Ajuntament de Barcelona

Desenvolupament web amb PHP
Conceptes clau de POO (II)

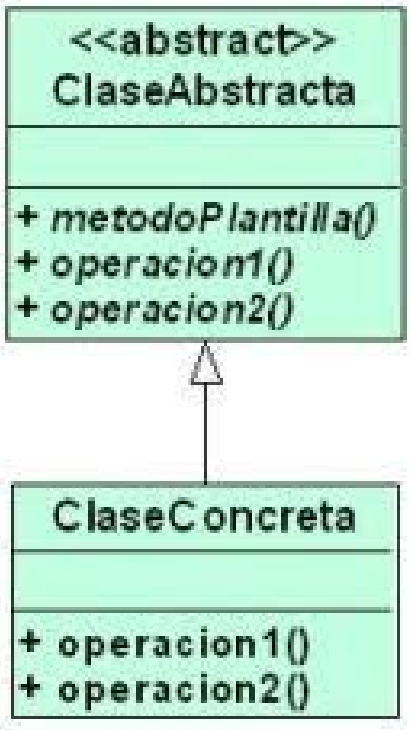
IT Academy

Desembre de 2020

Barcelona Activa

Unió Europea
Fons Europeu de Desenvolupament Regional

POO: Classes Abstractes i concretes



Una classe abstracta té per objectiu agrupar atributs i mètodes que després seran heretats per altres subclasses.

No es pot instanciar

POO: Classes Abstractes i concretes

La *keyword* **abstract** ens serveix per programar una classe abstracta en PHP

```php
<?php
//Clase abstracta padre
abstract class Operacion {
  //Atributos de clase
  protected $valor1;
  protected $valor2;
  protected $resultado;
  //Metodos de clase
  public function cargar1($v)
  {
    $this->valor1=$v;
  }
  public function cargar2($v)
  {
    $this->valor2=$v;
  }
  public function imprimirResultado()
  {
    echo $this->resultado.'<br>';
  }
}
```

POO: Classes Abstractes i concretes

```php
//Clase hija
class Suma extends Operacion{
  //Metodos de clase
  public function operar()
  {
    $this->resultado=$this->valor1+$this->valor2;
  }
}
//Clase hija
class Resta extends Operacion{
  //Metodos de clase
  public function operar()
  {
    $this->resultado=$this->valor1-$this->valor2;
  }
}
```

```php
//Creacion e inicializacion de objetos
$suma=new Suma();
$suma->cargar1(10);
$suma->cargar2(10);
$suma->operar();
echo 'El resultado de la suma de 10+10 es:';
$suma->imprimirResultado();
$resta=new Resta();
$resta->cargar1(10);
$resta->cargar2(5);
$resta->operar();
echo 'El resultado de la diferencia de 10-5 es:';
$resta->imprimirResultado();

?>
```

Que **Operacion** sigui una classe **abstracta** no ens evita que les classes que hereden d'ella funcionin com en exemples anteriors. Així doncs, aquí tindriem que **Suma** i **Resta** son **concrecions** de la **classe Operacion**

POO: Mètodes abstractes

Ara bé, si volem que les subclasses implementin **obligatoriament** determinats comportaments(mètodes) podem definir aquests **mètodes** com a **abstractes**.

Un **mètode abstracte** es declara en una classe però **no s'implementa**.

```
public abstract function nombreFuncion();
```

POO: Mètodes abstractes

Keyword **abstract**

```php
<?php
//Clase abstracta padre
abstract class Operacion {
  //Atributos de clase
  protected $valor1;
  protected $valor2;
  protected $resultado;
  //Metodos de clase
  public function cargar1($v)
  {
    $this->valor1=$v;
  }
  public function cargar2($v)
  {
    $this->valor2=$v;
  }
  public function imprimirResultado()
  {
    echo $this->resultado.'<br>';
  }
  public abstract function operar();
}
```

POO: Mètodes abstractes

```php
//Clase hija
class Suma extends Operacion{
  //Metodos de clase
  public function operar()
  {
    $this->resultado=$this->valor1+$this->valor2;
  }
}
//Clase hija
class Resta extends Operacion{
  //Metodos de clase
  public function operar()
  {
    $this->resultado=$this->valor1-$this->valor2;
  }
}
```

```php
//Creacion e inicializacion de objetos
$suma=new Suma();
$suma->cargar1(10);
$suma->cargar2(10);
$suma->operar();
echo 'El resultado de la suma de 10+10 es:';
$suma->imprimirResultado();
$resta=new Resta();
$resta->cargar1(10);
$resta->cargar2(5);
$resta->operar();
echo 'El resultado de la diferencia de 10-5 es:';
$resta->imprimirResultado();

?>
```

Tot funcionarà exactament igual amb la diferència de que,en aquest cas, **si no implementem** a les classes Suma i Resta el mètode **operar()** es produirà un **error**

POO: Interfícies

Una **interfície** és una col.leció de mètodes definits(no implementats) i que pot contenir també **valors constants**.

Les interfícies més que heredarse, més aviat **s'implementen**, lo qual implicarà en la classe on sigui utilitzada, la **obligatòria implementació** dels mètodes definits a la interfície, **podent ser les implementacions diferents** entre **diferents classes**.

Keyword **interface** per a definir interfícies

```
interface Barco {

}
```

POO: Interfícies – Exemple Vaixell(Barco)

```php
interface Barco {  //una interfaz solo puede tener métodos públicos
   function hundirse();
   function atracar();
   function desembarcar();
}
```

Per implementar una interfície a una classe, ho podem fer tal que així.

```php
class HidroAvion implements Barco {
```

POO: Interfícies – Exemple Vaixell(Barco)

```
class HidroAvion implements Barco, Avion {
```

Una classe pot implementar més d'una interfície**,** separant-les amb comes.

La idea de les **interficies** és que podem tenir mètodes comuns a classes que en principi no tenen una relació de **pare-filla**. Es podría decir que és como si fossin classes cosines.

Altra idea serie entendre les **interfícies** com a elements que "només" defineixen **comportaments**.

POO: Interfícies – Exemple Vaixell(Barco)

```php
<?php
interface Barco {  //una interfaz solo puede tener métodos públicos
    function hundirse();
    function atracar();
    function desembarcar();
}
interface Avion {
    function despegar();
    function aterrizar();
}
class HidroAvion implements Barco, Avion {
    public function aterrizar() {
    }
    public function atracar() {
    }
    public function desembarcar() {
    }
    public function despegar() {
    }
    public function hundirse() {
    }
}
$hidro = new HidroAvion();
?>
```

POO: Mètodes i classes finals

Si a un **mètode** li afegim la *keyword* "**final**" significa que **cap subclasse el pot sobreescriure.** També podriem aplicar aquest modificador a una **classe**, indicant, llavors, que aquesta classe **no es pot heretar**.

```php
<?php
//Clase padre
class Operacion {
  //Atributos de clase
  protected $valor1;
  protected $valor2;
  protected $resultado;
  //Constructor de clase
  public function __construct($v1,$v2)
  {
    $this->valor1=$v1;
    $this->valor2=$v2;
  }
  //Metodos de clase
  public final function imprimirResultado()
  {
    echo $this->resultado.'<br>';
  }
}
```

POO: Mètodes i classes finals

```php
//Clase hija final
final class Suma extends Operacion{
  //Atributos de clase
  private $titulo;
  //Constructor de clase
  public function __construct($v1,$v2,$tit)
  {
    Operacion::__construct($v1,$v2);
    $this->titulo=$tit;
  }
  //Metodos de clase
  public function operar()
  {
    echo $this->titulo;
    echo $this->valor1.'+'.$this->valor2.' es ';
    $this->resultado=$this->valor1+$this->valor2;
  }
}
```

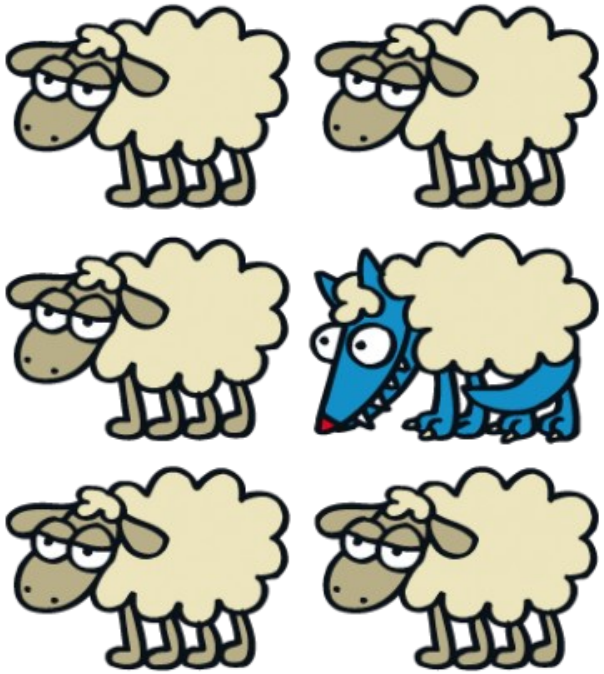
POO: Mètodes i classes finals

```
398  //Creacion e inicializacion de objetos
399  $suma=new Suma(10,10,'Suma de valores:');
400  $suma->operar();
401  $suma->imprimirResultado();
402
403  ?>
```

POO: Referència i clonació d'objectes

Quan assignem una variable de tipus objecte a altra variable, lo que estem fent és guardar **referència de l'objecte. No s'està creant un nou objecte**, sino una altra variable mitjançant la qual, podriem accedir al mateix objecte.

Si volem clonar un objecte idèntic hem d'utilitzar l'operador **clone.**



POO: Referència i clonació d'objectes

```php
<?php
//Definicion de Clase
class Persona {
  //Atributos de clase
  private $nombre;
  private $edad;
  //Metodos de clase
  public function fijarNombreEdad($nom,$ed)
  {
    $this->nombre=$nom;
    $this->edad=$ed;
  }
  public function retornarNombre()
  {
    return $this->nombre;
  }
  public function retornarEdad()
  {
    return $this->edad;
  }
}
```

POO: Referència i clonació d'objectes

```php
//Creacion e inicializacion de objetos
$personal=new Persona();
$personal->fijarNombreEdad('Juan',20);
$x=$personal;
echo 'Datos de la persona ($personal):';
echo $personal->retornarNombre().' - '.$personal->retornarEdad().'<br>';
echo 'Datos de la persona ($x):';
echo $personal->retornarNombre().' - '.$personal->retornarEdad().'<br>';
$x->fijarNombreEdad('Ana',25);
echo 'Después de modificar los datos<br>';
echo 'Datos de la persona ($personal):';
echo $personal->retornarNombre().' - '.$personal->retornarEdad().'<br>';
echo 'Datos de la persona ($x):';
echo $personal->retornarNombre().' - '.$personal->retornarEdad().'<br>';
$persona2=clone($personal);
$personal->fijarNombreEdad('Luis',50);
echo 'Después de modificar los datos de personal<br>';
echo 'Datos de la persona ($personal):';
echo $personal->retornarNombre().' - '.$personal->retornarEdad().'<br>';
echo 'Datos de la persona ($persona2):';
echo $persona2->retornarNombre().' - '.$persona2->retornarEdad().'<br>';

?>
```

POO: Funció clone()

PHP ens permet crear un mètode que es cridarà quan executem l'operador **clone**. Aquest mètode pot entre altres coses, inicialitzar alguns atributs.

Si no es defineix el mètode __clone, es farà una còpia idèntica de l'objecte que li passem com a paràmetre a l'operador clone.

```
public function __clone()
 {
    $this->atributo=0;
 }
```

POO: Funció clone()

```php
<?php
////Definicion de Clase
class Persona {
  //Atributos de clase
  private $nombre;
  private $edad;
  //Metodos de clase
  public function fijarNombreEdad($nom,$ed)
  {
    $this->nombre=$nom;
    $this->edad=$ed;
  }
  public function retornarNombre()
  {
    return $this->nombre;
  }
  public function retornarEdad()
  {
    return $this->edad;
  }
  public function __clone()
  {
    $this->edad=0;
  }
}
```

POO: Interfícies – Funció clone()

```
491  //Creacion e inicializacion de objetos
492  $personal=new Persona();
493  $personal->fijarNombreEdad('Juan',20);
494  echo 'Datos de $personal:';
495  echo $personal->retornarNombre().' - '.$personal->retornarEdad().'<br>';
496  $persona2=clone($personal);
497  echo 'Datos de $persona2:';
498  echo $persona2->retornarNombre().' - '.$persona2->retornarEdad().'<br>';
499
500  ?>
```

POO: Operador instanceof

Quan tenim una llista d'objectes de diferents tipus i volem saber **si un objecte és d'una determinada** classe, a PHP tenim l'operador **instanceof.**

```
if ($vec instanceof Gerente)
    echo 'Los objetos son del mismo tipo';
}
```

POO: Operador instanceof

```php
<?php
//Clase abstracta padre
abstract class Trabajador {
  //Atributos de clase
  protected $nombre;
  protected $sueldo;
  //Constructor de clase
  public function __construct($nom,$sue)
  {
    $this->nombre=$nom;
    $this->sueldo=$sue;
  }
  //Metodos de clase
  public function retornarSueldo()
  {
    return $this->sueldo;
  }
}
```

POO: Operador instanceof

```
528  //Clase hija
529  class Empleado extends Trabajador {
530  }
531  //Clase hija
532  class Gerente extends Trabajador {
533  }
```

POO: Operador instanceof

```php
//Creacion e inicializacion de objetos
$vec[]=new Empleado('juan',1200);
$vec[]=new Empleado('ana',1000);
$vec[]=new Empleado('carlos',1000);

$vec[]=new Gerente('jorge',25000);
$vec[]=new Gerente('marcos',8000);

$sumal=0;
$suma2=0;
for($f=0;$f<count($vec);$f++)
{
  if ($vec[$f] instanceof Empleado)
    $sumal=$sumal+$vec[$f]->retornarSueldo();
  else
    if ($vec[$f] instanceof Gerente)
      $suma2=$suma2+$vec[$f]->retornarSueldo();
}
echo 'Gastos en sueldos de Empleados:'.$sumal.'<br>';
echo 'Gastos en sueldos de Gerentes:'.$suma2.'<br>';
```

POO: Mètode _destruct()

•El seu objectiu principal és **alliberar recursos** que l'objecte va sol.licitar com per exemple: connexió a bases de dades, creació d'imatges dinàmiques…

• És l'últim mètode que s'executa de la classe.

•S'**executa de manera automática**, és a dir, que no l'hem de cridar.

• S'ha d'anomenar __**destruct**.

• **No retorna** dades.

POO: Mètode _destruct()

```php
<?php
////Definicion de Clase
class Banner {
  //Atributos de clase
  private $ancho;
  private $alto;
  private $mensaje;
  private $imagen;
  private $colorTexto;
  private $colorFondo;
  //Constructor de clase
  public function __construct($an,$al,$men)
  {
    $this->ancho=$an;
    $this->alto=$al;
    $this->mensaje=$men;
    $this->imagen=imageCreate($this->ancho,$this->alto);
    $this->colorTexto=imageColorAllocate($this->imagen,255,255,0);
    $this->colorFondo=imageColorAllocate($this->imagen,255,0,0);
    imageFill($this->imagen,0,0,$this->colorFondo);
  }
```

POO: Mètode _destruct()

```
581     //Metodos de clase
582     public function graficar()
583     {
584       imageString ($this->imagen,5,50,10, $this->mensaje,$this->colorFuente);
585       header ("Content-type: image/png");
586       imagePNG ($this->imagen);
587     }
588     public function __destruct()
589     {
590       imageDestroy($this->imagen);
591     }
592 }
```

POO: Mètode _destruct()

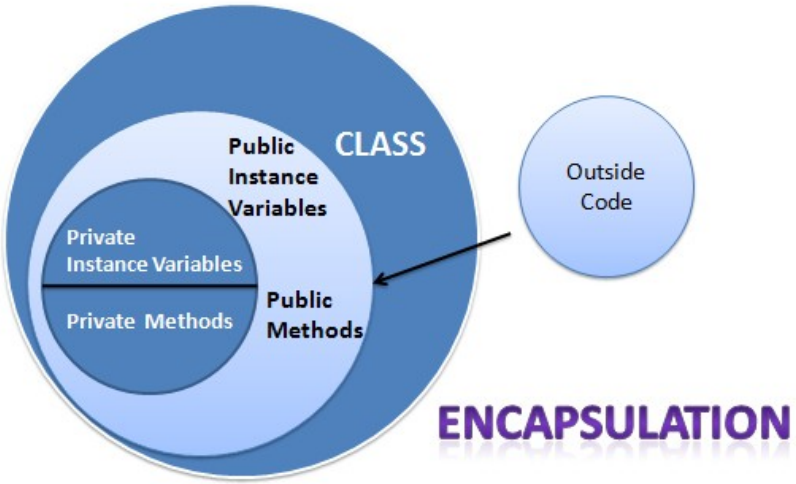
```
593  //Creacion e inicializacion de objetos
594  $baner1=new Banner(428,45,'Sistema de Ventas por Mayor y Menor');
595  $baner1->graficar();
596
597  ?>
```

POO: Mètodes estàtics

Un mètode estàtic pertany a la classe però no pot accedir als atributs d'una instància.

La característica fonamental és que un mètode estàtic es pot cridar sense haver d'instanciar la classe.

Un mètode estàtic és lo més semblant a una funció de llenguatge estructurat, només que, s'encapsula a dins d'una classe.



POO: Mètodes estàtics

```php
<?php
//Definicion de Clase
class Cadena {
  //Metodos de clase
  public static function largo($cad)
  {
    return strlen($cad);
  }
  public static function mayusculas($cad)
  {
    return strtoupper($cad);
  }
  public static function minusculas($cad)
  {
    return strtolower($cad);
  }
}
```

POO: Mètodes estàtics

```
624  //Creacion e inicializacion de objetos
625  $c='Hola';
626  echo 'Cadena original:'.$c;
627  echo '<br>';
628  echo 'Largo:'.Cadena::largo($c);
629  echo '<br>';
630  echo 'Toda en mayúsculas:'.Cadena::mayusculas($c);
631  echo '<br>';
632  echo 'Toda en minúsculas:'.Cadena::minusculas($c);
633
634  ?>
```

barcelona.cat/barcelonactiva