

Métodos y atributos estáticos y uso del operador ::

Static y ::

Referencias externas a clases sin instanciar

En POO, es posible definir **métodos y atributos estáticos** de forma que se pueda acceder a ellos sin la necesidad de instanciar la clase (crear el objeto).

```
public static $variable = "valor";
```

```
public static function funcion(){ //lógica }
```

El acceso a estos atributos o métodos se realiza mediante el operador ::



Ejemplo método y atributos static

```
class Calculadora {  
    public static $pi = 3.1416;  
    public static function sumar($a, $b) {  
        return $a + $b;  
    }  
    public static function calcularAreaCercle($radi) {  
        return $this->pi * pow($radi,2);  
    }  
}  
  
// Acceso a un atributo estatico  
echo "El valor de PI es: " . Calculadora::$pi . "<br>"; // El valor de PI es: 3.1416  
  
// Acceso a funcion estatica sin instanciar objeto  
echo Calculadora::sumar(5, 3); // 8  
  
echo "<br>El area de criculo con radio 2 és: " .  
    Calculadora::calcularAreaCercle(2);  
  
// El area de criculo con radio 2 és: 12.5664
```

Otros usos del operador :: Referencias internas

El operador **self::** permite acceder a métodos o propiedades **estáticas** de la misma clase. Es el mismo principio que usar **\$this->**

```
class MiClase {  
    private static $miAtributo = "miValor";  
    public function metodoPropio() {  
        return "valor de miAtributo: " . self::$miAtributo;  
    }  
    public function llamarMetodoPropio() {  
        return self::metodoPropio();  
    }  
}  
$varMiClase = new MiClase();  
echo $varMiClase->llamarMetodoPropio();  
// Salida: Valor de miAtributo: miValor
```

Otros usos del operador :: Late static binding

Cuando se trabaja con clases heredadas, **static::** se utiliza para acceder al contexto de **la clase en ejecución** (*late static binding*)

```
class ClaseBase {  
    public function metodoEstatico() {  
        return "Llamado desde ClaseBase";  
    }  
    public function llamar() {  
        return static::metodoEstatico();    // Usa la clase en ejecución  
    }  
}  
class ClaseDerivada extends ClaseBase {  
    public function metodoEstatico() {  
        return "Llamado desde ClaseDerivada";  
    }  
}  
$varClaseBase = new ClaseBase();  
$varClaseDerivada = new ClaseDerivada();  
echo $varClaseBase->llamar() . "<br>";    // Salida: Llamado desde ClaseBase  
echo $varClaseDerivada->llamar();    // Salida: Llamado desde ClaseDerivada
```

Otros usos del operador :: Constantes

Acceder a las **constantes** definidas en una clase.

```
class MiClase {  
    const MI_CONSTANTE = "Valor de la constante";  
}
```

```
echo MiClase::MI_CONSTANTE;    // Salida: Valor de la constante
```



Otros usos del operador :: métodos del padre

Llamar al **constructor o métodos de la clase padre**, dentro de una clase heredera. Ignora polimorfismo.

```
class ClasePadre {  
    public static function metodoPadre() {  
        return "Método de la clase padre";  
    }  
}  
class ClaseHija extends ClasePadre {  
    public function llamarPadre() {  
        return parent::metodoPadre();  
    }  
}
```

```
$obj = new ClaseHija();  
echo $obj->llamarPadre();    // Salida: Método de la clase padre
```



Otros usos del operador :: Nombre de la clase

Se usa con **::class** para **obtener el nombre completo de una clase** (incluido su namespace si lo tiene).

```
namespace MiNamespace;  
class MiClase {}
```

```
echo MiClase::class; // Salida: MiNamespace\MiClase
```



Resumen static y usos ::

1. Acceso EXTERNO a **métodos y atributos estáticos**. (**STATIC**)
2. Referencias INTERNAS a métodos o propiedades **propias** con `self::` (**STATIC**)
3. Uso de **late static binding** con `static::` (**STATIC**)
4. Acceso a **constantes de clase**
5. Llamadas a métodos de la **clase padre** con `parent::`
6. Obtener el **nombre completo de la clase** con `::class`



Ejemplo de uso: combinar self::class

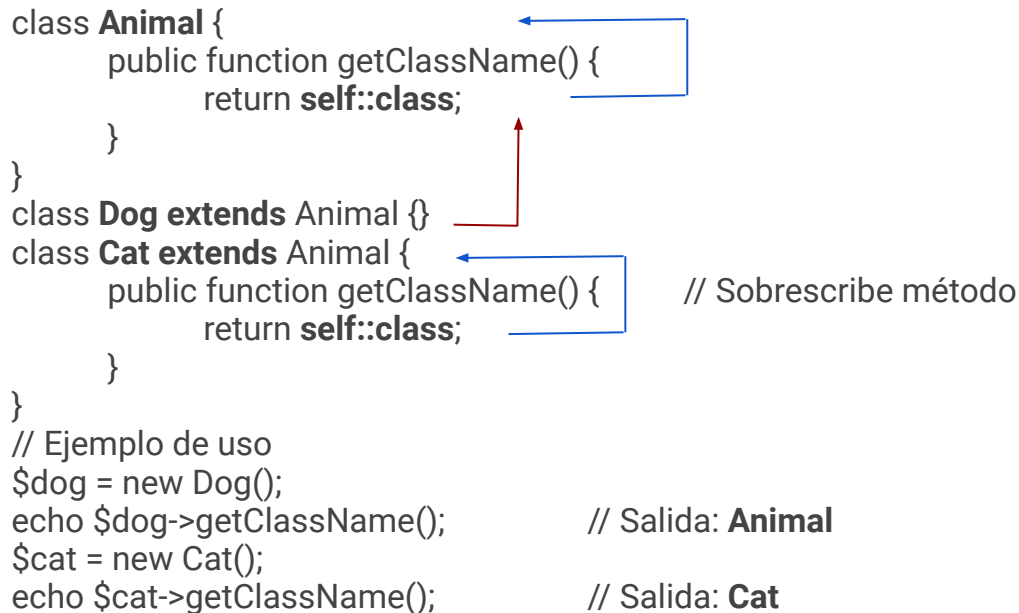
Mediante un método *return self::class*; podemos **obtener el nombre de la clase**. También sucede si se hereda (devuelve el nombre de la clase padre).

Sí el método se implementa en un trait, es como si la clase tuviera el método.



Ejemplo de uso: combinar self::class con HERENCIA

```
class Animal {  
    public function getClass_name() {  
        return self::class;  
    }  
}  
class Dog extends Animal {}  
class Cat extends Animal {  
    public function getClass_name() {  
        return self::class;  
    }  
}  
// Ejemplo de uso  
$dog = new Dog();  
echo $dog->getClass_name();           // Salida: Animal  
$cat = new Cat();  
echo $cat->getClass_name();           // Salida: Cat
```

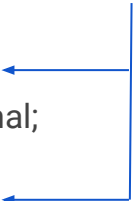


Se emplean los métodos con los datos de la clase origen



Ejemplo de uso: combinar self::class con TRAITS

```
trait Animal {  
    public function getClass_name() {  
        return self::class;  
    }  
}  
class Dog {  
    use Animal;  
}  
class Cat {  
    use Animal;  
}
```



```
// Ejemplo de uso  
$dog = new Dog();  
echo $dog->getClass_name();           // Salida: Dog  
$cat = new Cat();  
echo $cat->getClass_name();           // Salida: Cat
```

Se emplean los métodos del trait con los datos de la clase que usa el trait.

El valor viene después de la implementación del método.