Contents

[**Introducción** 1](#_Toc65156429)

[**Deuda técnica** 2](#_Toc65156430)

[**Code smell** 2](#_Toc65156431)

[**Cómo evitarlo** 2](#_Toc65156432)

[**Código espagueti** 2](#_Toc65156433)

[**Cómo evitar el código espagueti** 3](#_Toc65156434)

[**Inclusión de archivos** 3](#_Toc65156435)

[**Introducción a clases y objetos** 3](#_Toc65156436)

[**Características** 3](#_Toc65156437)

[**Signos** 4](#_Toc65156438)

[**Abstracción** 4](#_Toc65156439)

[**Formas de realizar la abstracción** 4](#_Toc65156440)

[ **Interface** 4](#_Toc65156441)

[ **Abstract** 5](#_Toc65156442)

[ **Class** 6](#_Toc65156443)

[**Alcance o Encapsulamiento** 7](#_Toc65156444)

[**Tipos de alcance** 7](#_Toc65156445)

[**Modularidad** 8](#_Toc65156446)

[**Polimorfismo** 9](#_Toc65156447)

[**Polimorfismo: interfaz** 10](#_Toc65156448)

[**Herencia** 11](#_Toc65156449)

[**Interfaces** 13](#_Toc65156450)

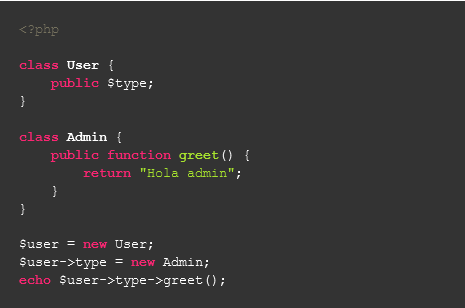
[**Resumen** 13](#_Toc65156451)

# **Introducción**

La **OOP** (**O**bject-**o**riented **p**rogramming) son un paradigma compuesto por técnicas que nos ayudará a hacer menos líneas de código y que este sea más reutilizable.

Estos serían los pasos que debemos seguir para tener una OOP:

1. Crear la **clase**, esta se trata de un molde para crear objetos.
2. Crear una **instancia**, el objeto es la instancia de la clase.



# **Deuda técnica**

Es el **coste** y los **intereses** a pagar por **hacer mal las cosas**. El sobre esfuerzo a pagar para mantener un producto software mal hecho, y lo que conlleva, como el coste de la mala imagen frente a los clientes, etc.

# **Code smell**

Hace referencia al **mal olor** del código. Este concepto no se refiere a errores técnicos, sino a **errores de orden** y **diseño**. Esto sucede mucho cuando intentamos crear soluciones a partir de otras soluciones.

La solución a estos casos es crear una abstracción.

## **Cómo evitarlo**

Para esto debemos hacer una programación más limpia, y reusable. Tenemos que evitar crear grandes métodos, o sea, programación estructurada dentro de clases. También evitar crear grandes clases o superclases.

Y sin duda, nosotros debemos evitar a toda costa copiar y pegar código.

**Recuerda**: El sistema va a funcionar, pero a futuro va a ser horrible de mantener, hasta imposible.

# **Código espagueti**

Un código espagueti es código que está estructurado netamente mediante **if**, **while**, **for**, todo en un mismo archivo donde solamente buscamos resolver el problema. Cuando creamos código estructurado corremos peligro de crear código espagueti. La OOP nos ayuda evitarlo.

💸 **El dinero en esta profesión está en el mantenimiento del código.**

## **Cómo evitar el código espagueti**

1. Resolver el problema con código espagueti o no.
2. Crea de forma lógica y coherente diferentes métodos que reemplacen tus estructuras de control.
3. Crea una o varias clases dependiendo del caso.

# **Inclusión de archivos**

Es un paso previo a la POO.

* **Include** => Nos permite incluir un archivo dentro de otro. Sino lo encuentra sólo genera un **warning** que permite continuar.
* **Require** => Lo mismo que include, pero más exigente, en caso de fallo producirá un **error fatal**, no permitirá continuar con el proceso.
* **Require\_once** => Idéntica a require excepto que PHP verificará si el archivo ya ha sido incluido varias veces y si es así, no se incluye nuevamente.

# **Introducción a clases y objetos**

Las clases son el molde y a partir de ese molde podemos crear diferentes objetos.

## **Características**

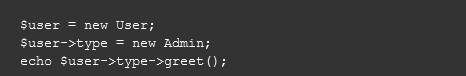
* Herencia.
* Modularidad.
* Abstracción.
* Polimorfismo.

## **Signos**

Una clase requiere la palabra reservada **class**



Para crear un objeto a partir de esa clase usamos la palabra reservada **new**



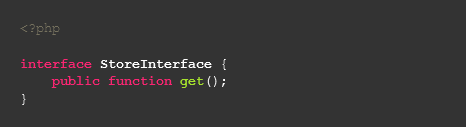
# **Abstracción**

Significa **aislar**, **separar** o **sacar**. Es básicamente lo que nos ayuda a abstraer, a pensar en el resultado final **antes** de que hagamos una programación.

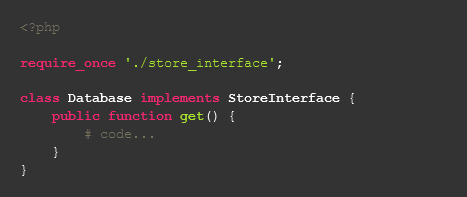
## **Formas de realizar la abstracción**

### **Interface**

* + Creamos el archivo **store.php** y declaramos lo que queremos como **resultado**.

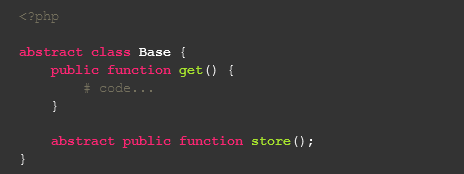


* + **Todos** archivos o clases que implemente esta interface está **obligado** a implementar el método **get()**.
  + Eso es abstracción, estamos **pensando en el resultado final** y lo estamos describiendo (**interface**).

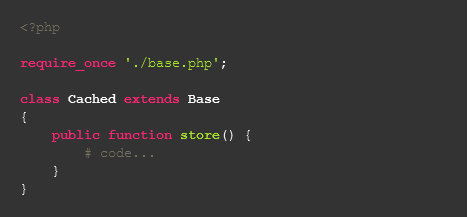


### **Abstract**

* + La abstracción se puede realizar con la interface o directamente con los métodos Abstract.
  + Un método abstracto no se define, **sólo se declara**.



* + Cualquier clase que extienda de Database deberá implementar el método **store()**.



### **Class**

* + Definiendo cada una de las cosas que requiero a nivel de autenticación.
  + Esto también es **abstracción**, es decir, **definir en esencia lo que yo requiero para mi clase**.

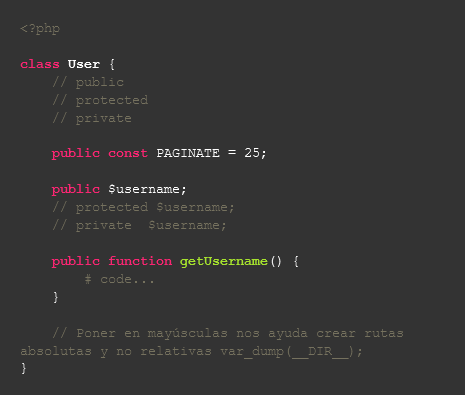


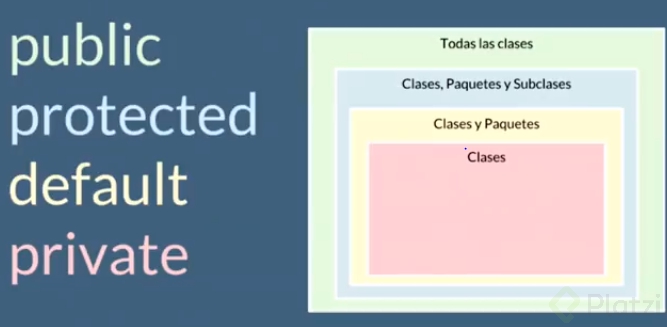
# **Alcance o Encapsulamiento**

El alcance hace referencia al encapsulamiento o principio de ocultación.

## **Tipos de alcance**

* **Public** → Puede ser accedido por **todos**, es decir, por cualquier elemento o clase.
* **Protected** → Podrá ser accedido a nivel de la **clases**, **paquetes** y **Subclases** (las clases hijas ← clases padres).
* **Defaulf** → Este nos permite el acceso a nivel de **clases** y **paquetes**. Es decir, a nivel de herencia no puede ser accedido.
* **Private** → Sólo puede ser accedido a nivel de **clases**, esto quiere decir, que sólo puede ser modificado a nivel de la clase, es decir, por ella misma.





# **Modularidad**

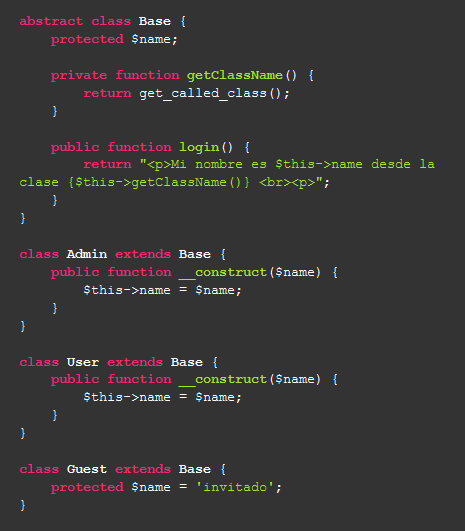
* Esta no es una técnica de programación, pero si es algo con lo que debemos cumplir para que en el futuro sea más fácil de mantener. Este concepto aplica **views**, **models**, **controllers**, **helpers**, etc.
* Debemos pensar en **módulos**, no en un sólo archivo con todo el código. ⚠
* Este hace referencia a tener pequeños archivos que al unirlos forman el sistema en sí.
* Lo más importante **es que en el futuro podamos entender lo que estamos programando hoy**.
* Si nosotros creamos pequeñas partes de código a lo largo de todo el sistema **va a ser mucho más fácil incluir un testeo** (PhpUnit.)
* **Crea cada cosa como una pequeña pieza aislada**.

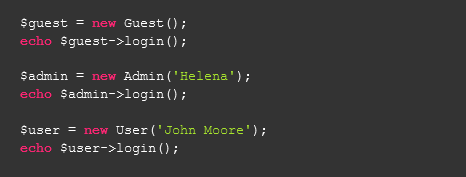
# **Polimorfismo**

El polimorfismo significa **varias formas**.

Esto quiere decir **que si un mismo elemento, si se comporta de diferentes maneras y otorga diferentes resultados, entonces aplica el término de polimorfismo**.

**Nota**: Sí es de varias cosas, si se comporta de diferentes maneras **un único elemento**, entonces es polimorfismo.

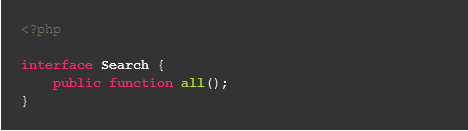




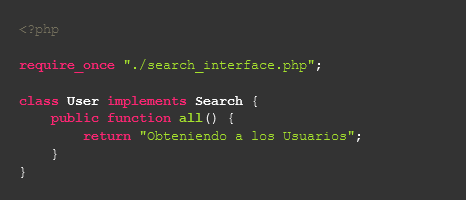
# **Polimorfismo: interfaz**

**En programación avanzada siempre vamos a trabajar sobre interfaces**. Nosotros programamos a en base a interfaces porque es lo que enviamos (nivel de configuración), preparamos todo el código necesario y no trabajamos sobre Users sino sobre la Interface Users.

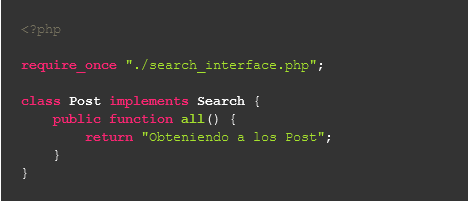
**Interface Search**



**Class User**

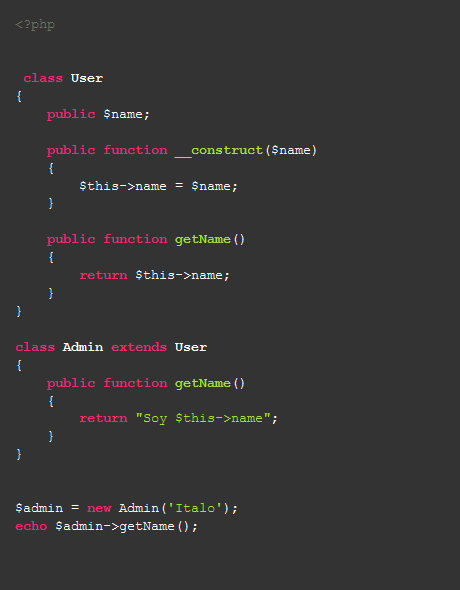


**Class Post**

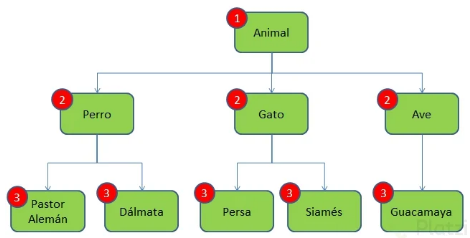


# **Herencia**

Tenemos una **clase** **padre** y una **clase hija**, si la clase padre tiene 5 funciones, la clase hija puede tener funciones propias y a estas se le **suman** las funciones de la clase padre.



Sí queremos que nadie pueda sobre escribir una función, colocamos la palabra **final** en la declaración de la función. Una clase también puede ser declarada como **final**, esto quiere decir que la clase no podará ser usada para que otra herede de ella.



# **Interfaces**

Una interface se desarrolla y se implementa en una clase, al implementar estamos obligados a desarrollar todos los métodos que la interface define.

El “**qué se va hacer**” se define en la interface, y el “**como va a ser y la implementación**” se define en la clase.

# **Resumen**

La programación orientada a objetos es una forma de programar, un paradigma o una técnica. Los conceptos que aquí aprendiste te servirán en PHP y en otros lenguajes de programación. Recordemos que para programar de esta forma en realidad debemos crear objetos, y un objeto es una instancia de una clase y una clase es el molde. Ejemplo:

* **Programación orientada a objetos**: Es la técnica.
* **PHP**: Es el lenguaje de programación (donde implementamos la técnica).

Podemos resumir los diferentes conceptos de la siguiente manera:

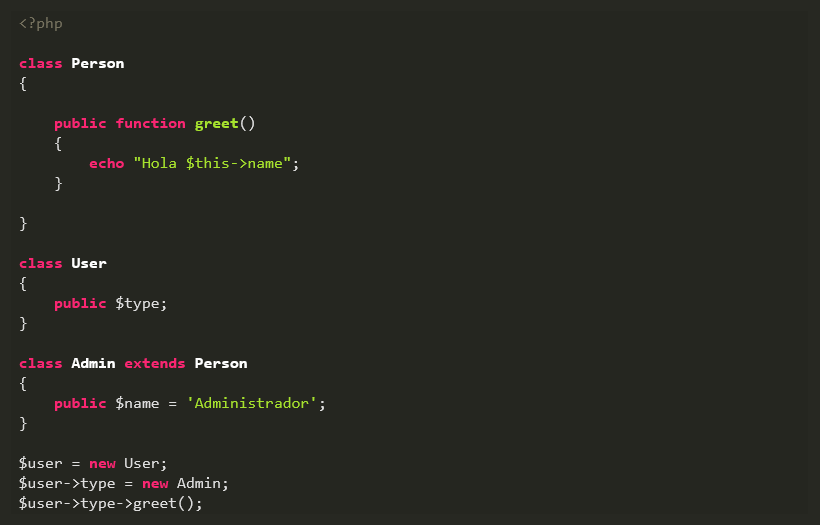
1. **Herencia**: Compartir métodos entre clases padres y clases hijas.
2. **Abstracción**: Significa aislar, separar y sacar.
3. **Polimorfismo**: Capacidad o virtud que tienen los métodos donde, por ejemplo, un mismo método puede tener diferentes comportamientos y dar diferentes resultados.
4. **Modularidad**: Este principio básicamente nos ayuda a tener cada vez piezas de código más pequeñas y entendibles, donde cada pieza es un módulo y muchos módulos forman el sistema entero.
5. **Encapsulamiento**: Un objeto debe estar aislado y ser un módulo natural. Esto se cumple aplicando la protección a las propiedades impidiendo su modificación y básicamente se refiere a controlar el acceso.

Al entender este estilo conseguimos organizar mucho mejor nuestro código agrupando tareas comunes para crear una sola solución y usarla las veces que sean necesarias en nuestro proyecto. Evitamos con esto repetir código y ganamos mucho al dar mantenimiento en el futuro.

1. Comienza con la palabra reservada **class**.
2. El código va entre llaves **{ }**.
3. La información se guarda en propiedades que pueden ser públicas, privadas o protegidas.
4. Cada acción la colocamos en métodos que básicamente son funciones o bloques de código dentro de una clase.
5. **$this** es una variable reservada por el lenguaje que podemos usar para acceder a elementos propios, siempre y cuando estemos en la instancia de la clase.
6. **new** es la palabra clave usada para crear un objeto a partir de una clase.

En el mundo de la programación tenemos muchas técnicas y formas, podemos instanciar una clase dentro de otra y navegar por sus métodos sin restricción.





Son conceptos, métodos o formas que usamos a veces sin saber que estos conceptos existen.

