Capítulo 3
Utilizar las funciones PHP

1. Preámbulo

El objetivo de este capítulo es presentar las funciones más útiles para el desarrollo de un sitio web.

PHP ofrece numerosas funciones; la descripción de cada función está disponible en línea en el sitio www.php.net.

Versión 8

Desde la **versión 8**, es posible pasar parámetros a una función utilizando el nombre del parámetro en lugar de su posición. Esta funcionalidad se presenta en el capítulo Escribir funciones y clases PHP, pero puede utilizarse para las funciones propias del lenguaje PHP y, por tanto, para las funciones que se presentan en este capítulo. Sin embargo, en este capítulo, los nombres reales de los parámetros de las funciones no se presentan (están traducidos); para conocerlos, consulte la documentación en línea de las funciones…

Desde la **version 8.1**, pasar el valor `NULL` en un parámetro que no es explícitamente opcional es obsoleto, y por lo tanto, genera una alerta `E_DEPRECATED`.

Ejemplo

```
<?php
$x = null;
$n = strlen($x);
?>
```

Resultado

```
Deprecated: strlen(): Passing null to parameter #1 ($string) of type string
is deprecated in /app/scripts/index.php on line 3
```

2. Manipular las constantes, las variables y los tipos de datos

2.1 Constantes

PHP ofrece una serie de funciones útiles sobre las constantes:

Nombre	Función
`defined`	Indica si una constante está definida o no.
`constant`	Devuelve el valor de una constante.

defined

La función `defined` permite saber si una constante está definida o no.

Sintaxis

booleano `defined(`*cadena* `nombre)`

`nombre` Nombre de la constante.

La función `defined` devuelve `TRUE` si la constante está definida y `FALSE` en caso contrario.

Ejemplo

```
<?php
// Probar si la constante CONSTANTE está definida.
$ok = defined('CONSTANTE');
if ($ok) {
  echo 'CONSTANTE está definida.<br />';
} else {
  echo 'CONSTANTE no está definida.<br />';
};
// Definir la constante CONSTANTE
define('CONSTANTE','valor de la CONSTANTE');
// Probar si la constante CONSTANTE está definida.
$ok = defined('CONSTANTE');
if ($ok) {
  echo 'CONSTANTE está definida.<br />';
} else {
```

```
  echo 'CONSTANTE no está definida.<br />';
};
?>
```

Resultado

```
CONSTANTE no está definida.
CONSTANTE está definida.
```

constant

La función `constant` devuelve el valor de una constante cuyo nombre se pasa como parámetro.

Sintaxis

mixto constant(*cadena* nombre)

Donde

`nombre` Nombre de la constante.

Esta función es útil para recuperar el valor de una constante cuyo nombre no se conoce a priori.

Ejemplo

```
<?php
// definir el nombre de la constante en una variable
$nombreConstante = 'OTRA CONSTANTE';
// definir el valor de la constante
define($nombreConstante,'valor de la OTRA CONSTANTE');
// mostrar el valor de la constante
echo $nombreConstante,' = ',constant($nombreConstante);
?>
```

Resultado

```
OTRA CONSTANTE = valor de la OTRA CONSTANTE
```

Otras funciones permiten conocer el tipo de una constante (véase la sección Manipular las constantes, las variables y los tipos de datos - Tipos de datos).

2.2 Variables

PHP ofrece una serie de funciones útiles en las variables:

Nombre	Función
`empty`	Indica si una variable está vacía o no.
`isset`	Indica si una o varias variables están definidas o no.

Nombre	Función
unset	Elimina una o varias variables.
var_dump	Muestra la información sobre una o varias variables (tipo y valor).

empty

La función `empty` permite probar si una variable está vacía o no.

Sintaxis

```
booleano empty(mixto variable)
```

`variable` Variable que se va a probar.

`empty` devuelve TRUE si la variable está definida y FALSE en caso contrario.

Una variable se considera vacía si no ha sido asignada o si contiene una cadena vacía (""), una cadena igual a 0 ("0"), un 0, NULL, FALSE o una tabla vacía.

La función `empty` también se puede utilizar para probar si una expresión está vacía o no.

Ejemplo

```
<?php
// Prueba de una variable no inicializada.
$está_vacía = empty($variable);
echo '$variable no inicializada<br />';
if ($está_vacía) {
  echo '=> $variable está vacía.<br />';
} else {
  echo '=> $variable no está vacía.<br />';
}
// Prueba de una variable que contiene una cadena vacía.
$variable = '';
$está_vacía = empty($variable);
echo '$variable = \'\'<br />';
if ($está_vacía) {
  echo '=> $variable está vacía.<br />';
} else {
  echo '=> $variable no está vacía.<br />';
}
// Prueba de una variable que contiene una cadena igual a 0.
$variable = '0';
$está_vacía = empty($variable);
echo '$variable = \'',$variable,'\'<br />';
```

```
if ($está_vacía) {
  echo '=> $variable está vacía.<br />';
} else {
  echo '=> $variable no está vacía.<br />';
}
// Prueba de una variable que contiene 0.
$variable = 0;
$está_vacía = empty($variable);
echo '$variable = ',$variable,'<br />';
if ($está_vacía) {
  echo '=> $variable está vacía.<br />';
} else {
  echo '=> $variable no está vacía.<br />';
}
// Prueba de una variable que contiene una cadena no vacía.
$variable = 'x';
$está_vacía = empty($variable);
echo '$variable = \'',$variable,'\'<br />';
if ($está_vacía) {
  echo '=> $variable está vacía.<br />';
} else {
  echo '=> $variable no está vacía.<br />';
}
?>
```

Resultado

```
$variable no inicializada
=> $variable está vacía.
$variable = ''
=> $variable está vacía.
$variable = '0'
=> $variable está vacía.
$variable = 0
=> $variable está vacía.
$variable = 'x'
=> $variable no está vacía.
```

isset

La función `isset` permite probar si una o varias variables están definidas o no.

Sintaxis

`booleano` `isset(`*`mixto`* `variable[,...])`

`variable` Variable que se va a probar; pueden ser varias, separadas por una coma.

`isset` devuelve **TRUE** si la variable está definida y **FALSE** en caso contrario.

Si se facilitan varios parámetros, la función devuelve TRUE únicamente si se definen todas las variables.

Una variable se considera como no definida si no se ha visto asignada o si contiene NULL. A diferencia de la función empty, una variable que contiene una cadena vacía (""), una cadena igual a 0 ("0"), un 0, un FALSE o una tabla vacía, no se considera como no definida.

Ejemplo

```
<?php
// Prueba de una variable no inicializada.
$está_definida = isset($variable);
echo '$variable no inicializada<br />';
if ($está_definida) {
  echo '=> $variable está definida.<br />';
} else {
  echo '=> $variable no está definida.<br />';
}
// Prueba de una variable que contiene una cadena vacía.
$variable = '';
$está_definida= isset($variable);
echo '$variable = \'\'<br />';
if ($está_definida) {
  echo '=> $variable está definida.<br />';
} else {
  echo '=> $variable no está definida.<br />';
}
// Prueba de una variable que contiene una cadena igual a 0.
$variable = '0';
$está_definida = isset($variable);
echo '$variable = \'',$variable,'\'<br />';
if ($está_definida) {
  echo '=> $variable está definida.<br />';
} else {
  echo '=> $variable no está definida.<br />';
}
// Prueba de una variable que contiene 0.
$variable = 0;
$está_definida = isset($variable);
echo '$variable = ',$variable,'<br />';
if ($está_definida) {
  echo '=> $variable está definida.<br />';
} else {
  echo '=> $variable no está definida.<br />';
}
// Prueba de una variable que contiene una cadena no vacía.
$variable = 'x';
```

```
$está_definida = isset($variable);
echo '$variable = \'',$variable,'\'<br />';
if ($está_definida) {
  echo '=> $variable está definida.<br />';
} else {
  echo '=> $variable no está definida.<br />';
}
?>
```

Resultado

```
$variable no inicializada
=> $variable no está definida.
$variable = ''
=> $variable está definida.
$variable = '0'
=> $variable está definida.
$variable = 0
=> $variable está definida.
$variable = 'x'
=> $variable está definida.
```

unset

La función `unset` permite eliminar una o varias variables.

Sintaxis

```
unset(mixto variable)
```

`variable` Variable que se va a eliminar (para eliminar varias, deben estar separadas por una coma).

Después de la eliminación, la variable se encuentra en el mismo estado que si no hubiera sido asignada. El uso de la función `isset` en una variable eliminada devuelve FALSE.

Ejemplo

```
<?php
// Definir una variable.
$variable = 1;
// Mostrar la variable y probar si está definida.
$está_definida = isset($variable);
echo '$variable = ',$variable,'<br />';
if ($está_definida) {
  echo '=> $variable está definida.<br />';
} else {
  echo '=> $variable no está definida.<br />';
}
```

```
// Eliminar la variable.
unset($variable);
// Mostrar la variable y probar si está definida.
$está_definida = isset($variable);
echo '$variable = ',$variable,'<br />';
if ($está_definida) {
  echo '=> $variable está definida.<br />';
} else {
  echo '=> $variable no está definida.<br />';
}
?>
```

Resultado

```
$variable = 1
=> $variable está definida.
$variable =
=> $variable no está definida.
```

Observación

Al asignar un 0 o una cadena vacía a una variable, no se borra.

var_dump

La función `var_dump` muestra información sobre una o varias variables (tipo y contenido).

Sintaxis

```
var_dump(mixto variable, [,...])
```

`variable` Variable que se va a mostrar (pueden ser varias, separadas por una coma).

La función `var_dump` es especialmente interesante en las fases de desarrollo.

Ejemplo

```
<?php
// mostrar la información sobre una variable no inicializada
$variable = NULL;
var_dump($variable);
// inicializar la variable con un número entero
$variable = 10;
// mostrar la información sobre una variable
echo '<br />';
var_dump($variable);
// modificar el valor (y el tipo) de la variable
$variable = 3.14; // número decimal
```

```
// mostrar la información sobre una variable
echo '<br />';
var_dump($variable);
// modificar el valor (y el tipo) de la variable
$variable = 'abc'; // cadena de caracteres
// mostrar la información sobre la variable
echo '<br />';
var_dump($variable);
?>
```

Resultado

```
NULL
int(10)
float(3.14)
string(3) "abc"
```

Para una variable no inicializada, `var_dump` devuelve `NULL`. Para un número, `var_dump` indica el tipo (`int` = entero, `float` = número decimal), seguido por el valor entre paréntesis. Para una cadena, `var_dump` indica el tipo (`string`), seguido de la longitud entre paréntesis, seguido por el valor entre comillas.

PHP también ofrece las funciones `print_r` y `var_export`, que son similares a la función `var_dump`. La función `print_r` muestra o devuelve el contenido de la variable en una forma más legible, sin mencionar el tipo de datos. La función `var_export` muestra o devuelve una cadena que ofrece un código PHP de definición de la variable.

Observación

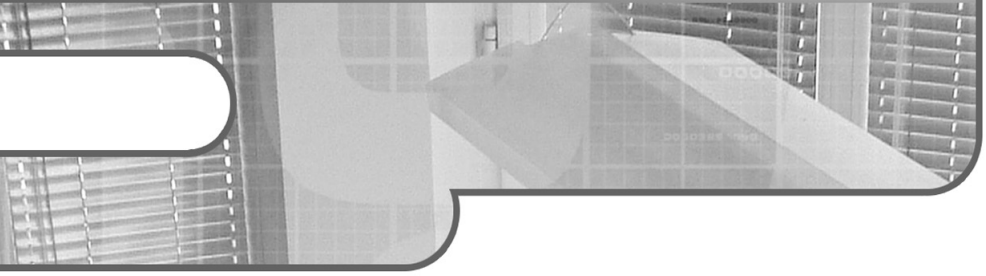
En la sección Tipos de datos de este capítulo, estudiaremos otras funciones que permiten determinar el tipo de una variable y realizar conversiones de tipos (de número a cadena, de cadena a número...).

2.3 Tipos de datos

2.3.1 Conversiones

PHP es capaz de realizar las conversiones automáticas implícitas de tipo.

Cuando un valor/expresión se asigna a una variable, la variable pasa a ser el tipo de valor/expresión.



Capítulo 12
El patrón de diseño Composite

1. Descripción

El objetivo del patrón de diseño `Composite` es materializar composiciones de objetos en forma de estructura de árbol. Estas composiciones están concebidas de tal forma que un cliente tratará indiferentemente un componente y un compuesto.

2. Ejemplo

En nuestro sistema de venta de vehículos, queremos representar las empresas cliente, en especial para conocer el número de vehículos de los que disponen y proporcionarles ofertas de mantenimiento para su parque de vehículos.

Las empresas que posean filiales solicitan ofertas de mantenimiento que tengan en cuenta el parque de vehículos de sus filiales.

Una solución inmediata consiste en procesar de forma diferente las empresas sin filiales y las que posean filiales. No obstante esta diferencia en el procesado entre ambos tipos de empresa haría que la aplicación fuera más compleja y totalmente dependiente de la composición interna de las empresas cliente.

El patrón de diseño `Composite` resuelve este problema unificando ambos tipos de empresa y utilizando la composición recursiva. Esta composición recursiva es necesaria puesto que una empresa puede tener filiales que posean, ellas mismas, otras filiales. Se trata de una composición en árbol tal y como se ilustra en la figura 12.1 donde las empresas madre se sitúan sobre sus filiales. Por razones evidentes de simplificación, suponemos que no hay filiales comunes a dos empresas.

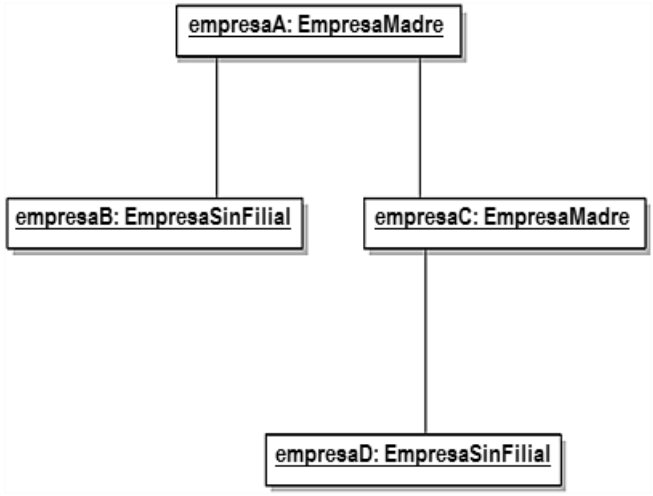


Figura 12.1 - Árbol de empresas madres y de sus filiales

La figura 12.2 muestra el diagrama de clases correspondiente. La clase abstracta `Empresa` contiene la interfaz destinada a los clientes. Posee dos subclases concretas, a saber `EmpresaSinFilial` y `EmpresaMadre`, esta última guarda una relación de agregación con la clase `Empresa` representando los enlaces con sus filiales.

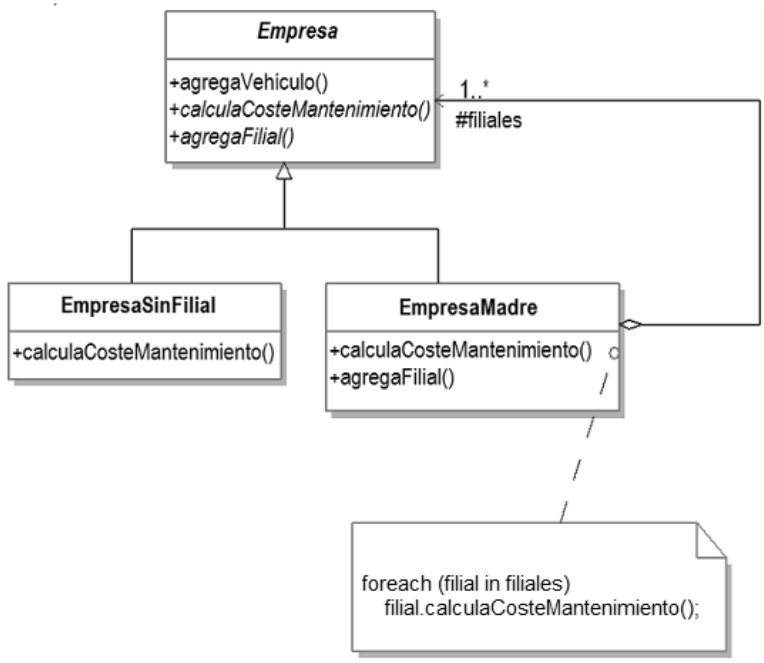


Figura 12.2 - El patrón de diseño `Composite` aplicado a la representación de empresas y sus filiales

La clase `Empresa` posee tres métodos públicos de los cuales dos son abstractos. El método concreto es el método `agregaVehiculo` que no depende de la composición en filiales de la empresa. En cuanto a los otros dos métodos, se implementan en las subclases concretas (`agregaFilial` no tiene implementación en `EmpresaSinFilial`, por eso no se representa en el diagrama de clases).

3. Estructura

3.1 Diagrama de clases

La figura 12.3 detalla la estructura genérica del patrón de diseño `Composite`.

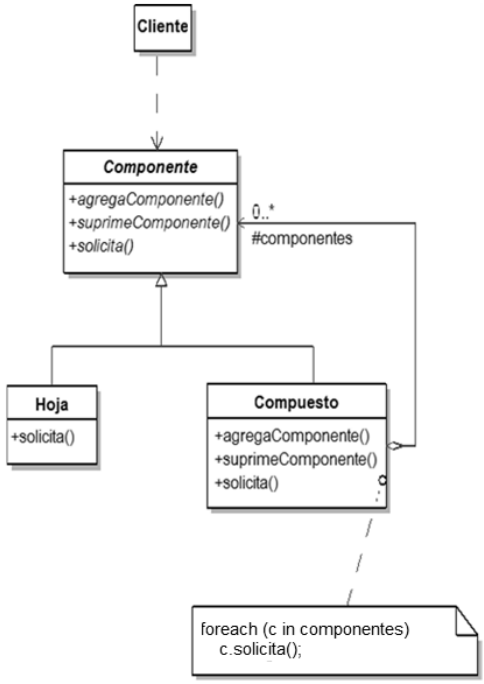


Figura 12.3 - Estructura del patrón de diseño `Composite`

3.2 Participantes

Los participantes del patrón de diseño `Composite` son los siguientes:

– `Componente` (`AbstractEmpresa`) es la clase abstracta que describe la interfaz de los objetos de la composición, implementa los métodos comunes e introduce la firma de los métodos que gestionan la composición agregando o suprimiendo componentes.

– `Hoja` (`EmpresasSinFilial`) es la clase concreta que representa las hojas de la composición (en una estructura de árbol, una hoja no posee componentes, es un nudo terminal).
– `Compuesto` (`EmpresaMadre`) es la clase concreta que representa los objetos compuestos de la jerarquía. Esta clase posee una asociación de agregación con la clase `Componente`.
– `Cliente` es la clase de los objetos que acceden a los objetos de la composición y los manipulan.

3.3 Colaboraciones

Los clientes envían sus peticiones a los componentes a través de la interfaz de la clase `Componente`.

Cuando un componente recibe una petición, reacciona en función de su clase. Si el componente es una hoja, procesa la petición tal y como se ilustra en la figura 12.4.

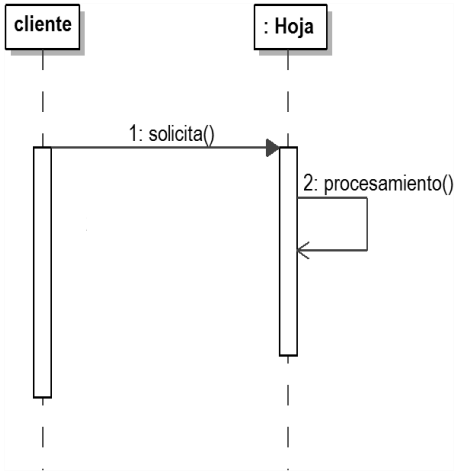


Figura 12.4 - Procesado de un mensaje por parte de una hoja

Si el componente es una instancia de la clase `Compuesto`, realiza un procesado previo, luego generalmente envía un mensaje a cada uno de sus componentes y realiza un procesamiento posterior. La figura 12.5 ilustra este comportamiento de llamada recursiva a otros componentes que van a procesar, a su vez, esta petición bien como hoja o bien como compuesto.

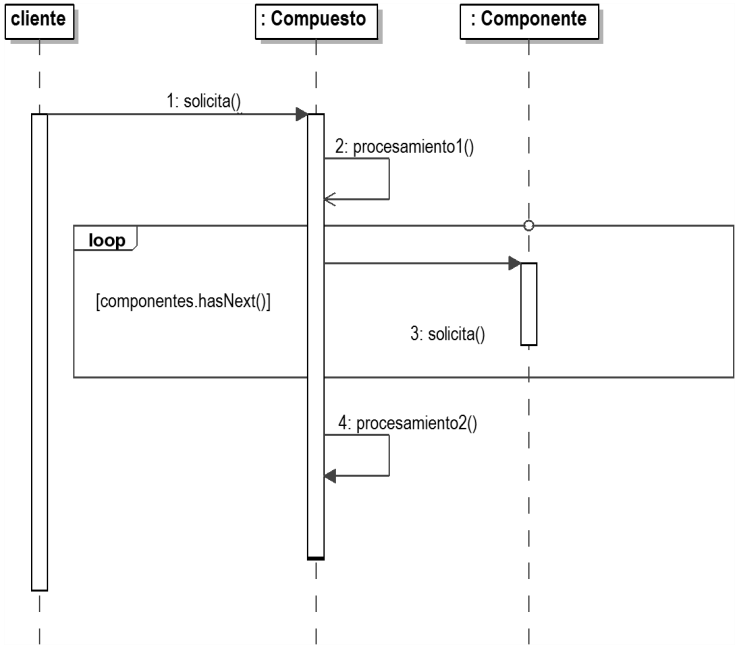


Figura 12.5 - Procesado de un mensaje por parte de un compuesto

4. Dominios de aplicación

El patrón de diseño Composite se utiliza en los siguientes casos:

– Es necesario representar jerarquías de composición en un sistema.

– Los clientes de una composición deben ignorar si se comunican con objetos compuestos o no.

5. Ejemplo en PHP

Retomemos el ejemplo de las empresas y la gestión de su parque de vehículos.

El código fuente en PHP de la clase abstracta `AbstractEmpresa` se describe a continuación. Conviene observar que el método `agregaFilial` devuelve un resultado booleano que indica si ha sido posible realizar o no la agregación.

```
<?php

declare(strict_types=1);

namespace ENI\DesignPatterns\Composite;

abstract class AbstractEmpresa
{
   protected static float $costeUnitarioVehiculo = 5;

   protected int $numVehiculos = 0;

   public function agregaVehiculo(): void
   {
       $this->numVehiculos++;
   }

   abstract public function calculaCosteMantenimiento(): float;

   abstract public function agregaFilial(AbstractEmpresa
$filial): bool;
}
?>
```

El código fuente de la clase `EmpresaSinFilial` aparece a continuación. Lógicamente, las instancias de esta clase no pueden agregar filiales.

```
<?php

declare(strict_types=1);

namespace ENI\DesignPatterns\Composite;

class EmpresaSinFilial extends AbstractEmpresa
{
   public function agregaFilial(AbstractEmpresa $filial): bool
```