getValores();

echo "<br/>valor de la constante con el nombre de la clase ".Constantes::K;
echo "<br/>valor de la constante con el nombre del objeto".$a::K;
?>
```

Ejemplo

Creamos la clase factura

- Constantes IVA
- Atributos Importe Base, fecha, impuestos, Importe bruto, estado (pagada o pendiente)
- Métodos: imprime

Variables

- Estas propiedades
- Las variables siguen la misma regla de construcción que vistas anteriormente.
- Las propiedades de la clase al igual que los métodos pueden especificárseles una determinada #visibilidad o alcance, siendo el valor por
- También puedes ser #static o estáticas, también conocidas como variables de la clase, si se especifica con la palabra reservada ***#static***
- Dado que php es de tipado dinámico, si dentro de un método de la clase quiero acceder a una propiedad de la misma he de indicarlo us
- En este caso no podremos el ***$*** delante del nombre de la propiedad

```php
<?php
class Propiedades{
     public $propiedad = "rojo";
     public function getPropiedad(){
        echo "\$propiedad ahora es una variable local a método y no tiene valor: --$propiedad--<br/>";
        $propiedad="azul";
        echo "Ahora visualizo el valor de \$propiedad del método:  --$propiedad--<br/>";
        echo "Ahora visualizo el valor de \$propiedad de la clase:  --$this->propiedad--<br/>";
     }
}

$a = new Propiedades();
$a->getPropiedad();
?>
```

Métodos

- Es la forma de especificar el comportamiento de la clase
- Es lo que el objeto va a saber hacer dentro del programa
- Los métodos de detallan usando la palabra reservada ***function***
- En php dentro de la programación orientada a objetos tenemos una serie o tipo de métodos que conviene conocer y que posteriormente

#métodos contructor y destructor

- son el método __construct(..) y __destruct(..)
- Su implementación es libre para cada clase,
- Su invocación es transparente para el programador y se realiza siempre respectivamente al crear el objeto, y cuando este es destruido
- El nombre es __construct y __destruct respectivamente, si bien, la lista de argumentos que pueden recibir, así como las acciones que pu
- Estos métodos que se invocan automáticamente cuando ocurre algo, en php se conocen como métodos mágicos.

#métodos mágicos

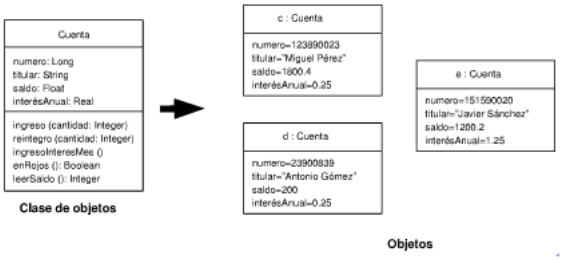
- Una serie de métodos cuyos nombres están reservados y se pueden usar con cualquier objeto de cualquier clase.
- Su nombre siempre empieza por __
- Un ejemplo son el __construct(...) y __destruct(...)

```
http://php.net/manual/es/language.oop5.magic.php
```

Instancias: Operador ***new***

- Una clase describe lo común de unos determinados objetos

- Las clases en principio no se usan durante la ejecución, salvo si queremos acceder a métodos o ṕropiedades estáticas como veremos un
- Lo que se usa son objetos
- Para ello debemos *instanciar* objetos de las clases
- Esto se hace con el operador **new**
- Una vez **instanciado** ya tenemos la referencia del objeto y lo podemos utilizar
- hay que pensar que en memoria tenemos **toda** la estructura del la clase por cada objeto




```
class MiCalse{
    //propiedades
    //métodos
}
$objeto = new Miclase();
```

Actividad

- Crea una clase llamada producto con los atributos código, nombre y precio

Accediendo a los atributos de un objeto

seudovariable $this

- $this es una seudovariable que referenica al objeto del ámbito en el cual está usado
- Se utiliza dentro de la definición de la propia clase y hará referencia a un objeto concreto en un momento dado de la clase en la que est

Ejemplo

```
<?php
class MyClase
{
    function ser_estar()
    {
        if (isset($this)) {
            echo '$this Ahora soy por que estoy';
        } else {
            echo "\$this ni es ni está.\n";
        }
    }
}


</div>
<div class="slide">
{{Actividad|
=Empleado=
Confeccionar una clase Empleado, definir como atributos su nombre y sueldo.
Definir un método inicializar que lleguen como dato el nombre y sueldo. Plantear un segundo método que imprima el nombre y un mensaje si debe o no pagar impuestos

<source lang=php>
<html>
<head>
<title>Pruebas</title>
</head>
<body>
<?php
class Empleado {
  private $nombre;
  private $sueldo;

  public function asigna($nom,$sue)
  {
    $this->nombre=$nom;
    $this->sueldo=$sue;
  }
  public function pagaImpuestos()
  {
    echo $this->nombre;
    echo '-';
    if ($this->sueldo>3000)
      echo 'Debe pagar impuestos';
    else
      echo 'No paga impuestos';
    echo '<br>';
  }
}

$empleado1=new Empleado();
$empleado1->inicializar('Luis',2500);
$empleado1->pagaImpuestos();
$empleado1=new Empleado();
$empleado1->inicializar('Carla',4300);
$empleado1->pagaImpuestos();
?>
</body>
</html>
```

Actividades

Actividad

Confeccionar una clase Menu. Permitir añadir la cantidad de opciones que necesitemos. Mostrar el menú en forma horizontal o ve

```
<html>
<head>
<title>Pruebas</title>
</head>
<body>
<?php
class Menu {
  private $enlaces=array();
  private $titulos=array();
  public function cargarOpcion($en,$tit)
  {
    $this->enlaces[]=$en;
    $this->titulos[]=$tit;
  }
  public function mostrarHorizontal()
  {
    for($f=0;$f<count($this->enlaces);$f++)
    {
      echo '<a href="'.$this->enlaces[$f].'">'.$this->titulos[$f].'</a>';
      echo "-";
    }
  }
  public function mostrarVertical()
  {
    for($f=0;$f<count($this->enlaces);$f++)
    {
      echo '<a href="'.$this->enlaces[$f].'">'.$this->titulos[$f].'</a>';
      echo "<br>";
    }
  }
}

$menu1=new Menu();
$menu1->cargarOpcion('http://www.google.com','Google');
$menu1->cargarOpcion('http://www.yahoo.com','Yhahoo');
$menu1->cargarOpcion('http://www.msn.com','MSN');
$menu1->mostrarVertical();
?>
</body>
</html>
```

Actividad

Construir una clase llamado racional que podamos inicializar con un string del tipo por ejemplo "8/5"

```
<?php
class racional {
    //put your code here
    private $numRacional;
    public function __construct($cadena) {
        $this->numRacional = $cadena;
    }
    public function visualiza(){
        return $this->numRacional;
    }
}

$a=new racional("8/5");
$b=new racional("6/4");
echo '<br>valor de $a: '. $a->visualiza();
echo '<br>valor de $b: '. $b->visualiza();
?>
```

Sobrecarga

Un concepto muy importante es la sobre carga.

- Este concepto es básico en la programación Orientada a objetos;
- Sin embargo este aspecto en php no es del todo intuitivo. No existe la sobrecarga como la entendemos en otros lenguajes
- No obstante tenemos técnicas para poder simular la sobrecarga.
- Muchas veces es fundamental o al menos cómodo sobrecargar el constructor de la clase.

- Vamos a ver como podemos implementar la sobrecarga en php
- Supongamos que en el ejemplo anterior quisiérmos poder instanciar de la siguiente manera los racionales

```
$a = new racional ("8/5");/*   8/5  */
$b = new racional (5,4);  /*   5/6  */
$c = new racional (5);    /*   5/1  */
$d = new racional ();     /*   1/1   */
```

- Para esta situación tenemos diferentes estrategias. En este tema vamos a analizar 2:

1. sobrecargar en el constructor
2. sobrecargar con ***métodos mágicos***

Sobrecargar en el constructor

- Una sencila es ir viendo de qué tipo son los parámetros
- Vemos que podemos tener 0, 1 o 2 paŕametros
- Por lo tanto la función constructora tendrá que tener 3 parámetros inicializados por si no le pasamos valores

```
public function __construct($cadenaOrNum=null, $den = null) {
 ....
}
```

Actividad

Vamos a simular la sobre carga de la clase racional

```
<?php
/**
 * Description of racional
 *
 * @author manolo
 */
class racional {
    //put your code here
    private $numRacional;
    private $num;
    private $den;
    public function __construct($cadenaOrNum=null, $den = null) {
        if (is_int($cadenaOrNum)){
            if (isset($den))
                $this->racionalNumDen($cadenaOrNum, $den );
            else
                $this->racionalNum($cadenaOrNum );
        }
        else{
            if (isset($cadenaOrNum))
                $this->racionalCadena($cadenaOrNum);
            else
                $this->racionalVacio();
        }
     }

     private function racionalNumDen($num, $den){
         $this->numRacional=$num."/".$den;
         $this->num=$num;
         $this->den=$den;
         $this->convierteNum();
     }
     private function racionalNum($num){
         $this->numRacional=$num."/1";
         $this->num=$num;
         $this->den=1;
         $this->convierteNum();
     }
     private function racionalCadena($cadenaOrNum){
         $this->numRacional=$cadenaOrNum;
         $pos = strpos($cadenaOrNum,"/");

         $num=substr($cadenaOrNum,0,$pos);
         $den=substr($cadenaOrNum,$pos+1);
         $this->num=$num;
         $this->den=$den;
     }
     private function racionalVacio(){
         $this->numRacional="1/1";
         $this->num=1;
         $this->den=1;
     }
     private function convierteNum(){
         settype($this->num, int);
         settype($this->den, int);
     }


    public function visualiza(){
        return $this->numRacional;
    }
    public function valorRacional(){
        return $this->num/$this->den;
    }
    private function simplifica(){

    }
    private function suma(racional $num){

    }

}

$a=new racional("8/5");
$b=new racional("6/4");
echo '<br>racional de $a: '. $a->visualiza();
echo '<br>valor de $a: '. $a->valorRacional();
echo '<br>racional de $b: '. $b->visualiza();
echo '<br>valor de $b: '. $b->valorRacional();
echo"<hr/>";
$a=new racional();
$b=new racional();
echo '<br>racional de $a: '. $a->visualiza();
echo '<br>valor de $a: '. $a->valorRacional();
```

```
echo '<br>racional de $b: '. $b->visualiza();
echo '<br>valor de $b: '. $b->valorRacional();
echo"<hr/>";
$a=new racional(8);
$b=new racional(4);
echo '<br>racional de $a: '. $a->visualiza();
echo '<br>valor de $a: '. $a->valorRacional();
echo '<br>racional de $b: '. $b->visualiza();
echo '<br>valor de $b: '. $b->valorRacional();
echo"<hr/>";
$a=new racional(124,6);
$b=new racional(7,123);
echo '<br>racional de $a: '. $a->visualiza();
echo '<br>valor de $a: '. $a->valorRacional();
echo '<br>racional de $b: '. $b->visualiza();
echo '<br>valor de $b: '. $b->valorRacional();

?>
```

- Código implementado en clase

Archivo:M.pdf