Desarrollo Web en Entorno Servidor

4.- Objetos en PHP

IES Severo Ochoa

Índice

- Objeto
- Encapsulación
- Herencia
- Clases abstractas
- Interfaces

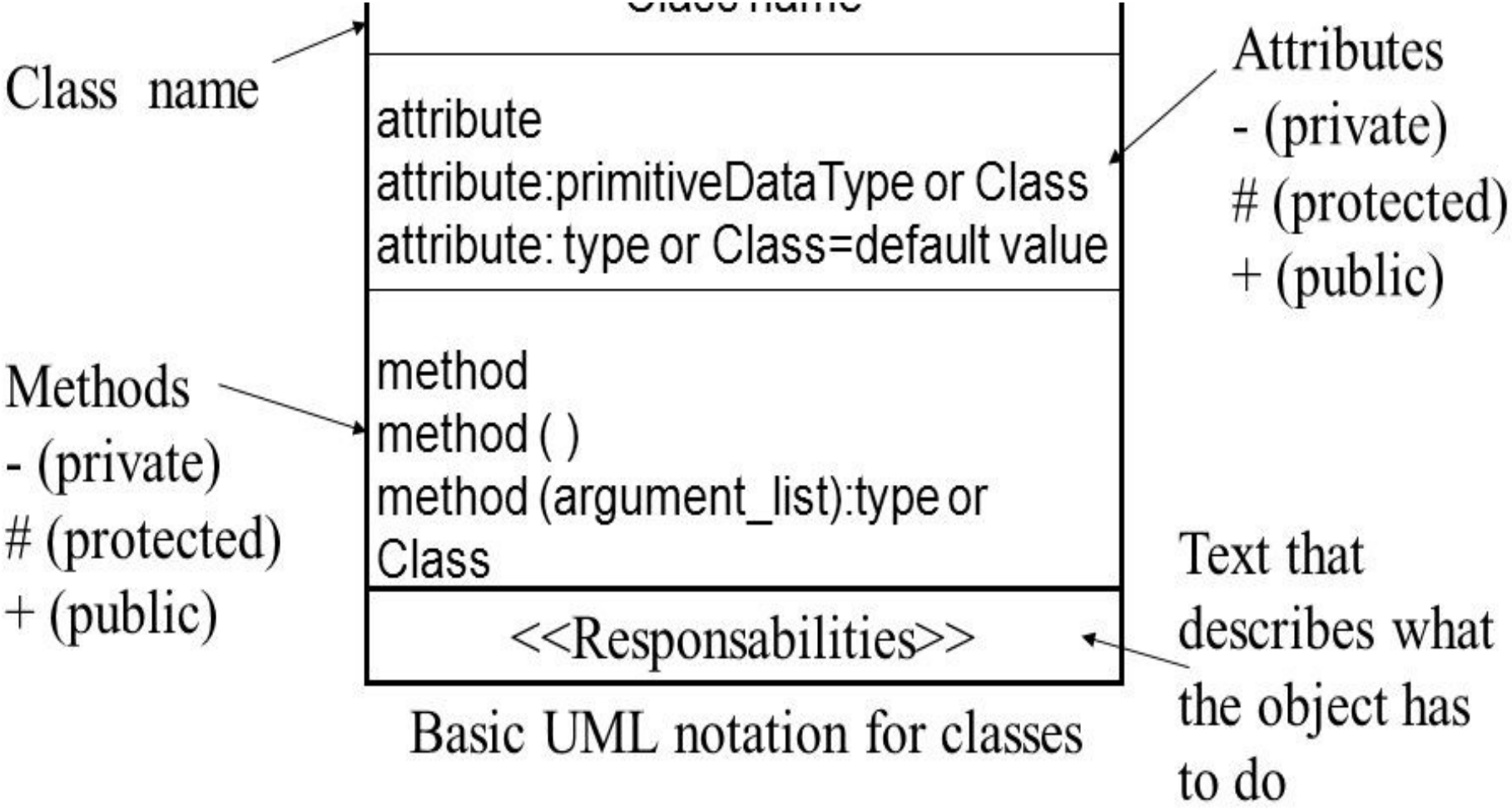
Clase

- Elementos
  - Propiedades
  - Métodos
- Visibilidad
  - Privado
    - `private`
  - Protegido
    - `protected`
  - Público
    - `public`

```
class NombreClase {
    // propiedades
    // y métodos
}

$ob = new NombreClase();
```

Recordando UML



Basic UML notation for classes

Invocando

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

- Si desde dentro de la clase, queremos acceder a una propiedad o método de la misma clase

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

Ejemplo Persona

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

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

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

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

Encapsulación de propiedades

ies severo ochoa

- 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($param)
public getPropiedad()
```

- Las constantes se definen públicas

Ejemplo getter/setter

```php
class MayorMenor {
    private $mayor;
    private $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;
    }
}
```

https://marketplace.visualstudio.com/items?itemName=phproberto.vscode-php-getters-setters

Devolviendo objetos

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

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

    return $result;
}

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

Constructor

- `__construct`
- Puede o no tener parámetros
- Sólo puede haber un constructor.

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

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

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

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

Clases estáticas

- Propiedades/métodos estáticos (o de clase)
  - se declaran con `static`
  - se referencian con `::`

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

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

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

Clase con parte static

```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();
?>
```

Funciones predefinidas

- `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.

Probando...

```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();
    }
}
?>
```

Copiando objetos

ies severo ochoa

- Al asignar dos objetos no se copian, se crea una nueva referencia
- Hay que clonarlo

```
clone(object) : object
```

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

Serializando objetos

ies severo ochoa

- Si necesitamos almacenar un objeto, ya sea en la sesión o en una BBDD.
  - `serialize(object): string`
  - `unserialize(string): object`
- Otra forma es crear su representación JSON:
  - `json_encode(mixed): string`
  - `json_decode(string): mixed`

Herencia

- PHP soporta herencia simple
- `class A extends B`
- El hijo hereda los atributos y métodos públicos y protegidos

- `get_parent_class(object): string`
- `is_subclass_of(object, string): bool`

Ejemplo Herencia

```php
class Producto {
   public $codigo;
   public $nombre;
   public $nombreCorto;
   public $PVP;
   public function mostrarResumen() {
          print "<p>Prod:".$this->codigo."</p>";
   }
}
```

```php
class Tv extends Producto {
        public $pulgadas;
        public $tecnologia;
}

$t = new Tv();
$t->codigo = 33;
if ($t instanceof Producto) {
        echo $t->muestra();
}

$padre = get_parent_class($t);
echo "<br>La clase padre es: " . $padre;
$objetoPadre = new $padre;
echo $objetoPadre->mostrarResumen();

if (is_subclass_of($t, 'Producto')) {
        echo "<br>Soy un hijo de Producto";
}
```

Sobreescribir métodos

- Método en los hijos con el mismo nombre
- Se puede invocar a los métodos del padre
  - `parent::nombreMetodo()`

```
class Tv extends Producto {
   public $pulgadas;
   public $tecnologia;

   public function mostrarResumen() {
      parent::muestra();
      echo "<p>TV ".$this->tecnologia." de ".$this->pulgadas."</p>";
   }
}
```

Constructor en hijos

- En los hijos no se crea ningún constructor de manera automática.
  - Si no lo hay, se invoca automáticamente al del padre
- Si lo definimos en el hijo, hemos de invocar al del padre de manera explícita

```php
public function __construct(string $codigo, int $pulgadas, string $tecnologia) {
    parent::__construct($codigo);
    $this->pulgadas = $pulgadas;
    $this->tecnologia = $tecnologia;
}
```

abstract

- Obliga a heredar de una clase
- No se permite su instanciación

  abstract class NombreClase {
- Una clase abstracta puede contener propiedad y métodos no-abstractos, y/o métodos abstractos.

```php
// Clase abstracta
abstract class Producto {
  private $codigo;
  public function getCodigo() : string {
    return $this->codigo;
  }
  // Método abstracto
  abstract public function mostrarResumen();
}
```

Ejemplo clase abstracta

- Cuando una clase hereda de una clase abstracta, obligatoriamente debe implementar los métodos marcados como abstractos.

```
class Tv extends Producto {
   public $pulgadas;
   public $tecnologia;

   public function mostrarResumen() { //obligado a implementarlo
      echo "<p>Código ".$this->getCodigo()."</p>";
      echo "<p>TV ".$this->tecnologia." de ".$this->pulgadas."</p>";
   }
}

$t = new Tv();
echo $t->getCodigo();
```

final

- Opuesto a `abstract`
- Evita que se pueda heredar una clase o método

```php
class Producto {
  private $codigo;
  public function getCodigo() : string {
    return $this->codigo;
  }

  final public function mostrarResumen() : string {
    return "Producto ".$this->codigo;
  }
}

final class Microondas extends Producto {
  private $potencia;
  public function getPotencia() : int {
    return $this->potencia;
  }
}
```

Interfaces

- Permite definir un contracto con las firmas de los métodos a cumplir
- Sólo contiene declaraciones de funciones
  - Todas las funciones públicas

```
interface Nombreable {
   // declaración de funciones
}
class NombreClase implements NombreInterfaz {
   // código de la clase
```

- Permite herencia de interfaces
- Una clase puede implementar varios interfaces

Ejemplo interfaces

```
interface Mostrable {
  public function mostrarResumen() : string;
}

interface MostrableTodo extends Mostrable {
  public function mostrarTodo() : string;
}

interface Facturable {
  public function generarFactura() : string;
}

class Producto implements MostrableTodo, Factura
ble {
  // Implementaciones de los métodos

}
```

¿Alguna pregunta?