بحث عن OOP في PHP

اعداد المهندسة :

إيفاء شرف علي الحمزي.

دكتور المادة :

إبراهيم الشامي.

البرمجة الكائنية التوجه (Object-Oriented Programming - OOP) :

هي نموذج برمجي يعتمد على فكرة تقسيم الكود إلى كائنات (Objects) تحتوي على بيانات) الخصائص (Properties - وسلوكيات) الدوال .(Methods - في PHP ، تدعم البرمجة الكائنية بشكل كامل منذ الإصدار ٥، مع تحسينات كبيرة في الإصدارات اللاحقة.

أساسيات البرمجة الكائنية في PHP :

البرمجة الكائنية تعتمد على عدة مفاهيم رئيسية:

1. الكائنات(Objects)

الكائن هو نسخة من الصنف(Class) ، ويمثل شيئًا يمكن التعامل معه برمجيًا.

2. الأصناف(Classes)

الصنف هو قالب أو مخطط يتم من خلاله إنشاء الكائنات.

3. الخصائص(Properties)

الخصائص هي المتغيرات الموجودة داخل الصنف، وتُمثل بيانات الكائن.

4. الدوال(Methods)

الدوال هي وظائف أو إجراءات يتم تنفيذها بواسطة الكائنات.

5. المحددات(Access Modifiers)

تحدد مستوى الوصول إلى الخصائص والدوال:

- `public`: يمكن الوصول إليه من أي مكان.
- `protected`: يمكن الوصول إليه من نفس الصنف أو الأصناف الموروثة.
- `private`: يمكن الوصول إليه فقط من داخل الصنف نفسه.

6. الوراثة(Inheritance)

تمكن صنفًا جديدًا من وراثة خصائص ودوال صنف آخر.

7. التعددية الشكلية(Polymorphism)

يمكن استخدام نفس الدالة أو الطريقة بأشكال مختلفة بناءً على السياق.

8. التغليف(Encapsulation)

يُستخدم لإخفاء البيانات وحمايتها من التعديل غير المصرح به.

تطبيق عملي: أمثلة توضيحية

إنشاء صنف وكائن :

```php
<?php
class Car {
    // خصائص
    public $brand;
    public $color;

    // دالة منشئ
    public function __construct($brand, $color) {
        $this->brand = $brand;
        $this->color = $color;
    }

    // دالة
    public function drive() {
        return "The $this->color $this->brand is driving!";
    }
}

// إنشاء كائن
$myCar = new Car("Toyota", "Red");
echo $myCar->drive(); // The Red Toyota is driving!
?>
```

الوراثة:

```php
<?php
// صنف رئيسي
class Animal {
    public $name;

    public function __construct($name) {
        $this->name = $name;
    }

    public function speak() {
        return "$this->name is making a sound.";
    }
}

// صنف فرعي
class Dog extends Animal {
    public function speak() {
        return "$this->name is barking.";
    }
}

$dog = new Dog("Buddy");
echo $dog->speak(); // Buddy is barking.
?>
```

التعددية الشكلية(Polymorphism) :

```php
<?php
class Shape {
    public function area() {
        return "Area is undefined.";
    }}
class Circle extends Shape {
    private $radius;

    public function __construct($radius) {
        $this->radius = $radius;
    }

    public function area() {
        return pi() * pow($this->radius, 2);
    }}
class Rectangle extends Shape {
    private $width, $height;
    public function __construct($width, $height) {
        $this->width = $width;
        $this->height = $height;
    }
    public function area() {
        return $this->width * $this->height;
    }}
$shapes = [
    new Circle(5),
    new Rectangle(4, 6),
];
foreach ($shapes as $shape) {
    echo $shape->area() . PHP_EOL;
}?>
```

التغليف(Encapsulation) :

```php
<?php
class BankAccount {
    private $balance;
    public function __construct($balance) {
        $this->balance = $balance; }
    public function deposit($amount) {
        $this->balance += $amount;}
    public function withdraw($amount) {
        if ($amount > $this->balance) {
            return "Insufficient balance.";}
        $this->balance -= $amount;
        return "Withdrawal successful.";}
    public function getBalance() {
        return $this->balance;}}
$account = new BankAccount(1000);
$account->deposit(500);
echo $account->getBalance(); // 1500
echo $account->withdraw(2000); // Insufficient balance.
?>
```

ميزات إضافية في PHP OOP

1.الواجهات(Interfaces)

تحدد مجموعة من الدوال التي يجب أن تحتويها أي صنف يطبق هذه الواجهة.

```php
<?php
interface Logger {
    public function log($message);
}
class FileLogger implements Logger {
    public function log($message) {
        echo "Logging to a file: $message";}}
$logger = new FileLogger();
$logger->log("Hello, World!");
?>
```

2.الأصناف المجردة(Abstract Classes)

تعمل كقالب عام، ولا يمكن إنشاء كائن مباشرة منها.

```php
<?php
abstract class Vehicle {
    abstract public function startEngine();
}
class Car extends Vehicle {
    public function startEngine() {
        echo "Car engine started.";}}
$car = new Car();
$car->startEngine(); // Car engine started.
?>
```

3.السمات(Traits)

تستخدم لإعادة استخدام الكود بين الأصناف دون الحاجة إلى الوراثة.

```php
<?php
trait Greet {
    public function sayHello() {
        return "Hello!";
    }}
class Person {
    use Greet;}
$person = new Person();
echo $person->sayHello(); // Hello!
?>
```

لبرمجة الكائنية التوجه (OOP) في PHP تتميز بعدة نقاط تجعلها مرنة وقوية، وتتيح للمطورين بناء تطبيقات ذات تنظيم عالٍ وقابلة للتوسع. فيما يلي أبرز ما يميز OOP في PHP:

1.سهولة التعلم والاستخدام

- لغة PHP توفر واجهات بسيطة ومرنة لتطبيق مبادئOOP ، ما يجعلها سهلة التعلم حتى للمبتدئين.
- يمكن الجمع بين البرمجة الكائنية والبرمجة الإجرائية في مشروع واحد.

2.التوافق مع معايير البرمجة الحديثة:

- PHPيدعم جميع مفاهيم البرمجة الكائنية مثل الوراثة(Inheritance) ، التغليف (Encapsulation)، التعددية الشكلية(Polymorphism) ، والواجهات.(Interfaces)
- يدعم PHP السمات (Traits) لإعادة استخدام الكود.

3.تعدد الميزات المتقدمة:

- **السمات (Traits):** تتيح إعادة استخدام الكود بين الأصناف دون الحاجة إلى الوراثة.
- **الأصناف المجردة (Abstract Classes):** يمكن استخدامها لتحديد تصميم عام يلزم الأصناف الفرعية بتطبيقه.
- **الواجهات (Interfaces):** تتيح الفصل بين تعريف السلوك وتنفيذه.

4.المرونة

- يدعم PHP البرمجة الكائنية بشكل كامل منذ الإصدار ٥، ما يتيح الجمع بين الأنماط البرمجية المختلفة.
- يمكن استخدام PHP لبناء تطبيقات صغيرة أو كبيرة بفضل دعمه للكائنات.

5.التكامل مع مكتبات وإطارات العمل

- معظم أطر العمل الشهيرة في PHP مثلLaravel ، Symfony، CodeIgniterتعتمد بشكل كبير على البرمجة الكائنية.
- تسهّل OOP في PHP بناء التطبيقات باستخدام هذه الأطر، مثل التطبيقات ذات البنية الطبقية (MVC).

6.دعم التعددية الشكلية(Polymorphism)

- يمكن إنشاء وظائف بنفس الاسم تؤدي سلوكيات مختلفة بناءً على الكائن الذي ينفذها، مما يسهل إضافة ميزات جديدة دون تعديل الكود الأساسي.

7.إعادة استخدام الكود

- الوراثة (Inheritance) والسمات (Traits) تسهم في إعادة استخدام الكود بين الأصناف.
- الكود يصبح أكثر تنظيمًا ومرونة وقابلية لإعادة الاستخدام.

8.دعم النطاقات(Namespaces)

- يوفر PHP ميزة **Namespaces** لتنظيم الكود ومنع تعارض الأسماء بين الأصناف أو الدوال، خاصة عند التعامل مع مكتبات أو مشاريع كبيرة.

9.التوافق مع معاييرPSR

- معايير PHP-FIG) مثل PSR-4 للتحميل التلقائي (تجعل الكود متوافقًا مع معايير الصناعة.
- تسهل هذه المعايير كتابة كود نظيف ومنظم باستخدامOOP

10.التحميل التلقائي(Autoloading)

- يدعم PHP ميزة التحميل التلقائي للأصناف باستخدام دوال مثل `spl_autoload_register()`، مما يقلل الحاجة إلى تضمين الملفات يدويًا.

11.دعم إدارة الأخطاء والاستثناءات:

- يدعم PHP التعامل مع **الاستثناءات (Exceptions)** فيOOP ، مما يتيح كتابة كود أكثر أمانًا واستقرارًا.

```php
<?php
class CustomException extends Exception {
    public function errorMessage() {
        return "Error on line {$this->getLine()} in {$this->getFile()}: {$this->getMessage()}";
    }}
try {
    throw new CustomException("Custom exception occurred!");
} catch (CustomException $e) {
    echo $e->errorMessage();}
?>
```

12.القابلية للتوسع:

- يمكن بناء نظام باستخدام OOP بحيث يكون قابلًا للتطوير والإضافة دون التأثير على الكود الأساسي.
- يتم تصميم الأنظمة بشكل موديولي (Modular) يسهل تعديله أو ترقيته

أمثلة على استخدامات مميزة لـ OOP فيPHP:

1.إنشاء تطبيقات على أساس التصميم الطبقي(MVC)

- **يتم استخدام OOP لتقسيم التطبيق إلى:**
  - ❖ **Model** : للتعامل مع البيانات وقواعد البيانات.
  - ❖ **View** : لتقديم الواجهة المرئية للمستخدم.
  - ❖ **Controller** : لإدارة تدفق البيانات بين الـ ModelوالـView

2.دمج البرمجة الكائنية مع قواعد البيانات:

- **باستخدام مكتبات مثل PDO، يمكن بناء أنظمة تعتمد على الكائنات للتعامل مع قواعد البيانات.**

3.إنشاء مكتبات أو حزم قابلة لإعادة الاستخدام:

- **باستخدامOOP ، يمكن إنشاء مكتبات مثل مكتبة لإرسال البريد الإلكتروني أو مكتبة للتعامل مع الملفات.**

مقارنه بسيطة للأنواع:

أساس.OOP إنشاء قوالب وكائنات	**ClassوObject**
أمان وتحكم أكبر بالبيانات. إخفاء التفاصيل	**Encapsulation**
يقلل التكرار ويتيح توسيع الوظائف. إعادة استخدام الكود	**Inheritance**
مرونة في التعامل مع الكائنات. نفس الدوال بسلوك مختلف	**Polymorphism**
تسهيل التصميم وإجبار الأصناف على الالتزام بواجهة معينة تجريد التفاصيل وفرض	**Abstraction**
دعم التعدد.(Multiple Inheritance) تعريف دوال للصنف	**Interfaces**
مرونة أكثر مقارنة بالوراثة. إعادة استخدام الكود في أصناف غير متعلقة	**Traits**

الخلاصة:

ما يميز OOP في PHP هو توازنها بين القوة والمرونة وسهولة الاستخدام. تجمع بين القدرة على بناء تطبيقات بسيطة وسريعة إلى بناء أنظمة معقدة وواسعة النطاق. تجعل OOP الكود أكثر تنظيمًا، وقابلية للإدارة، وإعادة الاستخدام، مما يسهم في تسريع تطوير البرمجيات وتحسين جودتها.

البحث الثاني عن الفرق بين الconst و ال static كود توضيحي:

const-1 :

- **تعريفها:**
  - تُستخدم لتعريف **ثوابت** (constants)في الكائنات.(classes)
  - لا يمكن تغيير قيمتها بعد تعريفها.
  - يجب أن تكون قيمتها معروفة أثناء وقت الترجمة (compile-time) وتقتصر على القيم السكالارية (مثل الأرقام أو النصوص).
- **الخصائص:**
  - مرتبطة مباشرة بالكلاس، وليست مرتبطة بأي كائن.(instance)
  - لا يمكن تعريفها باستخدام الكلمات المفتاحية `public`، `private`أو `protected`.
  - لا يمكن تعديل قيمتها أو إعادة تعريفها بعد تعيينها.
  - تُستخدم مباشرة باستخدام اسم الكلاس.(`ClassName::CONSTANT`)
- **مثال:**

```php
<?php
class MyClass {
    const MAX_LIMIT = 100; // تعريف ثابت

    public function showLimit() {
        echo self::MAX_LIMIT; // الوصول إلى الثابت داخل الكلاس
    }
}

echo MyClass::MAX_LIMIT; // الوصول إلى الثابت خارج الكلاس

```

static-2 :

- **تعريفها:**
  - تُستخدم لتعريف **الخصائص أو الوظائف (methods)** التي ترتبط بالكلاس وليس بالكائن.
  - يمكن تغيير قيمتها أثناء وقت التشغيل.
  - يتم مشاركة الخصائص أو القيم عبر جميع الكائنات من نفس الكلاس.
- **الخصائص:**
  - يمكن استخدامها مع `public`، `private` و `protected`لتحديد مستوى الوصول.
  - يتم الوصول إليها باستخدام اسم الكلاس.(`ClassName::staticProperty`)
  - يمكن استخدامها لتعريف دوال يتم استدعاؤها بدون إنشاء كائن (`ClassName::staticMethod`).
- **مثال:**

```php
<?php
class MyClass {
    public static $counter = 0; // تعريف خاصية static

    public static function incrementCounter() {
        self::$counter++;
    }
}

MyClass::incrementCounter(); // زيادة العدد
MyClass::incrementCounter();
echo MyClass::$counter; // 2 :طباعة
```

مقارنة رئيسية بين const و static:

static	const	الخاصية
لتحديد خصائص أو وظائف مرتبطة بالكلاس.	لتحديد ثوابت.	**الاستخدام**
يمكن تغيير قيم الخصائص أثناء وقت التشغيل.	ثابتة (لا يمكن تغييرها).	**التغيير**
`ClassName::$property`أو `ClassName::method()`	`ClassName::CONSTANT`	**الاستدعاء**
يمكن أن تتغير أثناء التشغيل.	يجب أن تكون قيمًا معروفة وقت الترجمة.	**القيم المسموح بها**
يمكن أن تكون وظائف أو خصائص متنوعة.	قيم ثابتة فقط مثل نصوص أو أرقام.	**المجال**

.