



PHP

+ OOP
Clase 2

PHP

Herencia



Las clases pueden ser **extensiones** de otras clases. La clase extendida puede usar todas las variables y funciones de la clase base, siempre y cuando no sean privados

A esto llamamos 'Herencia'.



Extends

```
class Silla
{
    private $color;
    private $resistenciaKG;

    public function resiste($peso)
    {
        return $this->resistenciaKG <= $peso;
    }
}
```

Extends

```
class SillaDeRuedas extends Silla
{
    private $cantidadRuedas;

    public function setCantidadRuedas($cantidad)
    {
        $this->cantidadRuedas = $cantidad;
    }
}
```

La clase SillaDeRuedas EXTIENDE Silla!



En algún punto ***debería de existir una restricción en la implementación de métodos y propiedades*** que refleje la realidad, y en OOP una de las herramientas que tenemos son...

PHP

Clases abstractas

Abstract Classes

- No se pueden instanciar.
- Si heredamos de una clase abstracta, todos sus métodos abstractos deberán ser definidos (al igual que en Interfaces).
- Si el método abstracto está definido como protegido, la implementación de la función debe ser definida como protegida o pública, pero nunca como privada.

Clase abstracta

Este va a ser mi molde para clientes, sin importar el tipo de cliente que sea

Atencion al scope!

```
abstract class Cliente
{
    protected $cuenta;
    protected $email;
    protected $password;

    public function __construct($cuenta,
                                $email, $password)
    {
        //...
    }
}
```



Abstract Class

Scope

Veníamos haciendo uso de **Public** y **Private**... Ahora también tenemos **PROTECTED**!

"A los miembros de clase declarados como 'protected' se puede acceder solo desde la misma clase o mediante clases heredadas."

Ejemplo

```
abstract class Cliente
{
    protected $cuenta;
    protected $email;
    protected $password;
```

```
    public function __construct($cuenta,
                                $email, $password)
    {
        //...
    }
}
```

```
class Persona extends Cliente {
    private $nombre;
    private $apellido;
    private $dni;

    public function __construct($cuenta, $email, $password,
                                $nombre, $apellido, $dni)
    {
        //...
    }
}
```

```
class Pyme extends Cliente
{
    private $razonSocial;
    private $cuil;

    public function __construct($cuenta, $email, $password,
                                $razonSocial, $cuil)
    {
        //...
    }
}
```



Parent

Al utilizar **parent**, hacemos referencia al objeto padre de nuestra instancia. Esto permite llamar a métodos de esa clase para ejecutar su comportamiento, por más de que hayamos sobrescrito el método.

Parent

```
class Pyme extends Cliente
{
    private $razonSocial;
    private $cuil;

    public function __construct($cuenta, $email, $password, $razonSocial, $cuil)
    {
        parent::__construct($cuenta, $email, $password);

        $this->razonSocial = $razonSocial;
        $this->cuil = $cuil;
    }
}

$nuevaPyme = new Pyme("10-22234/0", "pyme@dh.com", "123456", "Digital House", "30-12334546-0");
```



Parent


El ejemplo anterior muestra uno de los casos más comunes para el uso de **parent**. Pero también se puede utilizar en cualquier método para llamar a un método del padre.

No es necesario que el método llamante sea el que sobrescribe al padre.



Interfaces

Las interfaces permiten crear contratos entre el implementador de la clase y el utilizador.



Se especifica qué métodos **deben** ser implementados por una clase, sin tener que definir cómo estos métodos son implementados.



Entre otras cosas, **nos permiten hacer algo llamado Polimorfismo.**



Interfaces / Implements

Ejemplo

```
interface Liquidable {  
    public function liquidarHaberes($persona, $monto);  
}
```

Y cómo lo implementamos en nuestra clase Pyme?

```
class Pyme extends Cliente implements Liquidable
```

Interfaces

```
class Pyme extends Cliente implements Liquidable
{
    private $razonSocial;
    private $cuil;

    public function __construct($cuenta, $email, $password, $razonSocial, $cuil)
    {
        parent::__construct($cuenta, $email, $password);

        $this->razonSocial = $razonSocial;
        $this->cuil = $cuil;
    }
    //...
    public function liquidarHaber($persona, $monto)
    {
        $persona->getCuenta()->acreditar($monto);
        $this->cuenta->debitar($monto);
    }
}
```

instanceof

Es un constructor de PHP que nos permite preguntar si un objeto es de un tipo particular, es decir, cuál es su Clase.

```
if($nuevaPyme instanceof Pyme) {  
    echo "Soy una Pyme!";  
}
```


PHP

Static

- Declarar propiedades o métodos de clases como estáticos los hacen accesibles sin la necesidad de instanciar la clase.
- Una **propiedad** declarada como static no puede ser accedida con un objeto de clase instanciado (aunque un **método** estático sí lo puede hacer).

Propiedades estáticas

```
class Colectivo {  
    public static $precios = [11.00, 11.50, 11.75];  
}  
  
var_dump(Colectivo::$precios);
```

Métodos estáticos

```
class Colectivo {  
    public static $precios = [11.00, 11.50, 11.75];  
  
    public static function getPrecios($distancia) {  
        if($distancia < 5) {  
            return Colection::$precios[0];  
        } elseif($distancia < 10) {  
            return Colection::$precios[1];  
        } else {  
            return Colection::$precios[2];  
        }  
    }  
}  
  
Colectivo::getPrecio(20);
```

¡A practicar!



```
<?php  
    echo "Hora de practicar!";  
?>
```

¡Gracias!



¿Preguntas?