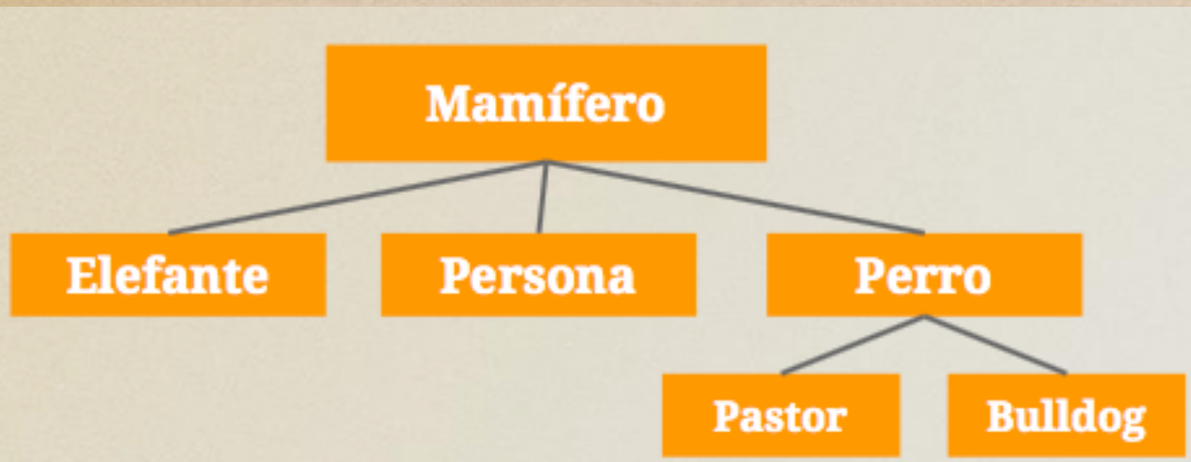
Parte IV

Herencia

Herencia

La herencia es un concepto que nos encontramos en la vida real.

Lo podemos apreciar en las jerarquías de clasificación.



Herencia

La herencia, en POO, consiste en la transmisión de los atributos y métodos desde una clase a otra.

Por ejemplo, la clase Mamífero define una serie de propiedades y atributos que transmite a todos los que heredan de Mamífero.

- Propiedades: Se definen Macho/Hembra. Las hembras producen leche.
- Métodos: Gestar (en la barriga de la madre), mamar (las crías de la madre)

Permite reutilizar el código cuando creamos "especializaciones" de las clases y facilita que la complejidad de la clase no se dispare en esas especializaciones.

- La clase perro, persona o gato reutiliza código de la clase mamífero.
- La clase moto o coche reutiliza de la clase vehículo.

Padre / Hija

La clase hija hereda los atributos y métodos (el código) de la clase padre.

La clase hija se especializa siempre, añadiendo nuevas propiedades y métodos o sobrescribiendo los de la clase padre.

Por ejemplo, la clase mamífero puede tener un método "comunicar", que se puede sobrescribir en las clases hijas indicando que las personas "hablan" y los perros "ladran".

¿Cuando debemos hacer herencia?

Si en nuestra clasificación encontramos objetos que son especialización de otros, muy probablemente la herencia nos produzca beneficios.

Sintaxis

Para poder crear una clase hija que hereda de una clase padre se utiliza la sintaxis siguiente:

```
class ClaseHija extends ClasePadre
{
  //Código de la clase hija
}
```

Aunque no figura explícitamente, dentro de la clase hija nos podemos imaginar que hay un include o require de la clase padre, por lo que el código de la clase padre está disponible en la clase hija.

Sólo hay una excepción a lo anterior. Los métodos y propiedades **private** no se pueden acceder desde las clases hijas.

Por eso existe **protected**, que es igual que **private**, pero sí se pueden ver y utilizar desde las clases hijas.

Redefinición de Métodos

La redefinición de métodos consiste en crear un método en la clase hija con el mismo nombre que otro de la clase padre. Por tanto, en la clase hija tendremos su método y no el del padre.

Para ello se requiere que el método a sobrescribir tenga el mismo nombre y la misma cantidad de parámetros que el del padre. Sólo hay una excepción y es el método __construct() que si puede tener una cantidad distinta de parámetros.

No confundir esto con Sobrecarga de métodos, que es otra cosa que no hemos visto todavía y que en PHP funciona de forma especial.

Si en una clase hija tenemos un método sobrescrito de la clase padre, y aun así queremos acceder al método de la clase padre, se invoca mediante la palabra reservada **parent.**

```
parent::metododelpadre();
```

Ejemplo

A partir de la clase Select hecha en los ejercicios anteriores, podemos ahora crearnos una clase hija llamada SelectPredeterminado.

En ella, tendremos que redefinir el método que genera las opciones del Select, para que cuando lleguemos al valor que queramos que sea predeterminado, añadir la opción SELECTED que sirve para que HTML tome esa opción por defecto.

```
protected function genera_opciones(){
  $opciones = "";
  foreach($this->datos as $key => $value){
    $opciones .= "<option ";
    if($key == $this->predeterminado){
      $opciones .= " selected ";
    }
    $opciones .= " value=\"$key\">$value</option>";
  }
  return $opciones;
}
```

Uso de parent

Cuando redefinimos un método, podemos (y generalmente será apropiado) apoyarnos en el código del método de la clase padre, y para acceder a él desde la clase hija se hace con la palabra reservada **parent.**

```
public function mostrar(){
  //Invocamos al método del padre reutilizando su cidigo
  parent::mostrar();
  //Hacemos la parte especifica de la clase hija
  echo $this->atributo_a_mostrar_de_la_clase_hija;
}
```

Podemos ver como ejemplo el método “escribe” de una clase SelectMultiple que es hija de la clase Select.

```
class SelectMultiple extends Select{
    public function escribe($atributos=""){
        parent::escribe("multiple " . $atributos);
    }
}
```

Constructores y parent

El constructor de la clase hija puede apoyarse en el constructor de la clase padre.

Veamos un ejemplo:

```
class Prisma extends Rectangulo
{
 function __construct ($altura, $anchura, $profundidad){
  parent::__construct($altura, $anchura);
  $this->profundidad = $profundidad;
 }
}
```

También podemos ver un ejemplo en el constructor de la clase SelectMultiple, como hija de la clase Select.

Uso de final

La palabra reservada **final** nos permite crear clases y métodos evitando los comportamientos de la herencia.

- **final** en un método. Es un método que no se puede redefinir en las clases hijas.

```
final public function(){
  //Código
}
```

- **final** en una clase. Es una clase que no se puede heredar. Es decir, esta clase no tendrá clases hijas.

```
final class MiClase
{
  //Código
}
```

Ejemplo de Herencia

Agregar herencia al ejemplo de "migas de pan"

MigasPanContenedor:

Es una clase que tiene que heredar de MigasPan, con la diferencia que tiene un contenedor configurable, donde se tiene que mostrar las migas de pan. El contenedor corresponderá a una etiqueta HTML, como podría ser p (<p>) o span () o div ( ).

MigasPanContenedor tiene que heredar de MigasPan, pero además tiene que usar la clase etiqueta, que es la que nos permite generar el contenedor.

Etiqueta:

Es una clase para implementar una etiqueta sencilla de HTML. Hacer que el enlace de MigasPan, donde Enlace es una especialización de Etiqueta, herede de ella.

Nota: El ejemplo de la etiqueta tiene sentido a nivel didáctico. Una página Web no se desarrolla a base de objetos etiqueta ya que el HTML generalmente se escribe fuera de código PHP, mediante plantillas o vistas (esquema MVC).

Clase Etiqueta

```php
<?php
class Etiqueta
{
    private $tag;
    private $contenido;
    private $atributos;

    public function __construct($tag, $contenido="", $atributos=""){
        $this->tag = $tag;
        $this->contenido = $contenido;
        $this->atributos = $atributos;
    }

    public function mostrar($contenido = null){
        $salida = "<$this->tag $this->atributos>";
        if(is_null($contenido)){
            $salida .= $this->contenido;
        }else{
            $salida .= $contenido;
        }
        $salida .= "</$this->tag>";
        return $salida;
    }
}
```

Clase Enlace

```php
<?php

class Enlace extends Etiqueta{

   //private $url;

   public function __construct($texto, $url){
      parent::__construct("a", $texto, "href=\"$url\"");
      //$this->url = $url;
   }
}

?>
```

Clase MigasPan

```php
<?php
class MigasPan
{

    private $separador;
    private $nodos = [];

    public function __construct($separador = "-"){
        $this->separador = $separador;
    }

    public function agregaNodo($texto, $ruta){
        $this->nodos[] = new Enlace($texto, $ruta);
    }

    public function mostrar(){
        $salida = '';
        foreach ($this->nodos as $indice => $enlace) {
            $salida .= $enlace->mostrar();
            if($indice != count($this->nodos) -1 ){
                $salida .= ' ' . $this->separador . ' ';
            }
        }
        return $salida;
    }
}
```

Clase MigasPanContenedor

```php
<?php

class MigasPanContenedor extends MigasPan
{

    private $contenedor;

    public function __construct($separador, $contenedor){
        parent::__construct($separador);
        $this->contenedor = new Etiqueta($contenedor);
    }

    public function mostrar(){
        return $this->contenedor->mostrar(parent::mostrar());
    }
}
?>
```

Usando MigasPanContenedor

```
<?php

include "etiqueta.php";
include "enlace.php";
include "MigasPan.php";
include "MigasPanContenedor.php";

$migas = new MigasPanContenedor("-", "section");
$migas->agregaNodo("Home", "http://www.iescierva.net");
$migas->agregaNodo("Noticias", "http://www.iescierva.net/
noticias");
$migas->agregaNodo("Noticias Académicas", "http://
www.iescierva.net/noticias/academicas");
echo $migas->mostrar();
```