

Abstracción e interfaces en PHP

Abstract Class, Abstract function e Interface/implements

Recordatorio: POLIMORFISMO

La capacidad de **usar un único método** para trabajar con **objetos de distintas clases**.

Cada clase puede implementar su propia versión de la tarea a realizar. Es decir, pueden tener su propio código.

Ventajas:

- Reutilización del código
- Desacoplamiento
- Extensibilidad (mejor adaptación, facilitar ampliación)



Abstracción en PHP

Se consigue mediante la palabras clave:

- Abstract class
 - ... extends
- Interface
 - ... implements



Clase Abstracta

Una clase abstracta no se puede instanciar y, por tanto, debe tener necesariamente una subclase heredera.

Las clases abstractas pueden tener **atributos** comunes a las subclases que las implementen.

En una clase abstracta, los **métodos** pueden ser:

- **Declarativos:** sin implementación, serán obligatoriamente sobrescritos en las subclases que los implementen. Se declaran con la palabra reservada `abstract nombreMetodo`.
- **Concretos:** con una implementación común para todas las subclases que los implementen. Estas implementaciones también podrán ser sobrescritas.

Además, también pueden contener métodos estáticos.



Ejemplo de clase Abstracta

```
// Clase abstracta
abstract class Animal {
    // Método abstracto
    abstract public function hacerRuido();
}

// Clase concreta para Perro con herencia de la clase
Animal
class Perro extends Animal {
    public function hacerRuido() {
        return "Guau Guau!";
    }
}

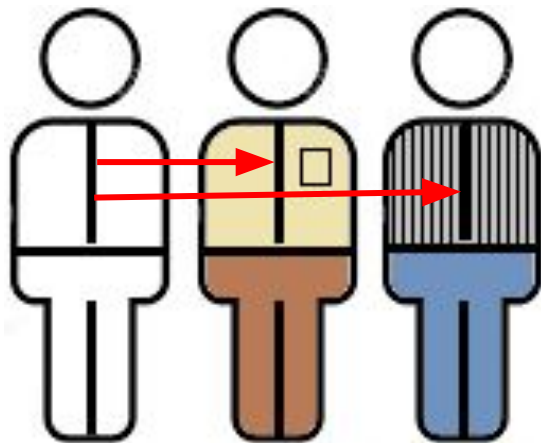
$perro = new Perro();
echo $perro->hacerRuido(); // Salida: Guau Guau!
$gato = new Gato();
echo $gato->hacerRuido(); // Salida: Miau!
```

```
// Clase concreta para Gato con herencia de la
clase Animal
class Gatoo extends Animal {
    public function hacerRuido() {
        return "Maiu!";
    }
}
```

Interface

Una Interfaz es una solución que proporciona PHP para implementar la abstracción, que también puede lograrse mediante Clases abstractas.

Ambas comparten muchos conceptos, pero es importante diferenciarlas.



Interface

La Interfaz que implementa PHP es, en esencia, una colección de **métodos abstractos** declarados **que deben ser implementados**.


Una interfaz no se puede instanciar y, por tanto, debe tener necesariamente una subclase heredera.

Las interfaces pueden tener **atributos que actúan como constantes**, comunes a las subclases que las implementan.

En una “interface”, igual que pasaba con las clases abstractas, los métodos pueden ser:

- **Declarativos:** sin implementación, serán obligatoriamente sobrescritos en las subclases que los implementen. Se declaran con la palabra reservada `abstract nombreMetodo`.
- **Concretos:** con una implementación común para todas las subclases que los implementen. Estas implementaciones también podrán ser sobrescritas.

Una clase puede implementar múltiples interfaces, aportando el concepto de herencia múltiple al lenguaje.



Ejemplo de Interface

```
// Interface Animal
interface Animal {
    // Método abstracto
    abstract public function hacerRuido();
}

// Clase concreta para Perro que implementa Animal
class Perro implements Animal {
    public function hacerRuido() {
        return "Guau Guau!";
    }
}

$perro = new Perro();
echo $perro->hacerRuido(); // Salida: Guau Guau!
$gato = new Gato();
echo $gato->hacerRuido(); // Salida: Miau!
```

```
// Clase concreta para Gato que implementa Animal
class Gato implements Animal {
    public function hacerRuido() {
        return "Maiu!";
    }
}
```



Ejemplo de uso de 2 Interfaces

```
// Interface Animal
interface Animal {
    // Método abstracto
    abstract public function hacerRuido();
}

// Clase concreta para Perro
class Perro implements Animal, Mamifero {
    public function hacerRuido() {
        return "Guau Guau!";
    }
}

$perro = new Perro();
echo $perro->hacerRuido(); // Salida: Guau Guau!
$gato = new Gato();
echo $gato->hacerRuido(); // Salida: Miau!
echo $perro->sueltaPelo(); // Salida: Suelta pelo!
echo $gato->sueltaPelo(); // Salida: También suelta!
```

```
// Interface 2
interface Mamifero {
    default function sueltaPelo(){
        return "Suelta pelo.";
    }
}


// Clase concreta para Gato
class Gato implements Animal, Mamifero {
    public function hacerRuido() {
        return "Maiu!";
    }
    public function sueltaPelo(){
        return "También suelta!";
    }
}
```

Clases abstractes vs Interfícies

Clases abstractas:

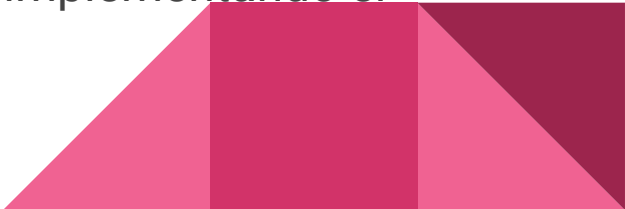
- Atributos y constantes se añaden a la clase heredera (hija/subclase)
- Puede tener constructores, que se llaman con `parent::__constructor` desde la subclase
- Métodos abstractos o implementados/concretos (que pueden ser sobrescritos)
- Una subclase sólo puede heredar de una única clase abstracta (herencia simple)

Interfícies:

- No permite atributos. Sí constantes
 - No permite constructores
 - Métodos abstractos (imprescindible sobrescribir con implementación) o concretos
 - Una clase puede implementar más de una Interficie (herencia múltiple)
- 

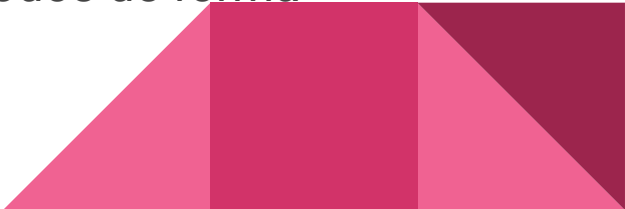
Clases abstractas vs interfícies

En definitiva...

- Una **Clase abstracta** no es más que una *clase común* que puede tener atributos, constructores, métodos abstractos y métodos concretos. Una clase abstracta no puede ser instanciada, solo heredada. Una clase hija puede extender una clase normal o abstracta.
 - Una **Interfaz** puede tener constantes, métodos abstractos y métodos por concretos. Una interfaz no puede ser instanciada, solo referenciada. Una clase puede implementar una, dos o más interfaces, implementando el concepto de herencia múltiple.
- 

Clases abstractas vs interfícies

Cuando usar Abstract o Interface:

- **Usaremos clases abstractas** si las subclases tienen **atributos comunes y métodos con un mismo comportamiento**. En este caso, los atributos y métodos se implementarán en la clase abstracta.
 - **Usaremos Interfaces** si las clases tienen atributos propios y **métodos con diferentes comportamientos**. En este caso, los métodos se implementarán en cada clase. También son útiles si se quiere obligar a que las clases que implementen la interfaz contengan una serie de métodos de forma obligatoria.
- 

Clases abstractas vs interfícies

En PHP **es posible trabajar con clases abstractas e interfaces al mismo tiempo.**

Esto es útil para aprovechar las ventajas de ambas estructuras.

```
class subClasse extends classePadre implements Interficie1, Interficie2, ... {  
    // Implementació de mètodes de la Classe i de cada Interfície  
}
```

Este enfoque también puede entenderse como un caso de implementación de herencia múltiple.

