1 Curso POO PHP Herdanza

1.1 Sumario

1.2 Herdanza

A herdanza é un mecanismo da POO que nos permite definir novas clases sobre a base doutra xa existente. As novas clases que herdan coñécense tamén co nome de subclases. A clase da que herdan chámase clase base ou superclase.

Por exemplo, podemos crear unha clase base chamada Produto, con algúns atributos e un método que xera unha saída personalizada en formato HTML do código.

```
class Produto {
    public $codigo;
    public $nome;
    public $nome_corto;
    public $PVP;

    public function mostra_codigo() {
        print "<p>" . $this->codigo . "</p>";
    }
}
```

Esta clase é moi útil se a única información que temos dos distintos produtos é a que se mostra arriba. Pero, se queres personalizar a información que vas tratar de cada tipo de produto (e almacenar, por exemplo para os televisores, as polgadas que teñen ou a súa tecnoloxía de fabricación), podes crear novas clases que herden de Producto. Por exemplo, TV, Ordenador, Movil.

```
class TV extends Produto {
    public $polgadas;
    public $tecnoloxia;
}
```

Como podes ver, para definir unha clase que herde doutra, simplemente tes que utilizar a palabra **extends** indicando a superclase. Os novos obxectos que se instancien a partir da subclase son tamén obxectos da clase base; pódese comprobar utilizando o operador **instanceof**.

```
$t = new TV();
if ($t instanceof Produto) {
// A condición é certa
?
}
```

Nas clases herdadas podes definir un novo construtor que redefina o comportamento do que existe na clase base, tal e como farías con calquera outro método. E dependendo de se programas ou non o construtor na clase herdada, se chamará ou non automaticamente ao construtor da clase base.

En PHP5, se a clase herdada non ten construtor propio, chamarase automaticamente o construtor da clase base (se existe). Non obstante, se a clase herdada define o seu propio construtor, deberás ser ti que realice a chamada ao construtor da clase base se o consideras necesario, utilizando para iso a palabra **parent** e o operador de resolución de ámbito.

```
class TV extends Produto {
    ...
    public function __construct($row) {
        parent::__construct($row);
        $this->polgadas = $row['polgadas'];
        $this->tecnoloxia = $row['tecnoloxia'];
    }
}
```

A palabra parent fai referencia á clase pai, e emprégase para facer referencia aos métodos orixinais que foron redefinidos na clase filla, como o construtor no exemplo anterior.

1.2.1 Herdanza e visibilidade

Como vimos cando falamos da definición de clases, os membros dunha clase poden ter tres tipos de visibilidade: public, private e protected.

Ademais das diferenzas no acceso aos membros segundo a súa visibilidade, esta tamén afecta á herdanza. Os membros privados dunha clase non están dispoñibles nas clases que herdan desta. Para crear un membro de acceso restrinxido (como os private) e herdable, débese declarar como **protected**. Os membros protected dunha clase hérdanse ao igual que os públicos e o acceso aos mesmos restrínxese aos propios membros da clase da mesma forma que sucede co acceso aos membros privados.

1.2.2 Enlace estático en tempo de execución

Consideremos o que acontece cando herdamos dunha clase que contén un método estático, como no noso exemplo.

```
class Produto {
    public static function novoProduto() {
        self::$num_produtos++;
    }
    ?
}

class TV extends Produto {
    private static $num_produtos = 0;
    ?
}

class Ordenador extends Produto {
    private static $num_produtos = 0;
    ?
}
```

O que pretendemos coa herdanza é empregar un mesmo método *novoProduto* que conte os números dos artigos de cada unha das clases, de xeito que cando fagamos:

```
TV()::novoProduto();
```

engada 1 á variable *$num_produtos* da clase TV, e cando fagamos:

```
Ordenador::novoProduto();
```

engada 1 á variable *$num_produtos* da clase Ordenador.

Non obstante o código anterior non funciona como esperamos, pois ao compilar encontra no método *novoProduto* unha referencia a **self::$num_produtos**, e intenta enlazar con unha variable da clase a que pertence (*Produto*) que non existe.

Para solventalo temos que indicar a PHP que enlace coa variable en tempo de execución, non en tempo de compilación, e empregue as propiedades da clase que fai a chamada ao método. Isto faise empregando a palabra **static** no canto de *self* co operador de resolución de ámbito **::**.

Tamén teremos que definir as propiedades *num_produtos* como *protected* para darlles acceso dende o método herdado. Isto é:

```
class Produto {
    public static function novoProduto() {
        static::$num_produtos++;
    }
    ?
}

class TV extends Produto {
    protected static $num_produtos = 0;
    ?
}

class Ordenador extends Produto {
    protected static $num_produtos = 0;
    ?
}
```

O mesmo acontece cando temos que acceder ao nome da clase. Se empregamos *__CLASS__* dende o método *novoProduto* devolveranos *"Produto"*. Para obter en tempo de execución o nome da clase que fai a chamada teremos que empregar a función **get_called_class**.

Outra forma de arranxar o problema da herdanza de métodos estáticos é emprengando *trazos*.

1.2.3 Funcións relacionadas coa herdanza

Función	Significado
get_parent_class	Devolve o nome da clase pai do obxecto ou a clase que se indica
is_subclass_of	Devolve true se o obxecto ou a clase do primeiro parámetro, ten como clase base á que se indica no segundo parámetro, ou false no caso contrario

--Víctor Lourido 14:08 28 jun 2013 (CEST)