

¿Qué es la programación orientada a objetos?

OOP - Object-Oriented Programming

Tipo de programación

OOP es una variante del estilo de programación imperativo.

Se trata de un desarrollo posterior de la programación procedimental:

- Programación imperativa: describe en pasos individuales cómo resolver un problema. Ejemplo: Algoritmo
 - Programación estructurada
 - Programación procedimental
 - Programación orientada a objetos



Tipo de Programación

	Paradigma	Características	Especialmente adecuado para	Idiomas
Imperativo	OOP	Objetos, clases, métodos, herencia, polimorfismo	Modelización, diseño de sistemas	Smalltalk, Java, Ruby, Python, Swift
Imperativo	Procedimental	Flujo de control, iteración, procedimientos / funciones	C, Pascal, Basic	
Declarativo	Funcional	Inmutabilidad, funciones puras, cálculo lambda, recursión, sistemas de tipos	Procesamiento paralelo de datos, aplicaciones matemáticas y científicas, analizadores sintácticos y compiladores	Lisp, Haskell, Clojure
Declarativo	Lenguaje específico del dominio (DSL)	Expresivo, amplia gama de lenguaje	Aplicaciones específicas del sector	SQL, CSS

Procedimental VS P00

Procedimental: trabaja con **datos** inertes que son procesados por un **código** ejecutable.

Datos: valores, estructuras de datos, variables

Código: expresiones, estructuras de control, funciones

P00: **combina datos y funciones en objetos**. Un objeto es una estructura de datos viva dado que tienen un comportamiento.

Mientras que con los datos solo se opera,
con los objetos es posible interactuar



Funcionamiento

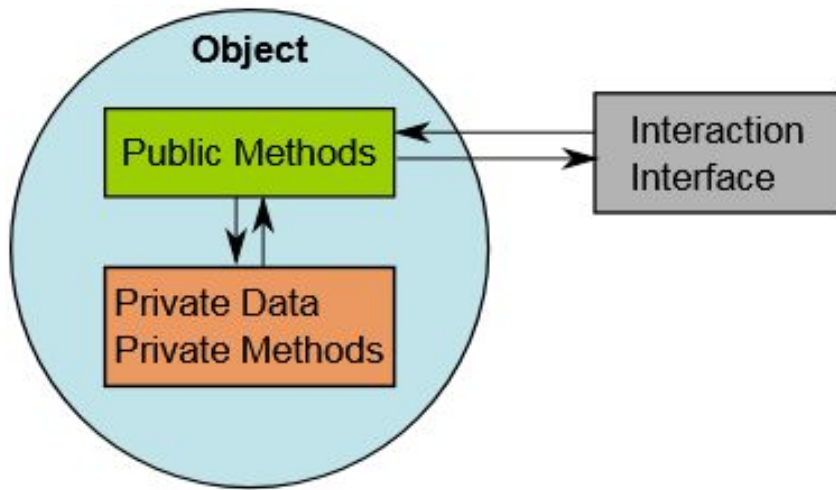
1. Los objetos **encapsulan** su estado interno.
2. Los objetos reciben mensajes a través de sus **métodos**.
3. Los **métodos se asignan dinámicamente** en tiempo de ejecución



Encapsular

El objeto decide / ejecuta: Dentro del mismo, el estado se almacena en variables.
El objeto gestiona los valores de las variables.

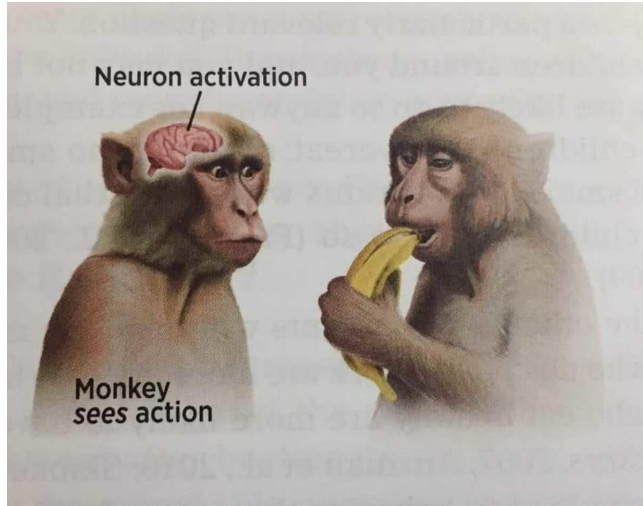
Este estado no puede ser modificado directamente desde el exterior.



Envío de mensajes / métodos de llamada

Los objetos reaccionan a los mensajes cambiando su estado interno.

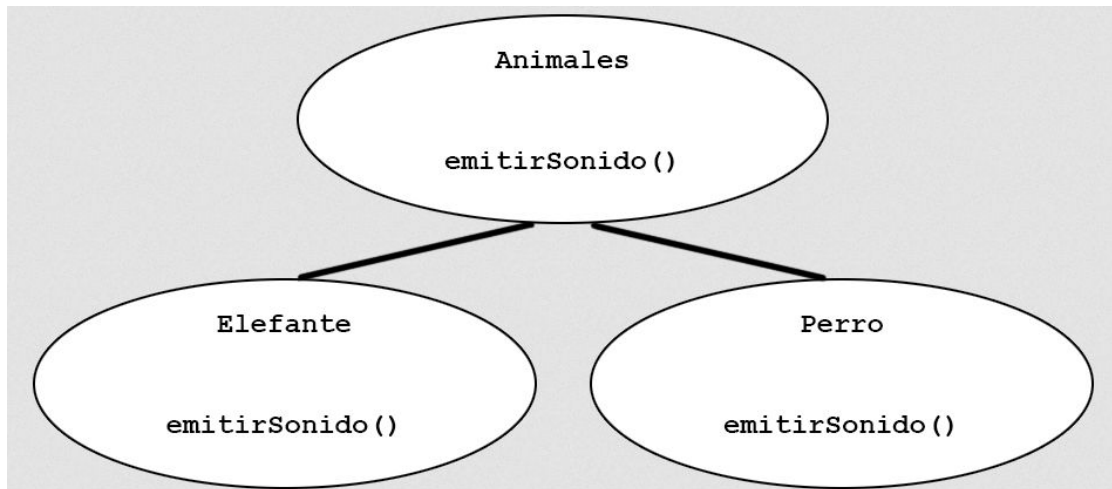
Estos mensajes reciben el nombre de métodos; son funciones que están vinculadas al objeto.



Polimorfismo

Es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos.

Aunque el mensaje sea el mismo, diferentes objetos pueden responder a él de manera única y específica. Esto permite incorporar nuevos comportamientos y funciones.




Asignación dinámica de los métodos

Capacidad de definir, invocar o **modificar métodos de manera dinámica**, en tiempo de ejecución, en lugar de hacerlo de forma estática durante la declaración de la clase.

En PHP:

1. Método mágico: `__call()`
2. Funciones anónimas: `function()`

Esta técnica es **valiosa para frameworks** o bibliotecas que requieren manejar métodos o propiedades dinámicamente, como en sistemas de ORM, servicios API, y otros patrones de diseño que demandan alta flexibilidad.



Instanciación e inicialización

Instanciación es el proceso por el que un **objeto pasa a existir**.

En PHP se realiza mediante la palabra clave ***new***, seguida del nombre de la clase.

Inicialización es el proceso de configurar las propiedades iniciales de un objeto. Establecer la configuración necesaria para que el objeto esté listo para usarse.

En PHP se realiza mediante una función llamada **constructor**, que ensambla un objeto cuando se llama. Al ejecutar la función, se le asigna atributos y estado inicial al objeto. A esta función se le pueden facilitar parámetros (valores).



Ejemplo de objeto - Clase PHP - Atributos



```
class Televisor {
```

```
// Atributos
```

```
public $marca;
```

```
public $color;
```

```
public $tamano;
```

```
public $peso;
```

Ejemplo de objeto - Clase PHP - Constructor



```
// Constructor para inicializar los atributos

public function __construct($marca, $color,
$tamano, $peso) {

    $this->marca = $marca;

    $this->color = $color;

    $this->tamano = $tamano;

    $this->peso = $peso;

}
```

Ejemplo de objeto - Clase PHP - Metodos



// Métodos

```
public function encender() {  
    echo "El televisor está encendido.\n";  
}  
public function apagar() {  
    echo "El televisor está apagado.\n";  
}  
public function subirCanal() {  
    echo "Subiendo el canal.\n";  
}  
public function bajarCanal() {  
    echo "Bajando el canal.\n";  
}  
public function subirVolumen() {  
    echo "Subiendo el volumen.\n";  
}  
public function bajarVolumen() {  
    echo "Bajando el volumen.\n";  
}  
}
```

Acceso a atributos y métodos

Acceso a atributo de un objeto:

```
$nameObject->nameAttribute;
```

Acceso a método de un objeto, si este se comporta como un procedimiento:

```
$nameObject->nameMethod();
```

o bien, si el método se comporta como una función:

```
$var = $nameObject->nameMethod();
```



Ejemplo de objeto - Clase PHP - Metodos



// Ejemplo de uso

```
$televisor = new Televisor("Samsung",  
"Negro", "42 pulgadas", "8 kg");
```

```
$televisor->encender();
```

```
$televisor->subirCanal();
```

```
$televisor->subirVolumen();
```

```
$televisor->apagar();
```

Visibilidad atributos y métodos

Pública: Visible para todos.

Protegida: Visible para objetos hijos.

Privada: Visible sólo dentro del mismo objeto.

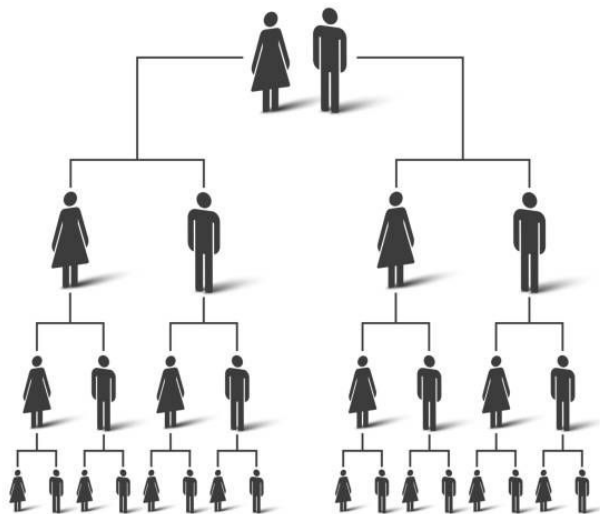
Nivel	Acceso en la misma clase	Acceso en clases hijas	Acceso externo
<code>public</code>	Sí	Sí	Sí
<code>protected</code>	Sí	Sí	No
<code>private</code>	Sí	No	No

Herencia

Mecanismo que permite a **una clase (hija)** **heredar propiedades y métodos de otra clase (padre)**.

Esto permite reutilizar código y establecer relaciones jerárquicas entre clases.

En PHP, se usa la palabra clave **extends** para crear una clase que herede de otra.



Polimorfismo en PHP mediante Herencia

```
class Animal {  
    public function hacerSonido() {  
        // Implementación genérica de hacerSonido()  
    }  
}
```

```
class Perro extends Animal {  
    public function hacerSonido() {  
        echo "El perro dice: Guau Guau\n";  
    }  
}
```

```
class Gato extends Animal {  
    public function hacerSonido() {  
        echo "El gato dice: Miau Miau\n";  
    }  
}
```

Misma función, diferente resultado

Abstracción

El proceso de definir clases y métodos abstractos que **representan conceptos** generales **sin implementar detalles** específicos.

Una **clase abstracta no se puede instanciar** directamente y actúa como una plantilla que define métodos que sus clases hijas deben implementar.

En PHP, las clases y métodos abstractos se definen usando la palabra clave **abstract**.




Ejemplo de Clase Abstracta

```
abstract class Vehiculo {  
    abstract public function arrancar();  
}
```

```
class Coche extends Vehiculo {  
    public function arrancar() {  
        echo "El coche está arrancando";  
    }  
}
```

```
$miCoche = new Coche();  
$miCoche->arrancar(); // Resultado: El coche está arrancando
```

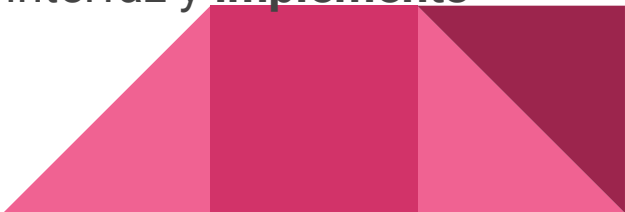


Interfaces

Una **interfaz define un conjunto de métodos** que una clase debe implementar, pero **sin** proporcionar **la implementación** de estos métodos.

Permiten la implementación múltiple (una clase puede implementar varias interfaces), lo cual es útil para definir **comportamientos comunes** en clases no relacionadas entre sí.

En PHP, se usa la palabra clave **interface** para definir una interfaz y **implements** para implementarla en una clase.



Ejemplo de Interfaz en PHP

```
interface Volador {  
    public function volar();  
}  
  
class Avion implements Volador {  
    public function volar() {  
        echo "El avión está volando";  
    }  
}  
class Pájaro implements Volador {  
    public function volar() {  
        echo "El pájaro está volando";  
    }  
}  
  
$miAvion = new Avion();  
$miAvion->volar(); // Resultado: El avión está volando  
$miPajaro = new Pájaro();  
$miPajaro->volar(); // Resultado: El pájaro está volando
```