Curso de Programación en PHP

Nivel I

Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología – Oro Verde – v2.1

Capítulo 3: Programación Orientada a Objetos en PHP

Algo de UML

Concepto de Abstracción

Relaciones entre clases utilizando PHP

Ejemplo de Clases y objetos en PHP

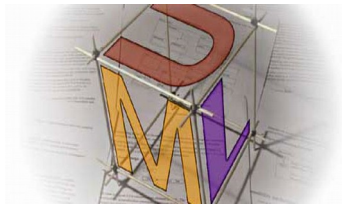
Encapsulamiento y principio de ocultación

Accesores/getter y modificadores/setter

Constantes de clases

$this vs self y el operador ::

Clase 6: Algo de UML



¿Qué es UML?

Es un **L**enguaje **U**nificado de **M**odelado estandarizado que nos permite escribir planos de Software.

¿Por qué usar UML?

Por que es apropiado para modelar casi cualquier tipo de aplicación (Escritorio, Web, de tiempo real, etc.). Es un lenguaje muy expresivo. No es difícil de aprender ni de utilizar.

Clase 6: Algo de UML

La única forma de aprender un nuevo lenguaje de programación es programando en él.

Brian Kernighan y Denis Ritchie

Para que nos sirve UML?

Es un lenguaje para:

- **Visualizar**
- **Especificar**
- **Construir**
- **Documentar**

UML es un lenguaje para Visualizar

La mayoría de los programadores piensa que la distancia entre pensar en una implementación y transformarla en código fuente es casi nula. Lo piensas, lo codificas.

Los modelos UML permiten una **mejor comunicación**.

UML **transciende** de lo que puede ser **representado** por un lenguaje de **programación**.

Todos **los símbolos** de UML poseen una semántica bien definida **evitando ambigüedades**.

UML es un lenguaje para Especificar

Al decir especificar nos referimos a construir **modelos precisos**, no ambiguos y completos en UML.

UML cubre las especificaciones de todas las **etapas del desarrollo** (análisis, diseño e implementación).

UML es un lenguaje para Construir

En primer lugar diremos que UML no es un lenguaje de programación Visual, pero nos **permite conectar** algunos **modelos** a una gran variedad de **lenguajes de programación**.

Como ejemplo, podemos citar Java, PHP, C++ o Visual Basic.

Incluso se podría conectar a tablas de una base de datos relacional como así también a bases de datos orientadas a objetos.

UML es un lenguaje para Documentar

UML cubre la documentación de la arquitectura de un sistema y todos sus detalles.

Posee un lenguaje para expresar los requisitos y pruebas.

También permite modelar las actividades de planificación y gestión.

Dónde puede utilizarse UML?

En desarrollos de gran cantidad de Software como por ejemplo:

- **Sistemas empresariales**
- **Bancos y entidades financieras**
- **Telecomunicaciones**
- **Electrónica médica**
- **Comercio**
- **Sistemas basados en la Web**

Clase 6: Algo de UML

Modelo conceptual de UML

- **Bloques básicos de UML** (Elementos, Relaciones y Diagramas)
- Reglas
- Mecanismos comunes

Clase 6: Algo de UML

Bloques básicos de UML

- **Elementos en UML**
  (Estructurales, comportamiento, agrupación y anotación)
- **Relaciones en UML**
  (Dependencia, Asociación, Generalización, Realización)
- **Diagramas en UML**
  (Clases, Objetos, Casos de uso, Secuencia, Estados)

Diagrama de clases

Un diagrama de clases nos permite **visualizar** un conjunto de **clases**, **interfaces** y **colaboraciones**, así como **sus relaciones**.

Es uno de los diagramas más utilizados en el modelado de sistemas POO.

Los diagramas de clases logran una **vista estática del sistema**.

Ejemplo de un diagrama de clases

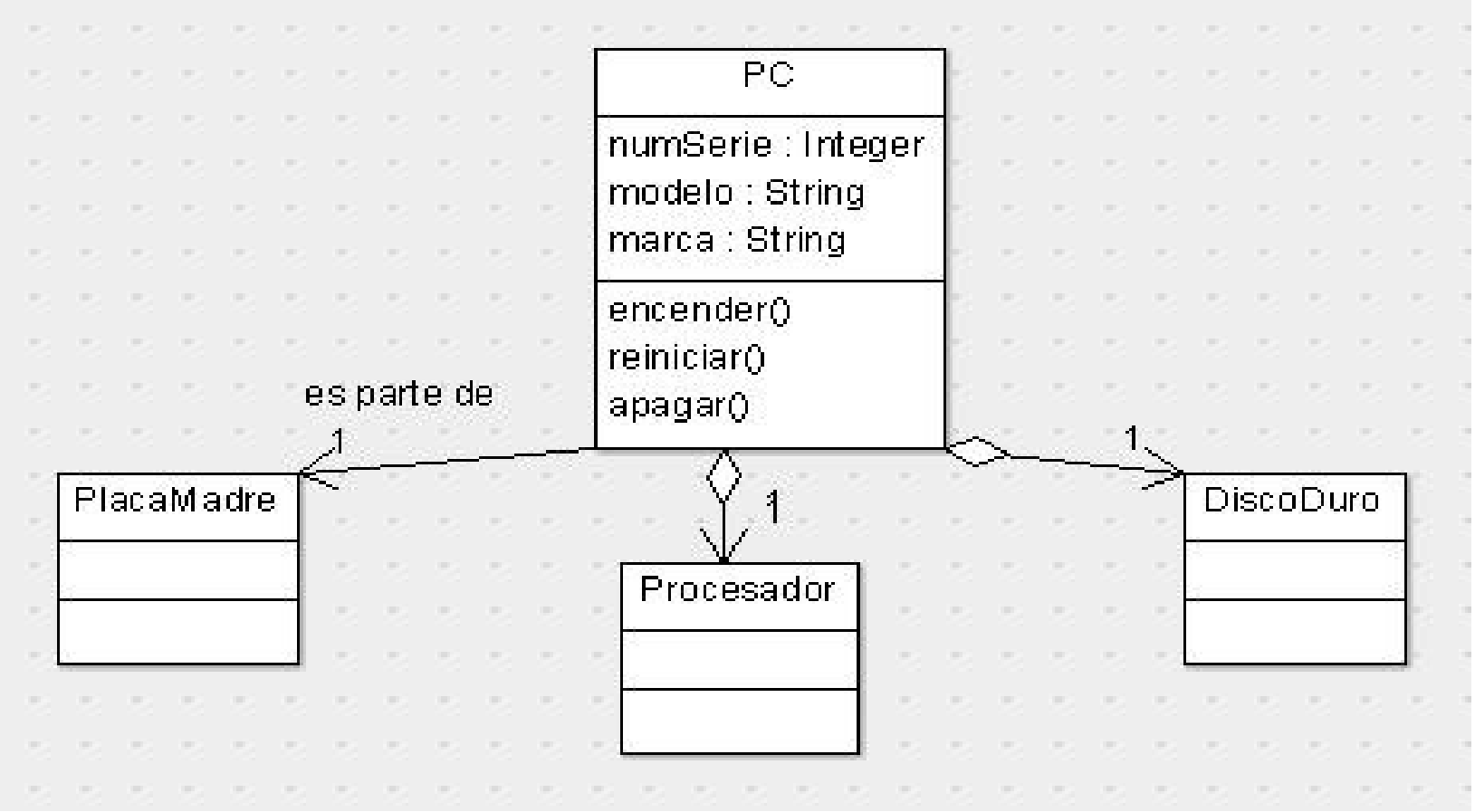


Diagrama de objetos

Un diagrama de objetos nos permite visualizar un **conjunto de objetos y sus relaciones**.

Es otro de los diagramas más utilizados en el modelado de sistemas POO.

Los diagramas de objetos representan **instantáneas estáticas de instancias de los elementos existentes en los diagramas de clases**.

Diagrama de casos de uso

Un diagrama de casos de uso se utilizan para **modelar aspectos dinámicos** de un sistema.

Es otro de los diagramas más utilizados en el modelado de sistemas POO.

Los diagramas de Casos de Uso son **importantes** para modelar el **comportamiento de un sistema**, subsistema o de una clase.

Cada diagrama muestra un conjunto de **casos de uso**, **actores** y sus **relaciones**.

Capítulo 3: Programación Orientada a Objetos en PHP

~~Algo de UML~~

Concepto de Abstracción

Relaciones entre clases utilizando PHP

Ejemplo de Clases y objetos en PHP

Encapsulamiento y principio de ocultación

Accesores/getter y modificadores/setter

Constantes de clases

$this vs self y el operador ::

Concepto de abstracción

¿Qué es abstracción?

Es la capacidad para encapsular y aislar la información del diseño y ejecución.

Los seres humanos han desarrollado una técnica excepcional y potente para tratar la complejidad: la abstracción.

Denota las características esenciales de un objeto, donde se capturan sus comportamientos y atributos

Capítulo 3: Programación Orientada a Objetos en PHP

~~Algo de UML~~

~~Concepto de Abstracción~~

Relaciones entre clases utilizando PHP

Ejemplo de Clases y objetos en PHP

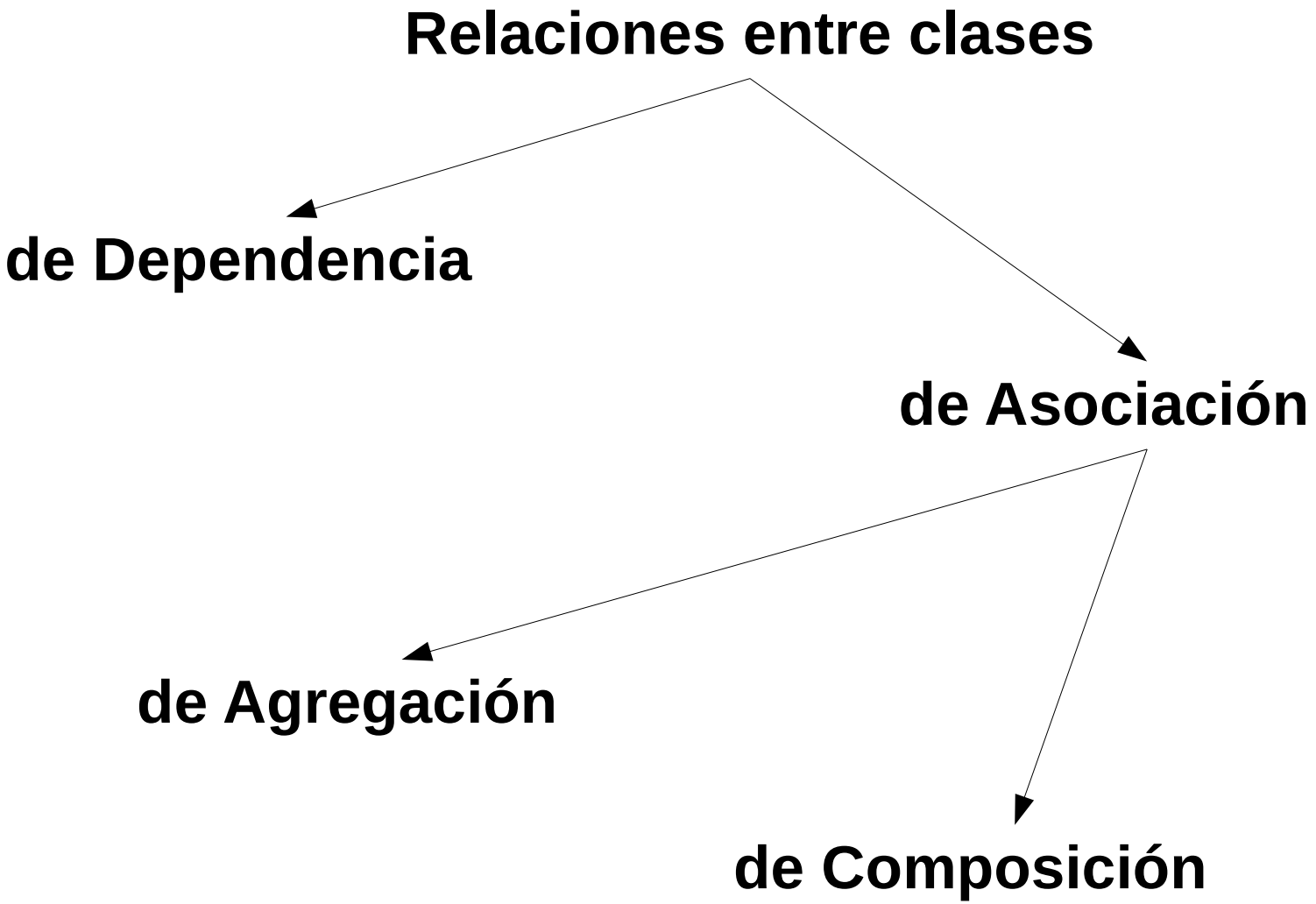
Encapsulamiento y principio de ocultación

Accesores/getter y modificadores/setter

Constantes de clases

$this vs self y el operador ::

Clase 6: Relaciones entre clases utilizando PHP



Relación de Dependencia

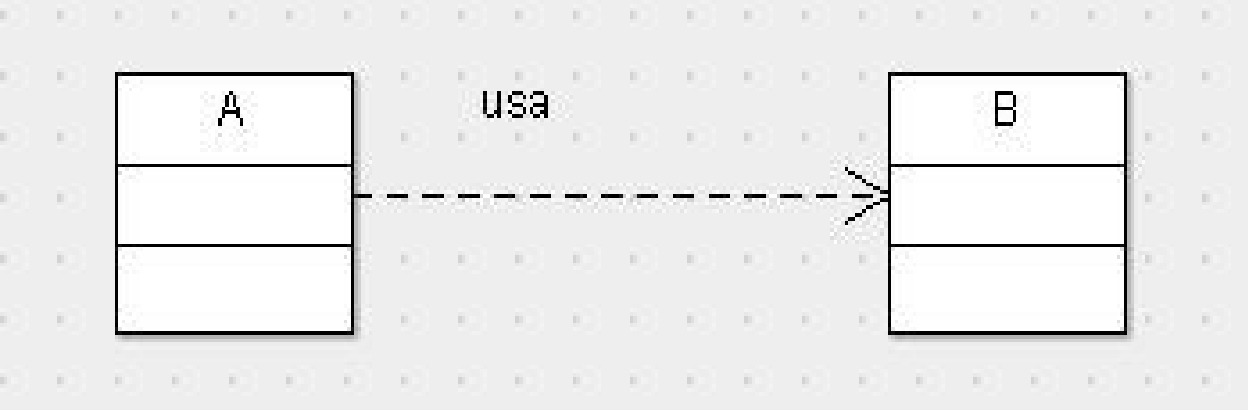
¿Qué es la relación de dependencia?

Es una relación de uso entre dos entidades (clases).

Donde tenemos una entidad independiente y otra dependiente.

Cualquier modificación de la entidad independiente afecta a la dependiente.

Relación de Dependencia utilizando UML



Relación de dependencia utilizando PHP
(dentro de un método)

```
<?php

require_once 'B.php';

class A
{

	public function mostrar()
	{

		$b = new B();

		/* codificación adicional */
	}

}
```

Relación de dependencia utilizando PHP

(como parámetro de un método)

```
<?php

require_once 'B.php';

class A
{

    public function mostrar(B $b)
    {

        echo $b;

        /* codificación adicional */
    }

}
```

Relación de Asociación

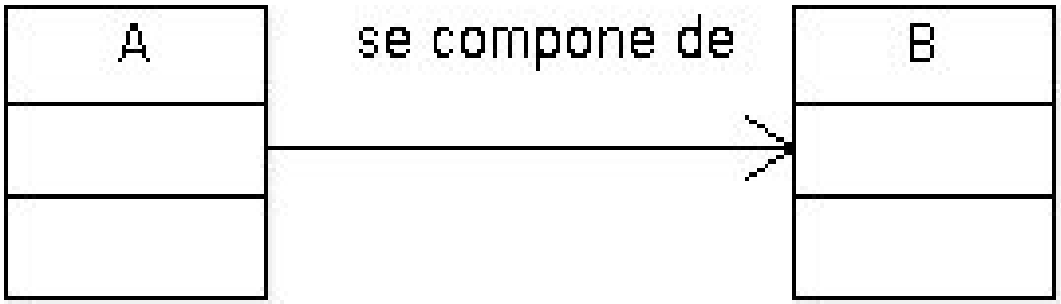
¿Qué es la relación de asociación?

Es una relación estructural entre entidades (clases).

Donde una clase se construye a partir de otras clases.

Este tipo de relación posee variaciones que se denominan relación de agregación y composición.

Relación de Asociación utilizando UML

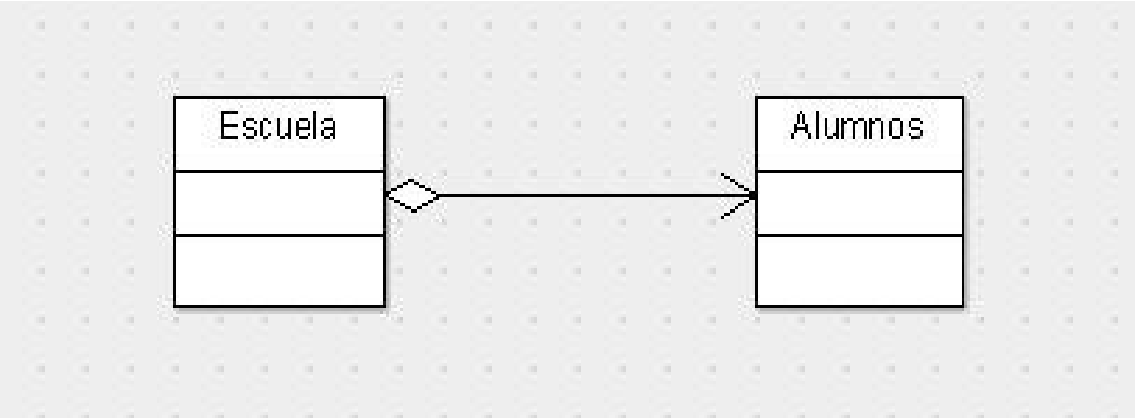


Relación de asociación utilizando PHP

```
<?php

require_once 'B.php';

class A
{
	private $_b;

	public function __construct()
	{

		$this->_b = new B();

		/* codificación adicional */
	}

}
```

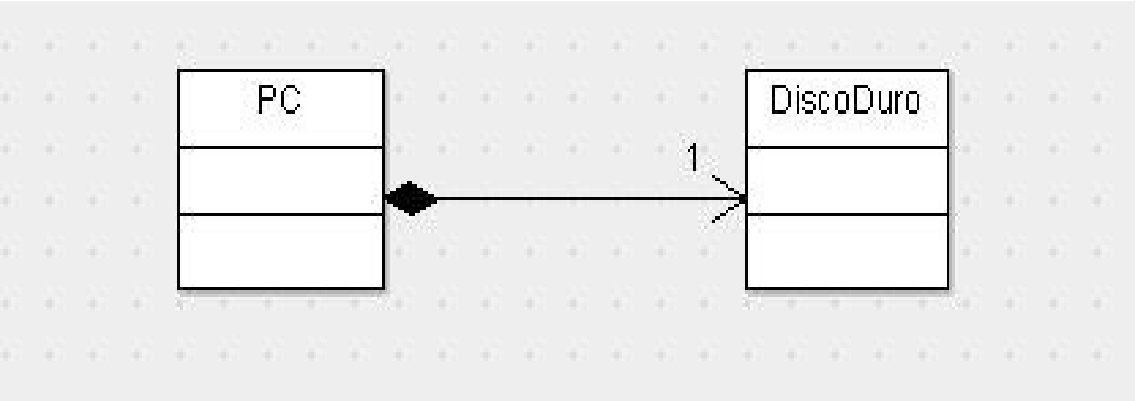
Clase 6: Relaciones entre clases utilizando PHP

Relación de Asociación (agregación)



Una escuela agrupa muchos alumnos

Relación de Asociación (composición)



Una computadora personal está compuesta de al menos un disco duro

Relación de asociación utilizando PHP
(de composición)

```
<?php

require_once 'DiscoDuro.php';

class Pc
{
	private $_discosDuros = array();
	public function agregarDiscoDuro(DiscoDuro $discosDuros)
	{
		$this->_discosDuros[] = $discosDuros;
	}
}

$computadora = new Pc();
$computadora->agregarDiscoDuro(new DiscoDuro());
$computadora->agregarDiscoDuro(new DiscoDuro());
```

Capítulo 3: Programación Orientada a Objetos en PHP

~~Algo de UML~~

~~Concepto de Abstracción~~

~~Relaciones entre clases utilizando PHP~~

Ejemplo de Clases y objetos en PHP

Encapsulamiento y principio de ocultación

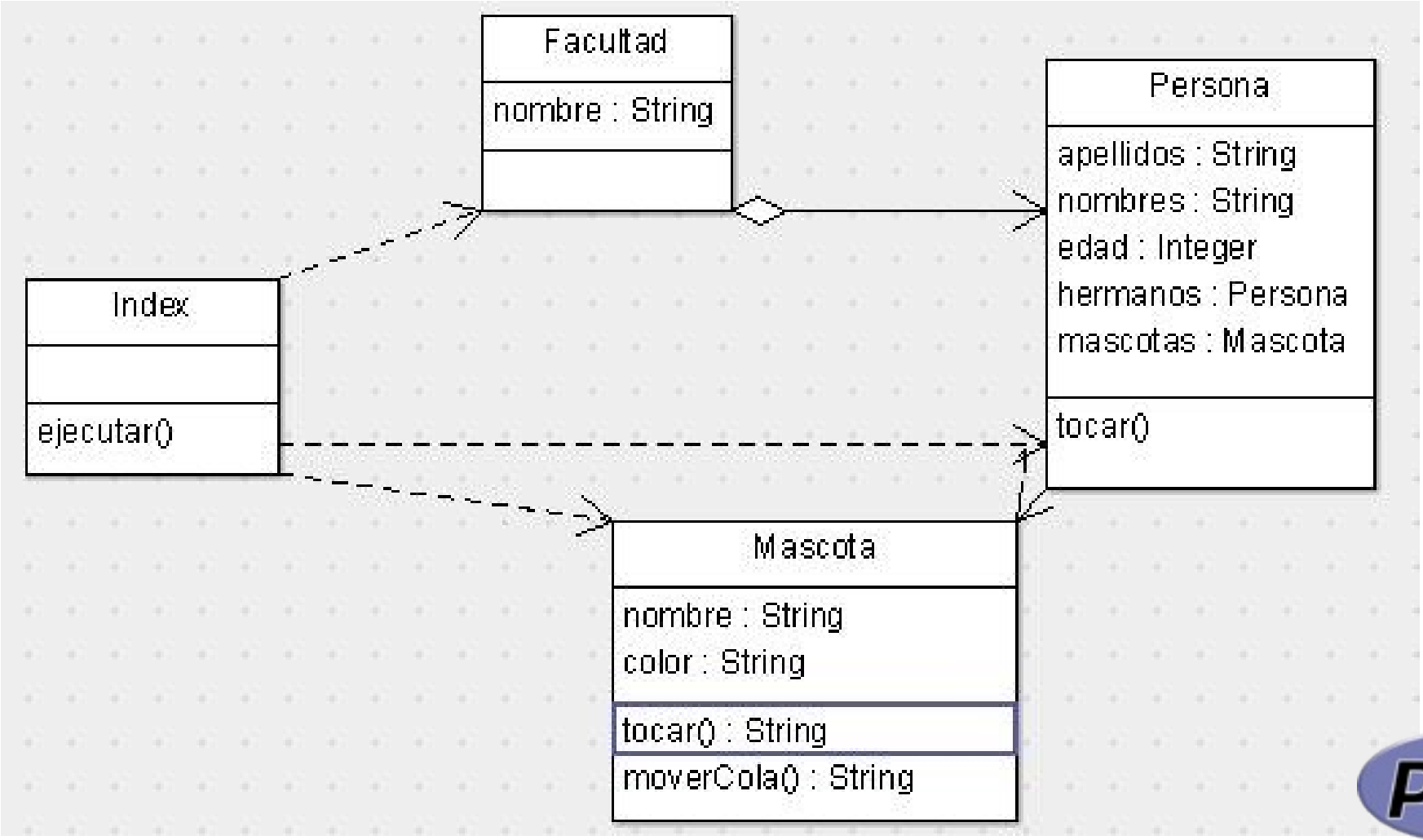
Accesores/getter y modificadores/setter

Constantes de clases

$this vs self y el operador ::

Clase 6: Ejemplo de Clases y objetos en PHP

Diagrama de clases (ejemplo de clases y objetos)



Codificación en PHP (ejemplo de clases y objetos)

index.php

```
<?php
require_once 'Persona.php';
require_once 'Facultad.php';
require_once 'Mascota.php';

class Index
{
	public static function ejecutar()
	{
		$persona = new Persona();
		$mascota = new Mascota();
		$facultad = new Facultad();
		/* código adicional */
	}
}

Index::ejecutar();
```

Clase 6: Ejemplo de Clases y objetos en PHP

Codificación en PHP (ejemplo de clases y objetos)

Facultad.php

```
<?php
require_once 'Persona.php';

class Facultad
{
	private $_nombre;
	private $_alumnos = array();
	public function __construct($nombre)
	{
		$this->_nombre = $nombre;
	}
	public function agregarAlumno(Persona $persona)
	{
		$this->_alumnos[] = $persona;
	}
}
```

Capítulo 3: Programación Orientada a Objetos en PHP

~~Algo de UML~~

~~Concepto de Abstracción~~

~~Relaciones entre clases utilizando PHP~~

~~Ejemplo de Clases y objetos en PHP~~

Encapsulamiento y principio de ocultación

Accesores/getter y modificadores/setter

Constantes de clases

$this vs self y el operador ::

Definición de encapsulamiento

Significa **reunir** a todos los **elementos** que pueden considerarse **pertenecientes a una misma entidad** y al mismo nivel de abstracción.

Se mantienen ocultos los procesos internos dando acceso solamente a los procesos que se necesitan.

Esto permite aumentar la coherencia de los componentes del sistema.

Capítulo 3: Programación Orientada a Objetos en PHP

~~Algo de UML~~

~~Concepto de Abstracción~~

~~Relaciones entre clases utilizando PHP~~

~~Ejemplo de Clases y objetos en PHP~~

~~Encapsulamiento y principio de ocultación~~

Accesores/getter y modificadores/setter

Constantes de clases

$this vs self y el operador ::

Métodos setter y getter

Método setter

Permite cargar un valor a un atributo. (asignar un valor de una variable)

Método getter

Permite retornar un valor de una propiedad. (solo se devuelve la información de un atributo para quién la solicita)

Ejemplo de setter y getter

Persona.php

```
<?php
class Persona
{
	private $_apellidos;
	private $_nombres;

	public function setApellidos($apellidos)
	{
		$this->_apellidos = $apellidos;
	}
	public function getApellidos()
	{
		return $this->_apellidos;
	}

} // Fin de la definición de la clase
?>
```

Capítulo 3: Programación Orientada a Objetos en PHP

~~Algo de UML~~

~~Concepto de Abstracción~~

~~Relaciones entre clases utilizando PHP~~

~~Ejemplo de Clases y objetos en PHP~~

~~Encapsulamiento y principio de ocultación~~

~~Accesores/getter y modificadores/setter~~

Constantes de clases

$this vs self y el operador ::

Constantes dentro de una clase

Se pueden definir valores constantes en función de cada clase manteniéndola invariable.

Las **constantes** se diferencian de variables comunes en que **no utilizan** el símbolo **$** al declararlas o usarlas.

El valor debe ser una expresión constante, no (por ejemplo) una variable, una propiedad, un resultado de una operación matemática, o una llamada a una función.

También es posible tener constantes para interfaces.

Ejemplo de definición y uso de constantes de clase

```
<?php
class Prueba
{
	// Definición de constantes y atributos
	const CADENA = 'Mi constante de tipo cadena';
	const NUMERICA = 10;
	const NUMERICAREAL = 2.1;
	const BOLEANA = true;

	// Definición de métodos
} // Prueba.php

echo Prueba::CADENA; // Salida: Mi constante de tipo cadena
```

Ejemplo de definición y uso de constantes de clase

```
<?php
class Prueba
{
	// Definición de constantes y atributos
	const CADENA = 'Mi constante de tipo cadena';
	const NUMERICA = 10;
	const NUMERICAREAL = 2.1;
	const BOLEANA = true;

	// Definición de métodos
} // Prueba.php

echo Prueba::CADENA; // Salida: Mi constante de tipo cadena
```

Capítulo 3: Programación Orientada a Objetos en PHP

~~Algo de UML~~

~~Concepto de Abstracción~~

~~Relaciones entre clases utilizando PHP~~

~~Ejemplo de Clases y objetos en PHP~~

~~Encapsulamiento y principio de ocultación~~

~~Accesores/getter y modificadores/setter~~

~~Constantes de clases~~

$this vs self y el operador ::

La palabra reservada y variable $this

La palabra reserva **this** nos permite **acceder** a **variables y/o métodos** de objetos de una **instancia particular**.

Se puede ver como una variable (**$this**) que apunta a una instancia particular brindando acceso a las propiedades y comportamientos de un objeto dentro de la definición de una clase.

La palabra reservada self

La palabra reserva **self** nos permite acceder a **variables estáticas y constantes de clases** desde el **interior** de la definición de la misma.

Se pueden acceder a las constantes y variables sin necesidad de instanciar la clase teniendo en cuenta que son propias de la clase y no de una instancia a la misma.

Self puede ser reemplaza con el nombre de la clase, pero el código queda más prolijo con el uso de **self**.

Operador de resolución de ámbito (::)

El Operador de Resolución de Ámbito (también denominado Paamayim Nekudotayim) o en términos simples, el doble dos-puntos, es un **token** que **permite acceder a elementos estáticos, constantes, y propiedades o métodos** de una clase.

Cuando se hace referencia a estos items desde el exterior de la definición de la clase, se utiliza el nombre de la clase.

Paamayim Nekudotayim podría, en un principio, parecer una extraña elección para bautizar a un doble dos-puntos. Sin embargo, mientras se escribía el Zend Engine 0.5 (que utilizó PHP 3), así es como el equipo Zend decidió bautizarlo. En realidad, significa doble dos-puntos - en Hebreo!

Ejemplo de uso de constantes de clase desde el exterior de la definición de la clase

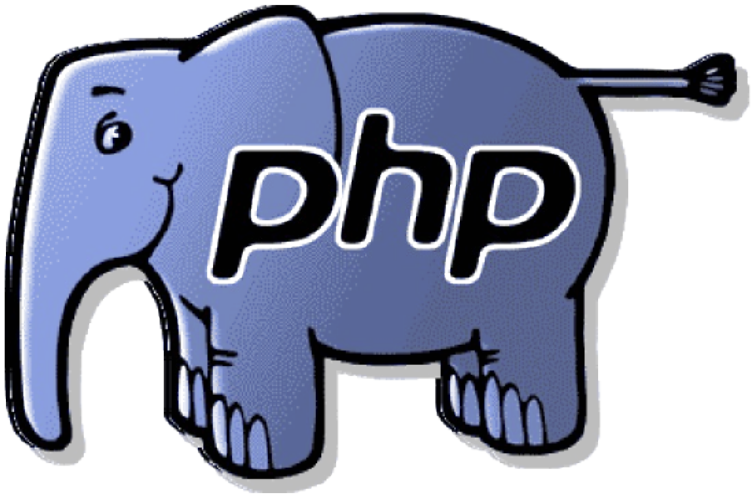
```
<?php
class Prueba
{
    // Definición de constantes y atributos
    const MICONSTANTE = 'Mi constante de tipo cadena';
    // Definición de métodos
} // Prueba.php
```

```
echo Prueba::MICONSTANTE;        // Salida: Mi constante de tipo cadena
```

Ejemplo de uso de constantes de clase desde el interior de la definición de la clase

```
<?php
class Prueba
{
	// Definición de constantes y atributos
	const MICONSTANTE = 'Mi constante de tipo cadena';

	Public function obtenerValorConstante()
	{
		return self::MICONSTANTE;
	}
	// Definición de métodos
}
```

¿Dudas?



¿Consultas?

Información de contacto

Web:

http://www.gugler.com.ar

http://campusvirtual.gugler.com.ar

http://www.facebook.com/gugler.com.ar

http://www.twitter.com/cgugler

Mail:

contacto@gugler.com.ar

academica@gugler.com.ar

administracion@gugler.com.ar

Versión 2.1