ALCYON
IT Services

POO

Programación orientada a objetos
Parte I: Introducción

ALCYON
IT Services

¿QUÉ ES LA ORIENTACIÓN A OBJETOS?

ALCYON
IT Services

INTRODUCCIÓN

Un **objeto** es una encapsulación de **datos** y **operaciones** que permiten modelar un objeto del mundo (realmente creamos clases a partir de las cuales se crean objetos).

Los datos se ocultan dentro del objeto, el cual presentará al exterior sólo unas operaciones para trabajar (interfaz del objeto -ej si utilizamos el simil de un interruptor estas operaciones serían on y off)).

Los objetos se especializan según el tipo de mundo que modelen: entidades de gestión (clientes, proveedores, etc), elementos visuales, tareas de control, etc.

ALCYON
IT Services

INTRODUCCIÓN

La OO consiste en organizar y agrupar los componentes normales de cualquier sistema informático (datos y procedimientos) haciendo énfasis no en los procesos a realizar sino en las entidades u objetos que participan así como la relación entre ellos (identificar y relacionar que clases intervienen).

ALCYON
IT Services

PROGRAMACIÓN CLÁSICA VS OOP

En el desarrollo de un sistema mediante la OO no nos preguntamos qué ha de hacer el sistema sino ¿cuántos objetos hay?¿qué relaciones hay entre ellos?

Ventaja: desarrollar el sistema de esta manera garantiza que será flexible en el futuro por no estar condicionado por ninguna función concreta.

ALCYON
IT Services

CLASE

Descripción de una entidad (por ejemplo, “Coche”) mediante una encapsulación de datos y procedimientos.

Los procedimientos (métodos) representan el comportamiento visible de las entidades (clases).

A partir de una clase se pueden crear tantas entidades (objetos) como se quiera.

Una clase es como una plantilla de definición de entidades.

OBJETO I

Un elemento creado (o instanciado) a partir de una clase.

También se dice que un objeto es una "concretización" de una clase. Por ejemplo, “Seat Leon" es un objeto (concreto) de la entidad (la clase) "Coche“.

INSTANCIA: Un objeto es una instancia de una clase.

Un objeto tiene:

identidad (implica que cada objeto es único y cada uno de ellos ocupa una zona de memoria diferente),

estado (el valor en particular que ese objeto tiene en sus datos)

comportamiento (los métodos del objeto que modifican los datos de éste).

ALCYON
IT Services

OBJETO II

Un objeto es una instancia de una clase de forma que todos los objetos de una clase tienen todas las propiedades y métodos de la clase pero con diferentes valores

```
class Persona{}
$persona1 = new Persona();
```

Clases
Son generadoras de objetos.
Por convenio, los nombres de clase comienzan siempre con mayúscula.
Objetos
Un objeto tiene **propiedades** y **métodos**.
Una persona tiene **nombre, edad, altura**, etc y puede **saludar**, **correr, saltar,** etc.
Por convenio, los nombres de los objetos comienzan por minúscula y si tienen un nombre compuesto, utilizan mayúsculas para separar las letras del mismo.

ALCYON
IT Services

ATRIBUTO I

Una propiedad de un objeto (aunque veremos que también hay atributos de clase, es decir, atributos genéricos de la clase que no pertenece en exclusiva a ningún objeto instanciado. Ej un contador de alumnos).

Por ejemplo, el atributo “modelo” puede almacenar el nombre comercial de un coche.

ATRIBUTO II

```
class Persona{
    public $nombre='Arthur';
}

$persona1 = new Persona();
echo $persona1->nombre;
```

no se pone $nombre ya que nos referimos a la propiedad de una instancia y no a una variable de php

000 Ver

ATRIBUTO III

Modificar una propiedad:

```
class Persona{
    public $nombre='Arthur';
}

$persona1 = new Persona();
echo $persona1->nombre."<br/>";

$persona1->nombre = 'David';
echo $persona1->nombre;
```

000 Ver

ALCYON
IT Services

MENSAJE

Forma de comunicación entre dos objetos para pedir la ejecución de un servicio. Similar a la llamada a función o procedimiento (es la invocación a un método de una clase).

Por ejemplo, una clase llamada “Circuito” puede utilizar objetos “Coche” y enviarles a estos el mensaje arrancar():

METODO I

Llamada a un **método** de una clase:

```
class Persona{
    public $nombre = 'Arthur';

    public function saludar(){
        echo "Hola Mundo";
    }
}

$persona = new Persona();
$persona->saludar();
```

000 Ver

METODO II

Utilizar una propiedad dentro de la clase:

Para utilizar una propiedad dentro de la propia clase tendremos que acceder a ella mediante el objeto $this

```
class Persona{
    public $nombre = 'Arthur';

    public function saludar(){
        echo "Hola ".$this->nombre;
    }
}

$persona = new Persona();
$persona->saludar();
```

000 Ver

ALCYON
IT Services

HERENCIA I

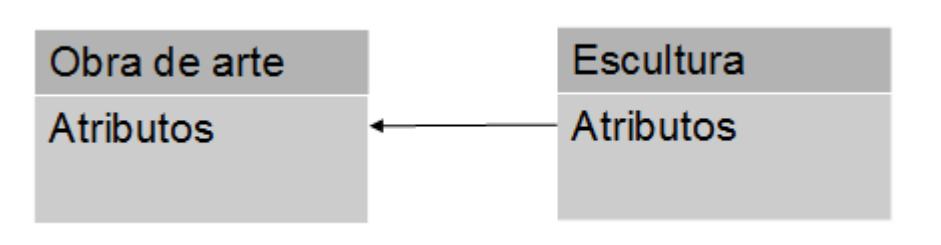
Una técnica que permite la definición de una clase a partir de otra considerando sólo las diferencias entre ellas. Permite la reutilización ya que puede haber instancias de la clase que, además de las caracterísiticas de la clase base (superclase) pueden tener otras exclusivas que no tienen porque compartir otras instancias de la misma clase. (ej: superclase coche puede tener clases del tipo: deportivos, berlinas, etc...)

Conseguimos que la nueva clase (la que hereda) disponga de la implementación de la original.

HERENCIA II

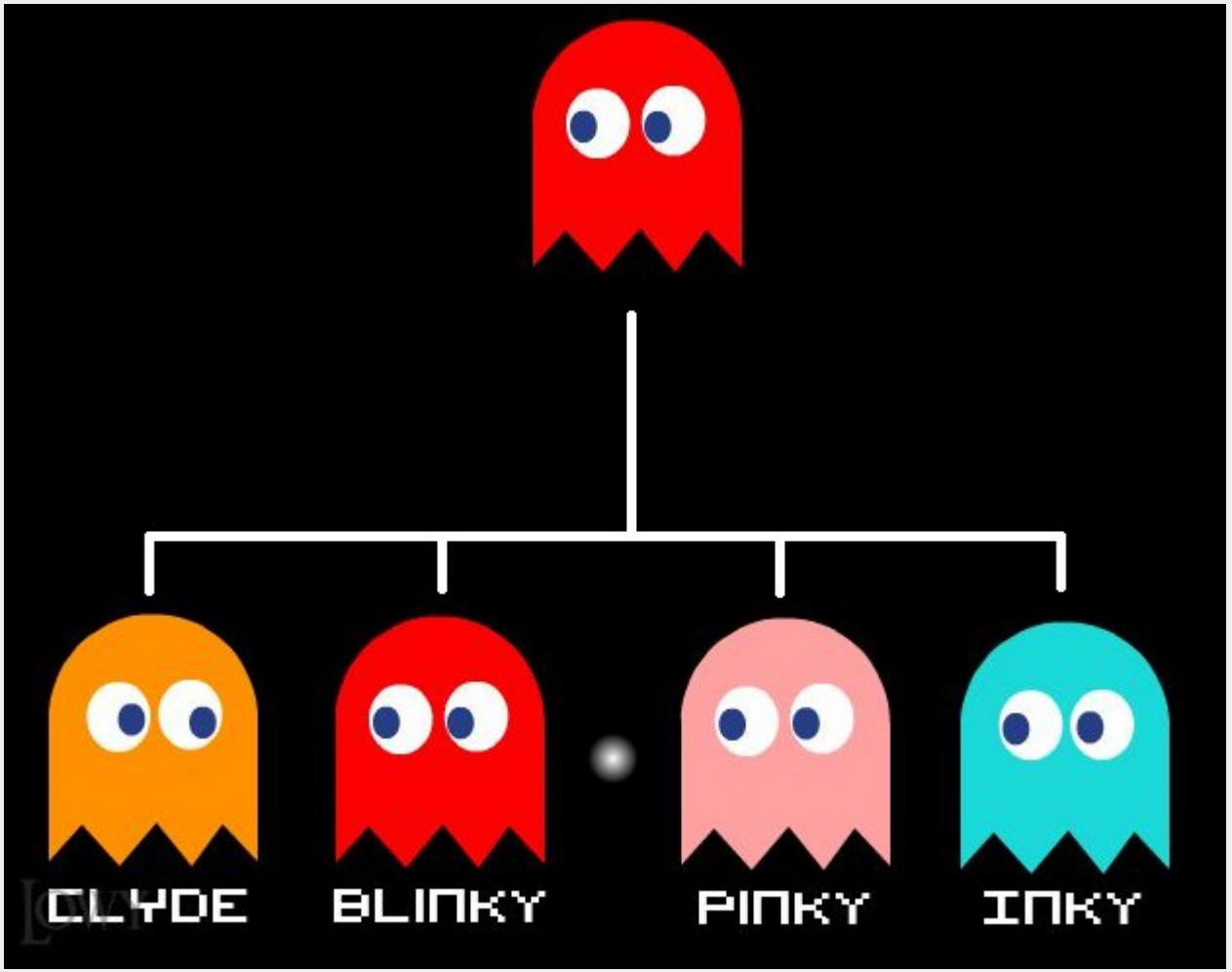
Tenemos la clase ‘ObraDeArte’ y queremos definir la clase ‘Escultura’ como subclase de ‘ObraDeArte’.

Una escultura **es una** obra de arte.



ALCYON
IT Services

EJEMPLO DE HERENCIA



ALCYON
IT Services

VISIBILIDAD PROPIEDADES I

Vamos a utilizar modificadores de acceso (definen la accesibilidad tanto de propiedades y metodos):

```
class Persona{
    public $a=1;
    protected $b=2;
    private $c=3;
}
$persona = new Persona();
echo $persona->a;
echo $persona->b; //error al imprimir
echo $persona->c; //error al imprimir
```

002 Ver

VISIBILIDAD PROPIEDADES I

Si creamos un método público que lee los atributos del objeto vemos como si que se visualizan los tres valores:

```
class Persona{
	public $a=1;
	protected $b=2;
	private $c=3;

	public function mostrar() {
		echo $this->a.'<br>';
		echo $this->b.'<br>';
		echo $this->c.'<br>';
	}
}
$persona = new Persona();
$persona->mostrar();
```

002 Ver

HERENCIA Y VISIBILIDAD

Vamos a crear una clase adolescente que herede los métodos y propiedades de la clase persona

```
class Persona{
      …/...
}
class Adolescente extends Persona {
      public $d = 4; //añadimos un nuevo atributo específico de adolescente

      public function mostrar() {
            echo $this->a.'<br>'; //se visualiza por ser public
            echo $this->b.'<br>'; //se visualiza por ser protected
            echo $this->c.'<br>'; //no se visualiza por ser private
            echo $this->d.'<br>'; //se visualiza por ser de la propia clase
      }
}
$adolescente = new Adolescente();
$persona->mostrar();
```

004 Ver

ALCYON
IT Services

POLIMORFISMO

Siguiendo con el ejemplo anterior vamos a instanciar un objeto de tipo persona y un objeto de tipo adolescente:

```
$persona1 = new Persona();
$persona2 = new Adolescente();
```

Ambos objetos contienen un método mostrar() y podemos ver como se ejcuta siempre el que corresponde a cada tipo de objeto:

```
$persona1->mostrar(); //se ejecuta el de la clase Persona
$persona2->mostrar(); //se ejecuta el de la clase Adolescente
```

Si eliminamos el método mostrar() de la clase adolescente veremos como se ejecuta entonces el que corresponde a la clase Persona (gracias a la herencia)

006 Ver

ALCYON
IT Services

BASES DE LA OO I

La OO se basa en la integración de seis conceptos o tecnologías:

Abstracción. Modelado de la estructura de clases de una aplicación

Clasificación. Crear una clase a partir de la simplificación que se hace del objeto

Encapsulación. Consiste en concentrar atributos y métodos

Ocultación de la información. Consiste en identificar que se puede ver y que o (cualificadores public, private, ...)

Herencia. Una subclase puede utilizar los métodos y heredar los atributos de la superclase

Polimorfismo. Permite que cada clase dentro de una jerarquia de herencia utilice los métodos que le corresponden en cada momento (no habrá necesidad de añadir IF para decidir que tipo de objeto estamos tratando)

ALCYON
IT Services

ENCAPSULACIÓN I

Al descomponer en módulos, tenemos que cada uno de ellos es una encapsulación de datos y procedimientos. NO se permite tener operaciones o datos desligados de clases

Los módulos (clases) se comunican entre ellos (paso de parámetros y valores de retorno) a través de mensajes (invocaciones a métodos).

Importante: La encapsulación por si sola no garantiza la ocultación.

ALCYON
IT Services

ENCAPSULACIÓN II

ACCESO A METODOS

- como norma generar las propiedades se definen como private y los metodos como public para evitar que se cambien propiedades si no es utilizando métodos.
- Esto es la base del encapsulamiento.
- Al no permitir que se modifique directamente el valor de las propiedades evitamos que se asigne un valor no permitido (usaremos un método específico que se encargará de validar el dato antes de asignarlo a la propiedad del objeto)

ENCAPSULACIÓN III

En este ejemplo vemos como encapsular las propiedades para que no sean visibles en el exterior de la clase y habilitamos unos métodos públicos para poder consultar y modificar el valor de cada una de ellas

```
class Persona {
	private $nombre;
	private $apellidos;

	//métodos getters
	public function getNombre() { return $this->nombre; }
	public function getApellidos() { return $this->apellidos; }

	//métodos setters
	public function setNombre($n) { $this->nombre = $n; }
	public function setApellidos($a) { $this->apellidos = $a; }
}
```

008 Ver

ALCYON
IT Services

OCULTACIÓN

Consiste en ocultar al exterior la estructura física de los datos y la implementación de las operaciones.

Sólo se permite ver lo que se denomina la **interfaz**, es decir, las operaciones que permiten a otras clases (clases clientes) utilizar con aprovechamiento la clase actual.

Aunque cambie la implementación (de los métodos), al no variar la interfaz se tiene gran independencia entre diferentes partes de un programa.

ALCYON
IT Services

ABSTRACCIÓN I

La abstracción permite que dispongamos de las características del objeto que necesitemos (simplificación de la realidad para aislar solo la información que realmente es útil para la operativa que se desea implementar).

Por ejemplo, si necesitamos el objeto “Persona” en un sistema administrativo, podríamos utilizar los atributos:

nombre, edad, direccion, estadoCivil.

ALCYON
IT Services

ABSTRACCIÓN II

En cambio, si requerimos al objeto “Persona” para el área de biología, dentro de sus atributos quizá tengamos, ADN, RND, Gen x1, Gen x2, etc.

Por tanto, los atributos antes mencionados no serán requeridos.

En general, podemos decir que “Persona” cuenta con todos los atributos mencionados aquí, pero debido al proceso de abstracción excluimos todos aquellos, que no tiene cabida en nuestro sistema.

ALCYON
IT Services

CLASIFICACIÓN I

Los conceptos de abstracción y clasificación están muy relacionados:

Abstracción: genera abstracciones de la realidad.

Clasificación: genera clases con estas abstracciones.

En POO la única división modular posible es la clase.

No pueden existir rutinas desligadas de objetos.

Todo han de ser tipos! (concepto similar a clase). Ej: clase Alumno que contiene la clase Asignatura y la clase String, Es decir, definimos una clase en función de otras y que no son más que tipos de datos.

ALCYON
IT Services

CLASIFICACIÓN II

Hay que ver la clase como una plantilla que permite crear muchos elementos (objetos instanciados), dado que la clase contiene la descripción de muchos.

ALCYON
IT Services

HERENCIA I

Heredar de un objeto quiere decir definir un nuevo objeto que tiene todos los atributos y todas las operaciones del objeto del que hereda.

El nuevo objeto, además de estos contenidos heredados, añade nuevos datos y operaciones (y puede sustituir las heredadas). Este objeto extiende/deriva/subclasifica la clase superior

Ej: clase Empleado y clase Ejecutivo extends Empleado

HERENCIA II

La herencia ofrece:

Reutilización.

Posibilidad de ampliación.

Un coste bajo de mantenimiento.

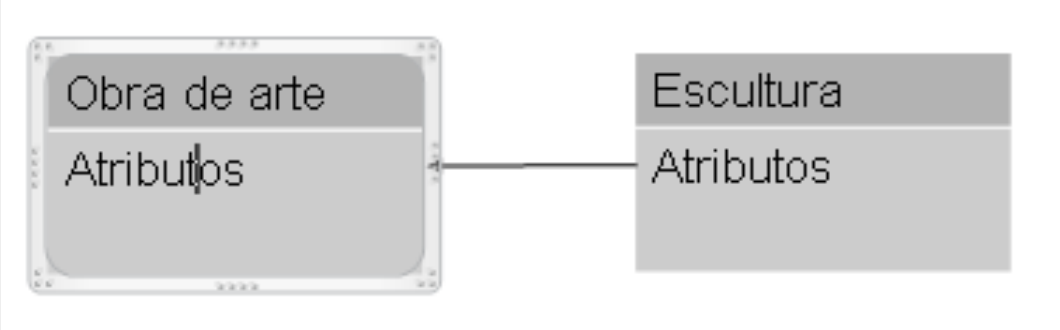
Un método de relación entre clases de una manera que sea semánticamente sensata. Una de las posibles relaciones entre clases es la ***is-a*** **(es-un)**.

HERENCIA III

Ejemplo de relación es-un:

Tenemos la clase ‘ObraDeArte’ y queremos definir la clase ‘Escultura’ como subclase de ‘ObraDeArte’.

Una escultura **es una** obra de arte.



POLIMORFISMO I

También denominado acoplamiento dinámico o acoplamiento tardío o acoplamiento en tiempo de ejecución.

El polimorfismo permite mejorar la organización y la legibilidad del código.

En programación no OO en tiempo de compilación ya se resuelve en que dirección de memoria se encuentra la rutina a llamar en cada momento

En OO es en tiempo de ejecución cuando se resuelve la dirección de la memoria donde se encuentra el método a utilizar

ALCYON
IT Services

POLIMORFISMO II

Gracias al polimorfismo podemos ampliar las clases de una aplicación sin necesidad de introducir modificaciones en los algoritmos principales de otras clases (En un lenguaje clásico habría que modificar la rutina principal para añadir el IF de la nueva función introducida).

La llamada a un método polimórfico permite que cada tipo (clase) exprese su distinción respecto a los otros tipos similares, siempre y cuando todos ellos deriven de un mismo tipo base (el método tiene que estar definido en la clase base).

POLIMORFISMO III

Ejemplo: Clientes y el cálculo de descuentos.

A partir de la clase Cliente definimos ClienteMayorista y ClienteMinorista.

Estas dos nuevas clases tendrán un método denominado calculoDescuento() (definido como método abstracto en la clase Cliente) e implementado en cada una de las subclases de manera diferente.

Así, un algoritmo que procese clientes y pida a cada uno que calcule su descuento, invocará a la implementación calculoDescuento() de cada cliente en particular.

ALCYON
IT Services

POLIMORFISMO IV

El polimorfismo nos permite enviar mensajes a objetos (llamar a métodos) sin tener la necesidad de saber exactamente el tipo (clase) de tales objetos.

La llamada a uno o a otro método se resuelve según el tipo del objeto:

- En tiempo de compilación (enlace estático).
- En tiempo de ejecución (enlace dinámico).

CONSTRUCTORES I

Un constructor es un método que se ejecuta automáticamente cuando se instancia un objeto

Normalmente se utilizan para inicializar propiedades o ejecutar métodos que solo se ejecutan una vez al ser instanciado el objeto

```
class Persona{
    function __construct(){
        echo "Objeto persona creado</br>";
    }
}
$persona = new Persona();
```

010 Ver

CONSTRUCTORES II

Una segunda opción para crear un constructor es utilizar el nombre de la clase.

¡¡OJO a partir de la versión 8 de php está opción está obsoleta!!

```
class Persona{
    function Persona(){
        echo "Objeto persona creado</br>";
    }
}
$persona = new Persona();
```

CONSTRUCTORES III

Podemos aprovechar el constructor para pasar parámetros cuando creamos el objeto

```
class Persona{
    function __construct($nombre){
        echo “Persona con nombre ".$nombre ." creado";
    }
}

$persona = new Persona("Rodolfo");
```

010 Ver

ALCYON
IT Services

CONSTRUCTORES IV

Utizaremos el constructor para asignar los valores a los atributos en el momento de realizar la instanciación del objeto

```
class Persona {
    private $nombre; private $apellidos;

    //método constructor
    public function __construct($n, $a, $e) {
        //asignación directa (no recomendable)
        //$this->nombre = $n;
        //$this->apellidos = $a;

        //asignación por delegación
        $this->setNombre($n);
        $this->setApellidos($a);
    }
}
$persona = new Persona('David', 'Alcolea');
```

ALCYON
IT Services

CONSTRUCTORES Y HERENCIA I

```
class Persona{
    function __construct(){
        echo "Objeto persona creado<br/>";
    }
}
class Funcionario extends Persona{
    function __construct(){
        echo "Objeto funcionario creado<br/>";
    }
}
$funcionario = new Funcionario();
```

si la clase funcionario no tiene constructor entonces se ejecuta el de la clase persona

012 Ver

ALCYON
IT Services

CONSTRUCTORES Y HERENCIA II

```
class Persona{
    function __construct(){
        echo "Objeto persona creado<br/>";
    }
}
class Funcionario extends Persona{
    function __construct(){
        parent::__construct();
        echo "Objeto funcionario creado<br/>";
    }
}
$funcionario = new Funcionario();
```

con parent podemos ejecutar el constructor de la superclase

CONSTRUCTORES Y HERENCIA III

EJERCICIO 1: Crear una clase Persona (o utilizar la que ya tenemos creada) con los siguientes atributos:

- Nombre
- Apellidos

Y los siguientes métodos

- Constructor
- Getters
- Setters

Tener en cuenta la encapsulación y la visibilidad de los atributos y métodos

012 Ver

CONSTRUCTORES Y HERENCIA IV

EJERCICIO 2: Crear una clase Funcionario que herede de la clase persona pero con un atrubuto adicunal:

- Salario

Y los siguientes métodos

- Constructor
- Getter delnuevo atributo
- Setter del nuevo atributo

Tener en cuenta la encapsulación y la visibilidad de los atributos y métodos

012 Ver

ALCYON
IT Services

CONSTRUCTORES Y HERENCIA V

EJERCICIO 3:

Instanciar un objeto de cada clase y mostrar sus atributos

ATRIBUTOS ESTÁTICOS I

El modificador static permite acceder a las variables y métodos aunque no tengamos creada una instancia del objeto que los contiene.

En estos casos usaremos :: en vez de ->

```
class Persona{
	public static $contador=0;

	public static function mostrarContador(){
		/*Con métodos estáticos no podemos usar $this*/
		//echo $this->contador;
		return self::$contador;
	}
}
echo Persona::$contador;
echo Persona::mostrarContador();
```

014 Ver

ATRIBUTOS ESTÁTICOS II

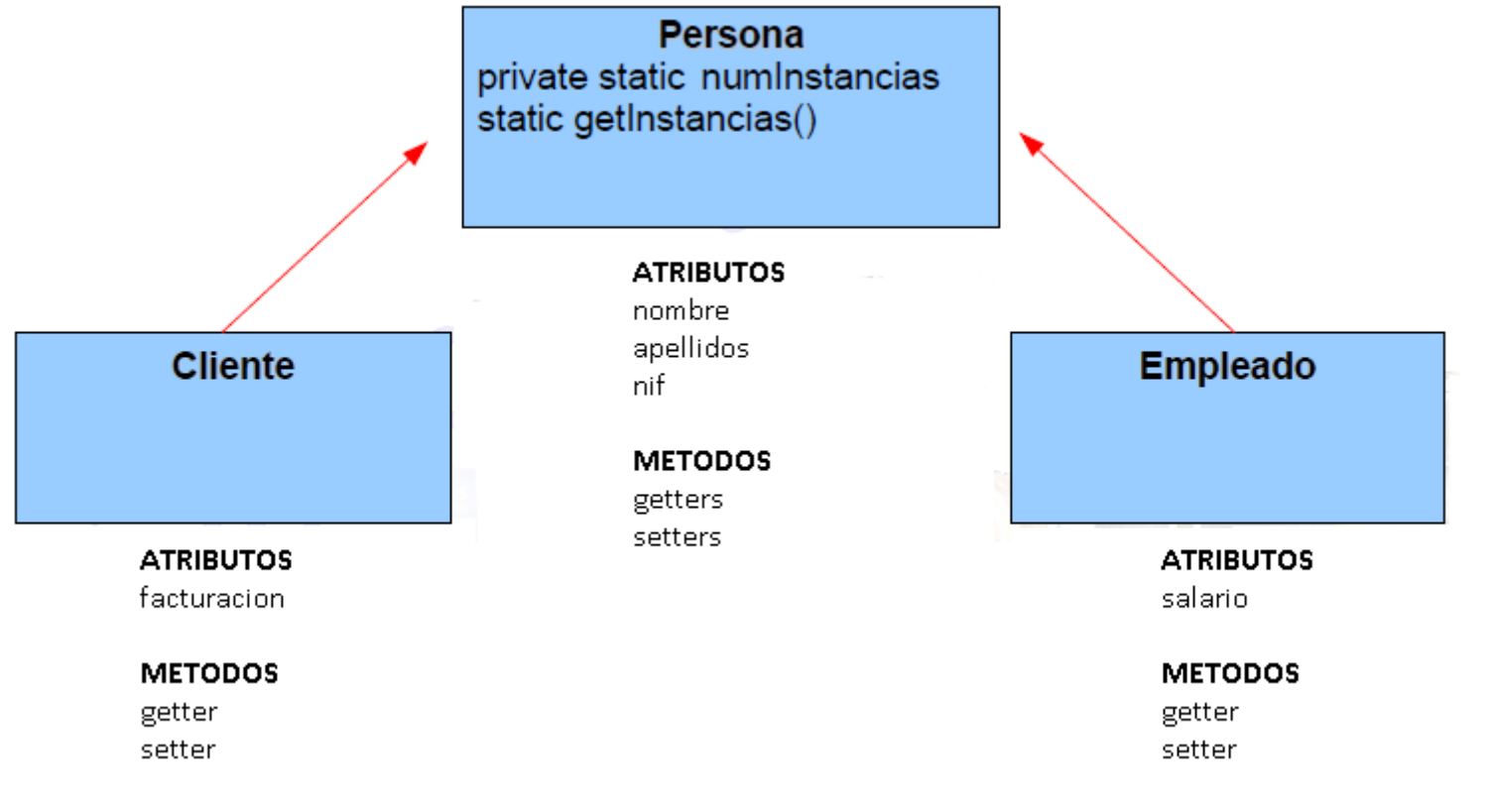
Podríamos crear un contador de instancias de forma que cada vez que creamos una sumemos una unidad al contador. Esta operación la podemos realizar en el constructor de la clase.

```
class Persona{

    public static $contador=0;

    public function Persona(){
        …/...
        //sumar 1 al contador
        self::$contador++;
    }
}
$persona = new Persona('David', 'Alcolea');
echo Persona::$contador;
echo Persona::mostrarContador();
```

014 Ver

EJERCICIO

Desarrollar una aplicación que contenga las siguientes clases:



EJERCICIO

1. Crear una instancia de empleado y una de cliente.

2. Imprimir el total de instancias de persona creadas, gracias a un método estático de la clase Persona.

3. Imprimir el valor de los atributos del empleado y el cliente creados

```
Cliente: David Alcolea 23456789L 30000
Empleado: John Rambo 99956789L 20000
2
```

016 Ver

Aviso Legal

David Alcolea
david-alcolea@alcyon-it.com
www.alcyon-it.com