cpifp Los Enlaces

DESARROLLO WEB EN ENTORNO SERVIDOR

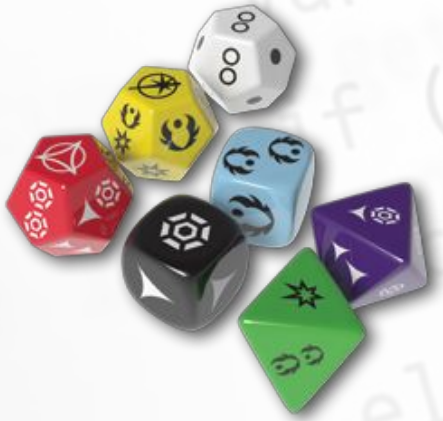
2º DESARROLLO DE APLICACIONES WEB

TEMA 11. PROGRAMACIÓN ORIENTADA A OBJETO

cpi'fp Los Enlaces

T11. PROGRAMACIÓN ORIENTADA A OBJETO

1. OBJECT ORIENTED PROGRAMMING (OOP)



```
class Car{
    public $color;
    public $model;
}
```

cpifp Los Enlaces

T11. PROGRAMACIÓN ORIENTADA A OBJETO

1. OBJECT ORIENTED PROGRAMMING (OOP)

La programación orientada a objetos (Object Oriented Programming OOP) es un modelo de programación organizado por **clases y objetos** constituidos por datos y funciones.

Una **clase** define de forma encapsulada los datos y la lógica de un tipo de objetos. Los datos se denominan **propiedades** y las funciones que actúan sobre esos datos se denominan **métodos**.

Las clases nos permiten agrupar información y acciones que podemos ejecutar con dicha información.

cpifp Los Enlaces

T11. PROGRAMACIÓN ORIENTADA A OBJETO

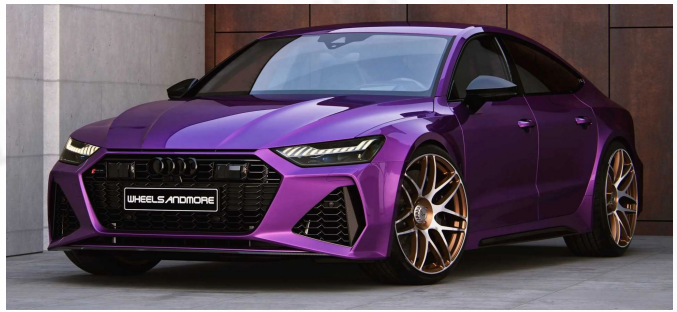
1. OBJECT ORIENTED PROGRAMMING (OOP)

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

La clase es como una plantilla que define características y funciones. El objeto es un ejemplar específico de la clase.

```
class Car {
    public $color;
    public $model;
    function Car($c, $m) {
        $this -> color = $c;
        $this -> model = $m; }
}
```

```
$audiRS = new Car ("Purple", "Audi RS7")
```



cpi'fp Los Enlaces

T11. PROGRAMACIÓN ORIENTADA A OBJETO

1. OBJECT ORIENTED PROGRAMMING (OOP)

PHP, desde la versión 5.0, implementa la definición de Clases y Objetos con el fin de permitir el desarrollo de páginas y aplicaciones web mediante Programación Orientada a Objetos (POO).

Aprender las bases de la programación orientada a objetos es esencial para poder desarrollar aplicaciones con los frameworks modernos de PHP, como Laravel o Symfony, ya que estos implementan potentes librerías de clases.

cpi fp Los Enlaces

T11. PROGRAMACIÓN ORIENTADA A OBJETO

1. OBJECT ORIENTED PROGRAMMING (OOP)

El crear una página o aplicación web haciendo uso de una correcta POO nos facilitará en gran medida la organización y reutilización del código y la depuración de errores, con lo cual nuestros proyectos web podrán ser mantenidos y ampliados de forma mucho más sencilla.

Los pasos a seguir son:

1. Definir una clase.
2. Definir objetos de dicha clase, es decir, instancias de la clase.
3. Realizar acciones con los objetos creados invocando a sus métodos.

2. NIVELES DE ACCESO

Tanto los nombres de las propiedades como los de los métodos irán precedidos de las palabras **public**, **private** o **protected**, que indican el nivel de acceso que cada uno permite de la siguiente forma:

- Las propiedades y métodos **public** pueden ser accedidos y alterados desde cualquier parte dentro y fuera de la clase.
- Las propiedades y métodos **private** solo pueden ser accedidos o alterados desde dentro de la propia clase.

2. NIVELES DE ACCESO

- Las propiedades y métodos **protected** pueden ser accedidos y alterados desde la propia clase y desde las clases derivadas.
- Para prevenir cambios indebidos en las propiedades se recomienda:
  - Definir las propiedades con nivel de acceso **private**.
  - Acceder o modificar los valores de las propiedades creando métodos **públicos** "getter" (para obtener su valor) y "setter" (para modificarlo).

T11. PROGRAMACIÓN ORIENTADA A OBJETO

3. DEFINIR UNA CLASE

La sintaxis usada para definir una clase en PHP es como sigue:

```php
class Persona {
    // Propiedades: son los datos que contiene una clase.
    private $nombre;
    private $apellidos;
    private $sexo;
    protected $fechaNacimiento; /*Ej: "03-11-2000" */
}
```

3. DEFINIR UNA CLASE

La sintaxis usada para definir una clase en PHP es como sigue:

```php
class Persona {
    // Constructor de un objeto de la clase Persona.
    // Es el método que se ejecutará cuando se cree un objeto Persona.
    function __construct($nom,$ape,$sexo,$fechaNaci) {
        $this->nombre=$nom;
        $this->apellidos=$ape;
        $this->sexo=$sexo;
        $this->fechaNacimiento=$fechaNaci; }
}
```

T11. PROGRAMACIÓN ORIENTADA A OBJETO

3. DEFINIR UNA CLASE

La sintaxis usada para definir una clase en PHP es como sigue:

```php
class Persona {
    // Métodos: son las funciones que actuarán sobre los datos.
    public function getNombre() { return $this->nombre; }
    public function setNombre($nombre) { $this->nombre = $nombre; }

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

T11. PROGRAMACIÓN ORIENTADA A OBJETO

3. DEFINIR UNA CLASE

La sintaxis usada para definir una clase en PHP es como sigue:

```
class Persona {
    // Métodos: son las funciones que actuarán sobre los datos.
    public function getSexo() { return $this->sexo; }
    public function setSexo($sexo) { $this->sexo = $sexo; }

    public function getFechaNacimiento() { return $this->fechaNacimiento; }
    public function setFechaNacimiento($fecha) { $this->fechaNacimiento =
$fecha; }
}
```

3. DEFINIR UNA CLASE

La sintaxis usada para definir una clase en PHP es como sigue:

```
class Persona {
    // Métodos: son las funciones que actuarán sobre los datos.
    public function getNombreCompleto() { return "$this->nombre,
$this->apellidos"; }

    public function calculaEdad() { /*Calcula y devuelve la edad a partir de
su fecha de nacimiento*/ }
    //Otros métodos
}
```

3. DEFINIR UNA CLASE

Método `calculaEdad()` para la clase Persona:

```php
class Persona {
    public function calculaEdad() {
        $fechaNacimiento = new DateTime($this->fechaNacimiento);
        $fechaActual = new DateTime();
        $diferencia = $fechaNacimiento->diff($fechaActual);
        $edad = $diferencia->y;
        return $edad;
    }
}
```

3. DEFINIR UNA CLASE

La clase `DateTime()` es una clase propia de PHP, cuyo constructor tiene asignado por defecto el valor de la fecha actual. Por eso, si no se le pasa ningún parámetro, se creará un objeto que almacena la fecha en la que ha sido creado.

Implementa varios métodos (add, sub, format,...), entre ellos el método `diff`, que calcula la diferencia entre dos objetos DateTime.

Para obtener los años de diferencia entre dos objetos DateTime bastará con tomar el atributo `y` (year) del objeto donde se haya almacenado dicha diferencia.

4. DEFINIR UN OBJETO DE UNA CLASE DETERMINADA

La sintaxis de PHP para crear un objeto e interactuar con él es:

```php
// Se crea un objeto, es decir una instancia, de la clase Persona:
$objPersona = new Persona("Manuel","Sánchez Pérez","H","3-12-2000");

/*Se interacciona con el objeto a través de sus métodos*/
echo "Nombre: ".$objPersona->getNombre()."<br>";
// Devuelve: "Nombre: Manuel"

echo "Apellidos: ".$objPersona->getApellidos()."<br>";
// Devuelve: "Apellidos: Sánchez Pérez"
```

4. DEFINIR UN OBJETO DE UNA CLASE DETERMINADA

La sintaxis de PHP para crear un objeto e interactuar con él es:

```php
// Se crea un objeto, es decir una instancia, de la clase Persona:
$objPersona = new Persona("Manuel","Sánchez Pérez","H","3-12-2000");

echo "Sexo: ".$objPersona->getSexo()."<br>"; // Devuelve: H

echo "Nombre completo: ".$objPersona->getNombreCompleto()."<br>";
// Devuelve: Manuel, Sanchéz Pérez

echo "Edad: ".$objPersona->calculaEdad()."<br>"; // Devuelve: 23
```

T11. PROGRAMACIÓN ORIENTADA A OBJETO

5. CONSTRUCTOR DE UNA CLASE

Un constructor es un método mágico o especial que puede aceptar una serie de argumentos y nos permite inicializar el objeto, por ejemplo asignando las propiedades mínimas requeridas para que nuestro objeto pueda funcionar (para que tenga una identidad única). En nuestro ejemplo consistirá en proporcionar el nombre, el apellido, el género y la fecha de nacimiento.

```
class Persona {
    public function __construct($nom,$ape,$sexo,$fechaNaci) {
        $this->nombre=$nom;
        $this->apellidos=$ape;
```

5. CONSTRUCTOR DE UNA CLASE

En PHP, a diferencia de Java, no podemos tener un método con el mismo nombre que la clase (en versiones anteriores a la 7.4 sí que se podía, pero actualmente genera un error).

El método constructor es un método que típicamente se suele sobrecargar, es decir, tener código diferente en función de los parámetros que aporte en su invocación.

```
class Racional {
    public function __construct($num=1, $den=1) {
        if (is_string($num)){
            $numero = explode("/",$num)
```

6. MÉTODOS MÁGICOS

Los métodos mágicos son métodos especiales que sobreescriben acciones por defecto cuando se realizan ciertas acciones sobre un objeto.

Todos los nombres de los métodos que comienzan con doble subrayado "__" son reservados por PHP. Por lo tanto, se recomienda no utilizar los nombres de métodos a menos que se sobreescriba su comportamiento.

__construct(), __destruct(), __call(), __callStatic(), __get(), __set(), __isset(), __unset(), __sleep(), __wakeup(), __serialize(), __unserialize(), __toString(), __invoke(), __set_state(), __clone(), __debugInfo()

6. MÉTODOS MÁGICOS

De entre todos ellos, es importante conocer al menos el funcionamiento de __construct(), __destruct(), __call(), y __toString().

El método constructor lo hemos visto antes. __destruct() se implementa automáticamente tan pronto como no haya otras referencias a un objeto, o en cualquier orden durante la secuencia de cierre.

También si le asignamos al objeto el valor `null` o utilizamos la función `unset()`.

6. MÉTODOS MÁGICOS

El método __toString() es invocado si queremos convertir el objeto en un string. No recibe parámetros, pues no se invoca de forma explícita.

```
class Racional {
    private $num;  private $den;
    public function __construct($num, $den){
        $this->num = $num; $this->den = $den; }
    public function __toString(){ return ("$this->num/$this->den"); }
}
$r1 = new Racional (8,5);
echo "Valor del objeto r1 = $r1";
```

6. MÉTODOS MÁGICOS

El método __call() es invocado si intentamos invocar un método que no existe en la clase instanciada. Crea un array indexado con la lista de parámetros con los que invocamos a la función.

```php
public function __call($funcion, $argumentos){
    echo "<h2>Has invocado a un método que no existe en esta clase </h2>";
    echo "Nombre de la función <strong>$funcion</strong><br />";
    echo "Lista de parámetros<br />";
    foreach ($argumentos as $param => $valor){
        echo "parámetro <strong>$param</strong> = <strong>".print_r($valor, true)."</strong> <br />";
    //Poner en print_r el segundo parámetro a true hace que, en lugar de imprimir, retorne el valor.
    }
}
```

7. HERENCIA

La **herencia** es un concepto fundamental de la POO consistente en crear nuevas clases a partir de otras ya existentes y permite la implementación de funcionalidad adicional en objetos similares sin la necesidad de reimplementar toda la funcionalidad compartida.

Una clase puede heredar métodos y propiedades de otra clase, y éstos se pueden sobreescribir empleando el mismo nombre que en la clase madre. Las clases sólo pueden heredar de una única clase (**no** es posible herencia **múltiple**), pero sí que pueden heredarse unas a otras (sí es posible herencia **multinivel**). La herencia se realiza mediante la palabra **extends**.

7. HERENCIA

Ejemplo: se va a definir la clase Usuario, hija de la clase Persona.

- La clase hija Usuario hereda las propiedades y métodos de su clase padre Persona.
- También puede añadir propiedades y métodos propios o sobreescribir los métodos de su clase padre.
- Los constructores también se heredan. Si una clase hija define un constructor propio y quiere heredar (no es obligatorio) el de la clase madre es necesario indicarlo con `parent::__construct()`

7. HERENCIA

Ejemplo: se va a definir la clase Usuario, hija de la clase Persona.

```
class Usuario extends Persona {
// Propiedades específicas de esta clase. Hereda las de la clase padre.
    private $nombreUsuario;   private $password;
// Constructor: si no lo ponemos hereda el de la clase padre.
    function __construct($nom,$ape,$sexo,$fechaNaci,$usu,$pass) {
            /*Primero invocamos al constructor de su clase padre, si es
útil*/
            parent::__construct($nom,$ape,$sexo,$fechaNaci);
            /*Después añadimos las acciones específicas*/
            $this->nombreUsuario=$usu;    $this->password=$pass; }
```

7. HERENCIA

Ejemplo: se va a definir la clase Usuario, hija de la clase Persona.

```
class Usuario extends Persona {
// Métodos específicos de esta clase. También hereda los de la clase padre.
    public function getNombreUsuario() { return $this->nombreUsuario; }
    public function setNombreUsuario( $nombreUsuario ) {
        return $this->nombreUsuario = $nombreUsuario;
    }
    public function getPassword() { return $this->password; }
    public function setPassword( $password ) { $this->password = $password;
}
```

7. HERENCIA

Ejemplo: se va a definir la clase Usuario, hija de la clase Persona.

```
class Usuario extends Persona {
/*Ejemplo de redefinición de un método de la clase padre*/
    public function getNombreCompleto() {
    /*Primero llamamos al método en la clase padre, si es útil*/
        $cadena=parent::getNombreCompleto(); /*Después lo completamos*/
        return "$cadena: {$this->nombreUsuario}*{$this->password}";
//Sería equivalente a:
     //return "{$this->getNombre()}, {$this->getApellidos()}:
                {$this->nombreUsuario}*{$this->password}";
```