ALCYON
IT Services

POO

CREANDO OBJETOS

ALCYON
IT Services

OBJETOS Y CLASES

ALCYON
IT Services

INTRODUCCIÓN

Una clase es una representación informática de algo que existe en el mundo (real o virtual) y tiene sentido para nosotros (tres tipos: frontera, control y de entidad -estás últimas equivaldrían a las tablas de una bbdd-).

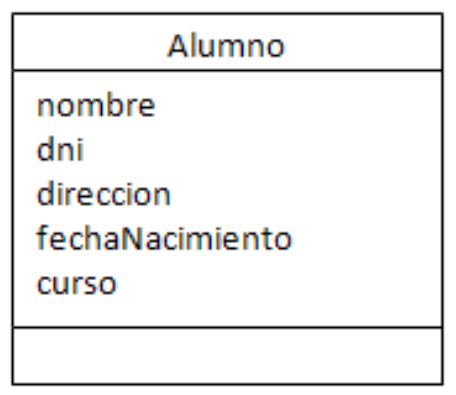
En OO una clase se representa como el encapsulamiento de unos datos y de unas operaciones sobre estos datos.

Una clase puede ser algo tangible (un libro, un artículo, etc) o intangible (una lista de tareas, un contador, etc).

DEFINICIÓN DE UNA CLASE I

Una clase representa a toda una familia de entidades que cumplen con la misma definición (los mismos datos y las mismas operaciones).

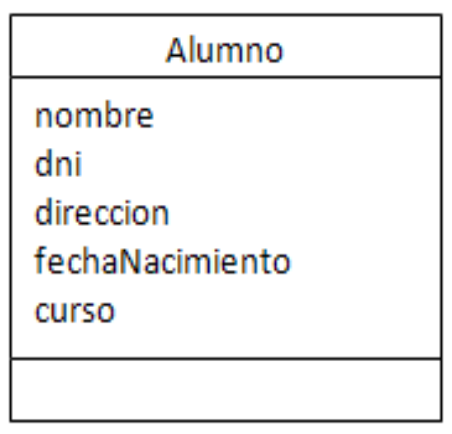
Ejemplo: Si definimos la siguiente clase *Alumno* no describimos a un estudiante sino a todos aquellos estudiantes que se pueden describir con los datos: nombre, dni, direccion, fechaNacimiento y curso.



DEFINICIÓN DE UNA CLASE II

El gráfico es un rectángulo dividido en tres partes y es la representación gráfica de una clase UML (Unified Modeling Language), un lenguaje gráfico de análisis y diseño OO estándar.

- La primera parte del rectángulo contiene el nombre de la clase (siempre un substantivo singular).
- Los nombres en la OO son muy importantes dado que son la base de la reutilización (miles de nombres en una librería) y del polimorfismo (operaciones con mismo nombre pero con implementación diferente).



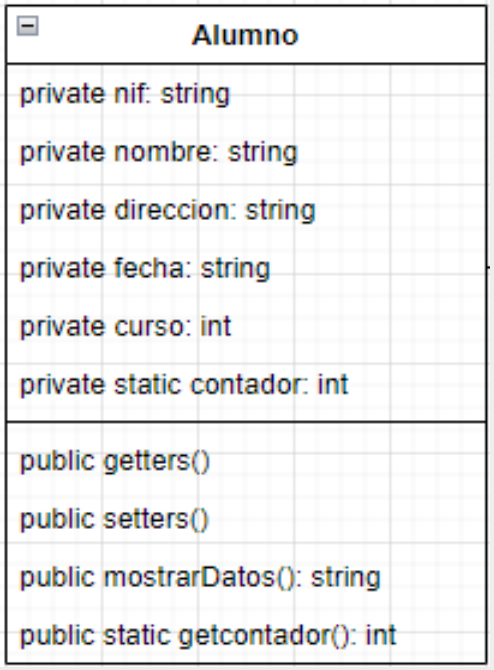
DEFINICIÓN DE UNA CLASE III

La segunda parte del rectángulo consta de la lista de los datos.

- Cada dato debe tener su identificador y tipo.
- El tipo y otra información suele documentarse en plantillas en herramientas CASE.

El tercer bloque del rectángulo corresponde a las rutinas (servicios, métodos) de la clase.

Se les deberá dar un nombre y documentar separadamente su firma (tipo de retorno+ nombre+parámetros) y el código asociado.



ALCYON
IT Services

DEFINICIÓN DE UNA CLASE IV

Los datos seleccionados para la clase Alumno no definen exactamente a un alumno del mundo real.

Sin embargo, sí que son relevantes para nuestro programa, es decir, se ha hecho una **abstracción** (una simplificación sobre lo que un alumno es en la realidad).

Con la definición de la clase Alumno podemos crear tantos estudiantes como queramos, cada uno con su particular dni, nombre, etc.

Cada uno de estos alumnos es lo que se conoce como **objeto**.

Es a partir de las clases que creamos los objetos. Esto se conoce como **instanciación**.

Un objeto es una **instancia** de una clase.

DEFINICIÓN DE UNA CLASE V

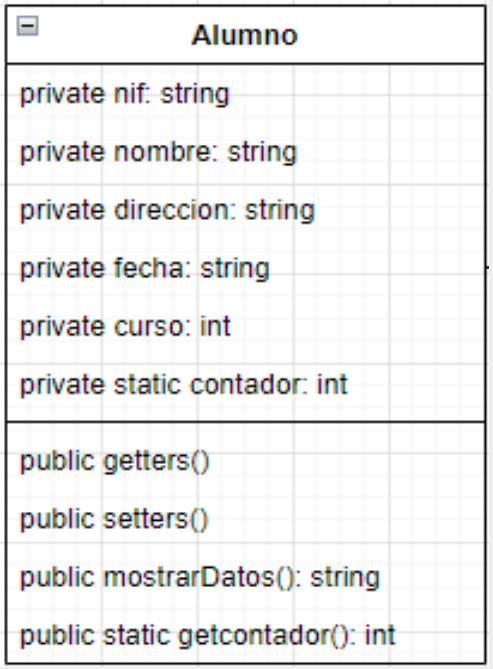
¿Qué operaciones podemos tener en la clase Alumno?

- Como mínimo una operación de lectura (getter) para acceder al valor de cada dato y una de escritura (setter) para modificar el valor de cada dato en caso que queramos modificarlos.

Ahora ya tenemos una clase completa, con datos y operaciones. Así, a un estudiante 'x':

- Le podemos pedir el nombre – x.leerNombre() - y nos devolverá una cadena de caracteres.

- Si escribimos x.escribirNombre("Juan") nos guardará la cadena "Juan" en el dato 'nombre'.



ALCYON
IT Services

DEFINICIÓN DE UNA CLASE VI

Los objetos se guardan en memoria y ya reservan espacios para sus atributos (ya que éstos son específicos de cada uno de ellos)

Las operaciones (métodos) solo se guardan en la clase ya que son comunes para todos los objetos instanciados. Java a estas operaciones las referencía con '**this**' de forma que que sepa sobre que objeto debe ejecutarse

Si la operación es estática no tienen la referencia this y no se pueden ejecutar sobre los objetos instanciados

DEFINICIÓN DE UNA CLASE VII

Consideraciones I

Es un convenio ampliamente aceptado utilizar para los métodos accesores y modificadores la nomenclatura getXXX() y setXXX(), respectivamente

```
public class Alumno {
  private $nombre;
  private $dni;
  private $direccion;
  private $fechaNacimiento;
  private $curso;

  public function getNombre() {
    return $this->nombre;
  }
  public function setNombre($nombre) {
    this->nombre = $nombre;
  }
}
```

DEFINICIÓN DE UNA CLASE VIII

Consideraciones II

private: oculta los datos al exterior. Sólo son accesibles desde el interior de la clase (forman parte de la implementación de la clase).

public: permite que el dato o el método sea accesible para el exterior de la clase. Ej:

podríamos ver este atributo desde otra clase;

Alumno->dni

```
public class Alumno {
  private $nombre;
  private $dni;
  private $direccion;
  private $fechaNacimiento;
  private $curso;

  public function getNombre() {
    return $this->nombre;
  }
  public function setNombre($nombre) {
    this->nombre = $nombre;
  }
}
```

ALCYON
IT Services

TIPADO I

Si bien php es un lenguaje no tipado podemos utilizar tipado en los atributos y métodos de nuestras clases (y, además, es mas que recomendable.

Ejemplo tipado de atributos

```
private string $nombre;
private int $curso;
```

Ejemplo tipado en atributos de entrada a un método:

```
public function setNombre(string $nombre) {…}
```

Ejemplo tipado en retorno de un método

```
public function getNombre(): string {
    return $this->nombre;
}
```

ALCYON
IT Services

TIPADO II

Ejemplo tipado de un método que no retorna ningún valor

```
public function setNombre(string $nombre): void {
    $this->nombre = $nombre;
}
```

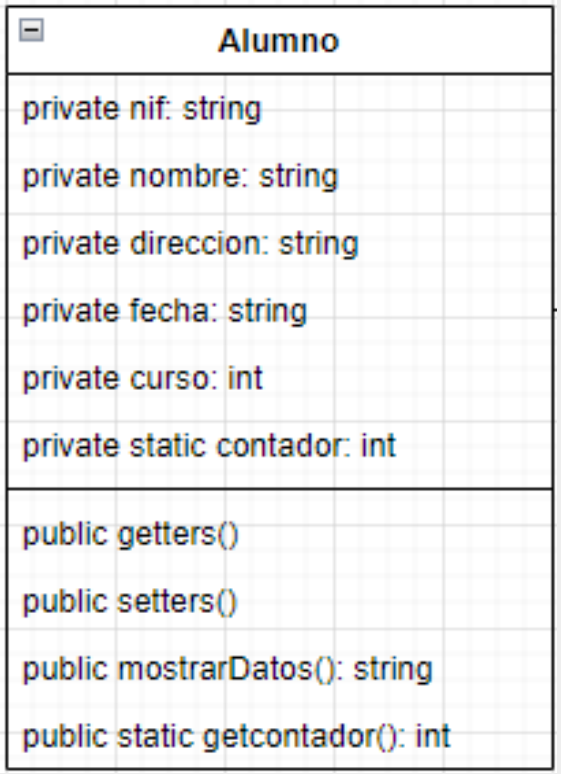
DEFINICIÓN DE UNA CLASE VIII

EJERCICIO:

Completad la anterior definición de la clase y ejecutarla.

Recordad utilizar el tipado de los atributos

Diseñar el diagrama UML utilizando draw.io



ALCYON
IT Services

INSTANCIACION Y USO

ALCYON
IT Services

INTRODUCCIÓN

Ahora veremos como se usa una clase, es decir, crear objetos a partir de una clase, manipularlos y destruirlos en un programa.

Haremos un programa OO. Veremos como el proceso de creación de objetos es la instanciación; un objeto es una instancia de una clase.

A partir de una clase se pueden crear tantos objetos como queramos.

ALCYON
IT Services

INSTANCIACIÓN I

Enunciado:

Crearemos un objeto de la clase Alumno y entonces preguntará al usuario el nombre del alumno, guardará el texto introducido por el usuario en el atributo ‘nombre’ del alumno.

Posteriormente, recuperará tal atributo y lo imprimirá por pantalla para comprobar que realmente se guardó el valor entrado por el usuario.

INSTANCIACIÓN II

Vamos a instanciar un objeto de la clase Alumno

```
$alumno = new Alumno();


$alumno->setNombre('nombre alumno');
// Ahora recuperamos del objeto para comprobar que se guardó
$nombreAlumno = $alumno->getNombre();
echo $nombreAlumno;
```

ALCYON
IT Services

INSTANCIACIÓN III

Hemos visto que podemos definir un dato del tipo ‘Alumno’ con la misma naturalidad que definimos un número entero o un carácter:

Tal declaración consiste sólo en definir una referencia (la variable alumno) que contendrá la dirección de memoria de un objeto.

Pero declarar una variable de tipo ‘Alumno’ no es suficiente para poder trabajar con el alumno.

Se necesitan dos pasos: declarar la variable e instanciar el objeto.

La instrucción para instanciar un objeto es muy similar en todos los lenguajes OO:

```
$alumno=new Alumno();
```

ALCYON
IT Services

USO

Una vez que obtenemos una referencia válida a un objeto podemos emplearla para usar los servicios del objeto.

Solicitar a un objeto que “ejecute una rutina”:

$alumno->setNombre(textoUsuario);

En OO no se emplea la expresión “ejecutar la rutina de un objeto” sino la de “enviar un mensaje a un objeto”.

ALCYON
IT Services

ATRIBUTOS Y SERVICIOS

ALCYON
IT Services

TERMINOLOGÍA

Hasta ahora hemos hablado de **datos** y **operaciones** de una clase o de un objeto.

Para referirnos a los datos se utiliza el término **atributo**, aunque también es frecuente emplear el término **propiedad**.

Para referirnos a las rutinas lo más usual es utilizar los términos **método** o **servicio**.

ALCYON
IT Services

TERMINOLOGÍA: ATRIBUTOS

Los atributos de una clase son sus datos y determinan el **estado** de cada objeto.

Cada atributo tiene un ***tipo*** y un ***nombre*** (también llamado ***identificador***).

Los tipos pueden ser simples, como un carácter o un entero, o complejos, como una lista de enteros, o quizás una clase como *Alumno*.

Pueden ser de tipo primitivo o referencia (guardan la dirección de memoria de un objeto)

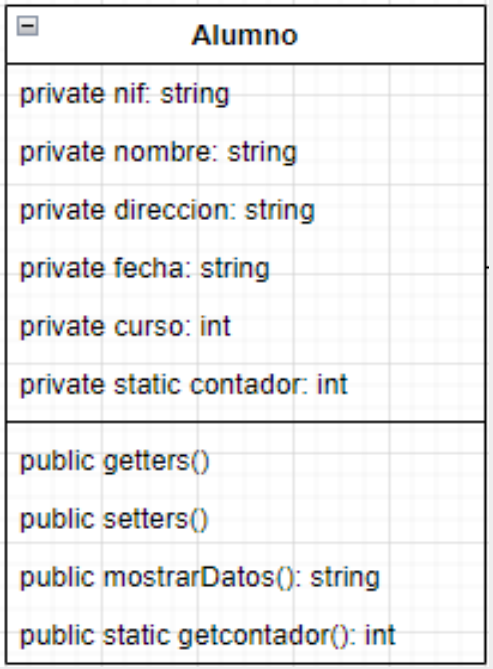
TERMINOLOGÍA: ATRIBUTOS

La clase Alumno tiene cinco atributos:

Cuatro de la clase **string**

Uno de la clase **int** (curso)

Un atributo estático que veremos más adelante



ALCYON
IT Services

TERMINOLOGÍA: SERVICIOS

Los **servicios** o **métodos** son las operaciones que se pueden hacer sobre los datos.

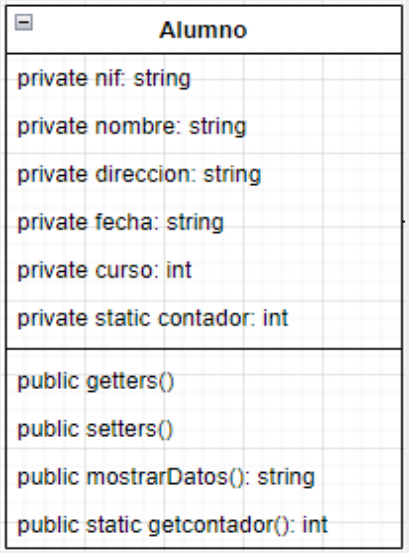
Los métodos determinan el comportamiento de un objeto.

Pueden ser funciones o procedimientos, esto es, pueden devolver un valor o no.

TERMINOLOGÍA: SERVICIOS

Alumno tiene dos métodos por cada atributo, uno para leer y otro para escribir.

Responsabilidad: Las clases son responsables de recordar cosas (los atributos) y de hacer cosas (los servicios o métodos).



Métodos de acceso (“getters” o “accessors”) son los que devuelven los valores de los atributos de una clase.

Métodos modificadores (“setters”) son los que escriben los valores en los atributos de una clase.

ALCYON
IT Services

VISIBILIDAD

Suele haber tres grados de visibilidad

Publico: Desde el exterior de la clase se tiene libre acceso para utilizar un atributo o un método público (desde otra clase se puede acceder a un atributo con NombreClase.atributo).

Privado: Desde el exterior de la clase no se puede acceder a un atributo o un método privado.

Protegido: Desde el exterior de la clase sólo pueden utilizar atributos y métodos protegidos las clases que hereden de la clase actual

ALCYON
IT Services

VISIBILIDAD

En php se establece la visibilidad anteponiendo en la declaración de un atributo o método la palabra **public**, **protected** o **private**.

Es muy poco recomendable utilizar el acceso público en los atributos de una clase (rompemos la ocultación y con ello la robustez programa).

Como norma general, la visibilidad ha de ser la menor posible (**private**).

ALCYON
IT Services

VISIBILIDAD

Respecto a los métodos:

Se declaran públicos todos los métodos que forman la interfaz pública de la clase (los procedimientos que generan/reciben los mensajes).

El resto de métodos forman parte de la implementación de la clase y se declaran como privados o protegidos o el acceso por defecto (no poner nada -clases de un mismo paquete son las únicas que tienen acceso-).

Declaramos un método como protegido cuando queremos que las clases del mismo paquete accedan a ese método y también que las subclases lo hereden.

ALCYON
IT Services

CONSTRUCTORES I

¿Qué ocurre cuando creamos un objeto?

```
$alumno = new Alumno();
```

“Crear un objeto de la clase Alumno y que se asocie a la referencia ‘alumno’”.

¿Quién se encargará de llevar a cabo este trabajo?

ALCYON
IT Services

CONSTRUCTORES II

Lo hará la clase *Alumno*. Toda clase tiene asociado un servicio para la creación de objetos. Este servicio se conoce con el nombre de **constructor**.

Los constructores son **métodos de clase** que permiten crear objetos de la clase.

CONSTRUCTORES III

En lenguajes como Java una clase puede tener varios constructores, diferenciados todos ellos por sus parámetros de invocación (sobrecarga). NO es el caso de PHP

Por ejemplo, a continuación se muestra cómo inicializar los atributos *nombre* y *nif* de un objeto Alumno, justo en el momento de su creación:

```
$alumno = new Alumno(“Arch Stanton”, '40000001A');
```

ALCYON
IT Services

CONSTRUCTORES IV

En general, en los lenguajes OO no es obligatorio que el programador tenga que definir los constructores.

Siempre hay uno predeterminado que el compilador se encarga de proporcionar (constructor por defecto).

A pesar de no ser obligatorio es recomendable que el programador declare y defina un constructor para garantizar que los objetos se crean consistentemente.

CONSTRUCTORES V

Se crea un método con el nombre __construct() y se le pueden definir parámetros o no y no tiene tipo de retorno.

PHP sólo admite un constructor por clase.

Ejemplo de constructor para la clase Alumno en donde utilizamos el tipado de los parámetros que esperamos recibir:

```
public function __construct (string $nif, string $nom, string $dir, string $fec, int $cur) {
	$this->nombre	= $nom;
	$this->nif		= $nif;
}
```

Para utilizarlo:

```
$alumno=new Alumno('40000001A', 'Arch Stanton', 'Malvavisco, 54', '2019-02-02', 6);
```

ALCYON
IT Services

CONSTRUCTORES VI

EJERCICIO:

Modificad la clase *Alumno*:

Añadid un constructor que permita crear alumnos con todos los datos proporcionados en los argumentos de entrada

DELEGACIÓN I

En el siguiente ejemplo se utiliza la *delegación*:

```
public function __construct (string $nif, string $nom, string $dir, string $fec, int $cur) {
        $this->setNombre($nom);
        $this->setNif($nif);
        ...
}

public function setNombre (string $nombre): void {
        $this->nombre = $nombre;
}
```

Notad que el constructor delega el trabajo de inicialización en los servicios setters

Esto tiene especial importancia cuando la tarea de validación de los datos la realizan los propios settes

ALCYON
IT Services

DELEGACIÓN II

La delegación nos permite centralizar en los métodos **setter** la validación de los datos.

Por ejemplo si el NIF tiene que estar informado obligatoriamente:

```
public function __construct (string $nif, ...) {
	try {
		$this->setNif($nif);
	} catch (Exception $e) {
		throw new Exception($e->getMessage());
	}
}

public function setDni(string $nif) {
	if (empty($nif)) {
		throw new Exception('NIF obligatorio')
	}

	$this->nif = $nif;
}
```

ALCYON
IT Services

CLASE THROWABLE

Algunos editores utilizan esta clase por defecto en las estructuras try...catch

Tiene el inconveniente, mientras estamos probando nuestra aplicación, que va a capturar todas las excepciones tanto propias como las del sistema.

En caso de estas últimas perderemos información sobre el número de linea y el archivo donde se ha producido el error

Es conveniente en estos casos utilizar la clase **Exception** que solo capturará nuestras excepciones

ALCYON
IT Services

ATRIBUTOS DE CLASE: DEFINICIÓN I

Hemos visto la necesidad de tener métodos de clase, pero ¿necesitamos **atributos de clase**?

Pues efectivamente, esta necesidad existe y la mayoría de los lenguajes OO lo permiten.

A menudo es necesario tener atributos de clase (calificador static) que den algún tipo de información sobre la clase, como por ejemplo el número de instancias que se han creado de la clase (¿cuántos objetos hay?).

ALCYON
IT Services

ATRIBUTOS DE CLASE: DEFINICIÓN II

Un atributo de **clase,** también llamado **estático**, es un dato asociado a la clase y no a las instancias. Sirve para describir algo que todos los objetos tienen en común.

En php se declara un atributo como estático anteponiendo la palabra **static** a la declaración.

ATRIBUTOS DE CLASE EN PHP I

Ejemplo:

La declaración en php de un atributo de clase llamado *contador* sería:

```
private static int $contador = 0;
```

Dado que debemos actualizarlo al crear un nuevo alumno, podemos incrementarlo en el cuerpo del constructor:

```
public function __construct(string $nif, ...) {
        self::$contador++;
        …/...
}
```

ATRIBUTOS DE CLASE EN PHP II

Para acceder al contador definimos un servicio de clase, que también se hace mediante el calificador static:

```
public static function getContador(): int {
        return self::$contador;
}
```

Desde un método no static se puede acceder a un dato o método static pero no al revés porque los datos carecen del apuntador **this**.

Para utilizarlo podemos emplear:

El nombre de la clase (forma más correcta ya que estamos solicitando un atributo que pertenece a la clase):

```
echo Alumno::getContador();
```

O bien la referencia de un objeto previamente creado:

```
echo $alumno->getContador();
```

ALCYON
IT Services

ATRIBUTOS DE CLASE EN PHP III

EJERCICIO:

Modificad la clase ‘Alumno’ para conseguir que se instancie más de un alumno y que el usuario pueda saber cuántos se han creado.

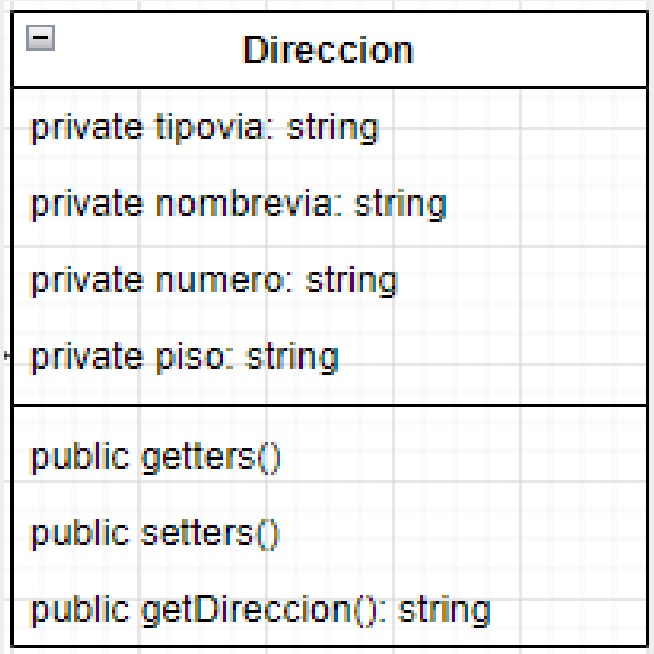
ALCYON
IT Services

USO

USO I

Una clase puede usar otra clase.

Por ejemplo, vamos a suponer que el atributo dirección de nuestra clase Alumno sea, a su vez, otro objeto:



USO II

0. Crearemos la clase ***Direccion*** según el esquema UML anterior y, además de los métodos getters y setters crearemos un método para recuperar la dirección completa

```
public function getDireccion(): string {
    return "$this->tipovia $this->nombrevia $this->numero
    $this->piso";
}
```

USO III

1. En la definición de atributos de la clase Alumno cambiamos el tipo del atributo dirección de string a Direccion

```
private Direccion $direccion;
```

2. En el constructor de la clase Alumno recogemos los datos ue componen la dirección y los enviamos al método setter

```
public function __construct (string $nom, string $nif, ..., string $tipo, string $via, string $num, string $piso, ...) {
    try {
        .../...
        $this->setDireccion($tipo, $via, $num, $tipo);
    } catch (Exception $e) {
        throw new Exception($e->getMessage());
    }
}
```

USO IV

3. Cambiamos el método setter para instanciar un objeto dirección

```
public function setDireccion(string $tipo, string $via, string $num, string $piso)) {
    $this->direccion = new Direccion($tipo, $via, $num, $piso);
}
```

4. Modificamos el método para recuperar la dirección utilizando el correspondiente getter de la clase Direccion

```
public function getDireccion(): string {
    return $this->direccion->getDireccion();
}
```

USO V

5. Si tenemos un método **mostrarDatos** en la clase Alumno también tendremos que modificarlo para recuperar la dirección a partir del getter y no directamente a partir del atributo **direccion** (y que ahora es un objeto)

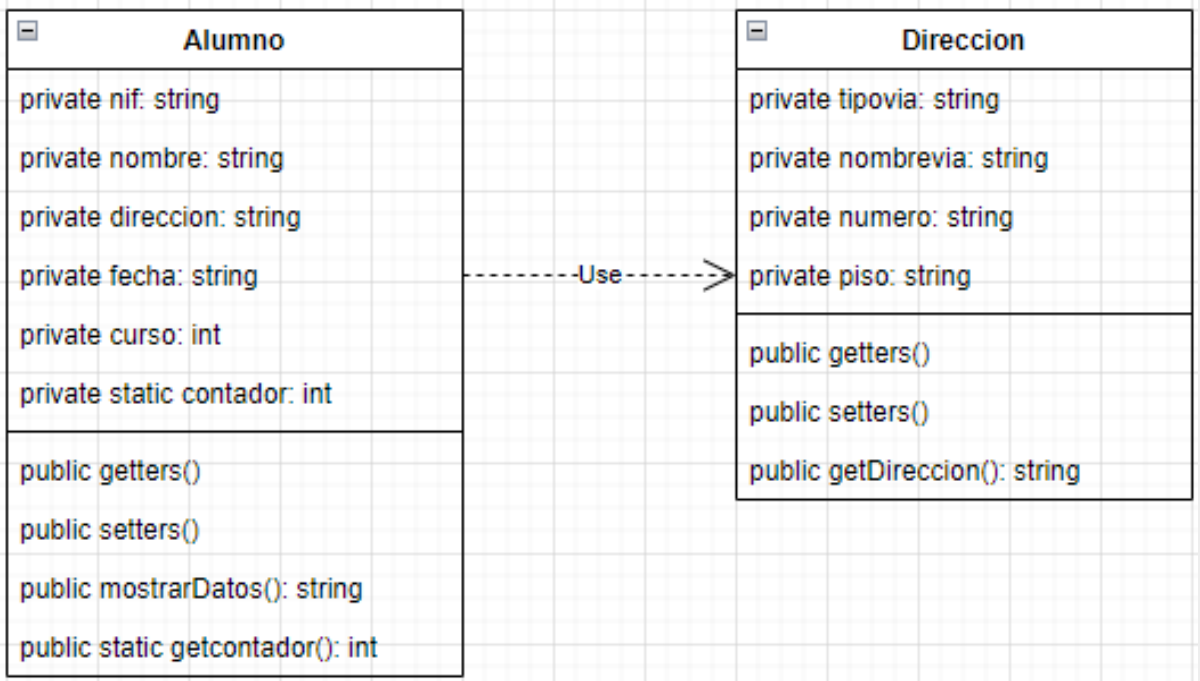
```
public function mostrarDatos(): string {
    return "$this->nif / $this->nombre / {$this->getDireccion()} /
            $this->fecha / $this->curso";
}
```

NOTA: Si queremos ejecutar un método para recuperar el retorno dentro de las comillas dobles envolveremos la llamada al método con { }

USO VI

Si necesitamos recuperar la dirección de nuestro objeto alumno invocamos al método getDireccion() y, dentro de éste, a los correspondientes métodos getters del objeto **Direccion**.

Se dice entonces que la clase **Alumno** usa la clase **Direccion**



ALCYON
IT Services

HERENCIA

ALCYON
IT Services

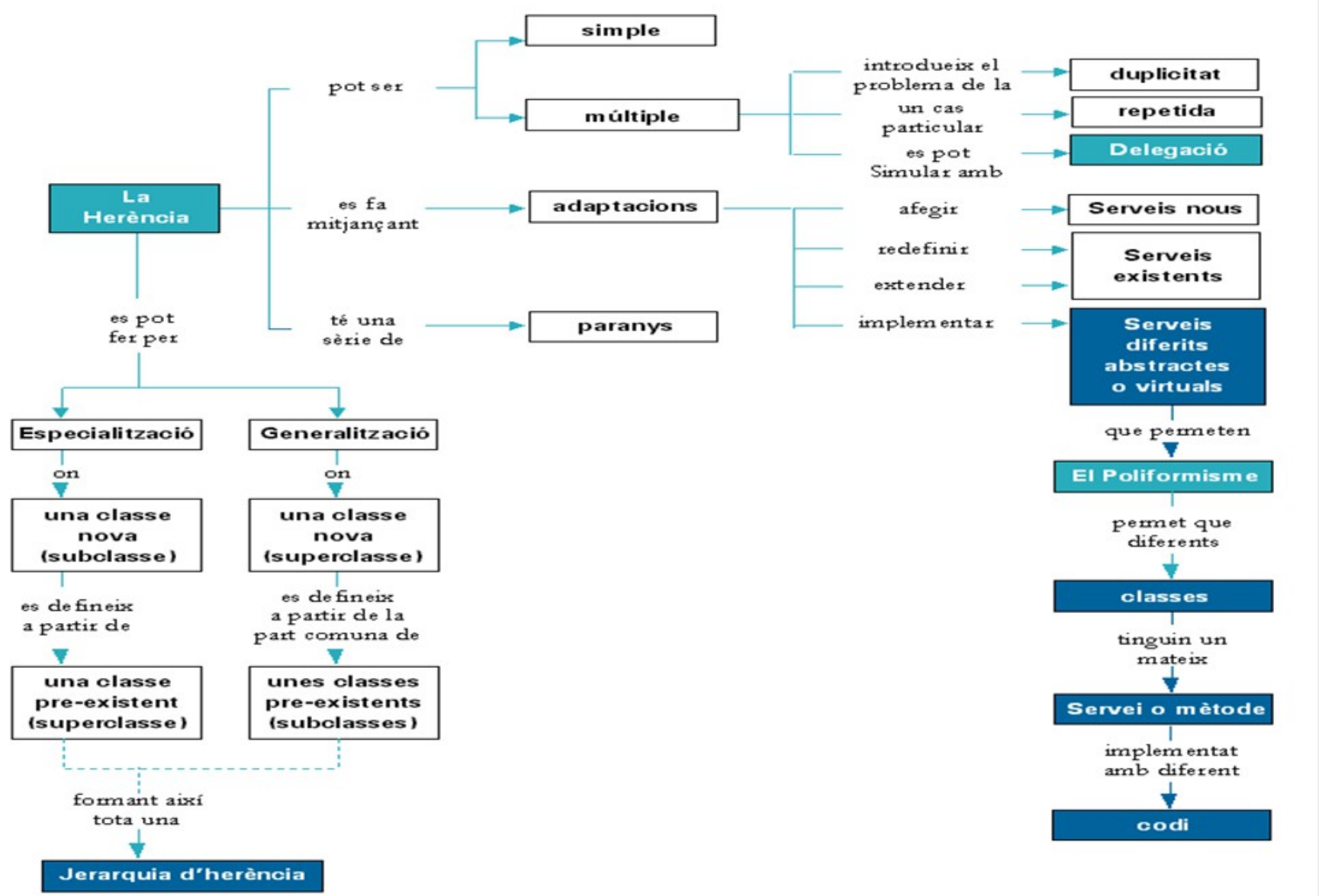
HERENCIA I

La herencia es la única tecnología propia de la OO.

La herencia permite niveles altos de reutilización y de aumento de la fiabilidad de las aplicaciones.

El resto de técnicas ya existía y la OO se las apropió: encapsulación, ocultación de la información, la división en módulos, etc.

HERENCIA II


La Herència
pot ser
simple
múltiple
introdueix el problema de la
duplicitat
un cas particular
repetida
es pot Simular amb
Delegació
es fa mitjançant
adaptacions
afegir
Serveis nous
redefinir
extender
Serveis existents
implementar
té una sèrie de
paranys
Serveis diferits abstractes o virtuals
es pot fer per
que permeten
El Poliformisme
Especialització
Generalització
on
on
permet que diferents
una classe nova (subclasse)
una classe nova (superclasse)
classes
es defineix a partir de
es defineix a partir de la part comuna de
tinguin un mateix
una classe pre-existent (superclasse)
unes classes pre-existents (subclasses)
Servei o mètode
implementat amb diferent
formant així tota una
codi
Jerarquia d'herència

ALCYON
IT Services

HERENCIA III

La herencia es una tecnología que permite definir una nueva clase a partir de otra mediante la descripción de las diferencias entre ellas.

Al aplicar la herencia:

*La nueva clase es la **subclase** de la anterior.*

*La clase de la que se hereda es la **superclase** de la nueva.*

*Las subclases podrían definirse como '**es como**'* o *'**es un**'*

ALCYON
IT Services

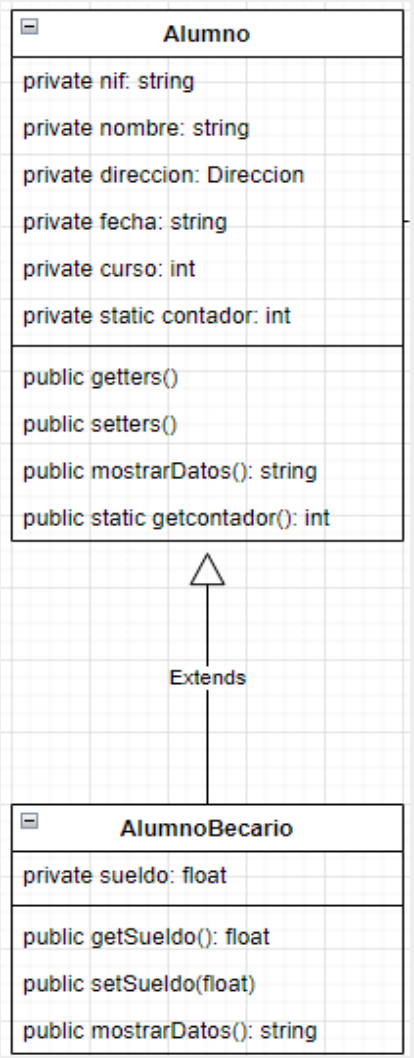
HERENCIA IV

Cuando no podemos reutilizar una clase tal y como es (y esto es muy frecuente) podemos aplicar la herencia para crear una subclase que se adapte a nuestras necesidades.

Puesto que la nueva clase es más específica que la anterior y además lo hacemos de forma descendente se dice que aplicamos herencia por especialización.

HERENCIA: JERARQUIA I

En el ejemplo del alumno, supongamos que queremos definir un **AlumnoBecario**, igual que el anterior pero con un atributo más (el importe de la beca –**sueldo**-) y servicios de acceso a este atributo: (getSueldo() / setSueldo())



HERENCIA: JERARQUIA III

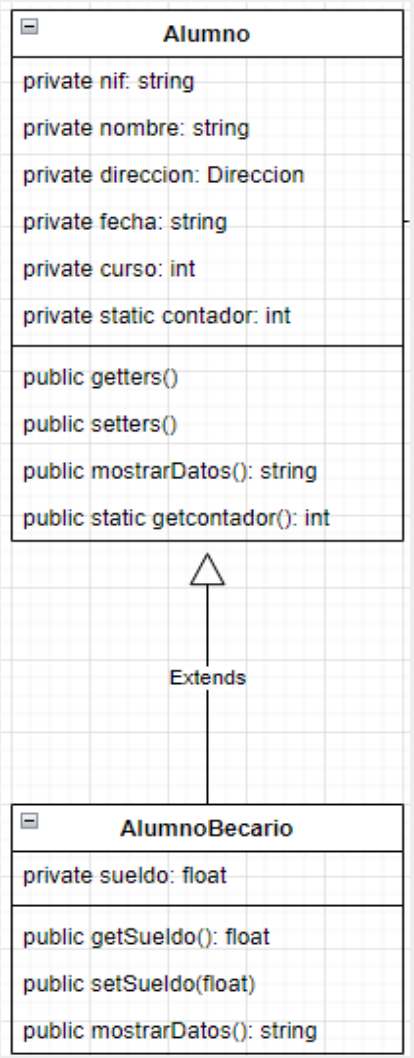
Todas las responsabilidades que tenía la clase Alumno las tiene también AlumnoBecario.

Por tanto, podemos escribir:

```
$becario->setNombre("Paco");
```

O también:

```
$becario.setSueldo(1200);
```



HERENCIA: JERARQUIA IV

Para definir la clase AlumnoBecario a partir de Alumno:

```
public class AlumnoBecario extends Alumno {
   private float $sueldo;

   public function __construct (string $nif, string $om, …, float $sal) {
      parent::__construct($nif, $nom, ...);
      $this->setSueldo($sal);
   }

   public function getSueldo(): float {
      return $this->sueldo;
   }

   public function setSueldo(float $sal): void {
      $this->sueldo = $sal;
   }
}
```

ALCYON
IT Services

HERENCIA: JERARQUIA V

extends se utiliza para derivar; indica que ***AlumnoBecario*** hereda de ***Alumno***.

parent sirve para hacer referencia desde una subclase a las responsabilidades de una superclase (en este caso se está llamando al constructor de la clase padre).

HERENCIA: JERARQUIA VI

EJERCICIO:

Vamos a crear un programa **main.php** o **principal.php** para instanciar un objeto de la clase alumno y otro objeto de la clase AlumnoBecario

Incorporar en este programa los ficheros con las clases utilizando require_once()

```
require_once('alumno.php');
require_once('becario.php');
try {
        $alumno = new Alumno('4000000A', 'O-Ren Ishii', 'c/', 'malvavisco', '54', '3o 4a', '2000-09-09', 5);
        echo $alumno->mostrarDatos();
        $becario = new Becario('4000001B', 'Pau Pou', 'c', 'malvavisco', '99', '1o 4a', '2006-09-09', 6, 1200);
        echo $becario->mostrarDatos();
} catch (Exception $e) {
       echo $e->getMessage();
}
```

ALCYON
IT Services

HERENCIA POR GENERALIZACIÓN I

Cuando observamos que varias clases tienen mucho en común y pequeñas diferencias entre ellas podemos extraer lo común y dejar en cada una sólo lo específico.

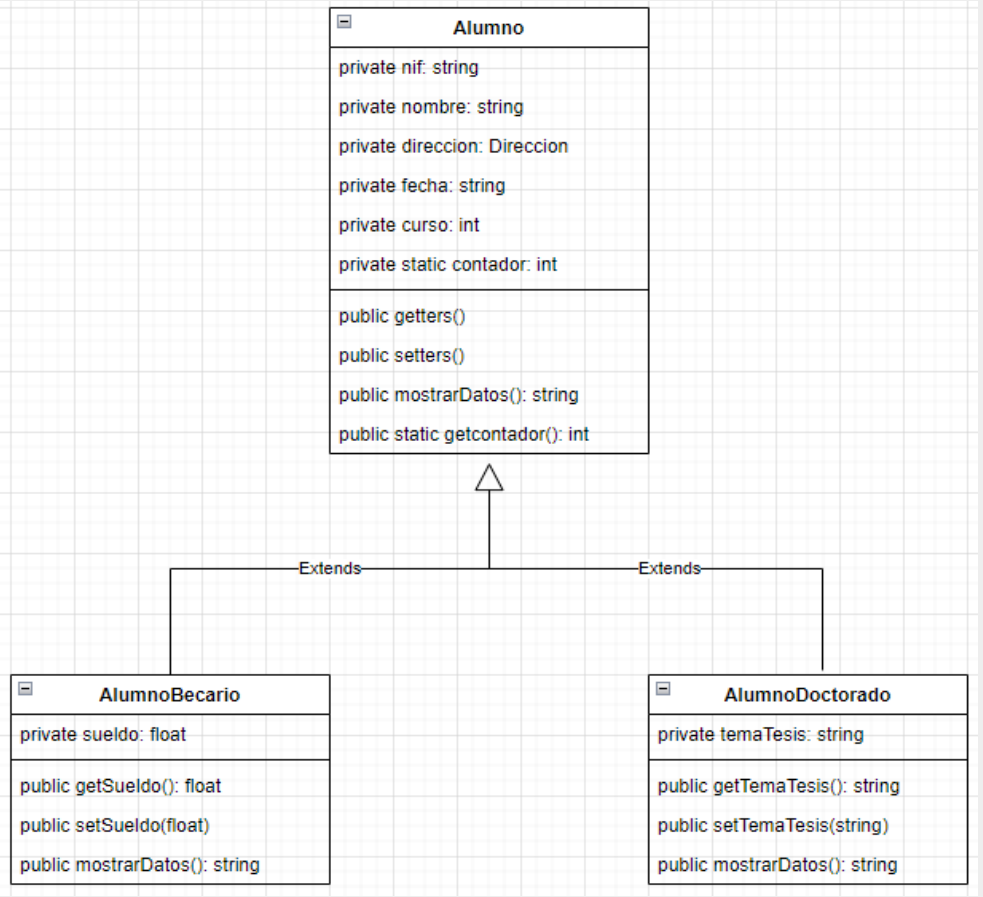
Este tipo de herencia recibe el nombre de herencia por generalización.

HERENCIA POR GENERALIZACIÓN II

En la figura se observa como se hace general, de manera ascendente, lo que es común.

En este caso la clase nueva es la superclase y las clases existentes pasan a ser subclases.

EJERCICIO: Crear una clase **alumnoDoctorado** con el atributo que se especifica en el diagrama UML



ALCYON
IT Services

ADAPTACIONES POSIBLES EN LA HERENCIA

¿Podemos añadir nuevos atributos en la subclase? Sí, por supuesto.

Respecto a los servicios, podemos:

Añadir nuevos servicios.

Implementar servicios.

Reimplementar servicios.

Ampliar servicios previamente definidos

ALCYON
IT Services

AÑADIR SERVICIOS

Consiste en añadir en la subclase nuevos métodos que no existían en la superclase.

Por ejemplo, es el caso de los servicios añadidos en ***AlumnoBecario***:

getSueldo()

setSueldo()

ALCYON
IT Services

IMPLEMENTAR SERVICIOS I

Concepto de servicio diferido (también llamado abstracto o virtual): servicio definido en una clase pero NO implementado.

El servicio se implementa en la subclase.

Es el caso del ejemplo en que cada tipo de cliente tiene un algoritmo diferente para calcular su descuento comercial.

ALCYON
IT Services

IMPLEMENTAR SERVICIOS II

En php los servicios diferidos se denominan servicios abstractos:

```
public abstract function calculoDescuento (string $albaran): void;
```

Una clase con un servicio diferido se conoce como clase diferida, abstracta o virtual aunque el resto de servicios o métodos no sean abstractos.

ALCYON
IT Services

IMPLEMENTAR SERVICIOS III

Una clase diferida NO permite crear instancias ya que está inacabada al tener un servicio no implementado.

Las subclases, de las cuales crearemos objetos, están obligadas a definir e implementar el servicio abstracto.

100% seguridad que todos los clientes instanciados tendrán implementado el método (el polimorfismo será viable).

ALCYON
IT Services

IMPLEMENTAR SERVICIOS IV

Podemos tener una subclase sin implementar un método abstracto de la superclase, pero entonces no podremos crear un objeto de tal subclase.

Descendiendo por la jerarquía de clases, nos encontraremos en algún punto en que ese método se implemente (en la nieta, bisnieta, …) con tal de poder crear una instancia.

ALCYON
IT Services

IMPLEMENTAR SERVICIOS V

Si observamos el ejemplo anterior:

```
public abstract function calculoDescuento(string $albaran): void;
```

Estamos obligados a implementar el método calculoDescuento en las clases hijas con exactamente un parámetro de entrada

Pero: ¿y si necesitamos distintos parámetros de entrada en cada clase?

Tenemos dos opciones:

ALCYON
IT Services

IMPLEMENTAR SERVICIOS VI

Opción array

Indicamos un array como parámetro de entrada

```
public abstract function calculoDescuento($args);
```

De forma que, al implementar el método en las clases hijas leemos el array que puede tener un numero indeterminado de valores

```
public function calculoDescuento($args) {
    foreach ($args as $item) {
        ...
    }
}
```

Y llamamos al método de la siguiente forma:

```
$cliente->calculoDescuento(['a', 'b', 'c']);
```

ALCYON
IT Services

IMPLEMENTAR SERVICIOS VII

Opción argumentos variables

A partir de php 5.6 tenemos la opción para pasar un número desconocido de argumentos

```
public abstract function calculoDescuento(...$args);
```

Al implementar el método en las clases hijas recogemos también un número indeterminado de argumentos

```
public function calculoDescuento(...$args) {
    foreach ($args as $item) {
        ...
    }
}
```

Y llamamos al método, por ejemplo para tres argumentos:

```
$cliente->calculoDescuento('a', 'b', 'c');
```

ALCYON
IT Services

IMPLEMENTAR SERVICIOS VIII

Opción argumentos variables

Esta opción, obviamente, se puede utilizar siempre que necesitemos pasar un número de argumentos indeterminado a cualquier método

EJERCICIO

Vamos a ver un ejemplo de implementación de método en el ejercicio de Alumnos

1. Creamos un nuevo atributo genérico para guardar el código de alumno.

```
protected string | null $codigo;
```

NOTA 1: Nos interesa que este atributo sea visible en las dos clases que heredan de Alumno ya que será en éstas donde lo informaremos

NOTA 2: Si necesitamos que un atributo pueda contener dos tipos de datos distintos podemos utilizar el símbolo *pipe* para especificarlos

EJERCICIO

2. Definimos un método sin implementar para obligar a que cada subclase lo implemente (consideraremos que la operativa de implementación será muy diferente en cada clase)

```
public abstract function codigoAlumno(string ...$args): void;
```

NOTA 1: Además consideraremos que cada alumno necesita un número diferente de parámetros de entrada para la obtención del código

3. El método abstracto nos obliga a que toda la clase sea también abstracta (no se permitirá instanciar objetos de esta clase)

```
abstract class Alumno
```

EJERCICIO

4. Modificaremos el método mostrarDatos() de la clase Alumno para consultar también el valor del nuevo atributo

5. En la clase ***AlumnoBecario*** implementamos el método (por ejemplo para este tipo de alumnos el código será un número aleatorio entre 5000 y 9999 al que concatenaremos dos caracteres)

```
public function codigoAlumno(string ...$args): void {
    $this->codigo = rand(5000, 9999) . $args[0] . $args[1];
}
```

6. En el constructor llamaremos al nuevo método implementado

```
$this->codigoAlumno('B', '1');
```

EJERCICIO

7. En la clase ***AlumnoDoctorado*** implementamos el método (por ejemplo para este tipo de alumnos el código será un número aleatorio entre 1000 y 4999 al que concatenaremos un caracter)

```
public function codigoAlumno(string ...$args): void {
    $this->codigo = rand(1000, 4999) . $args[0];
}
```

8. En el constructor llamaremos al nuevo método implementado

```
$this->codigoAlumno('D');
```

REIMPLEMENTAR SERVICIOS: OVERIDING I

Motivación: no nos satisface el servicio implementado en la superclase por lo que en la subclase lo sustituimos (sobreescritura de métodos).

Para reimplementar un servicio sólo hay que volverlo a definir con el mismo nombre e interficie (firma).

Hay que ser muy cauteloso al aplicar el overiding, ya que podríamos comprometer el correcto funcionamiento heredado de la clase base. **Si queremos que un método no pueda sobreescribirse lo definiremos con el cualificador 'final'**

REIMPLEMENTAR SERVICIOS II

Supongamos que todos los alumnos tienen un atributo llamado *usuario,* cuyo valor será un código proporcionado por el centro de estudios a partir del nif y el curso.

```
protected string $usuario;

private function setUsuario(): void {
    $this->usuario = $this->nif . $this->curso;
}
```

REIMPLEMENTAR SERVICIOS III

Por otro lado, el valor de este atributo para los becarios sigue una secuencia totalmente diferente a la de los alumnos (por ejemplo nif más la letra 'B').

```
private function setUsuario(): void {
    $this->usuario = $this->getNif() . 'B';
}
```

Por lo tanto, en la clase ***AlumnoBecario*** estamos reimplementando el servicio para la obtención del código de usuario (lo estamos sobreescribiendo)

AMPLIAR SERVICIOS I

Tenemos un método en la clase alumno para mostrar los valores de todos los atributos comunes:

```
public function mostrarDatos(): string {
    return "$this->nif / $this->nombre / {$this->getDireccion()} /
            $this->fecha / $this->curso / $this->codigo /
            $this->usuario";
}
```

AMPLIAR SERVICIOS II

Pero tenemos que ampliar este método en las dos subclases para obtener también el valor de sus atributos específicos. Ejemplo para becario:

```
public function mostrarDatos(): string {
    $datosComunes = parent::mostrarDatos();
    return "$datosComunes / $this->sueldo";
}
```

Al prefijar un servicio con *parent* ejecutamos el servicio de la superclase.

Ventaja: si por algún motivo se cambia el servicio del padre, el servicio del hijo queda automáticamente corregido.

ALCYON
IT Services

POLIMORFISMO

ALCYON
IT Services

POLIMORFISMO I

Objetos de clases diferentes, aunque dentro de una misma jerarquía de clases, tienen el mismo servicio implementado de manera diferente.

El polimorfismo permite hacer algoritmos independientes del tipo de datos (léase: de la clase de los objetos).

Recordemos el servicio descuentoCliente() para cada tipo de cliente (Minorista y Mayorista).

ALCYON
IT Services

POLIMORFISMO II

ENLACE ESTÁTICO Y ENLACE DINÁMICO

Enlace estático: En tiempo de compilación se conoce el tipo (la clase) del objeto.

Enlace dinámico: Al compilar el programa el compilador no puede deducir el tipo del objeto y no sabe a que función llamar. Se ve forzado a generar código para que sea en tiempo de ejecución cuando pueda interrogar al objeto y resolver entonces la llamada a la función.

(Nuevamente, el ejemplo de cada cliente con su propia rutina de calculo de descuento).

POLIMORFISMO III

ENLACE ESTÁTICO Y ENLACE DINÁMICO

Si tenemos un método sobreescrito, siempre se llamará al método de la subclase con la que hayamos creado el objeto al hacer el new

Ej:

```
$cli = new CliMayorista;

$cli->calcularDto();
```

Si ambos objetos (Cliente y CliMayorista) tienen el método calcularDto() se ejecutará en este caso el correspondiente a CliMayorista aunque la variable cli sea del tipo Cliente

Pero Cliente tiene que tener el método también

POLIMORFISMO IV

ENLACE ESTÁTICO Y ENLACE DINÁMICO

`EJERCICIO:`

Queremos disponer de una función en el programa ‘Principal’ llamado imprimirAlumno() que al pasarle cualquiera de los siguientes tipos de alumnos: *AlumnoBecario* o *AlumnoDoctorando* imprima la siguiente información:

Clase del objeto

Nombre del alumno

NIF

Codigo

A continuación se dan más datos:

POLIMORFISMO V

ENLACE ESTÁTICO Y ENLACE DINÁMICO

Notas

Para saber a qué clase pertenece un objeto sólo tenemos que llamar a su método **get_class($objeto)** .

```
function imprimirAlumno($obj) {
        echo 'Class: ' . get_class($obj) . '<br>';
        echo 'Nif: ' . $obj->getNif() . '<br>';
        echo 'Nombre: ' . $obj->getNombre() . '<br>';
        echo 'Código: ' . $obj->getCodigo() . '<br>';
}
```

Podemos ver como se ejecuta el método getCodigo() que corresponde a *Becario* o *Doctorado*. En esto consiste el polimorfismo

Aviso Legal

Los derechos de propiedad intelectual sobre el presente documento son titularidad de David Alcolea Martinez Administrador, propietario y responsable de www.alcyon-it.com El ejercicio exclusivo de los derechos de reproducción, distribución, comunicación pública y transformación pertenecen a la citada persona.

Queda totalmente prohibida la reproducción total o parcial de las presentes diapositivas fuera del ámbito privado (impresora doméstica, uso individual, sin ánimo de lucro).

La ley que ampara los derechos de autor establece que: “La introducción de una obra en una base de datos o en una página web accesible a través de Internet constituye un acto de comunicación pública y precisa la autorización expresa del autor”. El contenido de esta obra está protegido por la Ley, que establece penas de prisión y/o multa, además de las correspondientes indemnizaciones por daños y perjuicios, para quienes reprodujesen, plagiaren, distribuyeren o comunicaren públicamente, en todo o en parte, o su transformación, interpretación o ejecución fijada en cualquier tipo de soporte o comunicada a través de cualquier medio.

El usuario que acceda a este documento no puede copiar, modificar, distribuir, transmitir, reproducir, publicar, ceder, vender los elementos anteriormente mencionados o un extracto de los mismos o crear nuevos productos o servicios derivados de la información que contiene.

Cualquier reproducción, transmisión, adaptación, traducción, modificación, comunicación al público, o cualquier otra explotación de todo o parte del contenido de este documento, efectuada de cualquier forma o por cualquier medio, electrónico, mecánico u otro, están estrictamente prohibidos salvo autorización previa por escrito de David Alcolea. El autor de la presente obra podría autorizar a que se reproduzcan sus contenidos en otro sitio web u otro soporte (libro, revista, e-book, etc.) siempre y cuando se produzcan dos condiciones:

1. Se solicite previamente por escrito mediante email o mediante correo ordinario.

2. En caso de aceptación, no se modifiquen los textos y se cite la fuente con absoluta claridad.

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

Queda totalmente prohibida la reproducción total o parcial de las presentes diapositivas fuera del ámbito privado (impresora doméstica, uso individual, sin ánimo de lucro) así como su distribución sin la autorización explícita y por escrito de su autor.