unidad didáctica 3

PHP Orientado a Objetos

Índice

1. duración y criterios de evaluación

Duración estimada: 18 sesiones

Resultado de aprendizaje y criterios de evaluación:

5. Desarrolla aplicaciones Web identificando y aplicando mecanismos para separar el código de presentación de la lógica de negocio.

a) Se han identificado las ventajas de separar la lógica de negocio de los aspectos de presentación de la aplicación.

b) Se han analizado tecnologías y mecanismos que permiten realizar esta separación y sus características principales.

c) Se han utilizado objetos y controles en el servidor para generar el aspecto visual de la aplicación web en el cliente.

d) Se han utilizado formularios generados de forma dinámica para responder a los eventos de la aplicación Web.

e) Se han escrito aplicaciones Web con mantenimiento de estado y separación de la lógica de negocio.

f) Se han aplicado los principios de la programación orientada a objetos.

g) Se ha probado y documentado el código.

2. clases y objetos

PHP sigue un paradigma de programación orientada a objetos (POO) basada en clases.

Un clase es un plantilla que define las propiedades y métodos para poder crear objetos. De este manera, un objeto es una instancia de una clase.

Tanto las propiedades como los métodos se definen con una visibilidad (quien puede acceder)

- Privado - `private`: Sólo puede acceder la propia clase.
- Protegido - `protected`: Sólo puede acceder la propia clase o sus descendientes.
- Público - `public`: Puede acceder cualquier otra clase.

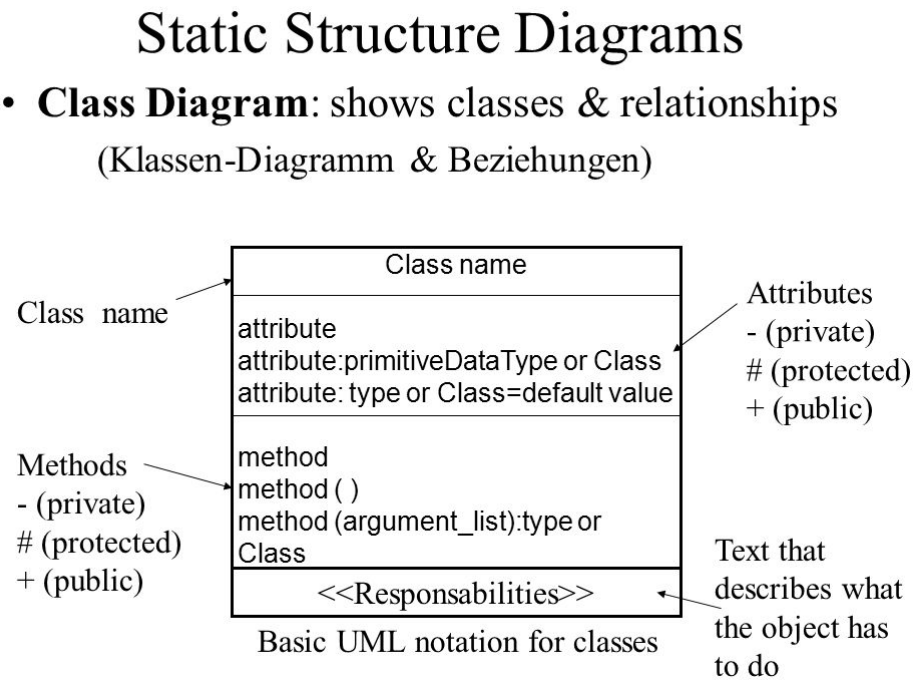
Para declarar una clase, se utiliza la palabra clave `class` seguido del nombre de la clase. Para instanciar un objeto a partir de la clase, se utiliza `new`:

```php
<?php
class NombreClase {
// propiedades
// y métodos
}

$ob = new NombreClase();
```

> **clases con mayúscula**
>
> Todas las clases empiezan por letra mayúscula.

Cuando un proyecto crece, es normal modelar las clases mediante UML (¿recordáis *Entornos de Desarrollo*?). La clases se representan mediante un cuadrado, separando el nombre, de las propiedades y los métodos:



Una vez que hemos creado un objeto, se utiliza el operador `->` para acceder a una propiedad o un método:

```
$objeto->propiedad;
$objeto->método(parámetros);
```

Si desde dentro de la clase, queremos acceder a una propiedad o método de la misma clase, utilizaremos la referencia `$this`;

```
$this->propiedad;
$this->método(parámetros);
```

Así pues, como ejemplo, codificaríamos una persona en el fichero `Persona.php` como:

```
<?php
class Persona {
    private string $nombre;

    public function setNombre(string $nom) {
        $this->nombre=$nom;
    }

    public function imprimir(){
        echo $this->nombre;
        echo '<br>';
    }
}

$bruno = new Persona(); // creamos un objeto
$bruno->setNombre("Bruno Díaz");
$bruno->imprimir();
```

Aunque se pueden declarar varias clases en el mismo archivo, es una mala práctica. Así pues, cada fichero contedrá una sola clase, y se nombrará con el nombre de la clase.

3. encapsulación

Las propiedades se definen privadas o protegidas (si queremos que las clases heredadas puedan acceder).

Para cada propiedad, se añaden métodos públicos (getter/setter):

```
public setPropiedad(tipo $param)
public getPropiedad() : tipo
```

Las constantes se definen públicas para que sean accesibles por todos los recursos.

```
<?php
class MayorMenor {
    private int $mayor;
    private int $menor;

    public function setMayor(int $may) {
        $this->mayor = $may;
    }

    public function setMenor(int $men) {
        $this->menor = $men;
    }

    public function getMayor() : int {
        return $this->mayor;
    }

    public function getMenor() : int {
        return $this->menor;
    }
}
```

3.1. Recibiendo y enviando objetos

Es recomendable indicarlo en el tipo de parámetros. Si el objeto puede devolver nulos se pone `?` delante del nombre de la clase.

> **objetos por referencia**
>
> Los objetos que se envían y reciben como parámetros siempre se pasan por referencia.

```
<?php
function maymen(array $numeros) : ?MayorMenor {
    $a = max($numeros);
    $b = min($numeros);

    $result = new MayorMenor();
    $result->setMayor($a);
    $result->setMenor($b);
```

```php
}

$resultado =  maymen([1,76,9,388,41,39,25,97,22]);
echo "<br>Mayor: ".$resultado->getMayor();
echo "<br>Menor: ".$resultado->getMenor();
```

4. constructor

El constructor de los objetos se define mediante el método mágico `__construct`. Puede o no tener parámetros, pero sólo puede haber un único constructor.

```php
<?php
class Persona {
    private string $nombre;

    public function __construct(string $nom) {
        $this->nombre = $nom;
    }

    public function imprimir(){
      echo $this->nombre;
      echo '<br>';
    }
}

$bruno = new Persona("Bruno Díaz");
$bruno->imprimir();
```

4.1. constructores en PHP8

Una de las grandes novedades que ofrece PHP 8 es la simplificación de los constructores con parámetros, lo que se conoce como promoción de las *propiedades del constructor*.

Para ello, en vez de tener que declarar las propiedades como privadas o protegidas, y luego dentro del constructor tener que asignar los parámetros a estás propiedades, el propio constructor promociona las propiedades.

Veámoslo mejor con un ejemplo. Imaginemos una clase `Punto` donde queramos almacenar sus coordenadas:

```php
<?php
class Punto {
    protected float $x;
    protected float $y;
    protected float $z;

    public function __construct(
        float $x = 0.0,
        float $y = 0.0,
        float $z = 0.0
    ) {
        $this->x = $x;
        $this->y = $y;
        $this->z = $z;
    }
}
```

```php
<?php
class Punto {
    public function __construct(
        protected float $x = 0.0,
        protected float $y = 0.0,
        protected float $z = 0.0,
    ) {}
}
```

el orden importa:

A la hora de codificar el orden de los elementos debe ser:

```php
<?php
declare(strict_types=1);

class NombreClase {
    // propiedades

    // constructor

    // getters - setters

    // resto de métodos
}
?>
```

5. clases estáticas

Son aquellas que tienen propiedades y/o métodos estáticos (también se conocen como de *clase*, por que su valor se comparte entre todas las instancias de la misma clase).

Se declaran con `static` y se referencian con `::`.

- Si queremos acceder a un método estático, se antepone el nombre de la clase: `Producto::nuevoProducto()`.
- Si desde un método queremos acceder a una propiedad estática de la misma clase, se utiliza la referencia `self: self::$numProductos`.

```php
<?php
class Producto {
    const IVA = 0.23;
    private static $numProductos = 0;

    public static function nuevoProducto() {
        self::$numProductos++;
    }
}

Producto::nuevoProducto();
$impuesto = Producto::IVA;
```

También podemos tener clases normales que tengan alguna propiedad estática:

```php
<?php
class Producto {
    const IVA = 0.23;
    private static $numProductos = 0;
    private $codigo;

    public function __construct(string $cod) {
        self::$numProductos++;
        $this->codigo = $cod;
    }

    public function mostrarResumen() : string {
        return "El producto ".$this->codigo." es el número ".self::$numProductos;
    }
}

$prod1 = new Producto("PS5");
$prod2 = new Producto("XBOX Series X");
$prod3 = new Producto("Nintendo Switch");
echo $prod3->mostrarResumen();
```

6. introspección

Al trabajar con clases y objetos, existen un conjunto de funciones ya definidas por el lenguaje que permiten obtener información sobre los objetos:

- `instanceof`: permite comprobar si un objeto es de una determinada clase
- `get_class`: devuelve el nombre de la clase
- `get_declared_class`: devuelve un array con los nombres de las clases definidas
- `class_alias`: crea un alias
- `class_exists` / `method_exists` / `property_exists`: true si la clase / método / propiedad está definida
- `get_class_methods` / `get_class_vars` / `get_object_vars`: Devuelve un array con los nombres de los métodos / propiedades de una clase / propiedades de un objeto que son accesibles desde dónde se hace la llamada.

Un ejemplo de estas funciones puede ser el siguiente:

```php
<?php
$p = new Producto("PS5");
if ($p instanceof Producto) {
    echo "Es un producto";
    echo "La clase es ".get_class($p);

    class_alias("Producto", "Articulo");
    $c = new Articulo("Nintendo Switch");
    echo "Un articulo es un ".get_class($c);

    print_r(get_class_methods("Producto"));
    print_r(get_class_vars("Producto"));
    print_r(get_object_vars($p));

    if (method_exists($p, "mostrarResumen")) {
        $p->mostrarResumen();
    }
}
```

clonado

Al asignar dos objetos no se copian, se crea una nueva referencia. Si queremos una copia, hay que clonarlo mediante el método clone(object) : object.

Si queremos modificar el clonado por defecto, hay que definir el método mágico __clone() que se llamará después de copiar todas las propiedades.

Más información en https://www.php.net/manual/es/language.oop5.cloning.php

7. referencias

- Manual de PHP
- Manual de OO en PHP - www.desarrolloweb.com