

# Bloque A

## Instrucciones básicas de programación

### Unidad 4. Objetos & Clases



# Contenidos

1. Introducción
2. Páginas web como modelos
3. Cómo crear y usar objetos
4. Ventajas de usar objetos
5. Ejemplo final
6. Resumen
7. Referencias



# Introducción

# Introducción

Los **objetos** agrupan un conjunto de variables y funciones que representan cosas que se encuentran en el día a día, como artículos de noticias, productos a la venta o usuarios de un sitio web.

- En la unidad 2 vimos cómo las **variables** pueden almacenar información individual. Cuando una variable se utiliza en un objeto, se denomina **propiedad** del objeto.
- En la unidad 3, viste cómo las **funciones** pueden representar una tarea que tu código necesita realizar. Cuando una función se utiliza en un objeto, se denomina **método** del objeto.

# Introducción

Los sitios web a menudo necesitan representar múltiples de los mismos tipos de cosas.

Un sitio de noticias publicará muchos artículos, una tienda venderá muchos productos y un sitio que permite a los usuarios registrarse tendrá muchos miembros.

Cada una de estas cosas puede ser representada en código utilizando un **objeto**.

# Introducción

PHP utiliza algo llamado **clase** como plantilla para crear objetos que representen un tipo de cosa.

Por ejemplo, podemos utilizar una clase para crear objetos que representen productos y otra clase para crear objetos que representen miembros.

Cada objeto que se crea utilizando una clase recibe automáticamente las propiedades y métodos que se definen en esa clase.

# Introducción

Los objetos y las clases ayudan a organizar el código y facilitan su comprensión.

También es importante aprender cómo funcionan los objetos porque el intérprete PHP tiene varios objetos incorporados, los cuales comenzaremos a conocer en el siguiente bloque de este módulo.

# Introducción

El comienzo de esta unidad introduce los conceptos detrás de los objetos y cómo son utilizados.

A continuación, aprenderemos el código requerido para crear y utilizar objetos y clases.



## Páginas web como modelos

# Páginas web como modelos

Los **modelos** son representaciones de cosas del mundo que nos rodea.

Los programadores creamos modelos utilizando datos; luego utilizamos código para realizar tareas que manipulan los datos almacenados en esos modelos.

# Páginas web como modelos

Los sitios web utilizan datos para crear modelos que representan cosas de la vida cotidiana. Los programadores suelen referirse a estas cosas como distintos **tipos de objetos**. Por ejemplo:

- Personas (como clientes o miembros de un sitio)
- Productos o servicios que los visitantes podrían comprar (como libros, coches, cuentas bancarias o suscripciones de televisión).
- Documentos que tradicionalmente se imprimían (artículos de prensa, calendarios, entradas, etc.).

# Páginas web como modelos

Por ejemplo, un banco puede necesitar ciertos datos para representar a cada **cliente**:

- Nombre
- Apellidos
- Correo electrónico
- Contraseña

Necesitaría los mismos datos sobre cada persona, pero los nombres, direcciones de correo electrónico y contraseñas de cada cliente serían diferentes.

# Páginas web como modelos

También necesitaría conocer los mismos datos sobre cada **cuenta bancaria**, pero los valores utilizados para representar cada cuenta serían diferentes.

Por ejemplo:

- Número de cuenta
- Tipo de cuenta
- Saldo

# Páginas web como modelos

El banco puede **realizar tareas** utilizando estos datos. Por ejemplo, las tareas realizadas con una cuenta bancaria podrían incluir:

- Comprobar el saldo
- Hacer un ingreso
- Retirar dinero

Tareas como éstas obtienen o actualizan datos contenidos en variables.

Por ejemplo, si se retira dinero o se hace un ingreso, cambiará la cantidad almacenada para el saldo de esa cuenta.

# Páginas web como modelos

Del mismo modo, las tareas relacionadas con un cliente podrían incluir:

- Autenticar a un usuario (confirmar que es quien dice ser) comprobando que el correo electrónico y la contraseña que facilita coinciden con los datos almacenados sobre él.
- Obtener su nombre completo (combinando nombre y apellidos).

# Páginas web como modelos

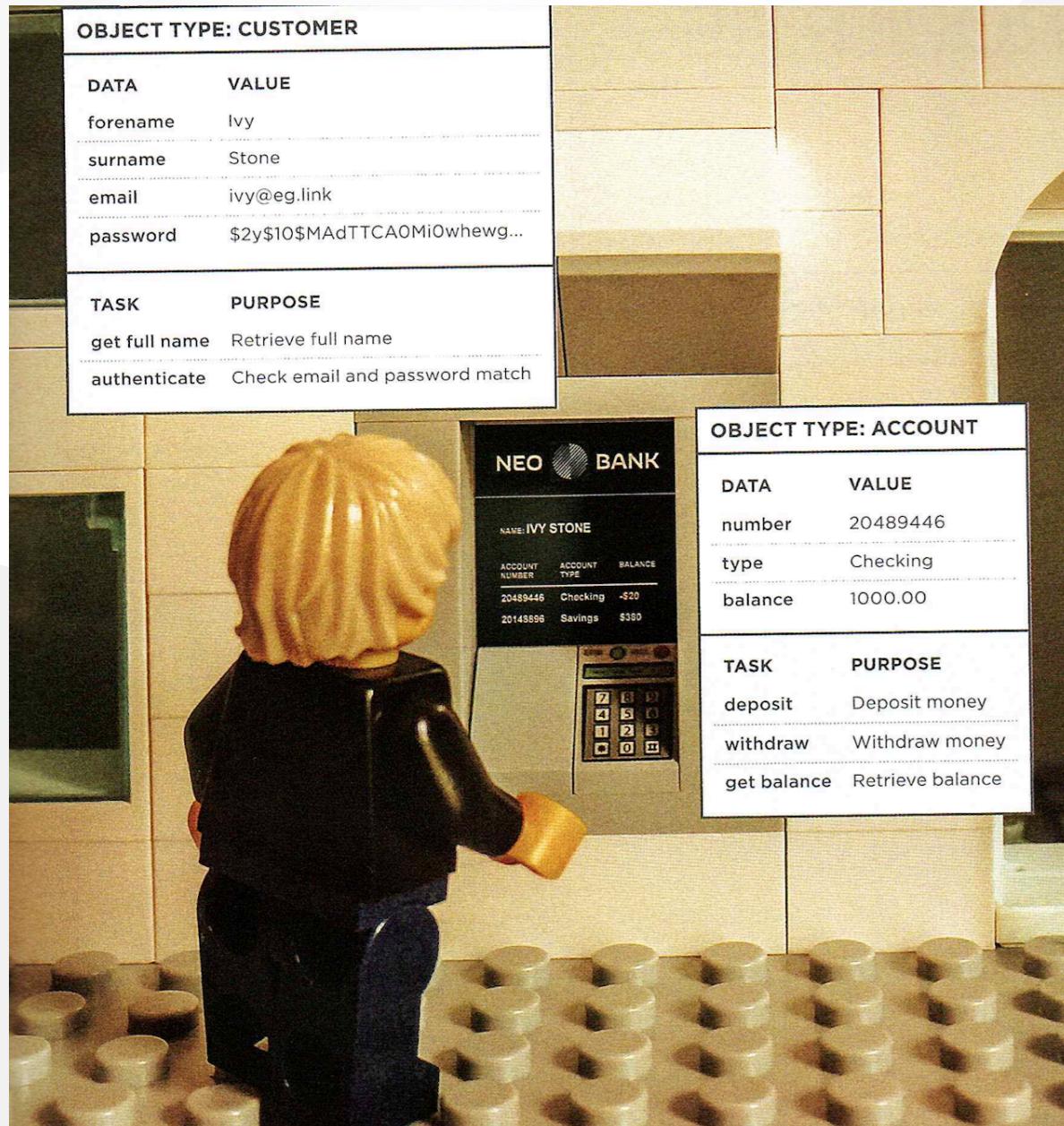
Un **objeto** agrupa todas las:

- **Variables que almacenan datos necesarios para crear un modelo de un concepto,** como un cliente o una cuenta
- **Funciones que representan tareas que puede realizar ese tipo de objeto** (definen el comportamiento del objeto)

# Páginas web como modelos

En la siguiente imagen, si quitamos la foto, todavía podrías decir mucho mirando la información de los recuadros: los tipos de objetos, los datos necesarios para representar cada uno y las tareas que los objetos pueden realizar.

# Páginas web como modelos



# Páginas web como modelos

Los cuadros de información de la figura anterior muestran dos tipos de objetos: **cliente** y **cuenta**. Para cada tipo de objeto, el sitio web debe hacer dos cosas:

- 1. Almacenar los datos que utiliza para representarlo.** Los datos individuales que almacena son los mismos para cada cliente o cuenta, pero los valores que representan a ese cliente o cuenta son diferentes.
- 2. Realizar las mismas tareas con ese tipo de objeto.** Puede realizar las mismas tareas con cada cliente. Puede realizar las mismas tareas con cada cuenta.

Estas tareas pueden acceder o modificar los datos almacenados para cada cliente o cuenta. Por ejemplo, cuando se realiza un ingreso en una cuenta, se actualizará el valor que representa el saldo de la cuenta.

# Propiedades y métodos

En un objeto, las variables se denominan **propiedades** y las funciones, **métodos**.

- Las propiedades almacenan los datos necesarios para crear un modelo de un concepto.
- Los métodos representan las tareas que puede realizar ese tipo de objeto.

# Propiedades y métodos

## Variables: las propiedades de un objeto

En la unidad 1, vimos que las variables pueden almacenar datos que cambian cada vez que se solicita una página. Cuando las variables se utilizan dentro de un objeto, se denominan propiedades del objeto.

Cuando se crea un objeto, un programador debe decidir qué datos necesita saber sobre ese tipo de objeto para que la página haga su trabajo.

# Propiedades y métodos

## Variables: las propiedades de un objeto

Por ejemplo, si se utilizan objetos para representar clientes, cada objeto cliente tendrá:

- Las *mismas* propiedades para contener su nombre, apellidos, dirección de correo electrónico y contraseña; pero
- Valores *diferentes* para representar a cada cliente

Si se utiliza un objeto para representar una cuenta, cada objeto cuenta tendrá también:

- Las *mismas* propiedades para el número de cuenta, el tipo de cuenta y el saldo, pero
- Valores *diferentes* para representar cada cuenta

# Propiedades y métodos

## Variables: las propiedades de un objeto

Las propiedades del objeto son un conjunto de variables que describen las **características individuales** que todos esos objetos tienen en común. Los valores que se almacenan para cada propiedad son los que hacen que un objeto sea diferente de otro.

# Propiedades y métodos

## Funciones: los métodos de un objeto

En la unidad 3, vimos que PHP puede agrupar todas las sentencias necesarias para realizar una tarea en una función. Cuando las funciones se utilizan dentro de un objeto, se llaman métodos del objeto.

Cuando se crea un objeto, un programador decide qué tareas pueden realizar los usuarios de su sitio web con cada tipo de objeto. Estas tareas suelen:

- *Hacer preguntas* que le digan algo sobre ese objeto utilizando datos almacenados en sus propiedades.
- *Cambiar los valores almacenados* en una o más de las propiedades de ese objeto

# Propiedades y métodos

## Funciones: los métodos de un objeto

Las tareas que puedes realizar con una cuenta (hacer un ingreso, retirar dinero o consultar el saldo) son aplicables a todas las cuentas, por lo que todos los objetos que representan una cuenta tienen los **mismos métodos**.

Del mismo modo, necesitarías realizar las mismas tareas para cada cliente (autenticarlos, obtener su nombre completo), por lo que cada objeto que representa a un cliente tendría los **mismos métodos**.

A continuación, puedes ver un diagrama similar al anterior, pero esta vez muestra los nombres de propiedades y métodos para dos objetos cliente y dos objetos cuenta.

# Propiedades y métodos

OBJECT TYPE: CUSTOMER	
DATA	VALUE
forename	Ivy
surname	Stone
email	ivy@eg.link
password	\$2y\$10\$MAdTTCA0Mi0whewg...

OBJECT TYPE: CUSTOMER	
DATA	VALUE
forename	Emiko
surname	Ito
email	emi@eg.link
password	\$2y\$10\$NN5HEAD3atarECjRiir...

TASK	PURPOSE
getFullName()	Return values of <b>forename</b> and <b>surname</b> properties
authenticate()	Check <b>email</b> and <b>password</b> match

TASK	PURPOSE
getFullName()	Return values of <b>forename</b> and <b>surname</b> properties
authenticate()	Check <b>email</b> and <b>password</b> match



# Propiedades y métodos

OBJECT TYPE: ACCOUNT	
DATA	VALUE
number	20489446
type	Checking
balance	1000.00
TASK	PURPOSE
deposit()	Increase value of balance property
withdraw()	Decrease value of balance property
getBalance()	Return value of balance property

OBJECT TYPE: ACCOUNT	
DATA	VALUE
number	10937528
type	Savings
balance	2346.00
TASK	PURPOSE
deposit()	Increase value of balance property
withdraw()	Decrease value of balance property
getBalance()	Return value of balance property

# Tipo de dato objeto (*object*)

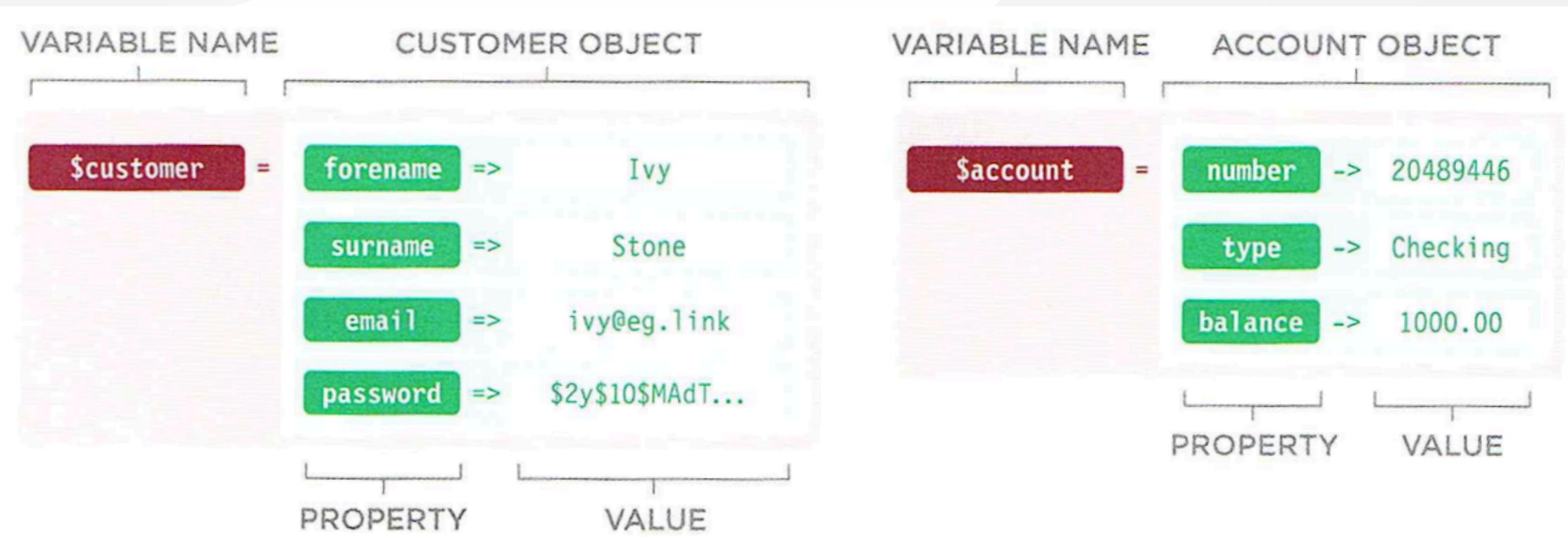
Un objeto es un ejemplo de tipo de datos **compuesto** porque puede almacenar múltiples valores.

Como ya sabemos, PHP tiene diferentes tipos de datos:

- Los tipos de datos **escalares** contienen valores individuales: cadenas, enteros, flotantes, booleanos
- Los tipos de datos **compuestos** contienen múltiples valores: arrays y objetos.

A continuación, puedes ver dos diagramas de objetos que representan un cliente y su cuenta.

# Tipo de dato objeto (*object*)



# Tipo de dato objeto (*object*)

Una variable que contiene un objeto puede nombrarse como cualquier otra variable (en minúsculas, con un guión bajo para separar cada palabra cuando el nombre utiliza varias palabras). Por ejemplo:

- Una variable llamada `$customer` podría almacenar un objeto que representa a un cliente.
- Una variable llamada `$account` podría almacenar un objeto que representa una cuenta.

# Tipo de dato objeto (*object*)

La gente suele decir que un objeto se almacena en una variable pero, como aprenderemos más adelante, la variable en realidad almacena algo llamado **referencia** a donde se creó el objeto en la memoria del intérprete PHP.

Cuando una página ha terminado de ejecutarse, y el intérprete PHP ha enviado una página HTML de vuelta al navegador, olvida el objeto (eliminándolo de su memoria, igual que olvida los valores almacenados en las variables).

# ¿Qué es una clase?

Para crear objetos, se utiliza una plantilla llamada **clase**.

Una definición de clase establece:

- Nombres de **propiedades** que describen los datos que es necesario almacenar para un tipo de objeto.
- **Métodos** que definen las tareas que se pueden realizar con ese tipo de objeto

Cada vez que creas un objeto utilizando una clase:

- Proporcionas los **valores para las propiedades** (esos valores hacen que un objeto sea diferente de otro)
- El objeto **obtiene automáticamente todos los métodos** que se definieron en la clase

Cada objeto individual creado utilizando la clase se denomina **instancia** de esa clase.

# ¿Qué es una clase?

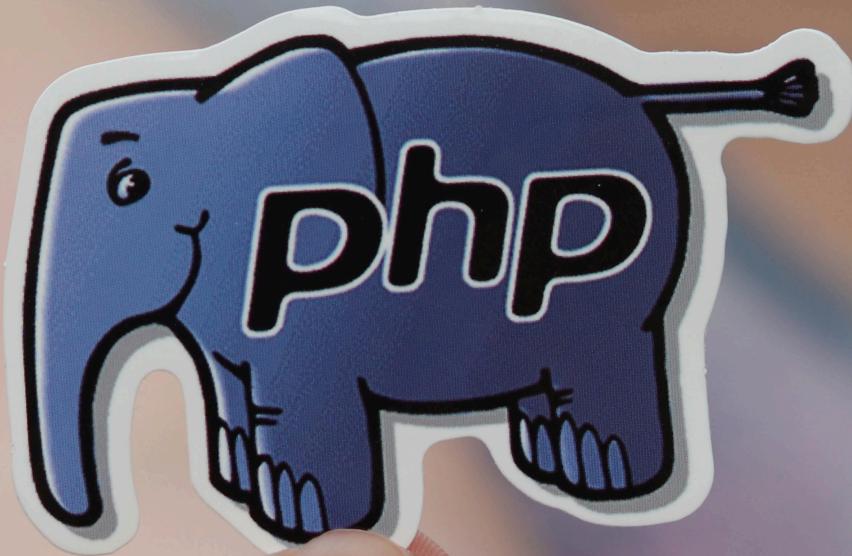
Por ejemplo, si creas un objeto para representar una cuenta bancaria (como se ilustra en el ejemplo anterior), proporcionarías los valores para las siguientes propiedades:

- \$number
- \$type
- \$balance

Y automáticamente obtendrías estos métodos

- deposit()
- withdraw()
- getBalance()

Algunos programadores utilizan los términos clase y objeto indistintamente pero, estrictamente hablando, **una clase es una plantilla que se utiliza para crear un objeto.**



## Cómo crear y utilizar objetos

# Cómo crear y utilizar objetos

A continuación, puedes ver los pasos que necesitas aprender para crear y utilizar clases y objetos.

## Definir una clase como plantilla de un objeto

Una clase es una plantilla que se utiliza para crear un tipo de objeto.

La clase define:

- Propiedades que almacenan los datos utilizados para representar ese tipo de objeto
- Métodos que contienen las sentencias para lograr las tareas que ese tipo de objeto puede realizar

Se crea una nueva clase para cada tipo de objeto que el sitio web necesite tratar.

# Cómo crear y utilizar objetos

## Crear un objeto y almacenarlo en una variable

Un objeto se crea:

- Especificando el nombre de la clase que debe utilizarse como plantilla para él
- Proporcionando valores para sus propiedades

El objeto obtendrá automáticamente los métodos definidos en la clase.

Cuando se crea un objeto, normalmente se almacena en una variable para que pueda ser utilizado por el resto del código de la página.

# Cómo crear y utilizar objetos

## Establecer y acceder a los valores de las propiedades

Una vez creado un objeto, es posible:

- Establecer valores para sus propiedades
- Acceder a los valores almacenados en sus propiedades y utilizarlos en el resto del código de la página.

Cada instancia de un objeto almacenará valores diferentes en sus propiedades porque representa una instancia diferente de ese tipo de objeto (como un cliente diferente o una cuenta diferente).

# Cómo crear y utilizar objetos

## Definir y llamar métodos de un objeto

Un método define las sentencias necesarias para lograr una tarea que un tipo de objeto es capaz de realizar. Se escriben igual que las definiciones de funciones, pero viven dentro de la clase. También suelen devolver un valor, como hacen las funciones.

Cuando un método de un objeto es llamado, a menudo necesitará acceder o actualizar valores almacenados en las propiedades de ese objeto.

PHP tiene una variable especial llamada `$this` que permite a los métodos trabajar con valores almacenados en las propiedades de este objeto.

# Cómo crear y utilizar objetos

## Asignar valores de propiedad al crear objetos

En lugar de crear un objeto y luego establecer los valores para cada una de sus propiedades individualmente, puedes crear un objeto y asignar valores a las propiedades en una sola línea de código. Para hacer esto, se añade a la clase algo llamado función **constructora o constructor**.

En PHP 8, un constructor también puede definir las propiedades de un objeto (por lo que no es necesario definirlas antes de establecer sus valores en una función constructora).

# Cómo crear y utilizar objetos

## Controlar qué código puede acceder a las propiedades

A veces, no querrás permitir que las páginas PHP accedan o actualicen las propiedades de un objeto directamente. En su lugar, puedes crear métodos para obtener o actualizar los valores almacenados en esas propiedades.

Por ejemplo, puedes ocultar la propiedad *balance* de un objeto cuenta, y luego utilizar los métodos `getBalance()`, `deposit()`, y `withdraw()` para trabajar con el valor que está almacenado en la propiedad *balance*.

# Clases: plantillas para objetos

Cuando se crea una **definición de clase**, las propiedades y métodos que tendrá un objeto se almacenan dentro de llaves.

Una definición de clase utiliza:

- La palabra clave `class` .
- Un **nombre** que describe el tipo de objeto que crea. El nombre debe utilizar *UpperCamelCase* donde la primera letra de cada palabra comienza con una letra mayúscula (no se utilizan guiones bajos).
- Un par de llaves para crear un bloque de código. Las llaves indican dónde empieza/termina la clase. Cada llave comienza en una nueva línea.

NOTA: No hay punto y coma después de la llave de cierre que termina la definición de la clase.

# Clases: plantillas para objetos

```
class Account
{
    PROPERTIES [
        public int      $number;
        public string   $type;
        public float    $balance;
    ]

    METHODS [
        public function deposit(float $amount): float
        {
            // Code to deposit money here
        }
        public function withdraw(float $amount): float
        {
            // Code to withdraw money here
        }
    ]
}
```

# Clases: plantillas para objetos

Dentro de las llaves de la clase, se enumeran las propiedades de este tipo de objeto utilizando:

- Una palabra clave de **visibilidad** (lo veremos más adelante). El ejemplo anterior utiliza la palabra clave *public*.
- El **tipo de datos** que contendrá la propiedad.  
(Esto se añadió en PHP 7.4 y es opcional).
- El **nombre de la propiedad** comenzando con el símbolo `$`.

Los **métodos utilizan la misma sintaxis que las definiciones de función**, pero están precedidos por una palabra clave de *visibilidad*; los métodos del ejemplo utilizan *public*.

# Crear un objeto utilizando una clase

Para crear el objeto, se utiliza:

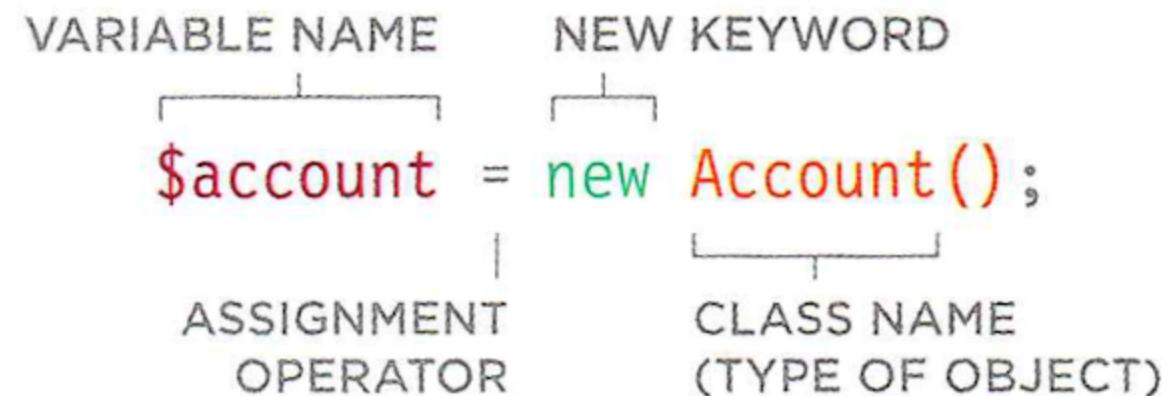
- La palabra clave `new`.
- El **nombre de la clase** que servirá de plantilla para el objeto.
- **Paréntesis.** Estos pueden contener nombres de parámetros (como una función tiene parámetros entre paréntesis). Pasan datos al objeto cuando se crea.

# Crear un objeto utilizando una clase

Como hemos visto previamente, una referencia a un objeto es a menudo **almacenada en una variable** para que pueda ser utilizada por el resto del código en una página PHP. Para hacer esto

- Crea una variable que contenga el objeto; su nombre debe describir el tipo de objeto que contiene.
- Añade el operador de asignación =
- Crea el objeto (como se ha descrito antes).

# Crear un objeto utilizando una clase



The diagram illustrates the components of the Java code `$account = new Account();`. It uses brackets and labels to identify each part:

- VARIABLE NAME**: Points to the identifier `$account`.
- NEW KEYWORD**: Points to the keyword `new`.
- ASSIGNMENT OPERATOR**: Points to the assignment operator `=`.
- CLASS NAME (TYPE OF OBJECT)**: Points to the class name `Account`.

# Crear un objeto utilizando una clase

En el ejemplo anterior, la variable `$account` almacenaría una referencia a un objeto creado utilizando la clase `Account` que se ha mostrado previamente.

Tendría tres propiedades: `$number`, `$type` y `$balance`. Ninguna de ellas tendría aún valores. Veremos cómo asignar sus valores a continuación.

El objeto también tendría automáticamente los dos métodos que estaban en la definición de la clase.

# Crear un objeto utilizando una clase

Para crear un segundo objeto que represente otra cuenta, haríamos lo mismo que se ha mostrado, pero con un nombre de variable diferente (de lo contrario, el segundo objeto sobrescribiría al primero).

**La definición de clase debe estar en cualquier página que cree un objeto utilizando la clase.**

Si una definición de clase es utilizada por más de una página, se coloca en un **archivo separado que puede ser incluido en ambas páginas**. Ese archivo recibe el mismo nombre que la clase (por ejemplo, `Account.php` ).

## Acceder y actualizar propiedades

Se accede y se actualizan las propiedades de un objeto como se hace con las variables. Si un objeto se almacena en una variable, se especifica primero el nombre de la variable y, a continuación, se utiliza el operador de objeto ( -> ) para especificar la propiedad con la que se desea trabajar.

# Acceder y actualizar propiedades

## Acceder a propiedades

Para acceder a un valor almacenado en una propiedad, se utiliza:

- **Nombre de la variable** que contiene el objeto
- **Operador de objeto** `->` sin espacio a ambos lados
- **Nombre de la propiedad** (ten en cuenta que aquí el nombre de la propiedad no empieza por el símbolo `$`)

El operador objeto indica que la propiedad a la derecha del operador pertenece al objeto almacenado en la variable a la izquierda del operador.

El siguiente código muestra cómo mostrar el valor del saldo de la cuenta en la página.

# Acceder y actualizar propiedades

## Acceder a propiedades



# Acceder y actualizar propiedades

## Establecer y actualizar propiedades

Para actualizar el valor almacenado en una propiedad, se utiliza:

- Nombre de la **variable que contiene el objeto**
- **Operador de objeto** `->` sin espacio a ambos lados
- Nombre de la **propiedad** que desea actualizar
- Operador de asignación `=`
- **Nuevo valor** (si el valor es una cadena, va entre comillas; los números y booleanos no van entre comillas)

Si se intenta establecer el valor de una propiedad que no estaba en la definición de la clase, la propiedad se añadirá a este único objeto. (No se añadiría a ningún otro objeto creado utilizando la clase).

# Acceder y actualizar propiedades

## Establecer y actualizar propiedades

OBJECT	PROPERTY	VALUE
\$account->number	=	20148896;
\$account->type	=	'Checking';
\$account->balance	=	1000.00;

|                                   |  
OBJECT                              ASSIGNMENT  
OPERATOR                          OPERATOR

# Acceder y actualizar propiedades

Si el tipo de dato de una propiedad fue establecido en la clase y se intenta acceder a la propiedad antes de que se le haya asignado un valor, el intérprete PHP creará un error que puede detener la ejecución de la página.

Como veremos más adelante, es posible especificar valores por defecto para las propiedades en el método `__construct()`. Esto asegurará que cada propiedad tenga un valor cuando el objeto sea creado utilizando la clase.

# Ejemplo: Usando objetos y propiedades

El siguiente ejemplo establece y muestra información básica de un cliente y su cuenta bancaria, y estructura la página web con encabezado y pie de página mediante la inclusión de archivos PHP externos:

1. Se define la clase `Customer` y sus propiedades.
2. Se define la clase `Account` y sus propiedades.
3. Se crea una instancia de la clase `Customer` y se almacena ese objeto en una variable llamada `$customer`.
4. Se crea una instancia de la clase `Account` y ese objeto se almacena en una variable llamada `$account`.

## Ejemplo: Usando objetos y propiedades

5. Se da un valor a la propiedad `email` del objeto `Customer`.
6. Se da un valor a la propiedad `balance` del objeto `Account`.
7. Un fichero *include* añade una cabecera a la página.
8. Se muestran las dos propiedades que se acaban de establecer.
9. Un archivo *include* añade las etiquetas HTML necesarias para cerrar la página.

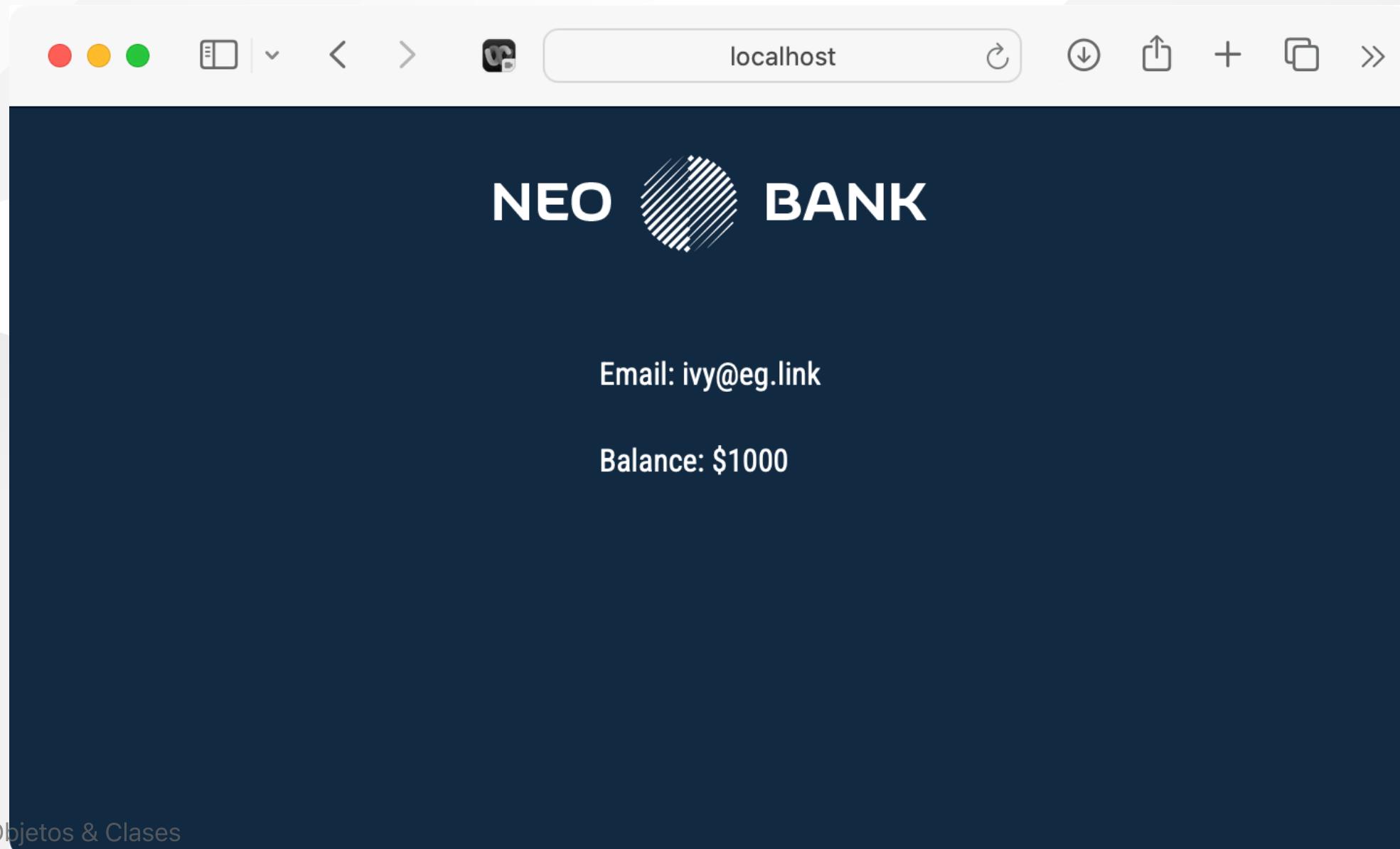
# Ejemplo: Usando objetos y propiedades

```
<?php
class Customer
{
    public string $forename;
    public string $surname;
    public string $email;
    public string $password;
}

class Account
{
    public int    $number;
    public string $type;
    public float  $balance;
}

$customer = new Customer();
$account  = new Account();
$customer->email  = 'ivy@eg.link';
$account->balance = 1000.00;
?>
<?php include 'includes/header.php'; ?>
<p>Email: <?= $customer->email ?></p>
<p>Balance: $<?= $account->balance ?></p>
<?php include 'includes/footer.php'; ?>
```

# Ejemplo: Usando objetos y propiedades



# Ejemplo: Usando objetos y propiedades

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

- Despues del Paso 5, establece el nombre y los apellidos del cliente en el objeto `Customer`.
- A continuación, en el Paso 8, muestra su nombre antes de la dirección de correo electrónico.

# Definiendo y llamando métodos

Un método es una función que se escribe dentro de la definición de una clase.

Para llamar a un método, utiliza el nombre de la variable que contiene el objeto, el operador objeto ( `->` ) y, a continuación, el nombre del método.

# Definiendo y llamando métodos

## Definir un método

Para añadir un método a una clase, utiliza una palabra clave de visibilidad (p.e: *public*), seguida de una definición de función. Si un método necesita acceder o actualizar una propiedad del objeto, utiliza:

- Una variable especial llamada `$this` (conocida como **pseudo-variable**) para indicar que se quiere acceder a una propiedad de este objeto.
- El operador de objeto `->`
- El nombre de la propiedad a la que desea acceder

El siguiente método `deposit()` tiene un parámetro llamado `$amount`. Cuando se llama al método, el valor utilizado para `$amount` se añade al valor de la propiedad `balance`, y se devuelve el nuevo valor en `balance`.

## Definir un método

```
class Account
{
    public int      $number;
    public string   $type;
    public float    $balance;

    public function deposit($amount)
    {
        $this->balance += $amount;
        return $this->balance;
    }
}
```

\_\_\_\_\_

\$this PSEUDO-VARIABLE

# Definiendo y llamando métodos

## Llamar a un método

Para llamar a un método, se utiliza:

- Nombre de la **variable que contiene el objeto**
- Operador objeto `->`
- **Nombre del método**
- **Argumentos para los parámetros del método**

El ejemplo siguiente deposita 50\$ en la cuenta. El método `deposit()` (mostrado en el ejemplo anterior) añade esta cantidad al valor de la propiedad `balance` y devuelve el nuevo `balance`.

El comando `echo` se utiliza para escribir el nuevo saldo en la página.

# Definiendo y llamando métodos

## Llamar a un método

The diagram illustrates the structure of a PHP method call. It shows the code: `echo $account->deposit(50.00);`. Brackets above the code identify the **OBJECT** (\$account) and the **METHOD NAME** (deposit). Brackets below the code identify the **OBJECT OPERATOR** (->) and the **ARGUMENT** (50.00).

```
echo $account->deposit(50.00);
```

OBJECT      METHOD NAME  
|  
|  
OBJECT OPERATOR      ARGUMENT

# Ejemplo: usando métodos de objetos

En este nuevo ejemplo se define una clase `Account` con propiedades y métodos para manejar depósitos y retiros, crea una instancia de `Account` con un saldo inicial, realiza un depósito y muestra el nuevo saldo en una página web estructurada con encabezado y pie de página mediante la inclusión de archivos PHP externos:

1. Definir la clase `Account` y sus propiedades.
2. Añade el método `deposit()`. El parámetro `$amount` es la cantidad a añadir al saldo.
3. La cantidad pasada a la función se añade al valor almacenado en la propiedad `balance`:
  - `$this->balance` obtiene la propiedad `balance` de este objeto.
  - `+=` añade el valor en `$amount` al saldo.
4. Se devuelve el nuevo valor almacenado en la propiedad `balance`.

# Ejemplo: usando métodos de objetos

5. `withdraw()` hace lo mismo que `deposit()`, pero resta una cantidad del saldo.
6. Se crea un objeto utilizando la clase `Account` y se almacena en una variable llamada `$account`.
7. La propiedad `saldo` del objeto se establece en 100,00\$.
8. Se llama al método `deposit()` añadiendo 50,00\$ a la cuenta. Devuelve el saldo actualizado, y ese valor se escribe en la página con la abreviatura de `echo`.

# Ejemplo: usando métodos de objetos

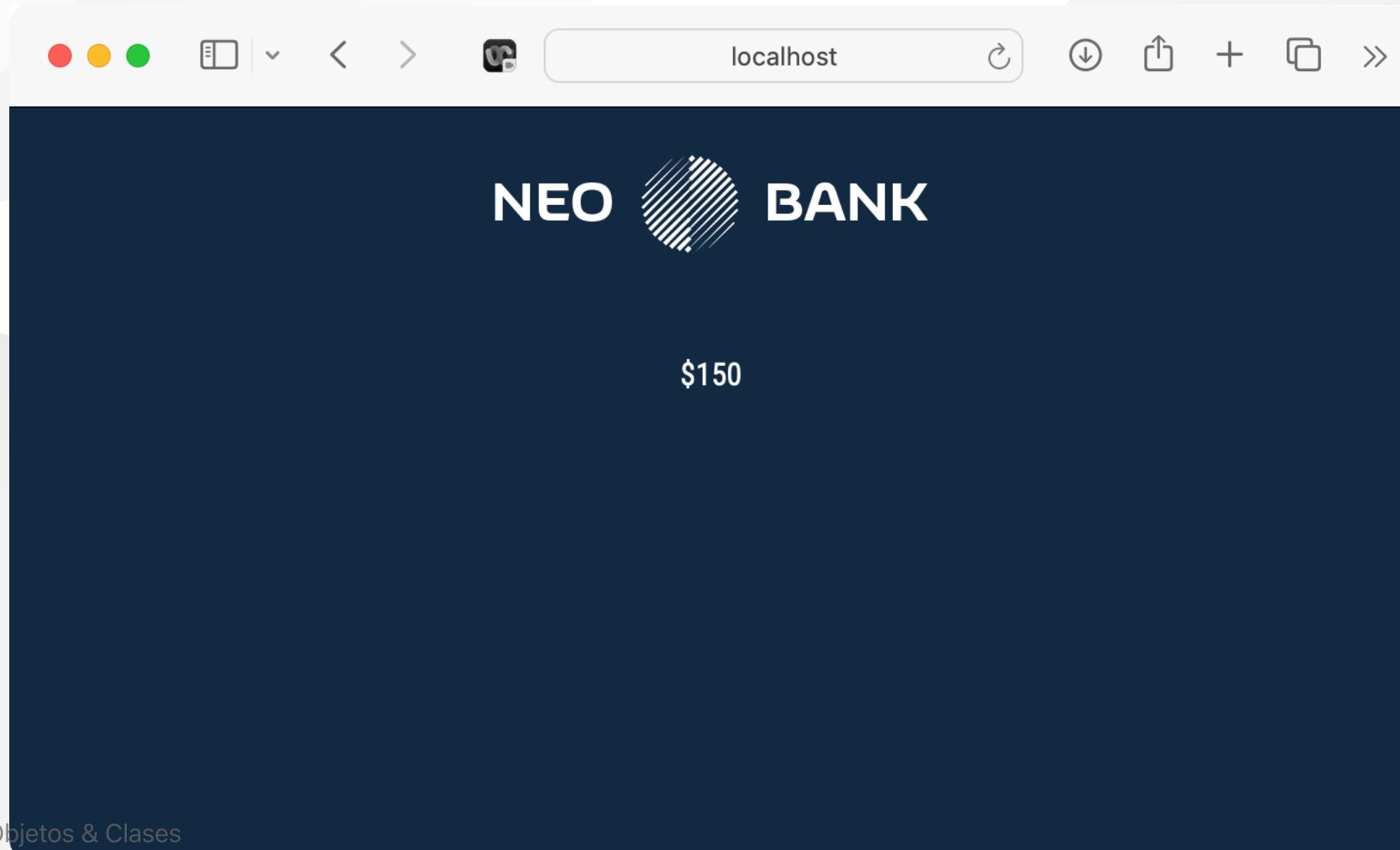
```
<?php
class Account
{
    public int    $number;
    public string $type;
    public float   $balance;

    public function deposit(float $amount): float
    {
        $this->balance += $amount;
        return $this->balance;
    }

    public function withdraw(float $amount): float
    {
        $this->balance -= $amount;
        return $this->balance;
    }
}

$account = new Account();
$account->balance = 100.00;
?>
<?php include 'includes/header.php'; ?>
<p>$<?= $account->deposit(50.00) ?></p>
<?php include 'includes/footer.php'; ?>
```

# Ejemplo: usando métodos de objetos



# Método constructor

El método `__construct()` se conoce como **constructor**. Se ejecuta automáticamente cuando se utiliza una clase para crear un objeto.

Es decir, si añades un método llamado `__construct()` a la definición de una clase (su nombre debe empezar con **dos guiones bajos**), las sentencias dentro de ese método se ejecutan automáticamente cuando la clase se utiliza para crear un objeto.

# Método constructor

Un método `__construct()` puede ser utilizado para permitirte crear un objeto y añadir valores a sus propiedades en una línea de código, en lugar de crear el objeto y luego establecer cada propiedad individualmente utilizando una sentencia separada.

# Método constructor

A continuación, se crea un objeto utilizando la clase `Account` y se almacena en una variable llamada `$account`. Cuando se crea el objeto, el intérprete de PHP busca en la clase un método llamado `__construct()`.

Los argumentos en los paréntesis después del nombre de la clase son pasados al método constructor. Dentro del método `__construct()`, esos valores se utilizan para establecer las propiedades del objeto.

# Método constructor

VARIABLE	CLASS NAME	VALUES USED TO CREATE OBJECT						
<code>\$account</code>	<code>Account</code>	<code>(20148896, 'Checking', 1000.00)</code>						
		<table><thead><tr><th>ACCOUNT NUMBER</th><th>ACCOUNT TYPE</th><th>BALANCE</th></tr></thead><tbody><tr><td>20148896</td><td>'Checking'</td><td>1000.00</td></tr></tbody></table>	ACCOUNT NUMBER	ACCOUNT TYPE	BALANCE	20148896	'Checking'	1000.00
ACCOUNT NUMBER	ACCOUNT TYPE	BALANCE						
20148896	'Checking'	1000.00						

# Método constructor

**Importante:** No se debe el nombre de sus propias funciones con dos guiones bajos; esta convención de nomenclatura sólo debe utilizarse para lo que PHP llama **métodos mágicos**.

Los métodos mágicos son llamados automáticamente por el intérprete de PHP; no necesita llamar a estos métodos en su propio código.

# Método constructor

A continuación, el método `__construct()` de la clase `Account` tiene tres parámetros: `$type`, `$number` y `$balance`, que corresponden a sus propiedades.

Las tres sentencias del método `__construct()` toman los valores de los parámetros y los utilizan para establecer las propiedades del objeto.

Como ya hemos comentado anteriormente en la unidad, la pseudo-variable `$this` te permite acceder o actualizar las propiedades de este objeto.

# Método constructor

Si se creara un objeto utilizando el código de la página de la izquierda, el método `__construct()` que se muestra a continuación se ejecutaría automáticamente y daría al parámetro:

- `$number` el valor 20148896.
- `$type` el valor 'Checking'.
- `$balance` el valor 100.00.

En el ejemplo siguiente, veremos cómo especificar valores por defecto para cualquiera de estas propiedades.

# Método constructor

```
class Account
{
    public int    $number;
    public string $type;
    public float  $balance;

    public function __construct($number, $type, $balance)
    {
        $this->number = $number;
        $this->type   = $type;
        $this->balance = $balance;
    }

    function deposit($amount) {...}
    function withdraw($amount) {...}
    function getBalance() {...}
}
```

The diagram illustrates the mapping between the constructor parameters and the class properties. Green arrows point from the parameters \$number, \$type, and \$balance to their respective assignments in the constructor. Red arrows point from the class properties \$this->number, \$this->type, and \$this->balance back to the same assignments.

# Método constructor

PHP 8 añade una forma más sencilla de escribir definiciones de clases al permitir declarar las propiedades de una clase dentro de los paréntesis del método `__construct()`.

Cuando se crea un objeto utilizando la clase, los argumentos proporcionados al constructor se asignan automáticamente como valores para estas propiedades. Esto se denomina **promoción de propiedades del constructor**.

Si una propiedad es opcional, se le puede dar un valor por defecto (ver la propiedad `$balance` del ejemplo a continuación) y este valor será utilizado si no se especifica un argumento.

# Método constructor

```
class Account
{
    public function __construct(
        public int    $number,
        public string $type,
        public float  $balance = 0.00,
    ) {}

    function deposit($amount) {...}
    function withdraw($amount) {...}
    function getBalance() {...}
}
```

## Ejemplo: usando constructor en una clase

El siguiente ejemplo define una clase `Account` con un constructor y métodos para manejar depósitos y retiros, crea dos instancias de `Account` con saldos iniciales, y muestra una tabla en una página web que detalla las transacciones y saldos de estas cuentas en fechas específicas, estructurada con encabezado y pie de página mediante la inclusión de archivos PHP externos.

# Ejemplo: usando constructor en una clase

1. La página PHP comienza habilitando tipos estrictos porque se han añadido declaraciones de tipo a los métodos.
2. Se define el nombre de la clase y sus propiedades.
3. Se añade el método `__construct()` de la página anterior para establecer los valores de las propiedades. Se añaden declaraciones de tipo de argumento a los parámetros. Si no se indica un saldo al crear el objeto, se utiliza un valor por defecto de 0,00.

# Ejemplo: usando constructor en una clase

4. Los métodos `deposit()` y `withdraw()` actualizan el valor almacenado en la propiedad saldo. El argumento y el tipo de retorno son float. (Cuando el tipo de datos es float se puede utilizar un int sin provocar un error).
5. Se crean dos objetos para representar una cuenta corriente y una cuenta de ahorro. El constructor asigna los valores del paréntesis a las propiedades de cada uno de los objetos.
6. Se dibuja una tabla HTML en la página. La primera fila muestra los encabezados utilizando la propiedad `type` de los dos objetos. Para acceder a una propiedad, utiliza el:
  - Nombre de la variable que contiene el objeto
  - Operador del objeto
  - Nombre de la propiedad

## Ejemplo: usando constructor en una clase

7. La siguiente fila de la tabla muestra la propiedad *balance* de los objetos.
8. En la tercera fila de la tabla, el saldo de cada cuenta se actualiza llamando a los métodos `deposit()` o `withdraw()`. Estos métodos devuelven el nuevo valor de la propiedad saldo, y éste se escribe en la página. Para llamar a un método, utiliza:
  - Nombre de la variable que contiene el objeto
  - Operador del objeto
  - Nombre del método con sus argumentos entre paréntesis
9. En la cuarta fila de la tabla, se repite el paso anterior utilizando valores diferentes.

# Ejemplo: usando constructor en una clase

```
<?php
declare(strict_types = 1);

class Account
{
    public int    $number;
    public string $type;
    public float   $balance;

    public function __construct(int $number, string $type, float $balance = 0.00)
    {
        $this->number    = $number;
        $this->type      = $type;
        $this->balance = $balance;
    }

    public function deposit(float $amount): float
    {
        $this->balance += $amount;
        return $this->balance;
    }

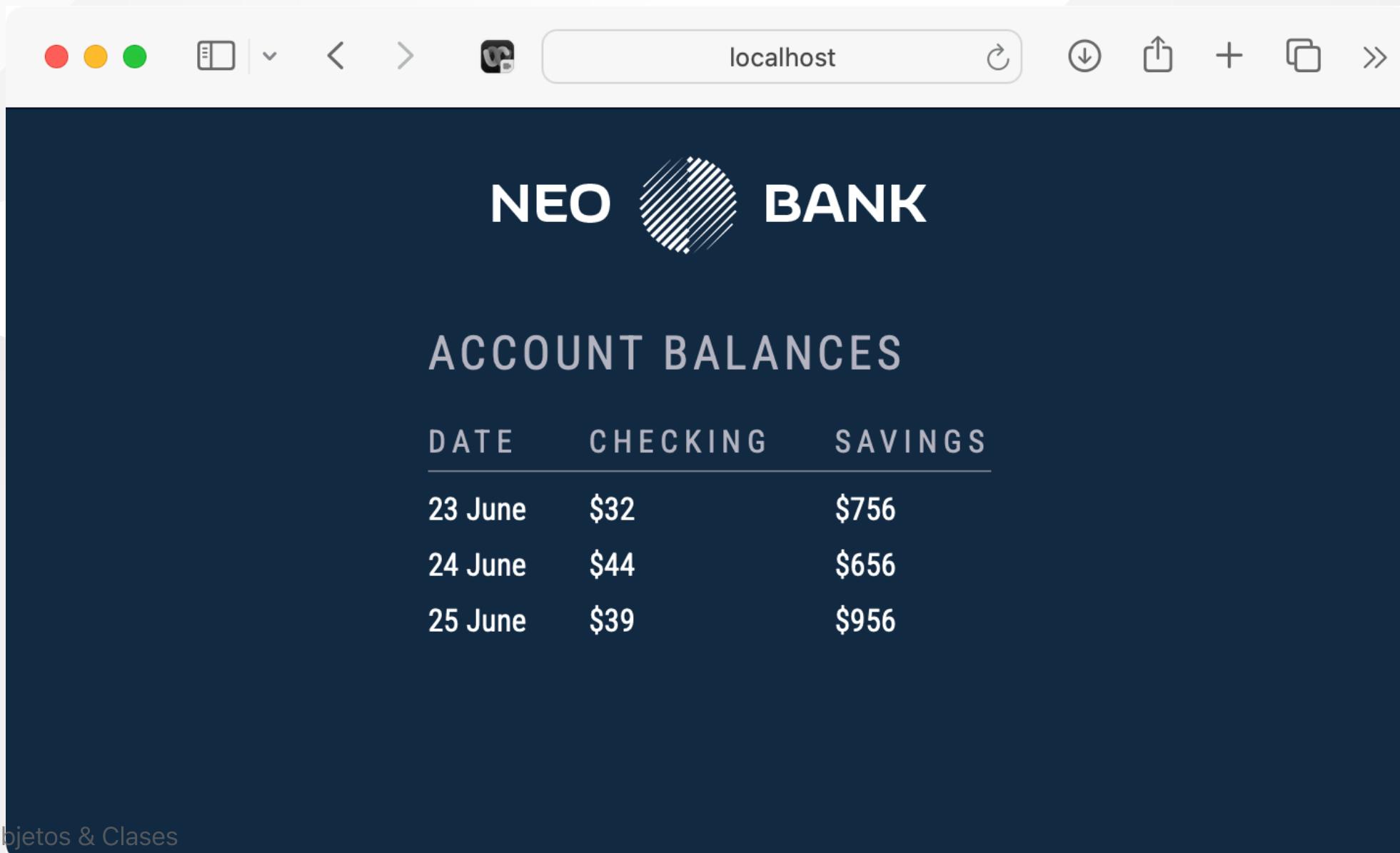
    public function withdraw(float $amount): float
    {
        $this->balance -= $amount;
        return $this->balance;
    }
}
```

# Ejemplo: usando constructor en una clase

```
$checking = new Account(43161176, 'Checking', 32.00);
$savings = new Account(20148896, 'Savings', 756.00);
?>

<?php include 'includes/header.php'; ?>
<h2>Account Balances</h2>
<table>
  <tr>
    <th>Date</th>
    <th><?= $checking->type ?></th>
    <th><?= $savings->type ?></th>
  </tr>
  <tr>
    <td>23 June</td>
    <td>$<?= $checking->balance ?></td>
    <td>$<?= $savings->balance ?></td>
  </tr>
  <tr>
    <td>24 June</td>
    <td>$<?= $checking->deposit(12.00) ?></td>
    <td>$<?= $savings->withdraw(100.00) ?></td>
  </tr>
  <tr>
    <td>25 June</td>
    <td>$<?= $checking->withdraw(5.00) ?></td>
    <td>$<?= $savings->deposit(300.00) ?></td>
  </tr>
</table>
<?php include 'includes/footer.php'; ?>
```

# Ejemplo: usando constructor en una clase



A screenshot of a web browser window titled "localhost". The page displays the "NEO BANK" logo at the top. Below it, the heading "ACCOUNT BALANCES" is centered. A table follows, showing account balances for three dates: June 23, June 24, and June 25. The table has columns for Date, Checking, and Savings.

DATE	CHECKING	SAVINGS
23 June	\$32	\$756
24 June	\$44	\$656
25 June	\$39	\$956

# Ejemplo: funciones que devuelven un valor

Crea una clase `Product` que represente un producto en una tienda en línea. Esta clase debe incluir propiedades básicas como `id`, `name`, `price` y `stock`. Se debe implementar un constructor que permita inicializar estas propiedades, con valores por defecto para `price` y `stock`.

Pasos a seguir:

1. Crear la clase `Product` (en un fichero aparte denominado `Product.php`).
2. Crear instancias de la clase `Product` en el fichero de página principal.
3. Actualiza las propiedades de una de las instancias creadas y muestra los cambios
4. Mostrar resultados en una Tabla HTML.

# Visibilidad de propiedades y métodos

Se puede impedir que el código externo a un objeto obtenga o establezca los valores almacenados en las propiedades de su interior. También podemos evitar que el código externo a un objeto llame a sus métodos.

# Visibilidad de propiedades y métodos

Las propiedades y métodos de una clase se denominan **miembros** de una clase. Puede especificar si el código externo a un objeto creado utilizando esta clase puede:

- Acceder o actualizar el valor almacenado en una propiedad
- Llamar a un método

Esto se hace estableciendo la **visibilidad** cuando se declara una propiedad o se define un método.

# Visibilidad de propiedades y métodos

Hasta ahora en esta unidad, todos los nombres de propiedades y métodos han sido precedidos por la palabra `public`, lo que significa que cualquier otro código puede trabajar con las propiedades y métodos del objeto.

Hay ocasiones en las que sólo quieras permitir que el código dentro del objeto acceda o actualice propiedades o llame a los métodos de ese objeto. Para ello, se cambia la palabra `public` por `protected`.

# Visibilidad de propiedades y métodos

Por ejemplo, la clase `Account` tiene una propiedad llamada `balance`. Si se declara utilizando la palabra clave de visibilidad `public`, cualquier código que cree un objeto utilizando esta clase puede obtener o actualizar el valor de esa propiedad.

Para evitar que cualquier otro código actualice el valor almacenado en la propiedad `balance`, se puede establecer su visibilidad a `protected`. Si se intenta acceder a una propiedad protegida utilizando código que está fuera de la clase, el intérprete de PHP generará un error.

Para ser exactos, `protected` permite el acceso a los miembros dentro de la clase donde fueron definidos y en cualquier clase que *herede* de ella, pero no desde fuera de estas clases.

# Visibilidad de propiedades y métodos

Si el código fuera del objeto necesita obtener el valor almacenado en una propiedad protegida, se añade un método a la clase que devuelve su valor. Este método se conoce como **getter** (porque obtiene un valor).

En el ejemplo que estudiaremos a continuación, se añade un nuevo método llamado `getBalance()` a la clase `Account`; su función es devolver el valor almacenado en la propiedad `$balance`.

# Visibilidad de propiedades y métodos

Para actualizar el valor almacenado en una propiedad protegida, se añade un método a la clase que actualiza su valor. Esto se conoce como **setter** (porque establece un valor).

Los métodos `deposit()` y `withdraw()` de la clase `Account` ya se utilizan para actualizar el valor almacenado en la propiedad `$balance`.

Estos cambios aseguran que el saldo sólo sea actualizado por los métodos `deposit()` o `withdraw()`. No podrá ser actualizado por ningún otro código.

# Visibilidad de propiedades y métodos

Si no especifica la visibilidad de una propiedad o método en la definición de la clase, por defecto será pública (*public*), pero indicar explícitamente si la propiedad o método es público o protegido se considera una buena práctica y hace que el código sea más fácil de entender.

También existe otra configuración para la visibilidad llamada `private`, utilizada en código orientado a objetos más avanzado. Estas opciones también se denominan **modificadores de acceso**.

## Ejemplo: usando getters y setters

El siguiente ejemplo define una clase `Account` para manejar cuentas bancarias, así como un script que instancia un objeto de esta clase y muestra información sobre la cuenta en una página web.

# Ejemplo: usando getters y setters

1. La propiedad `balance` antes era pública y ahora se cambia a protegida para que no sea visible fuera de la clase.
2. Los métodos `deposit()` y `withdraw()` existentes actúan como setters para actualizar el saldo (el código para ellos es el mismo que en el ejemplo anterior).
3. Se añade a la clase un getter llamado `getBalance()` para obtener el valor de la propiedad `balance` protegida si es necesario mostrarlo.

## Ejemplo: usando getters y setters

4. Se crea un objeto utilizando la clase `Account`. Se almacena en la variable `$account`.
5. Se muestra el tipo de cuenta. Como esta propiedad es pública, se puede acceder a ella directamente.
6. Se llama al método `getBalance()` para mostrar el valor de la propiedad `$balance`.
7. Se llama al método `deposit()`. Añade 35 \$ a la propiedad `$balance`. Este método también devuelve el nuevo saldo para que pueda ser escrito en la página.

# Ejemplo: usando getters y setters

```
<?php
declare(strict_types = 1);

class Account {
    public int $number;
    public string $type;
①   protected float $balance;

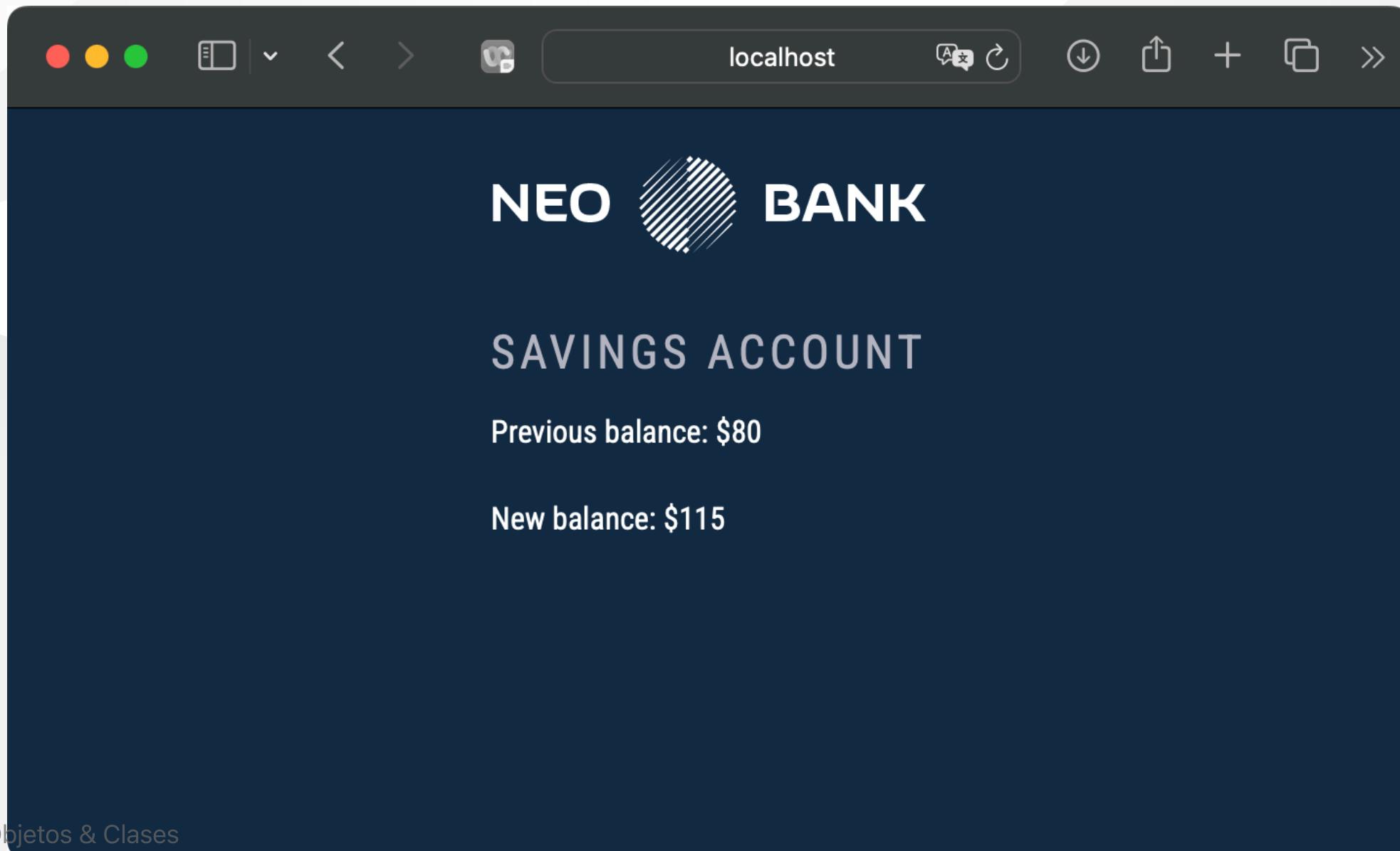
    public function __construct() {...}
    public function deposit() {...}
    public function withdraw() {...}

    public function getBalance(): float
    {
        return $this->balance;
    }
}

④ $account = new Account(20148896, 'Savings', 80.00);
?>

<?php include 'includes/header.php'; ?>
⑤ <h2><?= $account->type ?> Account</h2>
⑥ <p>Previous balance: <?= $account->getBalance() ?></p>
⑦ <p>New balance: <?= $account->deposit(35.00) ?></p>
<?php include 'includes/footer.php'; ?>
```

# Ejemplo: usando getters y setters



# Ejemplo: usando getters y setters

Sobre el ejemplo anterior, realiza las siguientes modificaciones:

Modifica la clase `Account` para agregar nuevas propiedades y métodos, aplicando diferentes niveles de visibilidad.

- Crear nueva propiedad privada y método:
  - Añadir una propiedad privada `owner` de tipo `string`, que representará el nombre del propietario de la cuenta.
  - Crear un método público `getOwner()` para obtener el nombre del propietario.
  - Crear un método público `setOwner()` para actualizar el nombre del propietario.
- Modificar el código para usar la nueva propiedad:
  - Crear una nueva instancia de `Account` y mostrar el nombre del propietario en el HTML.

# Almacenar un array en una propiedad de un objeto

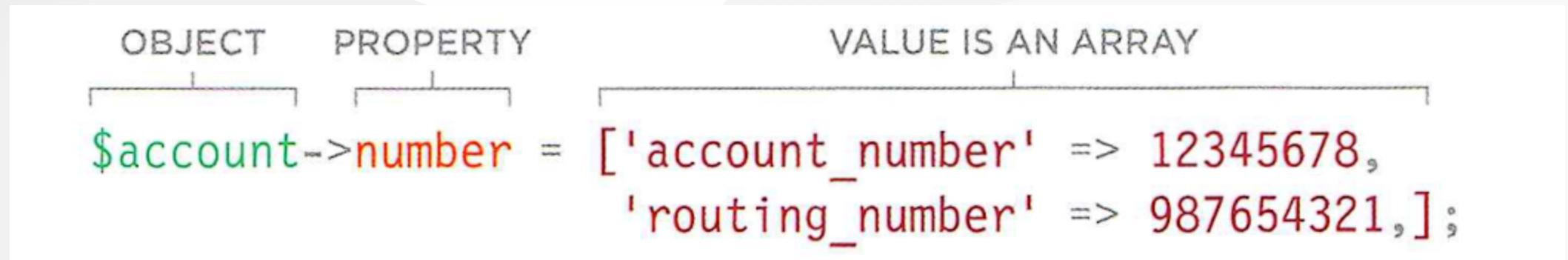
Una propiedad de un objeto puede almacenar un array. Se puede acceder a los elementos individuales del array utilizando la sintaxis de arrays.

Las propiedades de los objetos que has visto hasta ahora almacenan tipos de datos escalares (cadenas, números y booleanos). La propiedad de un objeto también puede almacenar un tipo de dato compuesto como un array. A continuación, un objeto `Account` se almacena en una variable llamada `$account`, y se establece su propiedad `number`.

# Almacenar un array en una propiedad de un objeto

El valor asignado a la propiedad número es un array asociativo que contiene dos valores distintos:

- Un número de cuenta
- Un número de ruta (conocido como código de clasificación o BSB en algunos países)



# Almacenar un array en una propiedad de un objeto

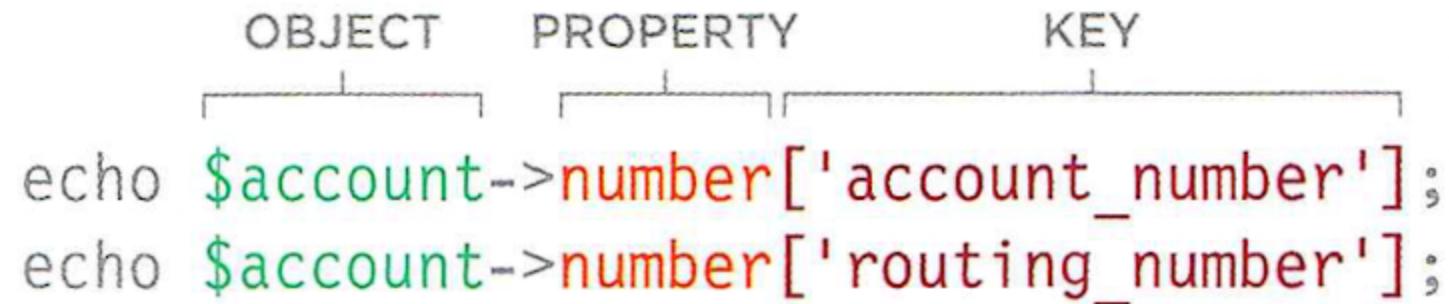
Para acceder a un valor almacenado como array en la propiedad número de un objeto, se utiliza el:

- Nombre de la variable que contiene el objeto
- Operador objeto
- Nombre de la propiedad que contiene el array
- Clave del elemento del array al que desea acceder

A continuación, el número de cuenta y el número de ruta que se han almacenado como un array asociativo en la propiedad `number` del objeto `Account` se recuperan utilizando sus nombres clave y se escriben en la página utilizando el comando echo. (Si el array fuera un array indexado, la clave sería el número de índice del elemento al que se quiere acceder).

# Almacenar un array en una propiedad de un objeto

A continuación, el número de cuenta y el número de ruta que se han almacenado como un array asociativo en la propiedad `number` del objeto `Account` se recuperan utilizando sus nombres clave y se escriben en la página utilizando el comando echo. (Si el array fuera un array indexado, la clave sería el número de índice del elemento al que se quiere acceder).



# Ejemplo: array en una propiedad de un objeto

En este ejemplo, se crea un objeto para representar una cuenta. Los números de cuenta y de ruta se almacenarán en la propiedad `$number`.

1. La clase `Account` es la misma que en el ejemplo anterior, pero la declaración del tipo de argumento para el parámetro `$number` en el método `__construct()` indica que el valor será un array.
2. Se declara una variable llamada `$number`. Contiene un array asociativo con dos claves:
  - `account_number`
  - `routing_number`

## Ejemplo: array en una propiedad de un objeto

1. Se crea un objeto utilizando la clase Cuenta. El primer argumento es la variable \$números creada en el paso 2.  
Esto asigna el array a la propiedad \$número del objeto.
2. El tipo de cuenta se escribe en la página.
3. v. Se muestra el número de cuenta.
4. vi. Se indica el número de ruta.

# Ejemplo: array en una propiedad de un objeto

```
<?php  
declare(strict_types = 1);  
  
① class Account {...}  
    // As the previous example, but number property is an array not int
```

```
        //Create an array to store in the property  
② [ $numbers = ['account_number' => 12345678,  
            'routing_number' => 987654321,];
```

```
        //Create an instance of the class and set properties
```

```
③ $account = new Account($numbers, 'Savings', 10.00);  
    ?>
```

```
    <?php include 'includes/header.php'; ?>
```

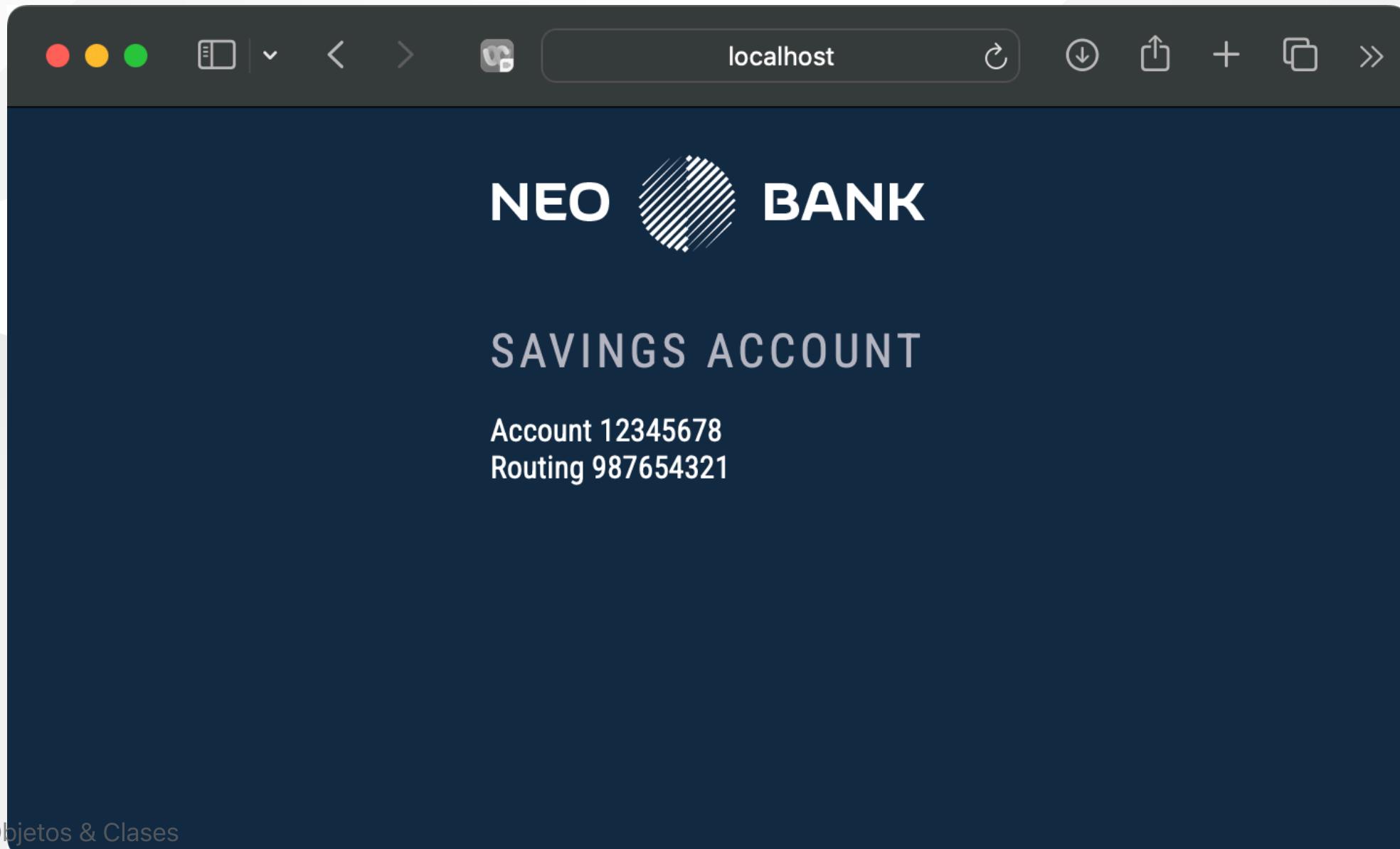
```
④ <h2><?= $account->type ?> account</h2>
```

```
⑤ Account <?= $account->number['account_number'] ?><br>
```

```
⑥ Routing <?= $account->number['routing_number'] ?>
```

```
    <?php include 'includes/footer.php'; ?>
```

# Ejemplo: array en una propiedad de un objeto



# Ejemplo: array en una propiedad de un objeto

Crea una clase `Library` que represente una biblioteca, utilizando un array para almacenar información sobre los libros. Deberás implementar métodos para añadir, eliminar y consultar libros, además de mostrar la información relevante en una página web.

**Pasos a seguir:**

1. Definición de la Clase:

- Crea una clase llamada `Library` .
- Define las propiedades `books` (de tipo array) y `libraryName` (de tipo string).
- Implementa el constructor que inicialice estas propiedades.

# Ejemplo: array en una propiedad de un objeto

## 2. Métodos de la Clase:

- Implementa un método `addBook` que permita añadir un libro a la biblioteca.
- Implementa un método `removeBook` que permita eliminar un libro de la biblioteca por su título.
- Implementa un método `getBooks` que devuelva la lista de libros en la biblioteca.

## 3. Instanciación y Manipulación del Objeto:

- Crea una instancia de la clase `Library` con un array que contenga información sobre los libros.
- Realiza algunas operaciones de añadido y eliminación de libros.
- Muestra la información de los libros en una página web.

# Almacenar un objeto en una propiedad de un objeto

En la sección anterior has visto que una propiedad de un objeto puede almacenar un array. También puedes almacenar otro objeto en la propiedad de un objeto.

A continuación, el valor asignado a la propiedad `$number` es un nuevo objeto creado con la clase `AccountNumber` (mostrado en el siguiente ejemplo).

La clase `AccountNumber` es una plantilla para un objeto que representa números de cuenta. Tiene dos propiedades:

- `$accountNumber` contiene el número de cuenta
- `$routingNumber` contiene el número de ruta (conocido como código de clasificación o BSB en algunos países).

# Almacenar un objeto en una propiedad de un objeto



# Almacenar un objeto en una propiedad de un objeto

Para acceder a una propiedad o método del objeto almacenado en la propiedad

`$number` del objeto, utiliza:

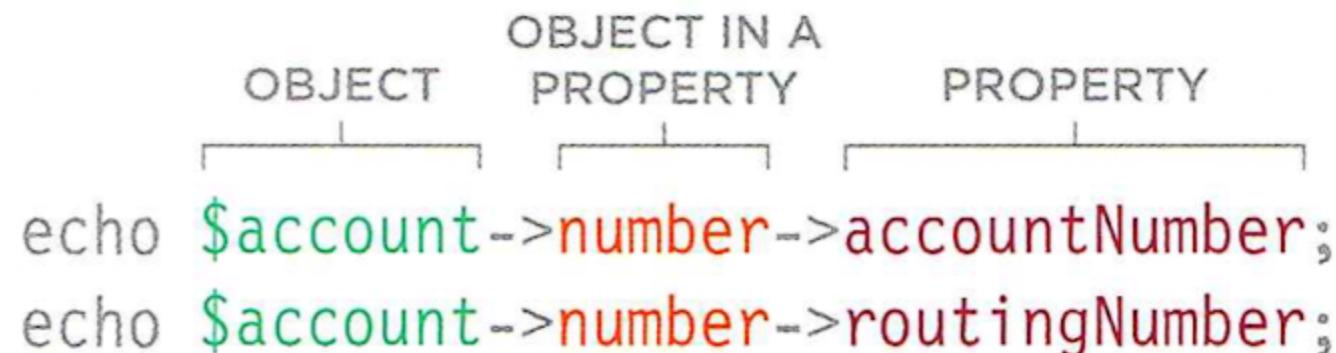
- Nombre de la variable que contiene el objeto `Account`
- Operador de objeto (`->`)
- Nombre de la propiedad que almacena los números de cuenta
- Operador de objeto nuevamente (para acceder a la propiedad del objeto contenido)
- Propiedad o método que desea utilizar

# Almacenar un objeto en una propiedad de un objeto

A continuación, la variable `$account` contiene el objeto que representa la cuenta bancaria.

Su propiedad `$number` almacena un segundo objeto creado utilizando la clase `AccountNumber`.

Sus propiedades, `$accountNumber` y `$routingNumber`, se escriben utilizando el comando `echo`.



## Ejemplo: objeto como propiedad de un objeto

El siguiente ejemplo define las clases `Account` y `AccountNumber` para manejar cuentas bancarias. La clase `Account` tiene propiedades y métodos para gestionar el número de cuenta, tipo de cuenta y balance. La clase `AccountNumber` almacena los detalles del número de cuenta y el número de ruta. El script crea una instancia de `Account`, incluye encabezado y pie de página, y muestra información de la cuenta en la página web.

# Ejemplo: objeto como propiedad de un objeto

1. La clase `Account` es la misma que en el ejemplo del array, excepto que la declaración del tipo de argumento para el parámetro `$number` en el método `__construct()` muestra que el valor para ese parámetro debe ser un objeto creado utilizando la clase `AccountNumber`.
2. Se añade una definición de clase para la clase `AccountNumber`. Tiene dos propiedades públicas:
  - `$accountNumber`
  - `$routingNumber`
3. Se utiliza un método constructor para asignar valores a estas propiedades cuando se crea un objeto utilizando esta clase.

# Ejemplo: objeto como propiedad de un objeto

4. Se crea un objeto utilizando la clase `AccountNumber`. Se almacena en una variable llamada `$numbers`.
5. Se crea un objeto para representar una cuenta utilizando la clase `Account`. El primer argumento es la variable que contiene el objeto que representa los números de cuenta.
6. El tipo de cuenta se escribe en el encabezado de la página.
7. Se muestra el número de cuenta.
8. Se indica el número de ruta (*routing number*).

# Ejemplo: objeto como propiedad de un objeto

```
<?php
declare(strict_types = 1);

class Account {
    public AccountNumber $number;
    public string $type;
    protected float $balance;

    public function __construct(AccountNumber $number, string $type, float $balance = 0.00)
    {
        $this->number = $number;
        $this->type = $type;
        $this->balance = $balance;
    }

    public function deposit(float $amount): float
    {
        $this->balance += $amount;
        return $this->balance;
    }

    public function withdraw(float $amount): float
    {
        $this->balance -= $amount;
        return $this->balance;
    }

    public function getBalance(): float
    {
        return $this->balance;
    }
}
...
...
```

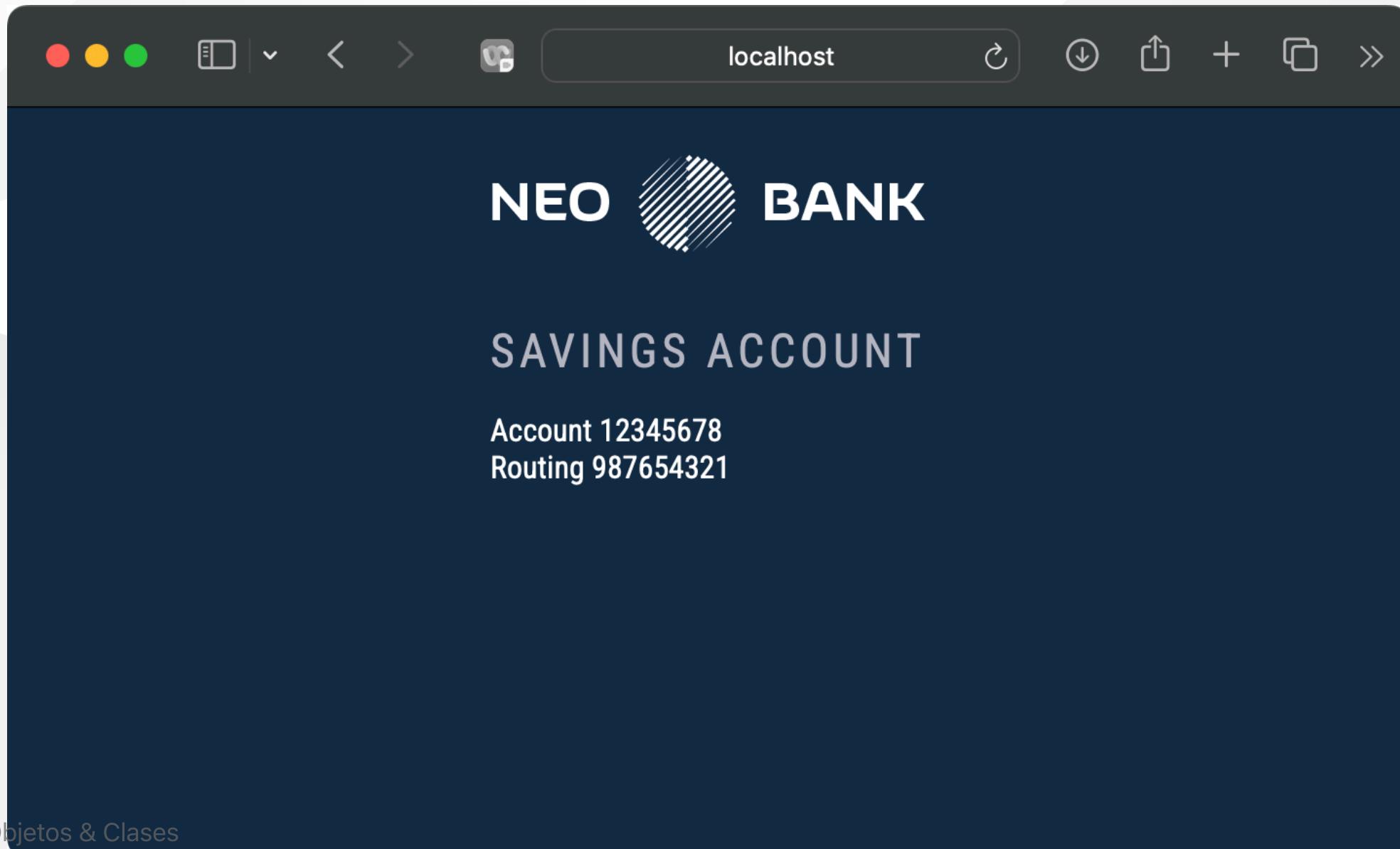
# Ejemplo: objeto como propiedad de un objeto

```
...
class AccountNumber
{
    public int $accountNumber;
    public int $routingNumber;

    public function __construct(int $accountNumber,
                                int $routingNumber)
    {
        $this->accountNumber = $accountNumber;
        $this->routingNumber = $routingNumber;
    }
}

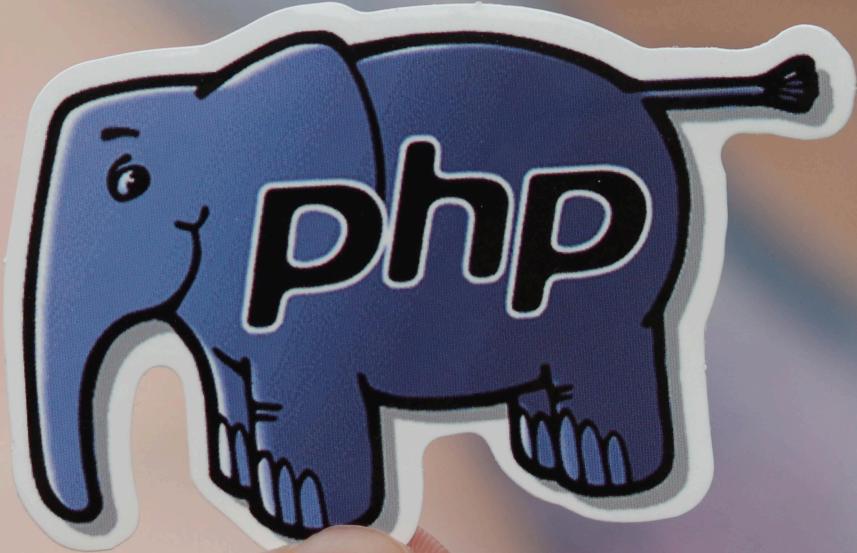
// Create an object to store in the property
$numbers = new AccountNumber(12345678, 987654321);
// Create instance of Account class + set properties
$account = new Account($numbers, 'Savings', 10.00);
?>
<?php include 'includes/header.php'; ?>
<h2><?= $account->type ?> Account</h2>
Account <?= $account->number->accountNumber ?><br>
Routing <?= $account->number->routingNumber ?><br>
<?php include 'includes/footer.php'; ?>
```

# Ejemplo: objeto como propiedad de un objeto



## Ejemplo: objeto como propiedad de un objeto

Crea una clase `Book` y una clase `Library`. La clase `Book` tendrá propiedades como título, autor y número de páginas. La clase `Library` gestionará una colección de objetos `Book`. Implementa métodos para agregar, eliminar y listar libros en la biblioteca.



## Ventajas de usar objetos

# Ventajas de usar objetos

Utilizar objetos ayuda a organizar el código, evita repetir el mismo código en diferentes páginas, facilita el mantenimiento y el uso compartido.

## Mejor organización

Si una página PHP tiene cientos de líneas de código, una detrás de otra, puede ser difícil averiguar qué hace cada línea de código.

Agrupar las variables y funciones utilizadas para representar un concepto, como un cliente o su cuenta, en una clase ayuda a mantener todo el código relacionado en un solo lugar.

# Ventajas de usar objetos

## Mejor organización

Cuando se crean objetos utilizando una clase, los programadores podemos consultar la definición de la clase para comprobar:

- Los datos disponibles en sus propiedades
- Las tareas que se pueden realizar utilizando sus métodos

Como verás en el ejemplo final de esta unidad, las definiciones de clase a menudo se almacenan en archivos separados (que se llaman **archivos de clase**). Esto facilita encontrar el código de una clase y separarlo del resto de nuestro sitio web.

# Ventajas de usar objetos

## Mejor reutilización

Puede haber varias páginas de un sitio que necesiten representar las mismas cosas; por ejemplo, varias páginas pueden representar a un cliente o a un usuario de un sitio.

En lugar de que cada página repita las declaraciones de variables (para almacenar los datos que representan al cliente) y las definiciones de funciones (para representar las tareas que pueden realizar), se puede utilizar una definición de clase como plantilla para crear un objeto que los represente.

Cualquier página que necesite representar a un cliente puede incluir el archivo de clase en la página y crear un objeto utilizando esa definición de clase como plantilla.

# Ventajas de usar objetos

## Mejor reutilización

Los programadores a veces se refieren al Principio de No Repetirse o **DRY (Don't Repeat Yourself)**. Según este principio, si se está repitiendo código, debes comprobar si una función o método de un objeto podría utilizarse para realizar la tarea en su lugar.

# Ventajas de usar objetos

## Mejor reutilización

Los programadores también se refieren al **principio de responsabilidad única**, que indica que cada función o método debe tener una única responsabilidad (en lugar de realizar múltiples tareas). Esto ayuda a maximizar la reutilización del código y facilita su comprensión.

# Ventajas de usar objetos

## Más fácil de mantener

Una organización cuidadosa del código y la maximización de su reutilización facilitan su mantenimiento. Por ejemplo:

- Si necesitas **almacenar alguna información adicional** sobre los clientes de un sitio, puedes añadir una propiedad a la definición de la clase que representa a los clientes, y esa información estará disponible en cada objeto que represente a un cliente.
- Si necesitas **cambiar la forma en que un programa realiza una tarea específica** (por ejemplo, cómo se calcula el interés de una cuenta), sólo tienes que actualizar el código en una clase, y actualizará todos los objetos que se creen utilizando esa clase.

# Ventajas de usar objetos

## Más fácil compartir código

Si piensas en cómo se escribe una clase, mientras conozcas su nombre y las propiedades y métodos que tiene, no necesitas saber cómo realiza todas las tareas que lleva a cabo.

Sólo necesitas saber

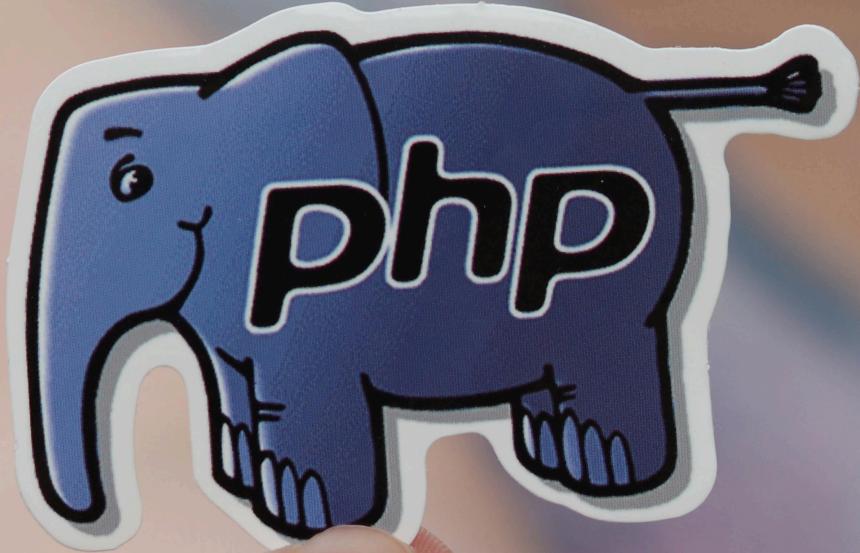
- Cómo crear un objeto utilizando esa clase.
- Qué datos puedes obtener de sus propiedades
- Qué tareas puede realizar con sus métodos

Esto ayuda a los programadores que están trabajando juntos en un equipo porque diferentes programadores pueden ser responsables de diferentes definiciones de clases.

# Ventajas de usar objetos

## Más fácil compartir código

En la siguiente sección del módulo, verás que el intérprete PHP tiene muchas funciones y clases incorporadas que te ayudan a crear páginas web. No necesitamos saber como logran sus tareas, solo como utilizarlas.



## Ejemplo final

# Ejemplo final

Este ejemplo mostrará la información del usuario y los saldos bancarios de un cliente con múltiples cuentas bancarias.

Se utilizan dos definiciones de clase:

- Una clase `Customer` se utiliza para crear un objeto que representa a un cliente del banco.
- Una clase `Account` se utiliza para crear objetos que representan las diferentes cuentas que tiene cada cliente

## Ejemplo final

Las clases serán puestas en dos archivos separados llamados `Customer.php` y `Account.php`, y esos archivos serán almacenados en una carpeta llamada *classes*.

Cualquier página que utilice las clases para crear un objeto incluirá las definiciones de las clases, al igual que se han incluido en cada página la cabecera y el pie de página.

# Ejemplo final

En la página que se mostrará como resultado:

- Se creará un objeto `Customer` utilizando la clase `Customer`
- Se añadirá una nueva propiedad a la clase `Customer` llamada `$accounts`
- La propiedad `$accounts` contendrá un array
- Ese array almacenará dos objetos `Account` que representan los dos tipos de cuenta que tiene el cliente (creados utilizando la nueva clase `Account`)

Esto muestra cómo se puede crear una jerarquía de objetos, donde un objeto contiene a otro.

## Ejemplo final

La página muestra el nombre del cliente y utiliza un bucle `foreach` para recorrer cada una de las cuentas que tiene el cliente. Dentro del bucle, se mostrará el *número de cuenta*, el *tipo* y el *saldo* de cada cuenta.

Una sentencia condicional comprueba si el usuario está en *descubierto* y, si es así, muestra el saldo en naranja en lugar de en blanco.

# Ejemplo final

```
// Account.php
<?php
class Account {
    public int $number;
    public string $type;
    protected float $balance;

    public function __construct(int $number, string $type, float $balance = 0.00)
    {
        $this->number = $number;
        $this->type = $type;
        $this->balance = $balance;
    }

    public function deposit(float $amount): float
    {
        $this->balance += $amount;
        return $this->balance;
    }

    public function withdraw(float $amount): float
    {
        $this->balance -= $amount;
        return $this->balance;
    }

    public function getBalance(): float
    {
        return $this->balance;
    }
}
```

# Ejemplo final

```
// Customer.php
<?php
class Customer
{
    public string $forename;
    public string $surname;
    public string $email;
    private string $password;
    public array $accounts;

    function __construct(string $forename, string $surname, string $email,
                        string $password, array $accounts)
    {
        $this->forename = $forename;
        $this->surname = $surname;
        $this->email = $email;
        $this->password = $password;
        $this->accounts = $accounts;
    }
    function getFullName()
    {
        return $this->forename . ' ' . $this->surname;
    }
}
```

# Ejemplo final

```
// example.php
<?php
include 'classes/Account.php';
include 'classes/Customer.php';

$accounts = [new Account(20489446, 'Checking', -20),
            new Account(20148896, 'Savings', 380),];

$customer = new Customer('Ivy', 'Stone', 'ivy@eg.link', 'Jup!t3r2684', $accounts);
?>

<?php include 'includes/header.php'; ?>
<h2>Name: <b><?= $customer->getFullName() ?></b></h2>






```

# Ejemplo final

Las dos definiciones de clase se crean en `Account.php` y `Customer.php` y se almacenan en una carpeta llamada `classes`. Las clases pueden ser incluidas en cualquier página que necesite crear ese tipo de objeto utilizando una sentencia PHP *include* (ver Paso 5).

1. La clase `Account` es la misma que fue creada previamente.
2. La clase `Customer` se basa en la que hemos estudiado previamente en esta unidad.  
La nueva propiedad `$accounts` contiene un array de objetos; cada objeto representa una de las cuentas del cliente.
3. La propiedad `$accounts` se añade al método constructor.

# Ejemplo final

4. Un nuevo método devuelve el nombre completo del cliente.
5. La página para mostrar las cuentas del usuario incluye las definiciones de las clases `Account` y `Customer` necesarias para crear dichos objetos.
6. Se crea un array indexado y se almacena en una variable llamada `$accounts`.  
Contiene dos objetos creados utilizando la clase `Account`. Cada objeto representa una de las cuentas bancarias del cliente.
7. Se crea un nuevo objeto `Customer` para representar al cliente y se almacena en una variable llamada `$customer`. El argumento final es el array de cuentas creado en el paso anterior.

# Ejemplo final

8. El nuevo método `getFullName()` del objeto `Customer` devuelve el nombre completo del cliente, que se muestra en el encabezado.
9. Un bucle `foreach` recorre el array almacenado en la propiedad `$accounts` del objeto `Customer`. En el bucle, cada cuenta se mantiene en una variable llamada `$account`.
10. El número y tipo de cuenta se escriben en la página.
11. Una sentencia `if` comprueba si el saldo es 0 o más.
12. Si lo es, se crea un elemento `<td>` con el crédito de clase.
13. Si no lo es, se sitúa en un elemento `<td>` con una clase llamada `overdrawn` (descubierto).
14. Se escribe el saldo.

# Actividad propuesta

Desarrollar una aplicación web sencilla que gestione un **parque de vehículos** utilizando PHP y HTML, aplicando los conceptos de clases y objetos aprendidos en esta unidad.

## Requisitos:

- Crear una clase `Vehicle` que represente un vehículo.
- Crear una clase `Fleet` que represente un parque de vehículos.
- Implementar las propiedades y métodos necesarios en cada clase.
- Crear un script PHP para mostrar la información del parque de vehículos y sus vehículos en una página web.

# Actividad propuesta

## Descripción de la Actividad

### Paso 1: Definición de la Clase Vehicle

Define una clase `Vehicle` que contenga las siguientes propiedades:

- `make` : la marca del vehículo.
- `model` : el modelo del vehículo.
- `licensePlate` : la matrícula del vehículo.
- `available` : un booleano que indica si el vehículo está disponible o no.

# Actividad propuesta

## Descripción de la Actividad

La clase `Vehicle` debe tener los siguientes métodos:

- `__construct($make, $model, $licensePlate, $available)` : inicializa las propiedades del vehículo.
- `getDetails()` : retorna un string con los detalles del vehículo.
- `isAvailable()` : retorna true si el vehículo está disponible, false en caso contrario.

# Actividad propuesta

## Descripción de la Actividad

### Paso 2: Definición de la Clase Fleet

Define una clase `Fleet` que contenga las siguientes propiedades:

- `name` : el nombre del parque de vehículos.
- `vehicles` : un array de objetos `Vehicle`.

# Actividad propuesta

## Descripción de la Actividad

La clase `Fleet` debe tener los siguientes métodos:

- `__construct($name, $vehicles)` : inicializa las propiedades del parque de vehículos.
- `addVehicle($vehicle)` : agrega un vehículo al parque.
- `listVehicles()` : lista todos los vehículos del parque.
- `listAvailableVehicles()` : lista solo los vehículos disponibles.

# Actividad propuesta

## Descripción de la Actividad

### Paso 3: Creación de Objetos y Visualización

Crea un script PHP que:

1. Incluya las clases definidas.
2. Cree una instancia de `Fleet` y varias instancias de `Vehicle`.
3. Agregue los vehículos al parque.
4. Muestre la información del parque de vehículos y sus vehículos en una página HTML.



## Resumen

# Resumen

- Los **objetos** agrupan variables y funciones que representan algo del mundo que nos rodea.
- En un objeto, las variables se llaman **propiedades** y las funciones, **métodos**.
- Una **clase** se utiliza como plantilla para crear objetos.
- Las **definiciones de clase** establecen las propiedades y métodos que tendrá cada objeto creado utilizando esa clase.

# Resumen

- El método `__construct()` se ejecuta cuando **se crea un objeto**. Puede ser utilizado para poner valores en las propiedades.
- `$this` accede a una propiedad o método **del propio objeto** donde se invoca.
- Las propiedades pueden ser declaradas como *públicas* (pueden ser accedidas por código fuera del objeto) o *protegidas* (sólo pueden ser utilizadas por código dentro del objeto).
- Las clases y los objetos **ayudan a organizar, reutilizar, mantener y compartir código de forma más eficaz**.

## Unidad 4: Objetos y Clases





## Referencias

# Referencias

- [PHP 8.0: Class constructor property promotion](#)
- [Visibilidad en clases y objetos \(php.net\)](#)
- [PHP – Classes and Objects](#)
- [PHP Objects \(phptutorial.net\)](#)

# Bloque A

## Instrucciones básicas de programación

### Unidad 4. Objetos & Clases

