Ajuntament de Barcelona

Desenvolupament web amb PHP
Paradigma de la orientació a objectes

IT Academy

Desembre de 2020

Barcelona Activa

Unió Europea
Fons Europeu de Desenvolupament Regional

POO: Definició

La programació orientada a objectes és un paradigma de la programació que descomposa el codi d'una **classe** en unitats anomenades **objectes**.

Un **objecte** en PHP és una entitat existent en la memoria d'un computador que té unes **propietats** anomenades **atributs** i unes **funcions associades** anomenades **mètodes**.

Aquests **objectes** venen de **clases.** En realitat,fins ara hem utilitzat una classe, només tenía un únic mètode implícit executable.

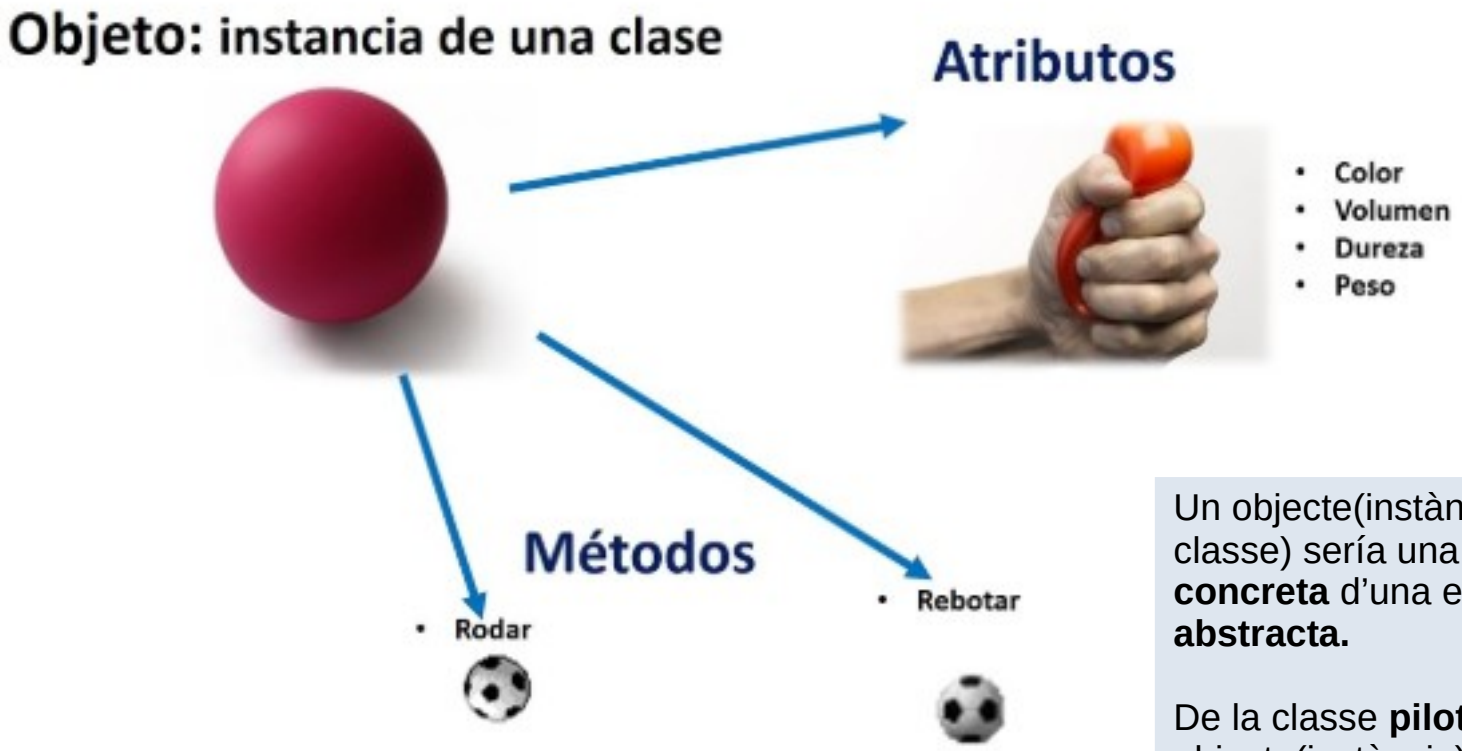
POO: Definició

Aquest paradigme és senzill d'entendre però la seva profunditat i efecte alhora de programar és ENORME.

Bàsicament perquè un objecte en aquest context pot ser QUALSEVOL representació d'una entitat en el món real.

I és per això que la presa de decisions alhora de generar un o més objectes per resoldre un problema concret del món real serà DECISIVA ja que marcarà l'estructura(inclús l'arquitectura) del programa que fem.

POO: Exemple amb l'entitat "Pilota"



Un objecte(instància d'una classe) sería una representació **concreta** d'una entitat **abstracta.**

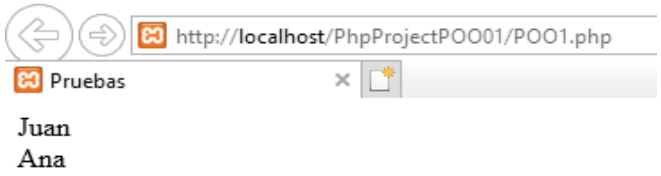
De la classe **pilota** tenim un objecte(instància) que representa una pilota concreta.

POO: Exemple entitat “Persona”

```php
<html>
<head>
<title>Pruebas</title>
</head>
<body>
<?php
    class Persona {

        //Atributos de clase
        private $nombre;

        //Metodos de clase
        public function inicializar($nom)
        {
            $this->nombre=$nom;
        }
            public function imprimir()
        {
            echo $this->nombre;
            echo '<br>';
        }
    }
    //Creacion y asignacion de objetos
    $per1=new Persona();
    $per1->inicializar('Juan');
    $per1->imprimir();
    $per2=new Persona();
    $per2->inicializar('Ana');
    $per2->imprimir();
?>
</body>
</html>
```



Aqui creem una clase que representa informació relativa a una Persona. Tenim un atribut **nombre** i uns mètodes per inicialitzar l’objecte i per imprimir el nom.

Instanciem(creem un objecte) de la classe Persona

La inicialitzem,en aquest cas, li donem nom.

Imprimim el nom d’aquest persona concreta

Mateix procés però amb diferent persona !

POO: Implementació classe persona en PHP

```php
<?php
    class Persona {

        //Atributos de clase
        private $nombre;

        //Metodos de clase
        public function inicializar($nom)
        {
            $this->nombre=$nom;
        }
            public function imprimir()
        {
            echo $this->nombre;
            echo '<br>';
        }
    }
```

Aquesta és “només” una idea de classe persona. Les implementacions dependran de les necessitats del software

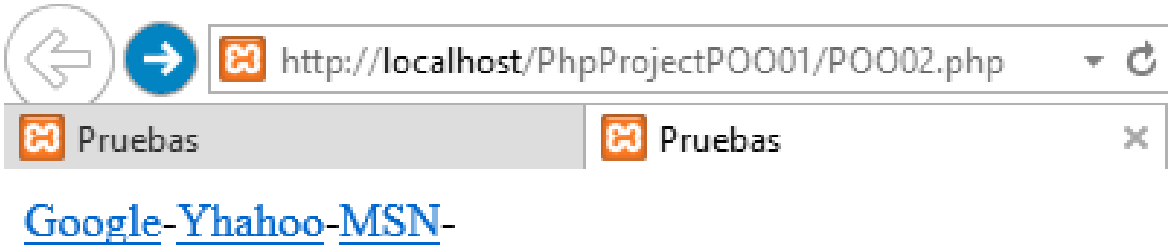
POO: Instanciant la classe persona

```
26      //Creacion y asignacion de objetos
27      $per1=new Persona();
28      $per1->inicializar('Juan');
29      $per1->imprimir();
30      $per2=new Persona();
31      $per2->inicializar('Ana');
32      $per2->imprimir();
33  ?>
34  </body>
35  </html>
```

Aqui fem ús de la implementació de la classe Persona per a fer una representació de dues persones concretes.

POO: Exemple “Navbar”

Implementar una classe que mostri una llista d’hipervincles en horitzontal(bàsicament un menú d’opcions)



POO: Exemple “Navbar”

```php
<?php
    class Menu {
        //Atributos de clase
        private $enlaces=array();
        private $titulos=array();

        //Metodos de clase
        public function cargarOpcion($en,$tit)
          {
            $this->enlaces[]=$en;
            $this->titulos[]=$tit;
          }
        public function mostrar()
          {
            for($f=0;$f<count($this->enlaces);$f++)
            {
              echo '<a href="'.$this->enlaces[$f].'">'.$this->titulos[$f].'</a>';
              echo "-";
            }
          }
    }
    //Creacion y asignacion de objetos
    $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->mostrar();

?>
```

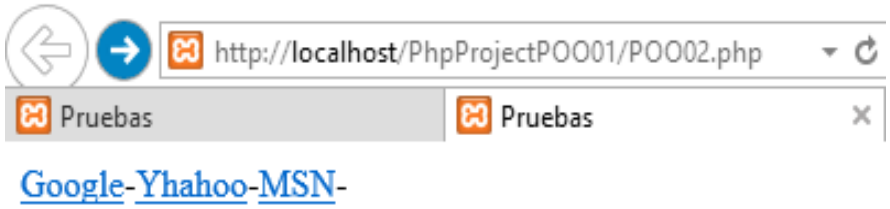
POO: Exemple “Navbar”

```php
<?php
    class Menu {
        //Atributos de clase
        private $enlaces=array();
        private $titulos=array();

        //Metodos de clase
        public function cargarOpcion($en,$tit)
          {
            $this->enlaces[]=$en;
            $this->titulos[]=$tit;
          }
        public function mostrar()
          {
            for($f=0;$f<count($this->enlaces);$f++)
            {
              echo '<a href="'.$this->enlaces[$f].'">'.$this->titulos[$f].'</a>';
              echo "-";
            }
          }
    }
```

POO: Exemple “Navbar”

```
66      //Creacion y asignacion de objetos
67      $menu1=new Menu();
68      $menu1->cargarOpcion('http://www.google.com','Google');
69      $menu1->cargarOpcion('http://www.yahoo.com','Yhahoo');
70      $menu1->cargarOpcion('http://www.msn.com','MSN');
71      $menu1->mostrar();
72
73  ?>
74  </body>
75  </html>
```



POO: Exemple *"Header* alineat"

Implementar una classe "CabeceraPagina" que permeti mostrar un títol, e indicar si el volem alineat al centre, a l'esquerra o a la dreta.

POO: Exemple "*Header* alineat"

```
<html>
    <head>
        <title>Pruebas</title>
    </head>
    <body>
    <?php
    class CabeceraPagina {
        //Atributos de clase
        private $titulo;
        private $ubicacion;
        //Metodos de clase
        public function inicializar($tit,$ubi)
          {
            $this->titulo=$tit;
            $this->ubicacion=$ubi;
          }
          public function graficar()
          {
            echo '<div style="font-size:40px;text-align:'.$this->ubicacion.'">';
            echo $this->titulo;
            echo '</div>';
          }
    }
    //Creacion y asignacion de objetos
    $cabecera=new CabeceraPagina();
    $cabecera->inicializar('El blog del programador','center');
    $cabecera->graficar();
    ?>
    </body>
</html>
```

POO: Exemple "*Header* alineat"

```php
<html>
    <head>
        <title>Pruebas</title>
    </head>
    <body>
    <?php
    class CabeceraPagina {
        //Atributos de clase
        private $titulo;
        private $ubicacion;
        //Metodos de clase
        public function inicializar($tit,$ubi)
          {
            $this->titulo=$tit;
            $this->ubicacion=$ubi;
          }
          public function graficar()
          {
            echo '<div style="font-size:40px;text-align:'.$this->ubicacion.'">';
            echo $this->titulo;
            echo '</div>';
          }
    }
```

POO: Exemple "*Header* alineat"


```
102        //Creacion y asignacion de objetos
103        $cabecera=new CabeceraPagina();
104        $cabecera->inicializar('El blog del programador','center');
105        $cabecera->graficar();
106        ?>
107        </body>
108    </html>
```

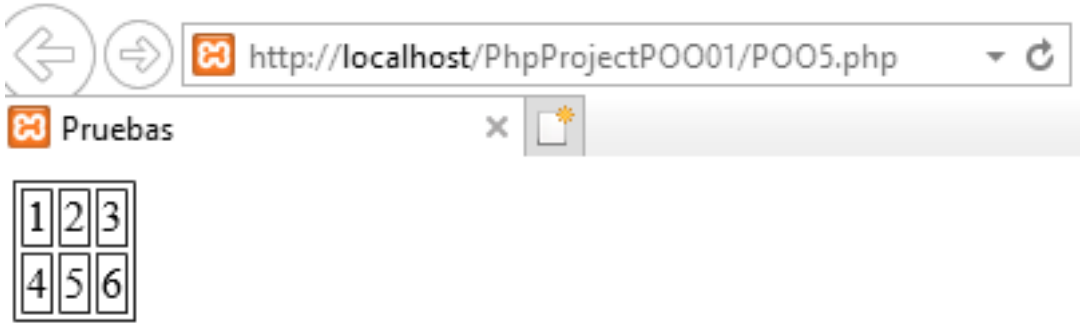
POO: Exemple *"Header* alineat" (amb constructor)

Un mètode típic a les classes és el **constructor,** que seria el mètode per defecte encarregat de definir e inicialitzar un objecte d'aquesta classe. Aquí ho aplicarem a l'exemple anterior.

POO: Exemple "*Header* alineat" (amb constructor)

```php
<?php
class CabeceraPagina {
    //Atributos de clase
    private $titulo;
    private $ubicacion;
    //Metodos de clase
    public function __construct($tit,$ubi)
      {
        $this->titulo=$tit;
        $this->ubicacion=$ubi;
      }
    public function graficar()
      {
        echo '<div style="font-size:40px;text-align:'.$this->ubicacion.'">';
        echo $this->titulo;
        echo '</div>';
      }
}
    //Creacion y asignacion de objetos
    $cabecera=new CabeceraPagina('El blog del programador','center');
    $cabecera->graficar();
?>
```

POO: Exemple “Taula”



Implementar una classe Taula que permeti indicar en el constructor la quantitat de files i columnes. Definir altre mètode que ens deixi carregar una dada concreta en una determinada cel.la. Finalment, mostrar el resultat per pantalla.

POO: Exemple “Taula”

```
219 <html>
220     <head>
221         <title>Pruebas</title>
222     </head>
223     <body>
224     <?php
225     //definicion clase objeto
226     class Tabla {
227
228         //Atributos de clase
229         private $mat=array();
230         private $cantFilas;
231         private $cantColumnas;
232
233         //Metodos de clase
234         public function __construct($fi,$co)
235           {
236             $this->cantFilas=$fi;
237             $this->cantColumnas=$co;
238           }
239
```

POO: Exemple “Taula”

```
233         //Metodos de clase
234         public function __construct($fi,$co)
235           {
236             $this->cantFilas=$fi;
237             $this->cantColumnas=$co;
238           }
239
240         public function cargar($fila,$columna,$valor)
241           {
242             $this->mat[$fila][$columna]=$valor;
243           }
244
245         private function inicioTabla()
246           {
247             echo '<table border="1">';
248           }
249
250         private function inicioFila()
251           {
252             echo '<tr>';
253           }
```

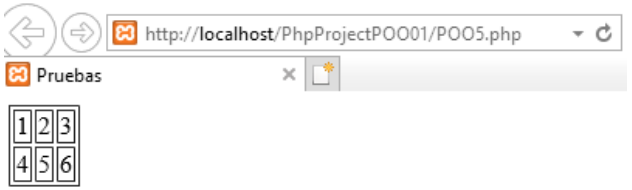
POO: Exemple “Taula”

```
254
255         private function mostrar($fi,$co)
256           {
257             echo '<td>'.$this->mat[$fi][$co].'</td>';
258           }
259
260         private function finFila()
261           {
262             echo '</tr>';
263           }
264
265         private function finTabla()
266           {
267             echo '</table>';
268           }
269
```

POO: Exemple “Taula”

```
270         public function graficar()
271           {
272             $this->inicioTabla();
273             for($f=1;$f<=$this->cantFilas;$f++)
274             {
275               $this->inicioFila();
276               for($c=1;$c<=$this->cantColumnas;$c++)
277               {
278                 $this->mostrar($f,$c);
279               }
280               $this->finFila();
281             }
282             $this->finTabla();
283           }
284     }
```

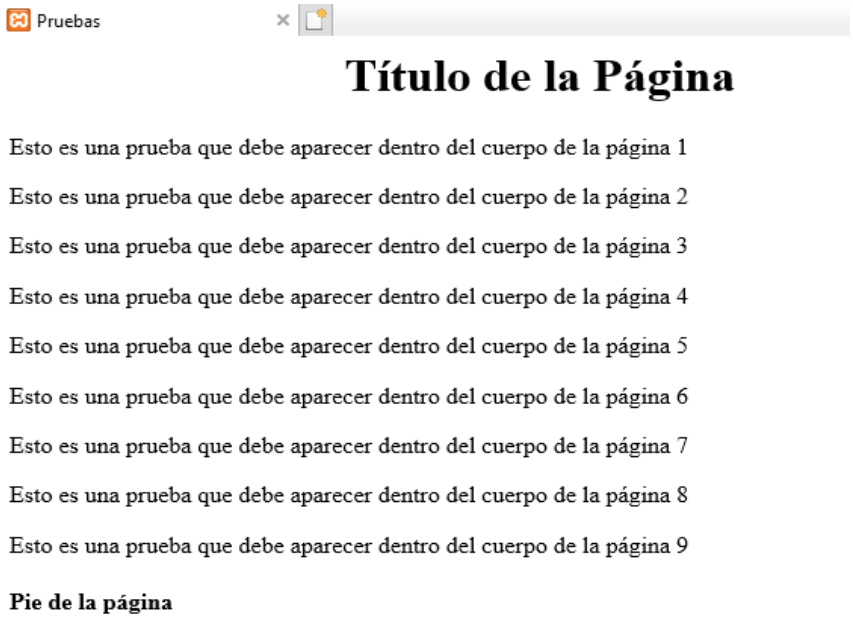
POO: Exemple “Taula”




```
285         //Creacion y asignacion de objetos
286         $tabla1=new Tabla(2,3);
287         $tabla1->cargar(1,1,"1");
288         $tabla1->cargar(1,2,"2");
289         $tabla1->cargar(1,3,"3");
290         $tabla1->cargar(2,1,"4");
291         $tabla1->cargar(2,2,"5");
292         $tabla1->cargar(2,3,"6");
293         $tabla1->graficar();
294         ?>
295         </body>
296     </html>
```

POO: Exemple “Pàgina”

Els objectes poden establir relacions entre si. De fet, de vegades és imprescindible per a tenir una bona solució a un problema concret.



Plantejar una classe Pagina que inclogui objectes de la classe Capçalera, Cos i Peu. Capçalera i Peu han de tenir un atribut on s’enmagatzemi el text a mostrar. La classe Cos ha de tenir un atributo tipus *array* on s’enmagatzemin tots els paràgrafs.

POO: Exemple “Pàgina”

```
79  <html>
80      <head>
81          <title>Pruebas</title>
82      </head>
83      <body>
84      <?php
85      class CabeceraPagina {
86          //Atributos de clase
87          private $titulo;
88          private $ubicacion;
89          //Metodos de clase
90          public function inicializar($tit,$ubi)
91          {
92            $this->titulo=$tit;
93            $this->ubicacion=$ubi;
94          }
95          public function graficar()
96          {
97            echo '<div style="font-size:40px;text-align:'.$this->ubicacion.'">';
98            echo $this->titulo;
99            echo '</div>';
100         }
101     }
```

POO: Exemple “Pàgina”

```php
//definicion clase objeto
class Cuerpo {
    //Atributos de objeto
    private $lineas=array();
    //Metodos de objeto
    public function insertarParrafo($li)
      {
        $this->lineas[]=$li;
      }
    public function graficar()
      {
        for($f=0;$f<count($this->lineas);$f++)
        {
          echo '<p>'.$this->lineas[$f].'</p>';
        }
      }
}
```

POO: Exemple “Pàgina”

```
339     //definicion clase objeto
340     class Pie {
341         //Atributos de objeto
342         private $titulo;
343         //Constructor de objeto
344         public function __construct($tit)
345           {
346             $this->titulo=$tit;
347           }
348         //Metodos de objeto
349         public function graficar()
350           {
351             echo '<h4 style="text-align:left">'.$this->titulo.'</h4>';
352           }
353     }
```

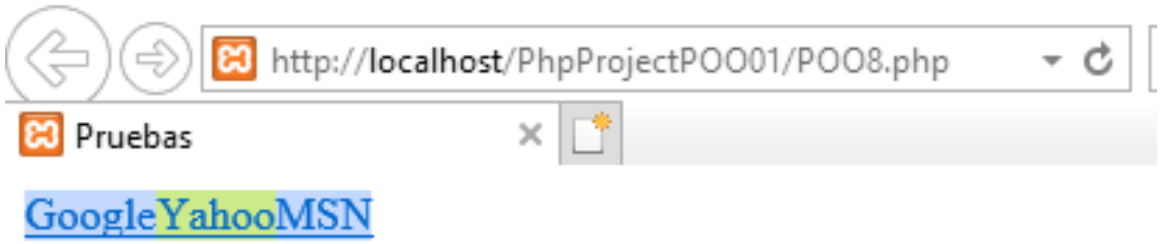
POO: Exemple “Pàgina”

```php
//definicion clase objeto
class Pagina {
    //Atributos de objeto
    private $cabecera;
    private $cuerpo;
    private $pie;
    //Constructor de objeto
    public function __construct($textol,$texto2)
      {
        $this->cabecera=new Cabecera($textol);
        $this->cuerpo=new Cuerpo();
        $this->pie=new Pie($texto2);
      }
    //Metodos de objeto
    public function insertarCuerpo($texto)
      {
        $this->cuerpo->insertarParrafo($texto);
    }
    public function graficar()
      {
        $this->cabecera->graficar();
        $this->cuerpo->graficar();
        $this->pie->graficar();
      }
}
```

POO: Exemple “Pàgina”

```
380     //Creacion y asignacion de objetos
381     $paginal=new Pagina('Título de la Página','Pie de la página');
382     $paginal->insertarCuerpo('Esto es una prueba que debe aparecer dentro del cuerpo de la página 1');
383     $paginal->insertarCuerpo('Esto es una prueba que debe aparecer dentro del cuerpo de la página 2');
384     $paginal->insertarCuerpo('Esto es una prueba que debe aparecer dentro del cuerpo de la página 3');
385     $paginal->insertarCuerpo('Esto es una prueba que debe aparecer dentro del cuerpo de la página 4');
386     $paginal->insertarCuerpo('Esto es una prueba que debe aparecer dentro del cuerpo de la página 5');
387     $paginal->insertarCuerpo('Esto es una prueba que debe aparecer dentro del cuerpo de la página 6');
388     $paginal->insertarCuerpo('Esto es una prueba que debe aparecer dentro del cuerpo de la página 7');
389     $paginal->insertarCuerpo('Esto es una prueba que debe aparecer dentro del cuerpo de la página 8');
390     $paginal->insertarCuerpo('Esto es una prueba que debe aparecer dentro del cuerpo de la página 9');
391     $paginal->graficar();
392     ?>
393     </body>
394 </html>
```

POO: Exemple “Objectes a un menú”



Plantejarem una classe Opcion i una altra classe Menu. La classe Opcion definirà com atribut el títol, l’enllaç i el color de fons.Els mètodes a implementar seran el constructor i el renderitzador/graficador.

Per altra banda, la classe Menú administrarà un *array* d’objectes de la classe Opcion i implementarà mètodes per a inserir objectes Opcion a la classe Menu i un altre per a renderitzar. A més, al constructor de la classe Menú l’indicarem si volem orientació horitzonal o vertical.

POO: Exemple “Objectes a un menú”

```
399 <html>
400     <head>
401         <title>Pruebas</title>
402     </head>
403     <body>
404     <?php
405     //definicion clase objeto
406     class Opcion {
407         //Atributos de objeto
408         private $titulo;
409         private $enlace;
410         private $colorFondo;
411         //Constructor de objeto
412         public function __construct($tit,$enl,$cfon)
413           {
414             $this->titulo=$tit;
415             $this->enlace=$enl;
416             $this->colorFondo=$cfon;
417           }
418         //Metodos de objeto
419         public function graficar()
420           {
421             echo '<a style="background-color:'.$this->colorFondo.'" href="'.$this->enlace.'">'.$this->titulo.'</a>';
422           }
423     }
```

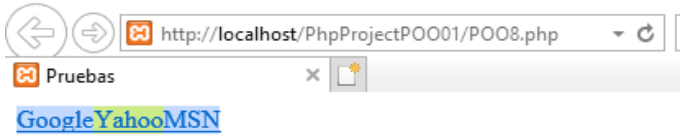
POO: Exemple “Objectes a un menú”

```
424     //definicion clase objeto
425     class Menu {
426         //Atributos de objeto
427         private $opciones=array();
428         private $direccion;
429         //Constructor de objeto
430         public function __construct($dir)
431           {
432             $this->direccion=$dir;
433           }
434         //Metodos de objeto
435         public function insertar($op)
436           {
437             $this->opciones[]=$op;
438           }
439
440         private function graficarHorizontal()
441           {
442             for($f=0;$f<count($this->opciones);$f++)
443             {
444               $this->opciones[$f]->graficar();
445             }
446           }
```

POO: Exemple “Objectes a un menú”

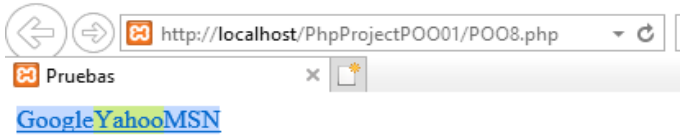
```
447         private function graficarVertical()
448           {
449             for($f=0;$f<count($this->opciones);$f++)
450             {
451               $this->opciones[$f]->graficar();
452               echo '<br>';
453             }
454           }
455         public function graficar()
456           {
457             if (strtolower($this->direccion)=="horizontal")
458               $this->graficarHorizontal();
459             else
460               if (strtolower($this->direccion)=="vertical")
461                 $this->graficarVertical();
462           }
463     }
```

POO: Exemple “Objectes a un menú”




```
464     //Creacion y asignacion de objetos
465     $menu1=new Menu('horizontal');
466     $opcion1=new Opcion('Google','http://www.google.com','#C3D9FF');
467     $menu1->insertar($opcion1);
468     $opcion2=new Opcion('Yahoo','http://www.yahoo.com','#CDEB8B');
469     $menu1->insertar($opcion2);
470     $opcion3=new Opcion('MSN','http://www.msn.com','#C3D9FF');
471     $menu1->insertar($opcion3);
472     $menu1->graficar();
473
474     ?>
475     </body>
476 </html>
```

POO: Exemple “Objectes a un menú”




```
464     //Creacion y asignacion de objetos
465     $menu1=new Menu('horizontal');
466     $opcion1=new Opcion('Google','http://www.google.com','#C3D9FF');
467     $menu1->insertar($opcion1);
468     $opcion2=new Opcion('Yahoo','http://www.yahoo.com','#CDEB8B');
469     $menu1->insertar($opcion2);
470     $opcion3=new Opcion('MSN','http://www.msn.com','#C3D9FF');
471     $menu1->insertar($opcion3);
472     $menu1->graficar();
473
474     ?>
475     </body>
476 </html>
```

POO: Exemple "*Header* configurat"

Desenvolupar la classe CabeceraDePagina, que ens mostrarà un títol alineat amb un determinat color de font i de fons. S'han de definir paràmetres opcionals tant pels colors com per l'alineació del títol.

POO: Exemple "*Header* configurat"

```php
<?php
//definicion clase objeto
class CabeceraPagina {
    //Atributos de objeto
    private $titulo;
    private $ubicacion;
    private $colorFuente;
    private $colorFondo;
    //Constructor de objeto
    public function __construct($tit,$ubi='center',$colorFuen='#ffffff',$colorFon='#000000')
      {
        $this->titulo=$tit;
        $this->ubicacion=$ubi;
        $this->colorFuente=$colorFuen;
        $this->colorFondo=$colorFon;
      }
    //Metodos de objeto
    public function graficar()
      {
        echo '<div style="font-size:40px;text-align:'.$this->ubicacion.';color:';
        echo $this->colorFuente.';background-color:'.$this->colorFondo.'">';
        echo $this->titulo;
        echo '</div>';
      }
}
```

POO: Exemple "*Header* configurat"

```
512     //Creacion y asignacion de objetos
513     $cabeceral=new CabeceraPagina('El blog del programador');
514     $cabeceral->graficar();
515     echo '<br>';
516     $cabecera2=new CabeceraPagina('El blog del programador','left');
517     $cabecera2->graficar();
518     echo '<br>';
519     $cabecera3=new CabeceraPagina('El blog del programador','right','#ff0000');
520     $cabecera3->graficar();
521     echo '<br>';
522     $cabecera4=new CabeceraPagina('El blog del programador','right','#ff0000','#ffff00');
523     $cabecera4->graficar();
524
525     ?>
526     </body>
527 </html>
```

barcelona.cat/barcelonactiva

Ajuntament de Barcelona

Desenvolupament web amb PHP
Conceptes clau de POO (I)

IT Academy

Desembre de 2020

Barcelona Activa

Unió Europea
Fons Europeu de Desenvolupament Regional

POO: Operador “this”

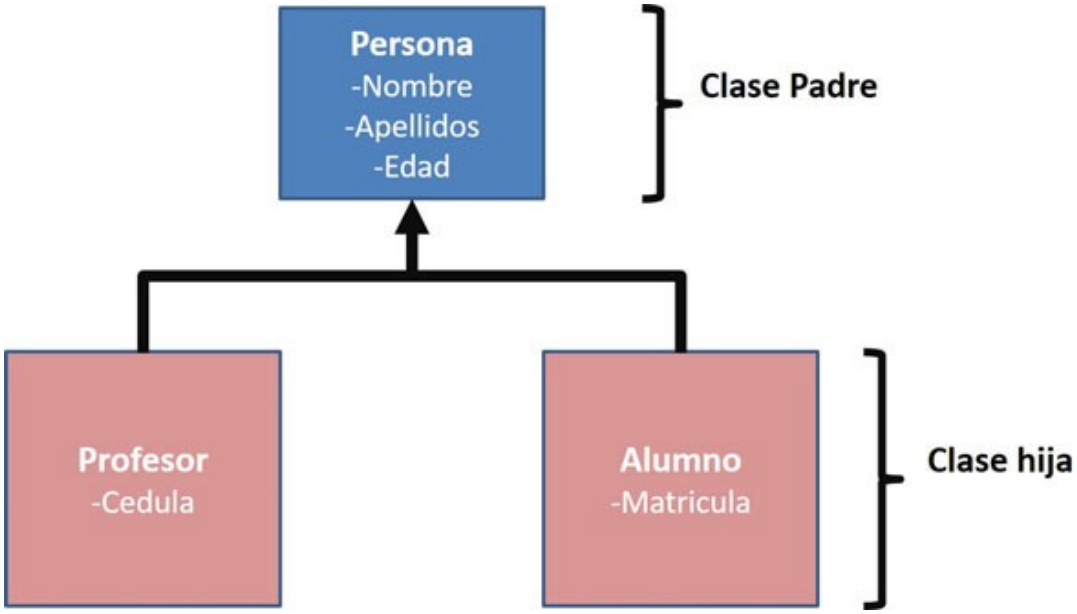
L’operador “this” es una variable especial que **només té sentit dins del context d’un objecte instanciat**. Representa al mateix objecte, i serveix, per tant, per referirse als mètodes i atributs d’ell mateix.

```php
<?php
//Clase Padre
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>';
  }
}
```

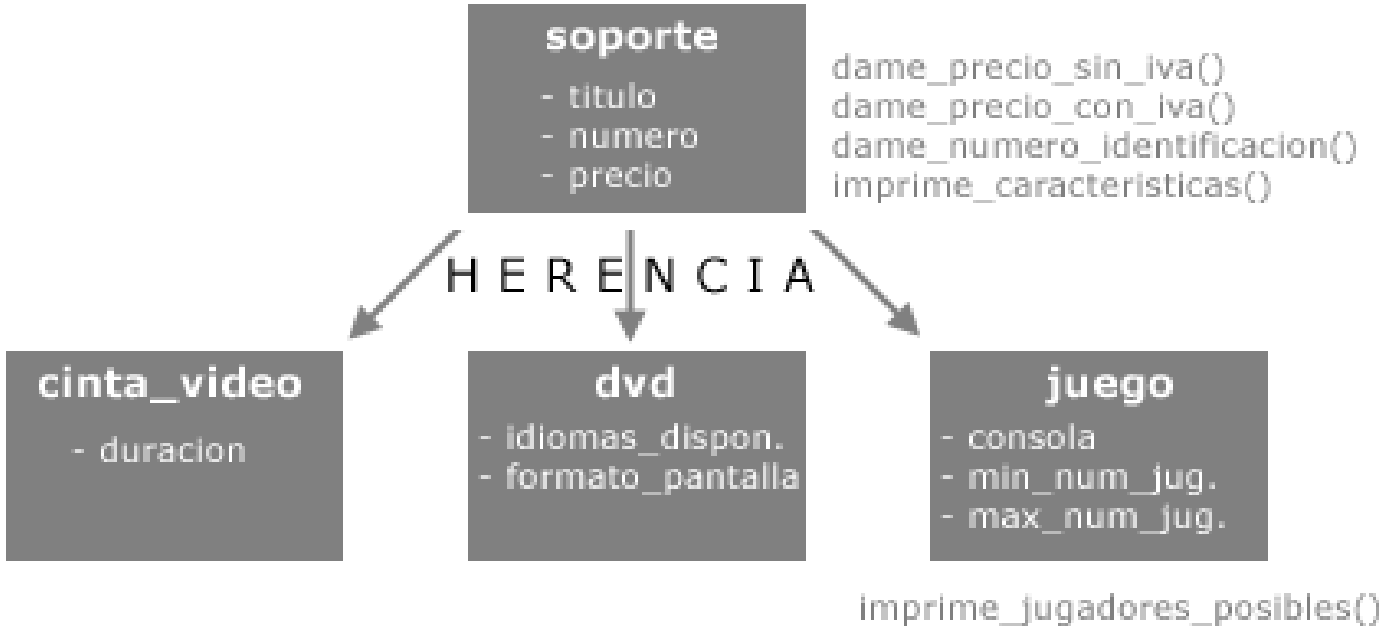
Aquí, per exemple, **$this→valor1** refereix a l’atribut de l’objecte, i li assigna el valor de la variable **$v.** Si la variable que entra pel paràmetre tingués el mateix nom($valor1) no hi hauria conflicte sempre i quan ens referíssim a l’atribut de l’objecte amb $this.

POO: Herència

L'**herència** és un mecanisme de POO mitjançant el qual podem crear **jerarquies** de classes.



POO: Herència



POO: Herència

```php
<?php
//Clase Padre
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>';
  }
}
```

La *keyword* **extends** ens serveix per a indicar de quina classe heretarem

```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;
      }
}
```

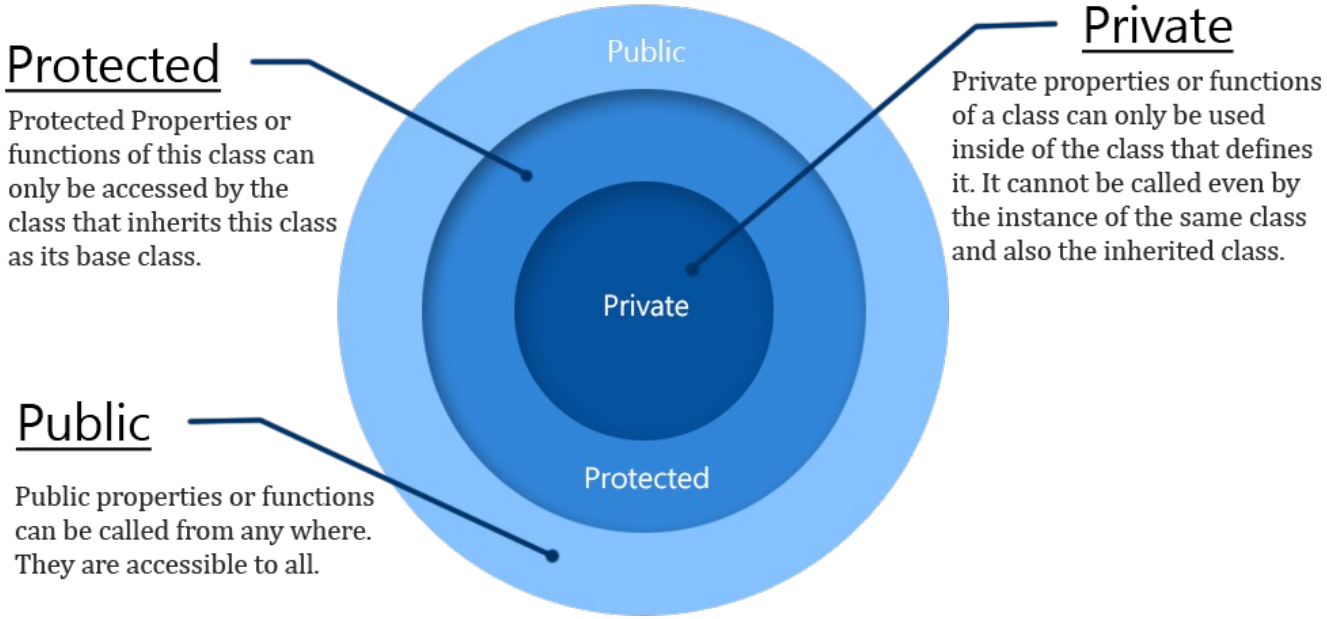
La principal “gràcia” d’aquest mecanisme es que s’hereten els mètodes i atributs depenent de la seva **visibilitat(**terme del qual parlem més endavant). En aquest cas, les classes Suma i Resta hereten $valor1,$valor2,$valor3 i els mètodes cargar1,cargar2 e imprimirResultado, afegint-ho a les seves pròpies característiques.

POO: Herència

```
45      //Creacion e inicializacion de objetos
46      $suma=new Suma();
47      $suma->cargar1(10);
48      $suma->cargar2(10);
49      $suma->operar();
50      echo 'El resultado de la suma de 10+10 es:';
51      $suma->imprimirResultado();
52      $resta=new Resta();
53      $resta->cargar1(10);
54      $resta->cargar2(5);
55      $resta->operar();
56      echo 'El resultado de la diferencia de 10-5 es:';
57      $resta->imprimirResultado();
58
59      ?>
```

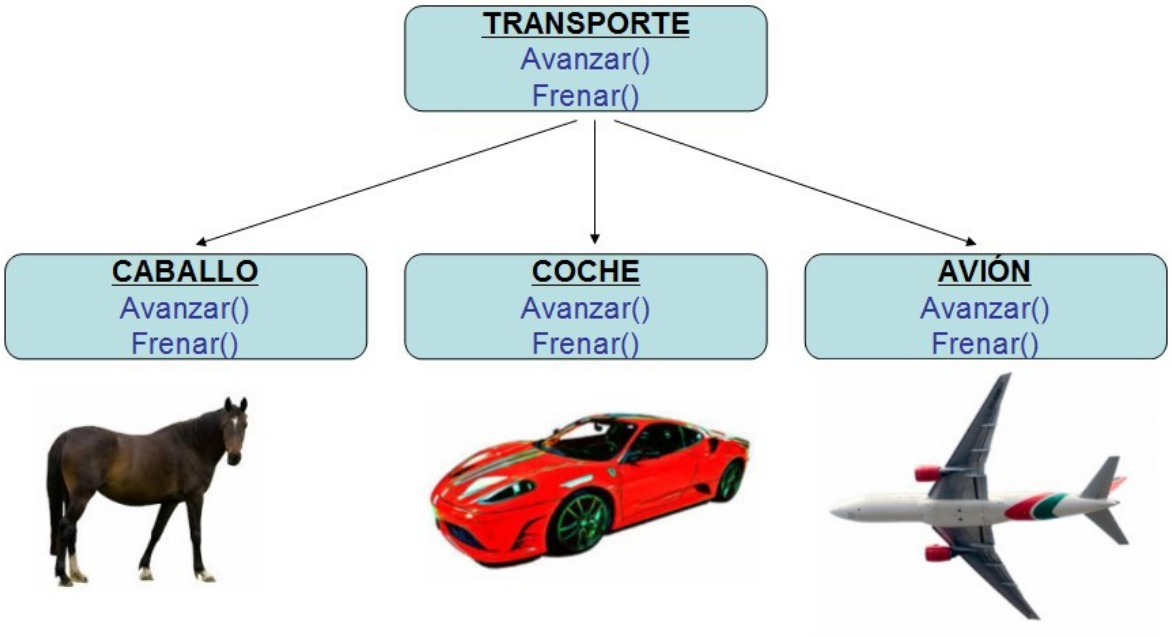
POO: Polimorfisme

La **visibilitat** és un mecanisme de POO que ens permet **limitar l'accés de mètodes o atributs d'una classe**.
Les *keywords* **protected, private** o **public** davant d'un atribut o mètode d'una classe signifiquen, doncs:



POO: Polimorfisme

El **polimorfisme** és un mecanisme de POO mitjançant el qual podem donar varies formes a un mateix objecte, permetent, entre d'altres coses, respondre de manera diferent a crides a funcions del mateix nom. El polimorfisme té sentit gràcies a l'**herència** en POO.



POO: Polimorfisme i sobreescriptura de mètodes

Recuperem l'exemple anterior on la classe "pare" Operacion té com a "filles" les classes Suma y resta

```php
<?php
//Clase Padre
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>';
  }
}
```

```php
//Clase hija
class Suma extends Operacion{
  //Metodos de clase
  public function operar()
  {
    $this->resultado=$this->valor1+$this->valor2;
  }
  public function imprimirResultado()
  {
    echo "La suma de $this->valor1 y $this->valor2 es:";
    parent::imprimirResultado();
  }
}
```

```php
//Clase hija
class Resta extends Operacion{
  //Metodos de clase
  public function operar()
  {
    $this->resultado=$this->valor1-$this->valor2;
  }
  public function imprimirResultado()
  {
    echo "La diferencia de $this->valor1 y $this->valor2 es:";
    parent::imprimirResultado();
  }
}
```

POO: Polimorfisme i sobreescriptura de mètodes

```
164  //Creacion e inicializacion de objetos
165  $suma=new Suma();
166  $suma->cargar1(10);
167  $suma->cargar2(10);
168  $suma->operar();
169  $suma->imprimirResultado();
170  $resta=new Resta();
171  $resta->cargar1(10);
172  $resta->cargar2(5);
173  $resta->operar();
174  $resta->imprimirResultado();
175
176  ?>
```

"La suma de $this→valor1 y $this→valor2 es: "

"La diferencia de $this→valor1 y $this→valor2 es: "

Gràcies a la **sobreescriptura de mètodes** que ens permet el **polimorfisme**, el mètode **imprimirResultado** es comportarà de manera diferent **depenent del tipus d'objecte**(suma o resta en aquest cas) que l'executi

POO: Polimorfisme i sobreescriptura de mètodes(constructor)

També es poden sobreescriure mètodes per defecte de les classes com en **constructor**

```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 function imprimirResultado()
  {
    echo $this->resultado.'<br>';
  }
}
```

```php
//Clase hija
class Suma extends Operacion{
  //Atributos de clase
  protected $titulo;
  //Constructor de clase
  public function __construct($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: Polimorfisme i sobreescriptura de mètodes(constructor)

```
223  //Creacion e inicializacion de objetos
224  $suma=new Suma(10,10,'Suma de valores:');
225  $suma->operar();
226  $suma->imprimirResultado();
227
228  ?>
229  </body>
230  </html>
```

barcelona.cat/barcelonactiva

Ajuntament de Barcelona

Desenvolupament web amb PHP
Conceptes clau de POO (II)

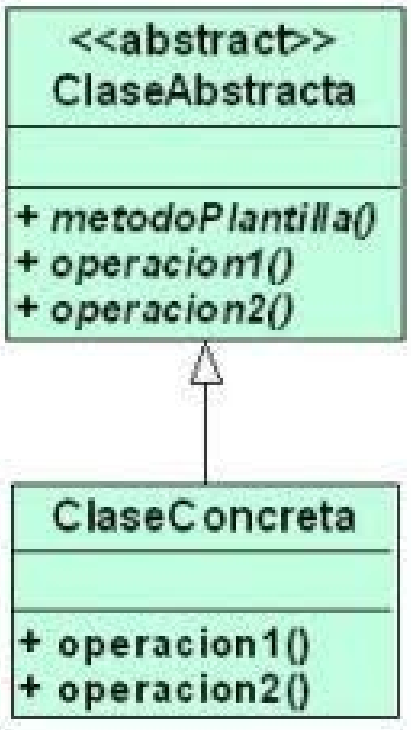
IT Academy

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;
  }
}
```

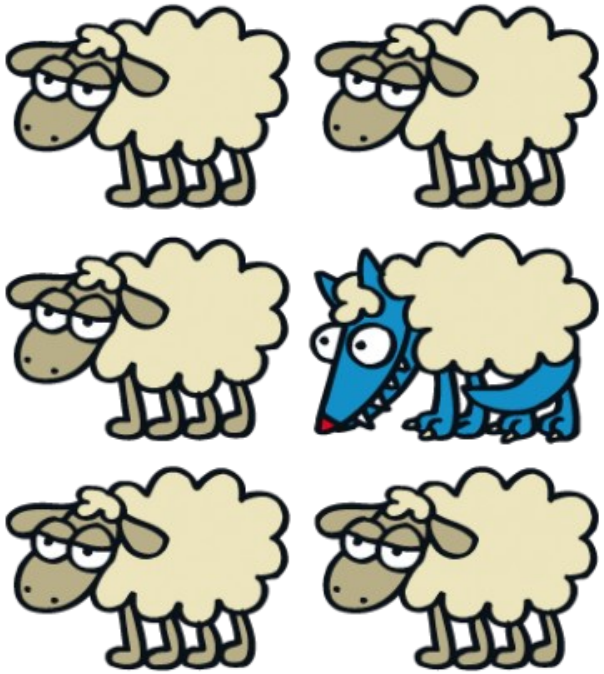
POO: Mètodes i classes finals

```
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()

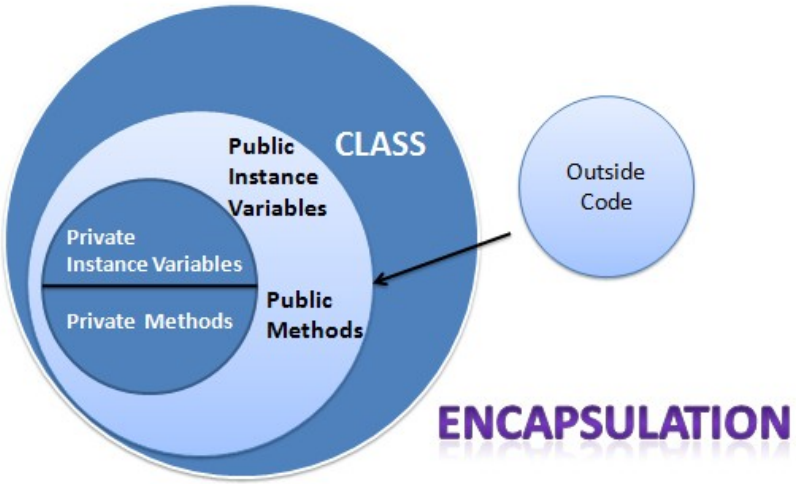
```
593  //Creacion e inicializacion de objetos
594  $baner1=new Banner(428,45,'Sistema de Ventas por Mayor y Menor');
595  $baner1->graficar();
596
597  ?>
```

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