

**College of Science and Engineering**



الجمهورية اليمنية  
الجامعة الوطنية  
كلية العلوم والهندسة



Dr. Ibrahim AL\_shami.

# Object Oriented Programming:

هي أسلوب برمجي يعتمد على تقسيم الكود إلى كائنات (Objects) تتفاعل فيما بينها، بدلاً من كتابة الكود بشكل إجرائي.

تساعد OOP في PHP، في تنظيم الكود وجعله أكثر قابلية لإعادة الاستخدام والصيانة، مما يجعل تطوير التطبيقات الكبيرة أكثر سهولة وفعالية. تُستخدم OOP في PHP لإنشاء تطبيقات أكثر تنظيماً، وقابلة لإعادة الاستخدام والصيانة بسهولة.

## المفاهيم الأساسية في OOP:

Classes

Objects

Properties

Methods

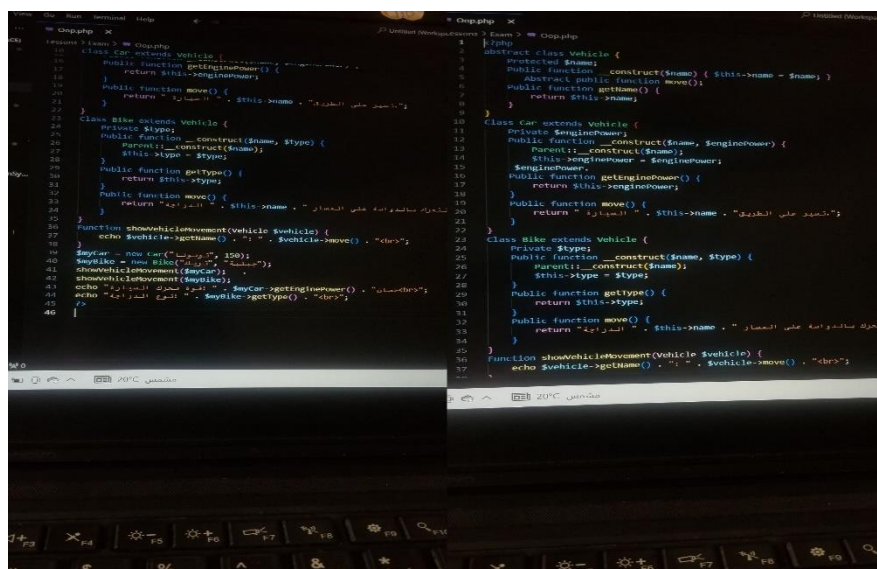
Inheritance

Abstraction

Encapsulation

Polymorphism

## كود يوضح المفاهيم OOP:



```
1 class Vehicle {
2     public function __construct($name) {
3         $this->name = $name;
4     }
5     public function getName() {
6         return $this->name;
7     }
8 }
9
10 class Car extends Vehicle {
11     private $enginePower;
12     public function __construct($name, $enginePower) {
13         parent::__construct($name);
14         $this->enginePower = $enginePower;
15     }
16     public function getEnginePower() {
17         return $this->enginePower;
18     }
19 }
20
21 class Bike extends Vehicle {
22     private $type;
23     public function __construct($name, $type) {
24         parent::__construct($name);
25         $this->type = $type;
26     }
27     public function getType() {
28         return $this->type;
29     }
30 }
31
32 function showVehicleMovement($vehicle) {
33     echo $vehicle->getName() . " : " . $vehicle->move() . "<br>";
34 }
35
36 $car = new Car("Car", 150);
37 $bike = new Bike("Bike", "Mountain");
38 showVehicleMovement($car);
39 showVehicleMovement($bike);
40 echo "أقوى محرك للسيارة: " . $car->getEnginePower() . "<br>";
41 echo "نوع الدراجة: " . $bike->getType() . "<br>";
42 }
```

## مميزات وعيوب وأهمية OOP في PHP:

### أولاً: مميزات:

#### \*إعادة استخدام الكود (Code Reusability):

باستخدام الوراثة (Inheritance) يمكن إعادة استخدام الأكواد دون الحاجة إلى كتابتها مجدداً، مما يقلل التكرار.

#### \*تحسين تنظيم الكود (Code Organization):

يسهل تقسيم المشروع إلى فئات (Classes) مستقلة، مما يجعل الكود أكثر تنظيماً وسهولة في الفهم.

#### \*تسهيل الصيانة والتعديل (Maintainability & Scalability):

أي تعديل على الكود يصبح أسهل لأنه يتمركز داخل الفئات بدلاً من تغييره في جميع أنحاء المشروع.

#### \*إخفاء البيانات (Encapsulation):

يمكن تقييد الوصول إلى الخصائص والوظائف باستخدام private، protected، مما يحسن أمان البيانات داخل الكائنات.

#### \*التعددية الشكلية (Polymorphism):

يسمح باستخدام نفس الدالة بطرق مختلفة حسب الفئة التي تستدعيها، مما يضيف مرونة في التصميم.

#### \*التجريد (Abstraction):

يجبر المطورين على استخدام هيكل معين للبرمجة، مما يسهل التعاون بين فرق العمل.

### \*سهولة التوسع (Extensibility):

يمكن إضافة ميزات جديدة بسهولة عن طريق إنشاء فئات جديدة أو تعديل الفئات الموجودة بدون الحاجة إلى تغيير الكود الأساسي.

### \*تحسين التعاون بين المطورين:

يمكن لعدة مطورين العمل على نفس المشروع بسهولة عن طريق تقسيم العمل إلى فئات مستقلة.

### ثانيًا: عيوب:

#### \*زيادة استهلاك الذاكرة والأداء:

إنشاء كائنات والتعامل مع الوراثة والواجهات يستهلك المزيد من الموارد مقارنةً بالبرمجة الإجرائية.

#### \*تعقيد الكود للمشاريع الصغيرة:

في المشاريع البسيطة، قد تكون OOP معقدة وغير ضرورية مقارنةً بالبرمجة الإجرائية.

#### \*زيادة وقت التطوير

قد يستغرق تصميم الفئات وإنشاء العلاقات بينها وقتًا أطول مقارنةً بالبرمجة العادية.

#### \*صعوبة في تصحيح الأخطاء (Debugging)

عندما تكون الكائنات مترابطة، قد يكون من الصعب تتبع مصدر الأخطاء.

#### \*منحنى تعليمي أكثر انحدارًا

يحتاج المبتدئون إلى وقت أطول لفهم مفاهيم مثل الوراثة، التجريد، والتعددية الشكالية مقارنةً بالبرمجة الإجرائية.

#### \*استهلاك زائد للموارد عند الاستخدام غير الصحيح:

استخدام OOP بشكل غير منظم قد يؤدي إلى تحميل غير ضروري على السيرفر، خاصة عند عدم تطبيق مفاهيم الأداء الجيد مثل التحميل الكسول (Lazy Loading).

### ثالثاً: أهمية

#### **\*تطوير تطبيقات كبيرة ومعقدة بسهولة**

تستخدم OOP في بناء أنظمة إدارة المحتوى (مثل ، WordPress وأنظمة التجارة الإلكترونية (مثل Magento و OpenCart) والتطبيقات الضخمة.

#### **\*تعزيز الإنتاجية والكفاءة:**

تسهيل عملية التطوير عبر إعادة استخدام الكود، مما يسرع من عملية البرمجة ويساعد على تقليل الأخطاء.

#### **\*تحسين الأداء العام للتطبيقات:**

عند تصميم النظام بشكل صحيح، يمكن أن يؤدي إلى تحسين أداء التطبيق من خلال تقليل التكرار وتحسين الهيكلية.

#### **\*دعم البرمجة المتقدمة مثل MVC:**

يمكن استخدام OOP في إطار عمل MVC (Model-View-Controller) مثل Laravel و CodeIgniter لتنظيم الكود بشكل أفضل.

#### **\*مرونة وسهولة التوسع (Scalability):**

أي ميزة جديدة يمكن إضافتها بسهولة دون الحاجة إلى تعديل الكود القديم، مما يجعل المشاريع المستقبلية أسهل في التطوير.

#### **\*تحسين تجربة المستخدم:**

من خلال جعل التطبيقات أكثر استقراراً وأماناً وأسرع في الأداء، مما يؤدي إلى تجربة مستخدم أفضل.

---



## المفاهيم ال OOP فى PHP :

1- الكائنات (Objects) والفئات (Classes):

**الكائن (Object):** هو نسخة من الفئة (Class) تحتوي على بيانات (خصائص) وسلوكيات (طرق).

**الفئة (Class):** هي قالب يُستخدم لإنشاء كائنات.

**مثال:** على تعريف فئة وإنشاء كائن منها

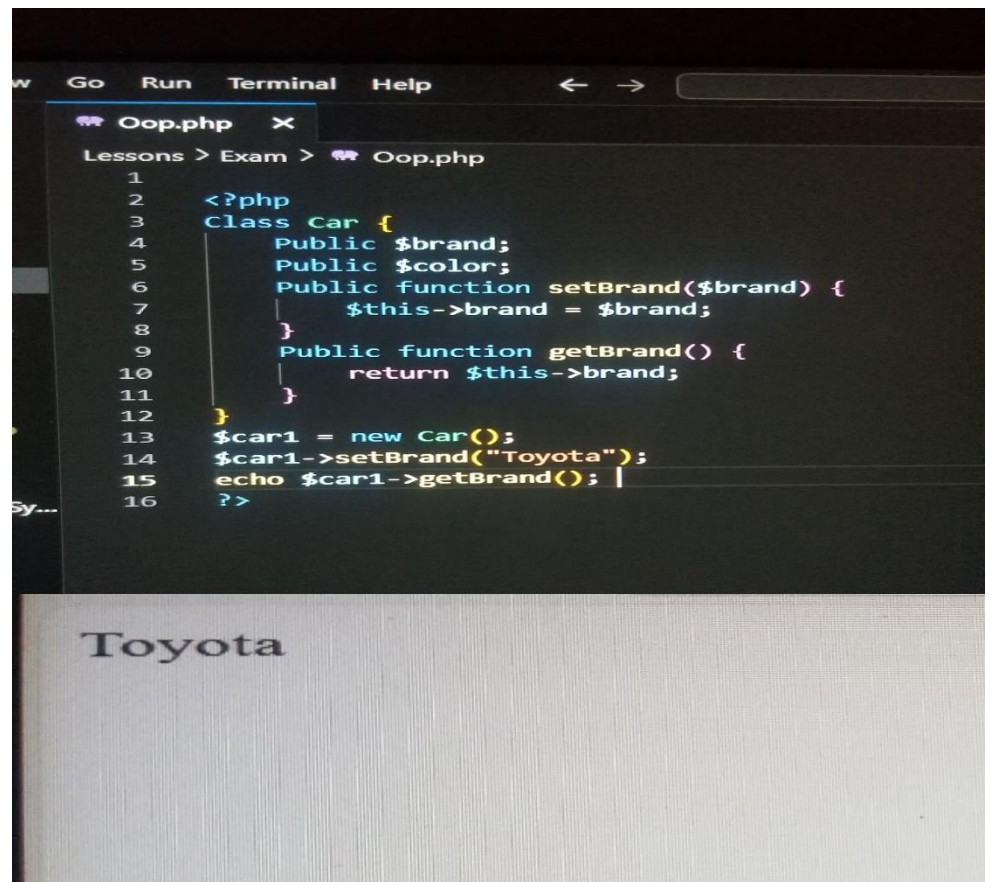
<?php

```
Class Car {  
    Public $brand;  
    Public $color;  
    Public function setBrand($brand) {  
        $this->brand = $brand; }  
    Public function getBrand() {  
        return $this->brand;  
    }  
}  
  
$car1 = new Car();  
$car1->setBrand("Toyota");  
echo $car1->getBrand();  
?>
```

## بعد تنفيذ الكود:

هنا تم تنفيذ الكود  
وتم طباعة

**Toyota**



```
1  <?php
2
3  Class Car {
4      Public $brand;
5      Public $color;
6      Public function setBrand($brand) {
7          $this->brand = $brand;
8      }
9      Public function getBrand() {
10         return $this->brand;
11     }
12 }
13 $car1 = new Car();
14 $car1->setBrand("Toyota");
15 echo $car1->getBrand(); |
16 ?>
```

Toyota

## 2- الخصائص (Properties) والطرق (Methods)

**الخصائص:** هي متغيرات تخزن بيانات داخل الكائن.

**الطرق:** هي دوال داخل الفئة تُحدد سلوك الكائن.

**كود:**

```
<?php
```

```
Class Person {
```

```
    Public $name;
```

```
    Public function sayHello()
```

```
{return "Hello, my name is " . $this->name;}  
}
```

```
$person1 = new Person();
```

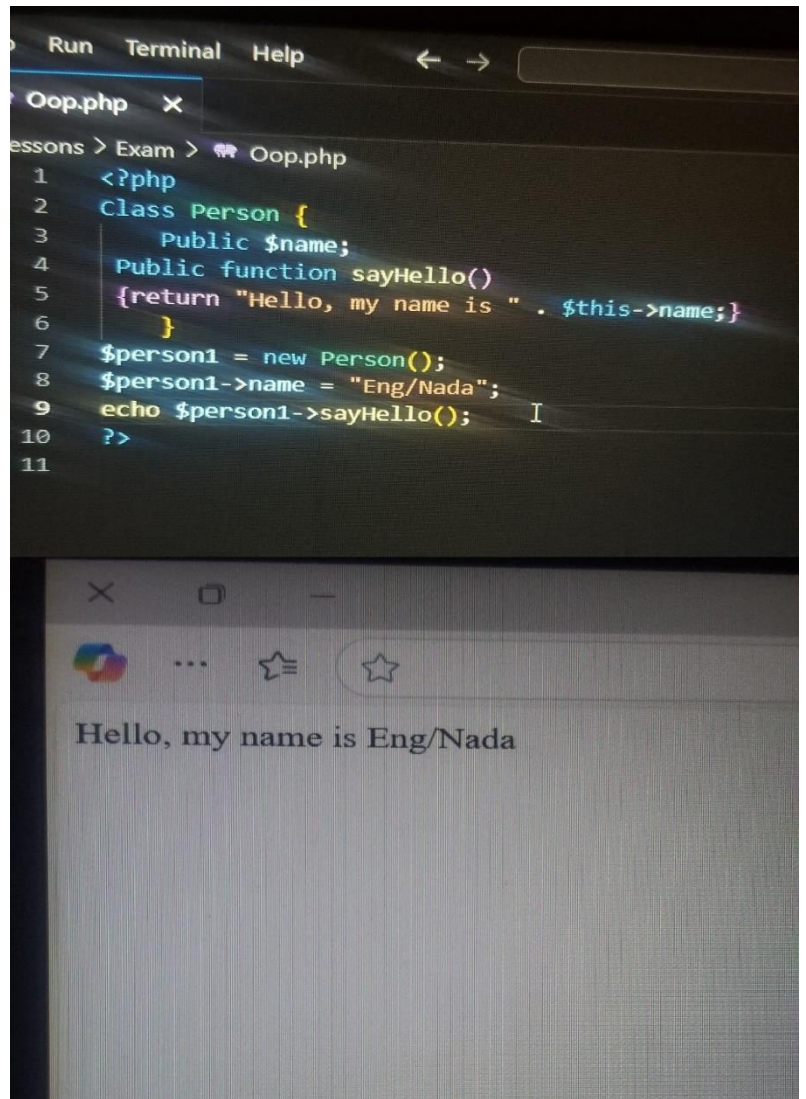
```
$person1->name = "Eng/Nada";
```

```
echo $person1->say Hello();
```

بعد تنفيذ الكود:

هنا تم تنفيذ الكود  
وتم طباعة

**Hello, my name is  
Eng/Nada**



The image shows two parts of a computer screen. The top part is a code editor window titled 'Oop.php' with a menu bar (Run, Terminal, Help) and a breadcrumb 'Lessons > Exam > Oop.php'. The code in the editor is as follows:

```
1 <?php  
2 Class Person {  
3     Public $name;  
4     Public function sayHello()  
5     {return "Hello, my name is " . $this->name;}  
6     }  
7 $person1 = new Person();  
8 $person1->name = "Eng/Nada";  
9 echo $person1->sayHello();  
10 ?>  
11
```

The bottom part of the image shows a web browser window with the output of the script: 'Hello, my name is Eng/Nada'.



### 3-المحددات (Access Modifiers)

تحدد مدى إمكانية الوصول إلى الخصائص والطرق داخل الكائنات.

1. – **Public** يمكن الوصول إليه من أي مكان.
2. – **Private** يمكن الوصول إليه فقط داخل الفئة نفسها.
3. – **Protected** يمكن الوصول إليه داخل الفئة أو الفئات الموروثة.

**كود:**

```
<?php
```

```
Class BankAccount {  
    Private $balance = 0;  
    Public function deposit($amount) {  
        $this->balance += $amount;  
    }  
    Public function getBalance() {  
        Return $this->balance;  
    }  
}  
  
$account = new BankAccount();  
$account->deposit(500);  
Echo $account->getBalance();  
?>
```

## بعد تنفيذ الكود:

هنا يتم تنفيذ الكود  
ويتم طباعة

**500**

```
1  <?php
2
3  Class BankAccount {
4      Private $balance = 0;
5      Public function deposit($amount) {
6          $this->balance += $amount;
7      }
8      Public function getBalance() {
9          return $this->balance;
10     }
11 }
12 $account = new BankAccount();
13 $account->deposit(500);
14 Echo $account->getBalance();
15 ?>
16 |
```

500

#### 4- المُنشئ (Constructor) والمُدمر (Destructor)

المُنشئ (construct) يُستخدم لتهيئة الكائن عند إنشائه.

المُدمر (destruct) يُستخدم لتنفيذ كود عند تدمير الكائن.

كود:

```
<?php
```

```
Class User {
```

```
    Public $name;
```

```
    Public function __construct($name) {
```

```
        $this->name = $name;
```

```
        echo "Welcome, $this->name!";}
```

```
    echo "User $this->name is deleted."; }}
```

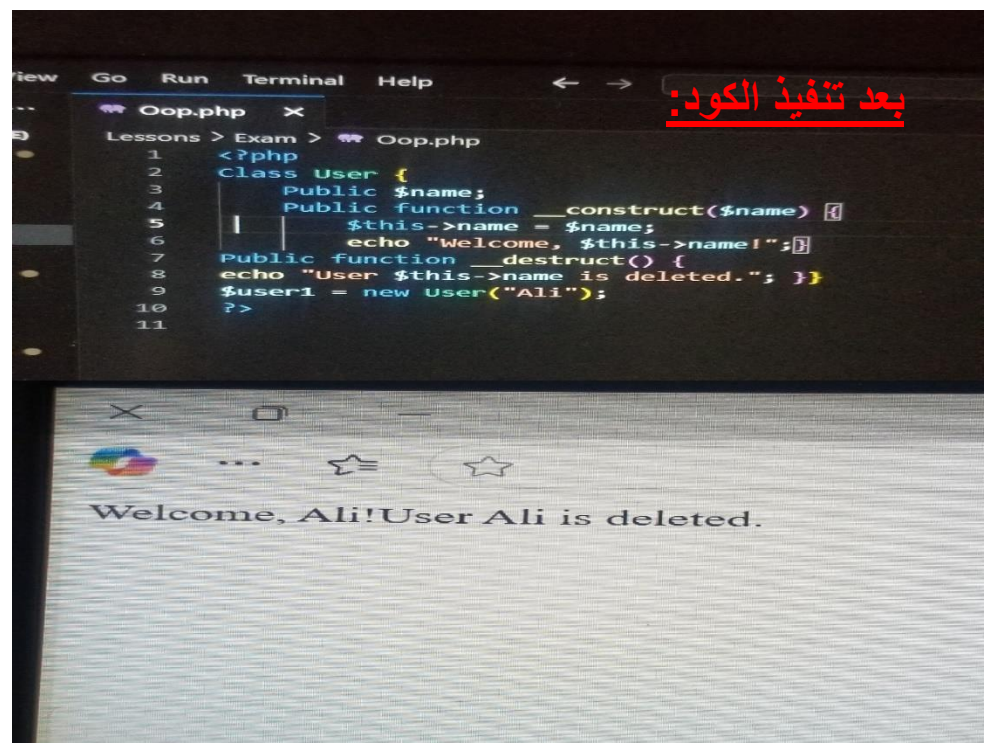
```
$user1 = new User("Ali");!
```

```
?>
```

هنا تم تنفيذ الكود

وتم طباعة

**Welcome ,Ali**



## \*الوراثة (Inheritance)

تمكّن الفئات من إعادة استخدام الكود من فئات أخرى.

**كود:**

```
<?php
```

```
Class Animal {
```

```
    Public $name;
```

```
    Public function makeSound() {
```

```
        return "Some generic sound"; }}
```

```
Class Dog extends Animal {
```

```
    Public function makeSound() {
```

```
        return "Bark!"; }}
```

```
$dog = new Dog();
```

```
$dog->name = "Buddy";
```

```
echo $dog->makeSound(); // Bark!
```

```
?>
```

بعد تنفيذ الكود:

هنا تم تنفيذ الكود  
ثم طباعة  
**Bark!**

```
Oop.php X
Lessons > Exam > Oop.php
1
2  <?php
3  Class Animal {
4      Public $name;
5      Public function makeSound() {
6          return "Some generic sound"; }}
7  Class Dog extends Animal {
8      Public function makeSound() {
9          return "Bark!"; }}
10 $dog = new Dog();
11 $dog->name = "Buddy";
12 echo $dog->makeSound(); // Bark!
13 ?>
14
```

A screenshot of a web browser window showing the output of the PHP code. The browser's address bar is visible with a search icon, a menu icon, and a star icon. The main content area of the browser displays the text "Bark!" in a large, bold, black font.

#### 4-التجريد (Abstraction)

يُستخدم لإنشاء فئات تحتوي على دوال مجردة (بدون تنفيذ)، تُجبر الفئات المتفرعة على تنفيذها.

**كود:**



**<?php**

```
Abstract class Shape {  
    Abstract public function getArea();}  
  
Class Square extends Shape {  
    Private $side;  
  
    Public function __construct($side) {  
        $this->side = $side;}  
  
    Public function getArea() {  
        return $this->side * $this->side;}}  
  
$square = new Square(4);  
echo $square->getArea();  
  
?>
```

#### 4. الواجهات (Interfaces)

تُشبه التجريد لكنها لا تحتوي على أي تنفيذ، وتُجبر الفئات على تنفيذ جميع الدوال الموجودة في الواجهة.

**<?php**

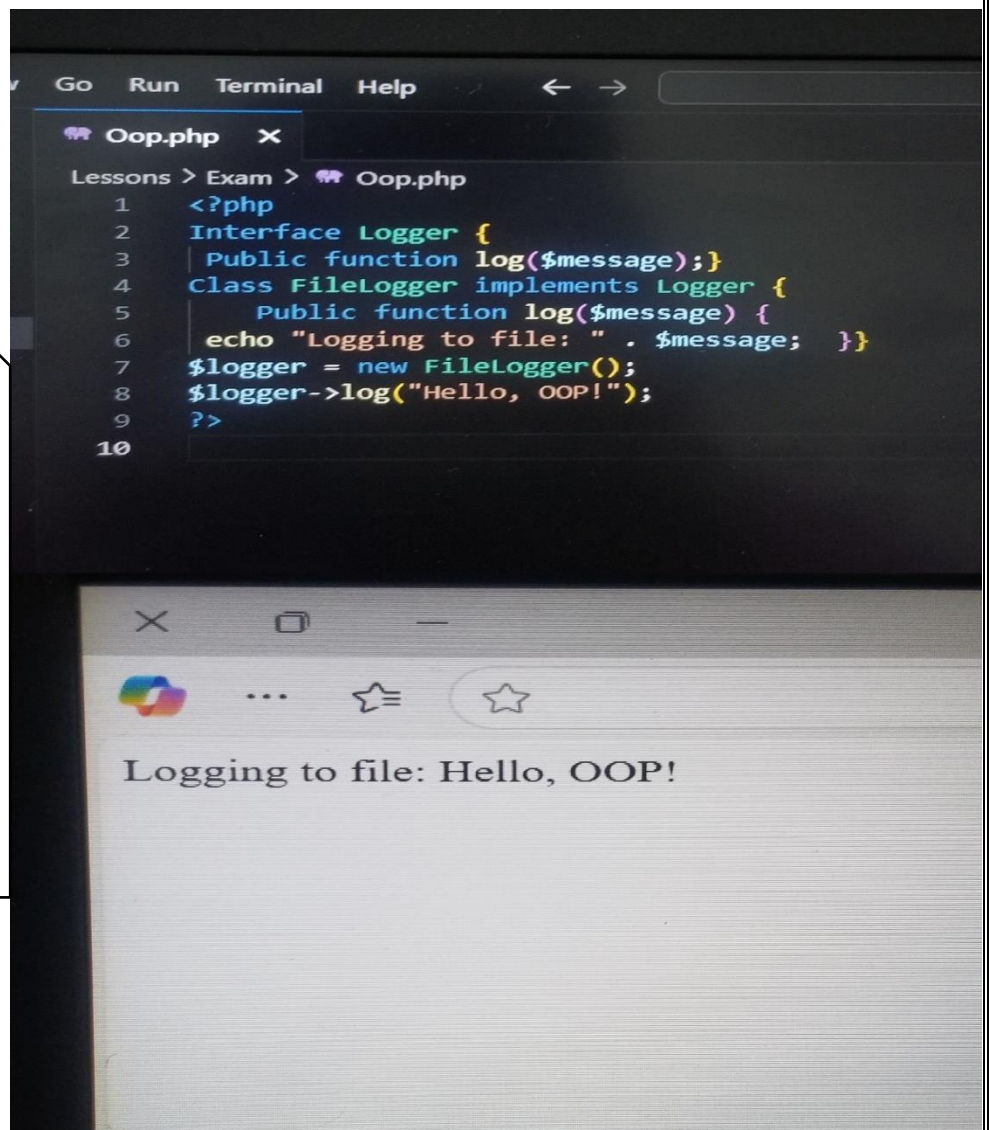
```
Interface Logger {  
  
    Public function log($message);}  
  
Class FileLogger implements Logger {
```

```
Public function log($message) {  
    echo "Logging to file: " . $message; }  
$logger = new FileLogger();  
$logger->log("Hello, OOP!");  
?>
```

الكود بعد التنفيذ:

هنا تم تنفيذ الكود  
تم طباعة

**Logging to  
file:Hello,OOP**



The image shows a screenshot of a code editor and a web browser. The code editor, titled 'Oop.php', contains the following PHP code:

```
1 <?php  
2 Interface Logger {  
3     Public function log($message);  
4 Class FileLogger implements Logger {  
5     Public function log($message) {  
6         echo "Logging to file: " . $message; }  
7 $logger = new FileLogger();  
8 $logger->log("Hello, OOP!");  
9 ?>  
10
```

Below the code editor, a web browser window displays the output of the code: "Logging to file: Hello, OOP!".

## 5. التعددية الشكلية (Polymorphism)

تمكين الفئات المختلفة من تنفيذ نفس الدالة بطريقة مختلفة.

**<?php**

```
Class Bird {
```

```
    Public function sound() {
```

```
        return "Chirp";
```

```
    }
```

```
}
```

```
Class Duck extends Bird {
```

```
    Public function sound() {
```

```
        return "Quack";
```

```
    }
```

```
}
```

```
$bird = new Bird();
```

```
$duck = new Duck();
```

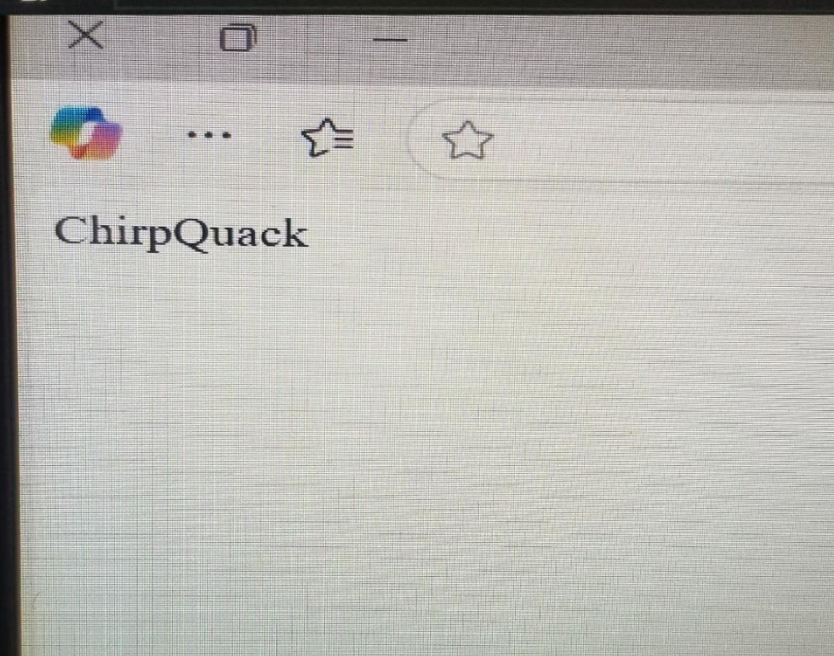
```
echo $bird->sound();
```

```
echo $duck->sound();
```

```
?>
```

## بعد تنفيذ الكود:

```
Lessons > Exam > Oop.php
1  <?php
2  Class Bird {
3      Public function sound() {
4          return "Chirp";
5      }
6  }
7  Class Duck extends Bird {
8      Public function sound() {
9          return "Quack";
10     }
11 }
12 $bird = new Bird();
13 $duck = new Duck();
14 echo $bird->sound();
15 echo $duck->sound();
16 ?>
17
```

A screenshot of a web browser window. The address bar shows a star icon and a menu icon. The main content area displays the text 'ChirpQuack' in a black serif font.

هنا تم تنفيذ الكود

ويتم طباعة

**Chirp**

**Quack**

## 6- السمات (Traits)

تسمح بإضافة وظائف للفئات دون الحاجة إلى الوراثة.

**<?php**

Trait Greet {

Public function sayHi() {

return "Hi!";

}}

Class User { Use Greet;}

\$user = new User();

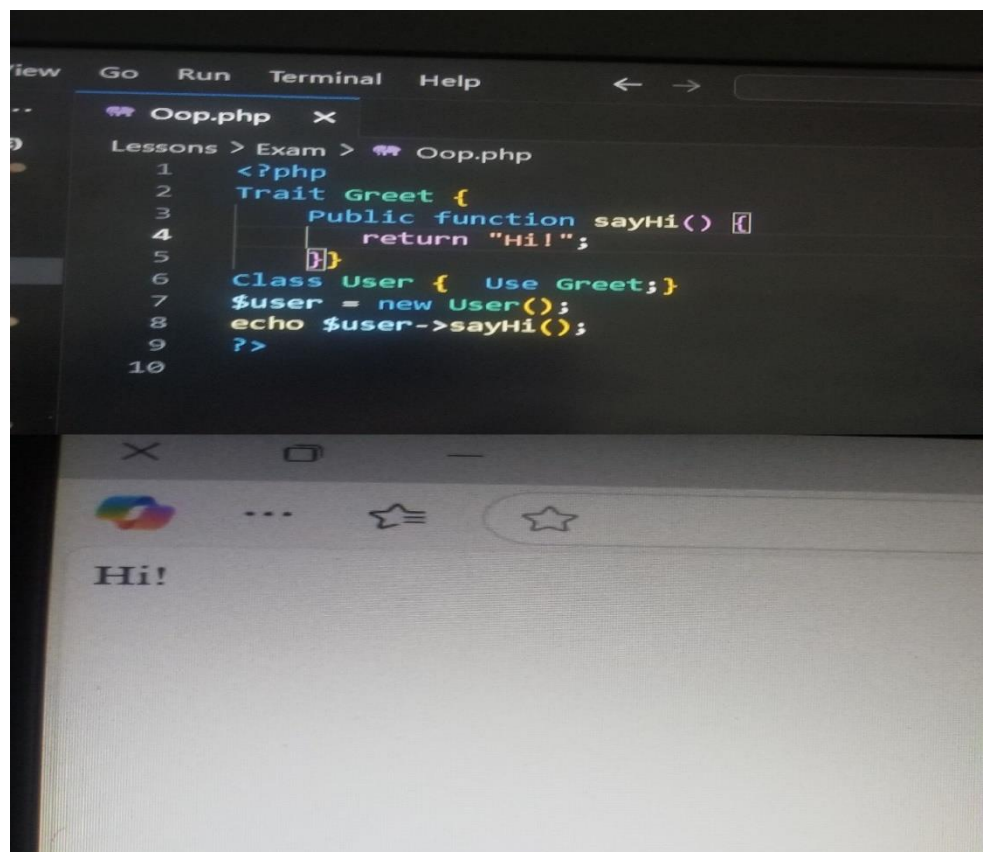
echo \$user->sayHi();

?>

هنا تم تنفيذ الكود

وتم طباعة

**Hi !**





## الدوال السحرية OOP في PHP:

هي دوال تبدأ ب (\_\_)، وتُستخدم لتنفيذ عمليات معينة عند استدعاء أو تعديل أو حذف الكائنات وخصائصها.

- تُنفَّذ عند إنشاء كائن جديد()\_\_construct
- تُنفَّذ عند تدمير الكائن()\_\_destruct
- تُنفَّذ عند محاولة الوصول إلى خاصية غير معرفة()\_\_get(\$name)
- تُنفَّذ عند محاولة تعيين قيمة لخاصية غير معرفة()\_\_set(\$name, \$value)
- على خاصية()empty أو()isset تُنفَّذ عند استخدام()\_\_isset(\$name) غير معرفة.
- على خاصية غير معرفة()unset تُنفَّذ عند استخدام()\_\_unset(\$name)
- تُنفَّذ عند محاولة استدعاء دالة غير معرفة()\_\_call(\$name, \$arguments) معرفة في الكائن.
- تُنفَّذ عند محاولة استدعاء دالة غير معرفة في نطاق ثابت()\_\_callStatic(\$name, \$arguments)
- تُنفَّذ عند محاولة طباعة الكائن كسلسلة نصية()\_\_toString
- تُنفَّذ عند استنساخ الكائن()\_\_clone
- تُنفَّذ عند استدعاء الكائن كدالة()\_\_invoke

## مثال عملي:

```
Oop.php X
Lessons > Exam > Oop.php
1 <?php
2 class MagicExample {
3     private $data = [];
4
5     public function __construct() {
6         echo "اتم إنشاء الكائن<br>";
7     }
8
9     public function __get($name) {
10        return $this->data[$name] ?? "لاغير موجودة '{$name}' الخاصة";
11    }
12
13    public function __set($name, $value) {
14        $this->data[$name] = $value;
15        echo "{$name} تم تعيين {<br>";
16    }
17
18    public function __toString() {
19        return "هذا كائن من الصنف MagicExample";
20    }
21
22    public function __invoke($message) {
23        echo "{$message} اتم استدعاء الكائن برسالة<br>";
24    }
25 }
26
27 $obj = new MagicExample();
28 $obj->name = "عقري";
29 echo $obj->name . "<br>";
30 echo $obj . "<br>";
31 $obj("امرحنا بك");
32 ?>
33
```

## بعد التنفيذ:



## المتغيرات السحرية OOP في PHP :

هي ثوابت معرفة مسبقًا تُستخدم للحصول على معلومات حول النص البرمجي

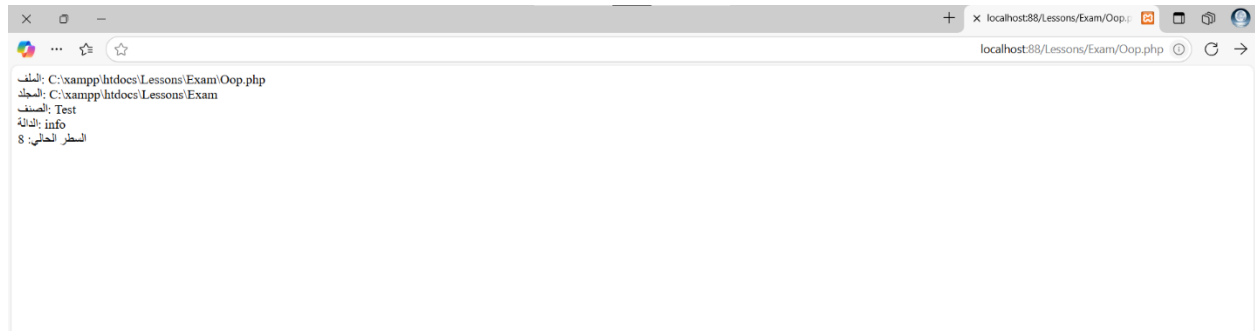
| الوصف                                   | السحرية المتغير |
|---|-----------------|
| رقم السطر الحالي في الملف.              | __LINE__        |
| المسار الكامل للملف الحالي.             | __FILE__        |
| اسم المجلد الذي يحتوي على الملف الحالي. | __DIR__         |
| اسم الدالة التي يتم استدعاؤها حاليًا.   | __FUNCTION__    |
| اسم الصنف الذي يتم استدعاؤه حاليًا.     | __CLASS__       |
| اسم الدالة مع اسم الصنف.                | __METHOD__      |
| اسم النطاق الحالي.                      | __NAMESPACE__   |

### مثال عملي:

```
Go Run Terminal Help  ← →  Untitled (Workspace)

Oop.php x
Lessons > Exam > Oop.php
1  <?php
2  class Test {
3      public function info() {
4          echo "الملف: " . __FILE__ . "<br>";
5          echo "المجلد: " . __DIR__ . "<br>";
6          echo "الصنف: " . __CLASS__ . "<br>";
7          echo "الدالة: " . __FUNCTION__ . "<br>";
8          echo "السطر الحالي: " . __LINE__ . "<br>";
9      }
10 }
11
12 $obj = new Test();
13 $obj->info();
14 ?>
15
```

### بعد التنفيذ:



## متى تستخدم OOP في PHP؟

✓ مناسبة عندما يكون لديك تطبيق معقد يحتاج إلى إعادة استخدام الكود، مثل أنظمة إدارة المحتوى (CMS) أو تطبيقات التجارة الإلكترونية.

✗ غير مناسبة عندما يكون لديك مشروع صغير مثل سكريبت بسيط لمعالجة نموذج (Form)

إذا كنت تعمل على مشروع متوسط إلى كبير، فإن استخدام OOP هو الخيار الأفضل لضمان الجودة، القابلية للتوسع، وسهولة الصيانة.

## الخلاصة:-

توفر البرمجة الكائنية التوجه OOP في PHP بنية قوية لإنشاء تطبيقات مرنة وقابلة للصيانة. من خلال استخدام الفئات، الكائنات، الوراثة، التجريد، التعددية الشكلية، والواجهات، يمكنك تطوير مشاريع احترافية، أسلوبًا أساسيًا لبناء تطبيقات حديثة قابلة للتطوير والصيانة. من خلال تنظيم الكود، تحسين الأمان، وإعادة استخدام الكود، تسهم OOP في جعل عملية التطوير أكثر احترافية وكفاءة. الدوال السحرية: تسهل التعامل مع الخصائص والدوال غير المعرفة. المتغيرات السحرية: تقدم معلومات مفيدة عن الكود أثناء التشغيل.

تم بحمد الله