


Reutilizar código de una clase

Traits

Antes de continuar...

Hemos visto que es posible reutilizar código mediante:

- **Require/Include:** Carga de scripts dentro de archivos
 - **Namespace:** Agrupar partes de código bajo un mismo nombre
 - **Extends:** Herencia simple donde una clase (hija) hereda atributos y/o métodos de otra (padre)
 - **Abstract:** Definir clases abstractas que pueden contener declaración o implementación a heredar
 - **Interface:** Colección de métodos abstractos declarados que deben ser implementados.
- 

Traits vs Namespace

Traits replica el funcionamiento de Namespace; da un nombre específico a una clase (con sus atributos y métodos).

Esta “clase” con nombre se puede emplear dentro de la declaración de otra clase mediante “use”, de forma similar a los alias de Namespace.

Diferencia principal;

- Namespace da nombre a un **fragmento de código** - Trait nombra **una clase**
- Namespace se usa en la lógica del **programa** - Trait se usa **dentro de la clase**



Características de las “clases” tipo Traits

Replican el comportamiento de una clase abstracta: se usa para declarar o implementar atributos o métodos que puedan reutilizar otras clases.

- Se declara como **Trait** en lugar de Class.
- No son instanciables. No tiene constructor.
- Se deben emplear con **Use** dentro de la clase que desea reutilizar el código.
- Los atributos de un Trait se comportan como propios de la clase que los use.
- Los métodos pueden ser declarativos mediante **abstract**
- O tener implementación “por defecto” con métodos desarrollados (código).
- Una clase puede usar múltiples Traits.

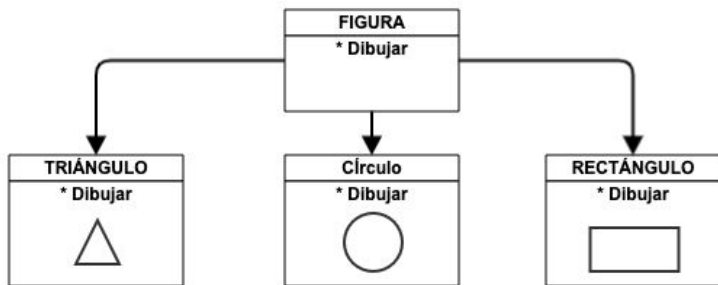


Polimorfismo mediante Traits

Mediante las penúltimas propiedades listadas:

- Los métodos pueden ser declarativos mediante **abstract**
- O tener implementación “por defecto” con **código por defecto**.

Tenemos que estos métodos se pueden “sobreescribir”, por lo que **Trait es otra herramienta para tener polimorfismo** (mismo nombre de método, diferente resultado)

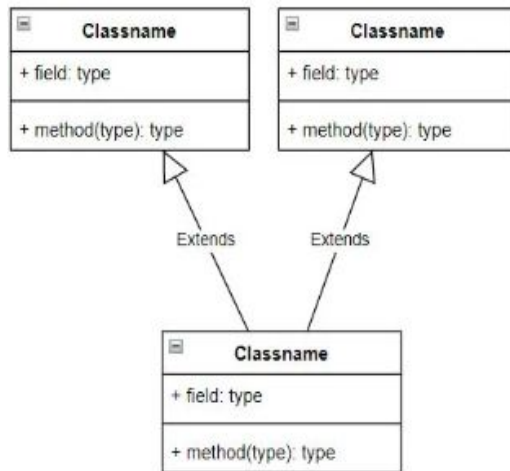


Herencia múltiple mediante Traits

Mediante la última propiedad listada:

- Una clase puede usar múltiples Traits.

Se puede conseguir replicar el funcionamiento de **herencia múltiple en PHP**.



Ejemplo Traits

```
trait Animal { // Trait Animal con metodo a utilizar
    public function haceSonido($sonido) {
        echo "Este animal hace: ".$sonido."<br>";
    }
}

trait Mamifero { // Trait Mamifero con metodo a REutilizar
    public function tienePelo() {
        echo "Este animal tiene pelo<br>";
    }
}

class Dog { // Clase Perro que usa los traits
    use Animal;
    use Mamifero;
    public function ladra() {
        $this->haceSonido("Guau");
    }
}

// Ejemplo de uso
$dog = new Dog();
$dog->ladra(); // Salida: Este animal hace: Guau
$dog->tienePelo(); // Salida: Este animal tiene pelo
$cat = new Cat();
$cat->maulla(); // Salida: Este animal hace: Miau
$dog->tienePelo(); // Salida: Este animal tiene pelo
```

Todo lo que está declarado dentro de *Trait* lo puede emplear la clase que haga *uso*, como si fuera propio.

Es decir, se puede llamar con **\$this->**

```
// Clase Gato que también usa los traits
class Cat {
    use Animal;
    use Mamifero;
    public function maulla() {
        $this->haceSonido("Miau");
    }
}
```

Interfaces vs Traits

Interfaces

- **Propósito: Definir métodos.**
- Implementación: Implementación no obligatoria, solo declara métodos. Sin atributos.
- Flexibilidad: Permite implementar múltiples interfaces en una clase.
- Polimorfismo: Facilita el uso de polimorfismo y desacoplamiento.
- Se utilizan cuando es necesario definir un método que las clases deben seguir, para **garantizar que ciertas clases implementen métodos específicos** y para **facilitar el intercambio de implementaciones y el desacoplamiento del código.**

Traits

- **Propósito: Reutilizar código.**
 - Implementación: Proporciona atributos e implementación de métodos.
 - Flexibilidad: Permite incluir métodos y atributos en varias clases.
 - Múltiples Traits: Se pueden usar múltiples traits en una clase.
 - Se utilizan cuando **es necesario compartir métodos y atributos concretos entre varias clases**, especialmente cuando estos **métodos tienen una implementación común que se desea reutilizar.**
- 