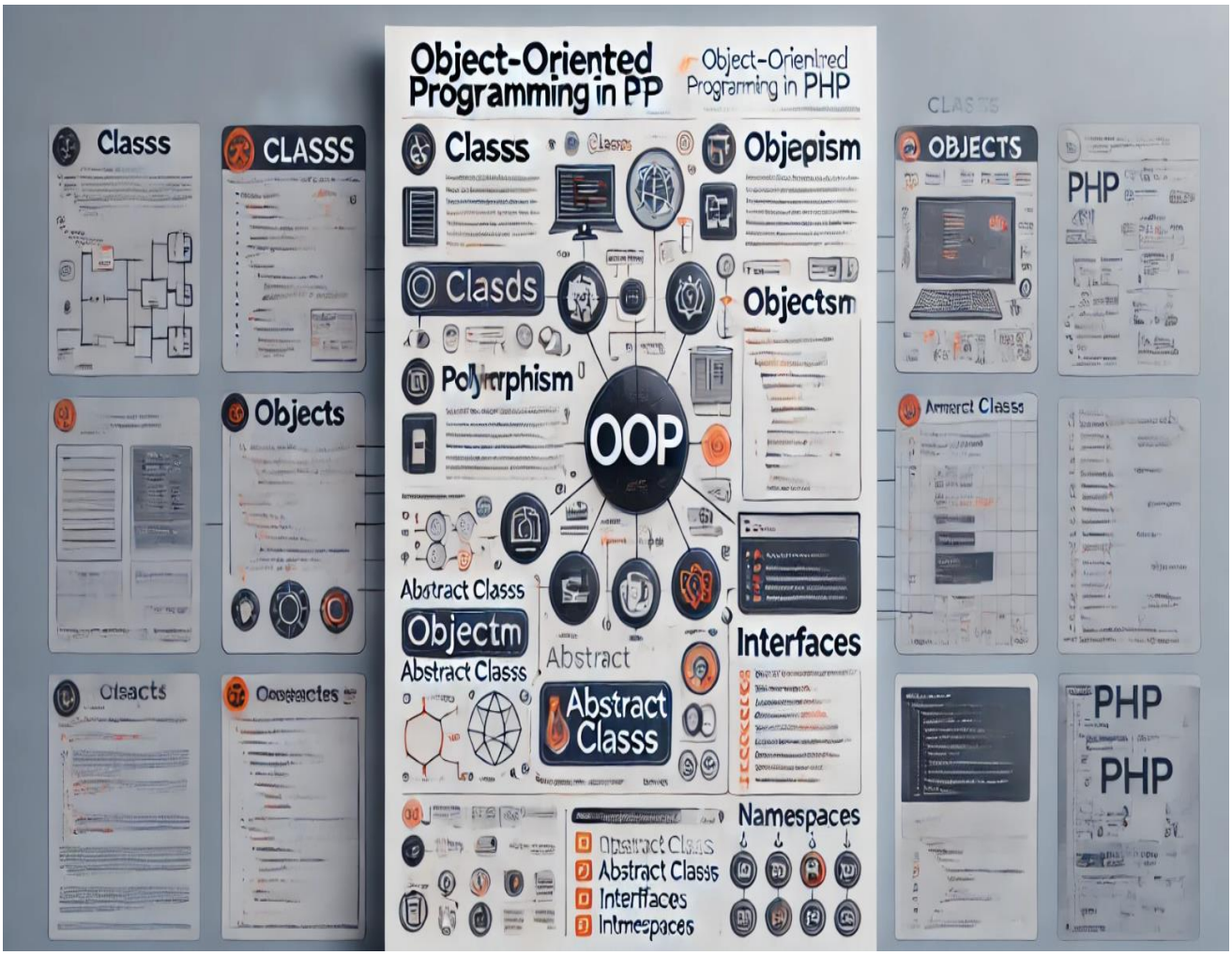
Republic of Yemen
Ministry of Higher Education
National University
Faculty of Science and
Engineering

الجمهورية اليمنية
وزارة التعليم العالي
الجامعة الوطنية
كلية العلوم والهندسة



المُهندسة:

ذكرى عادل الحبابي

الدكتور:

إبراهيم الشامي

❖ ما هي البرمجة الكائنية (OOP) في PHP ؟

OOP هي اختصار لـ Object-Oriented Programming(البرمجة الكائنية التوجه)

في البرمجة الإجرائية(Procedural Programming) ، يتم كتابة إجراءات أو وظائف تقوم بعمليات على البيانات. بينما في البرمجة الكائنية التوجه(OOP) ، يتم إنشاء كائنات تحتوي على كل من البيانات (الخصائص) والوظائف (الدوال).

❖ مزايا البرمجة الكائنية مقارنة بالبرمجة الإجرائية:

- **أسرع وأسهل في التنفيذ:**

البرمجة الكائنية تجعل تنفيذ الكود أكثر كفاءة وسرعة.

- **توفر هيكلًا واضحًا للبرامج:**

تُنظم الكود بطريقة تجعل قراءته وفهمه أسهل.

- **تساعد على الالتزام بمبدأ(DRY (Don't Repeat Yourself :**

تُقلل تكرار الكود مما يجعل الصيانة والتعديل وتصحيح الأخطاء أسهل.

- **إعادة الاستخدام:**

تجعل من الممكن إنشاء تطبيقات قابلة لإعادة الاستخدام بالكامل باستخدام كود أقل ووقت تطوير أقصر.

❖ عيوب البرمجة الكائنية:

- **التعقيد:**

قد يكون تصميم الأصناف والكائنات معقدًا مقارنة بالبرمجة الإجرائية

- **استهلاك الموارد:**

قد تكون البرامج الكائنية أبطأ في الأداء بسبب إنشاء الكائنات والتفاعل بينها

- **الحاجة الى التخطيط المُسبق:**

يتطلب تصميم البرنامج الكائني الى تخطيط لضمان فعاليته

❖ المفاهيم الأساسية في OOP:

1. الفئات (Classes):

الفئة هي قالب أو تصميم لإنشاء الكائنات وتحديد خصائصها وسلوكياتها.
يتم تعريف الفئة باستخدام الكلمة المحجوزة class، متبوعة باسم الفئة وزوج من الأقواس المتعرجة .({ }) يتم وضع جميع خصائصها وطرقها داخل الأقواس.

```php
class Person {
    public $name;
    public $age;

    public function introduce() {
        echo "Hi, my name is $this->name and I am $this->age years old.";
    }
}
```

2. الكائنات (Objects):

الكائن هو وحدة أساسية تحتوي على البيانات والوظائف والدوال المرتبطة بها الفئات لا قيمة لها بدون الكائنات! يمكننا إنشاء عدة كائنات من فئة واحدة. يحتوي كل كائن على جميع الخصائص والوظائف المعرفة في الفئة، ولكن كل كائن سيحتوي على قيم مختلفة للخصائص. يتم انشاء الكائناتnew عن طريق الكلمة المحجوزة

```php
class Car {
    public $color;

    public function drive() {
        echo "The car is driving";
    }
}

$myCar = new Car();
$myCar->color = "red";
$myCar->drive();
```

3. الخصائص(Properties):

الخصائص هي البيانات أو الصفات التي يمتلكها الكائن. يتم تعريفها داخل الصنف

```php
class Student {
    public $name;
    public $age;
}
```

4. الدوال (Methods):

الدوال هي السلوكيات أو العمليات التي يمكن للكائن تنفيذها. يتم تعريفها داخل الصنف

```php
class Lamp {
    public function turnOn() {
        echo "المصباح مضاء";
    }
}
```

5. المُنشىء (Constructor):

هو دالة خاصة تُستدعى تلقائيًا عند إنشاء كائن جديد من الصنف. تُستخدم لتعيين القيم الابتدائية للخصائص

```php
class Book {
    public $title;

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

$book = new Book("PHP Basics");
echo $book->title; // طباعة: PHP Basics
```

6. الوراثة(Inheritance) :

الوراثة تسمح لفئة جديدة باستخدام خصائص ودوال فئة موجودة.
يتم تعريف الفئة الموروثة باستخدام الكلمة المحجوزة **extends**

```php
class Animal {
    public $name;

    public function eat() {
        echo "$this->name is eating.";
    }
}

class Dog extends Animal {
    public function bark() {
        echo "$this->name is barking.";
    }
}

$dog = new Dog();
$dog->name = "Buddy";
$dog->eat();
$dog->bark();
```

7. التغليف(Encapsulation) :

هو إخفاء التفاصيل الداخلية للكائن باستخدام المحددات Privateو Publicو Protected.

```php
class BankAccount {
    private $balance;

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

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

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

$account = new BankAccount(1000);
$account->deposit(500);
echo $account->getBalance();
```

8. تعدد الأشكال(Polymorphism) :

يسمح باستخدام نفس الدالة بأشكال مختلفة بناءً على السياق.

```php
class Shape {
    public function draw() {
        echo "Drawing a shape.";
    }
}

class Circle extends Shape {
    public function draw() {
        echo "Drawing a circle.";
    }
}

$shape = new Shape();
$circle = new Circle();

$shape->draw(); // Drawing a shape.
$circle->draw(); // Drawing a circle.
```

❖ **محددات الوصول(Access Modifiers):**

- **Public: يمكن الوصول إليه من أي مكان.**
- **Protected: يمكن الوصول إليه من نفس الفئة والفئات الموروثة.**
- **Private: يمكن الوصول إليه فقط من داخل نفس الفئة.**

```php
class Example {
    public $publicVar = "Public";
    protected $protectedVar = "Protected";
    private $privateVar = "Private";

    public function showVariables() {
        echo $this->publicVar; // متاح
        echo $this->protectedVar; // متاح
        echo $this->privateVar; // متاح
    }
}

class ChildExample extends Example {
    public function accessVariables() {
        echo $this->publicVar; // متاح
        echo $this->protectedVar; // متاح
        // echo $this->privateVar; // غير متاح
    }
}
```

❖ **الدوال السحرية(magic methods):**

في PHP ، الدوال السحرية (Magic Methods) هي مجموعة من الدوال الخاصة التي يتم استدعاؤها تلقائيًا في مواقف معينة أثناء تشغيل البرنامج. يتميز كل من هذه الدوال بوجود أسماء تبدأ بعلامتين مائلتين __(الشرطتين السابقتين)، وهذه الدوال لا يتم استدعاؤها مباشرة بواسطة المطور، بل يتم تفعيلها بشكل تلقائي بناءً على العمليات التي تتم على الكائنات أو الصفوف.

بعض الدوال السحرية الشهيرة في PHP :

1. __construct()

يتم استدعاؤها تلقائيًا عند إنشاء كائن جديد من فئة. تُستخدم عادةً لتهيئة الكائن.

```php
class MyClass {
    public function __construct($name) {
        echo "Welcome, $name!";
    }
}

$obj = new MyClass("Ali"); // الإخراج: Welcome, Ali!
```

2. __destruct()

تُستدعى عندما يتم تدمير الكائن (عند انتهاء مدة حياته).
تستخدم لتنظيف الموارد أو إغلاق الاتصالات.

```php
class MyClass {
    public function __construct() {
        echo "Object created!\n";
    }

    public function __destruct() {
        echo "Object destroyed!";
    }
}

$obj = new MyClass(); // الإخراج: Object created!
// عند نهاية السكربت: Object destroyed!
```

3. __call()

تُستدعى عندما يتم استدعاء دالة غير موجودة أو غير معرفة في الكائن.
تسمح بتنفيذ أكواد بديلة أو تحويلات عند محاولة استدعاء دالة مفقودة.

```php
class MyClass {
    public function __call($name, $arguments) {
        echo "Method $name does not exist. Arguments: " . implode(", ", $arguments);
    }
}

$obj = new MyClass();
$obj->nonExistentMethod("arg1", "arg2"); // الإخراج: Method nonExistentMethod does not exist. Arguments: arg1, arg2
```

4. __get()

تُستدعى عند محاولة الوصول إلى خاصية غير موجودة في الكائن.
تستخدم للقراءة من الخصائص غير الموجودة.

```php
class MyClass {
    private $data = [];

    public function __get($name) {
        return $this->data[$name] ?? "Property $name does not exist";
    }

    public function __set($name, $value) {
        $this->data[$name] = $value;
    }
}

$obj = new MyClass();
$obj->name = "Ali";
echo $obj->name; // الإخراج: Ali
echo $obj->age;  // الإخراج: Property age does not exist
```

5. __set()

تُستدعى عند محاولة تعيين قيمة إلى خاصية غير موجودة في الكائن.
تستخدم لضبط قيم الخصائص غير الموجودة.

```php
<?
class MyClass {
    private $data = []; // مكان لتخزين القيم

    // تنفيذ __set عند محاولة تعيين قيمة لخاصية غير معرفة
    public function __set($name, $value) {
        echo "Setting '$name' to '$value'.\n";
        $this->data[$name] = $value;
    }

    // تنفيذ __get لاسترجاع القيم
    public function __get($name) {
        return $this->data[$name] ?? "Property '$name' is not set.";
    }
}

$obj = new MyClass();

// تعيين قيم للخصائص غير المعرفة
$obj->name = "Ali"; // Setting 'name' to 'Ali'.
$obj->age = 25;     // Setting 'age' to '25'.

// طباعة القيم
echo "Name: " . $obj->name . "\n"; // Name: Ali
echo "Age: " . $obj->age . "\n";   // Age: 25
```

6. __isset()

تُستدعى عند استخدام isset() أو empty() على خاصية غير موجودة.

```php
class MyClass {
    private $data = [];

    public function __isset($name) {
        return isset($this->data[$name]);
    }

    public function __set($name, $value) {
        $this->data[$name] = $value;
    }
}

$obj = new MyClass();
$obj->name = "Ali";
var_dump(isset($obj->name)); // الإخراج: bool(true)
```

7. __unset()

تُستدعى عند محاولة استخدام unset() على خاصية غير موجودة

```php
class MyClass {
    private $data = [];

    public function __set($name, $value) {
        $this->data[$name] = $value;
    }

    public function __get($name) {
        return $this->data[$name] ?? null;
    }

    public function __unset($name) {
        echo "Unsetting '$name'.\n";
        unset($this->data[$name]);
    }
}

$obj = new MyClass();

// تعيين قيم
$obj->name = "Ali";
$obj->age = 30;

// حذف خاصية
unset($obj->name); // Unsetting 'name'.

// محاولة الوصول إلى الخاصية المحذوفة
echo $obj->name; // لا يظهر أي شيء لأنها حذفت
```

8. __toString()

تُستدعى عندما يتم تحويل الكائن إلى سلسلة نصية، مثل استخدامه مع echo أو print

```php
class MyClass {
    public function __toString() {
        return "This is a string representation of the object.";
    }
}

$obj = new MyClass();
echo $obj; // الإخراج: This is a string representation of the object.
```

9. __invoke()

تُستدعى عندما يتم استدعاء الكائن كما لو كان دالة.

```php
class MyClass {
    public function __invoke($param) {
        echo "Object called with parameter: $param";
    }
}

$obj = new MyClass();
$obj("Hello!"); // الإخراج: Object called with parameter: Hello!
```

10. __clone()

تُستدعى عند محاولة نسخ الكائن باستخدام clone

```php
class MyClass {
    public $name;

    public function __clone() {
        $this->name = "Cloned: " . $this->name;
    }
}

$obj1 = new MyClass();
$obj1->name = "Ali";
$obj2 = clone $obj1;

echo $obj2->name; // الإخراج: Cloned: Ali
```

❖ المتغيرات السحرية(magic variables):

هي متغيرات تُعرف مسبقًا داخل اللغة وتوفر معلومات ديناميكية بناءً على السياق الذي يتم استخدامها فيه. تبدأ أسماء هذه المتغيرات بشرطتين سفليتين (__) ويتم استخدامها بشكل واسع لتسهيل تتبع الأكواد أو الحصول على معلومات عن البيئة.
هذه المتغيرات السحرية تكون مفيدة جدًا أثناء التصحيح (debugging) وتسجيل الأخطاء (logging).

1. __LINE__

تعطي رقم السطر الحالي في الملف الذي يتم تنفيذ الكود فيه.

```php
// 1. __LINE__
echo "This is line number " . __LINE__;
```

2. __FILE__

تعطي المسار الكامل والاسم للملف الحالي.

```php
// 2. __FILE__
echo "This file is located at " . __FILE__;
```

3. __DIR__

تعطي مسار الدليل الذي يحتوي على الملف الحالي.

```php
// 3. __DIR__
echo "This directory is " . __DIR__;
```

4. __FUNCTION__

تعطي اسم الدالة الحالية التي يتم تنفيذ الكود بداخلها.

```php
// 4. __FUNCTION__
function myFunction() {
    echo "This function is called " . __FUNCTION__;
}
myFunction();
```

5. __CLASS__

تعطي اسم الصف (class) الذي يتم تنفيذ الكود بداخله.

```php
// 5. __CLASS__
class MyClass {
    public function displayClass() {
        echo "This class is " . __CLASS__;
    }
}
$obj = new MyClass();
$obj->displayClass();
```

6. __TRAIT__

تعطي اسم الـ Trait المستخدم داخل الصف (إذا كان موجودًا).

```php
// 6. __TRAIT__
trait MyTrait {
    public function showTrait() {
        echo "This trait is ".__TRAIT__;
    }
}
class MyClass {
    use MyTrait;
}
$obj = new MyClass();
$obj->showTrait();
```

7. __METHOD__

تعطي اسم الدالة مع اسم الصف إذا كان الكود ينفذ داخل صف.

```php
// 7. __METHOD__
class MyClass {
    public function myMethod() {
        echo "This method is " . __METHOD__;
    }
}
$obj = new MyClass();
$obj->myMethod();
```

8. __NAMESPACE__

تعطي اسم الـ Namespace الذي ينتمي له الكود الحالي.

```php
// 8. __NAMESPACE__
namespace MyNamespace;

echo "This namespace is " . __NAMESPACE__;
```

❖ ثوابت الصف(Class Constants):

يمكن أن تكون مفيدة إذا كنت بحاجة إلى تعريف بيانات ثابتة داخل الصف، يتم الإعلان عن ثابت الصف باستخدام الكلمة المفتاحية const داخل الصف. لا يمكن تغيير قيمة الثابت بعد الإعلان عنه.

ثوابت الصف حساسة لحالة الأحرف .(case-sensitive) ومع ذلك، يُوصى دائمًا بتسمية الثوابت باستخدام أحرف كبيرة بالكامل.(uppercase letters)

يمكننا الوصول إلى الثابت من خارج الصف عن طريق استخدام اسم الصف متبوعًا بعامل النطاق (::) ثم اسم الثابت

```php
class MyClass {
    const MY_CONSTANT = "Hello, World!";
}

// Access the constant
echo MyClass::MY_CONSTANT;
```

أو يمكننا الوصول إلى ثابت من داخل الصف باستخدام الكلمة المفتاحية self متبوعة بعامل النطاق (::) ثم اسم الثابت، كما في المثال التالي:

```php
class MyClass {
    const GREETING = "مرحبًا بالعالم!";

    public function showGreeting() {
        echo self::GREETING;
    }
}

$obj = new MyClass();
$obj->showGreeting(); // يطبع: مرحبًا بالعالم!
```

❖ **الصفوف المجردة (Abstract Classes):**

الصفوف والدوال المجردة هي عندما يحتوي الصف الأب على دالة مُسماة، ولكنه يحتاج إلى الصف (أو الصفوف) الأبناء لتنفيذ المهام المطلوبة.
الصف المجرد هو صف يحتوي على دالة مجردة واحدة على الأقل. الدالة المجردة هي دالة يتم الإعلان عنها، ولكنها لا تحتوي على تنفيذ في الكود.
يتم تعريف الصف أو الدالة المجردة باستخدام الكلمة المفتاحية abstract

```php
abstract class ParentClass {
    abstract public function someMethod1();
    abstract public function someMethod2($name, $color);
    abstract public function someMethod3() : string;
}
```

عند الوراثة من صف مجرد، يجب تعريف دالة الصف الابن بنفس الاسم وبنفس مستوى أو مستوى أقل تقييدًا من مُحدد الوصول (Access Modifier). على سبيل المثال، إذا كانت الدالة المجردة معرفة كمحمية(protected) ، فيجب أن تُعرّف دالة الصف الابن كمحمية (protected)أو عامة(public) ، ولكن ليس خاصة (private) كذلك، يجب أن يكون نوع وعدد المعاملات المطلوبة (Required Arguments) هو نفسه، ومع ذلك، يمكن أن يحتوي الصف الابن على معاملات اختيارية إضافية.

❖ **القواعد عند وراثة صف ابن من صف مجرد:**

1. **يجب تعريف دالة الصف الابن بنفس الاسم، بحيث تعيد تعريف الدالة المجردة في الصف الأب.**
2. **يجب أن تكون دالة الصف الابن بنفس مستوى أو بمستوى أقل تقييدًا من مُحدد الوصول الخاص بالدالة المجردة في الصف الأب.**
3. **جب أن يكون عدد المعاملات المطلوبة هو نفسه تمامًا.**

مثال عملي:

```php
abstract class Shape {
    // تعريف دالة مجردة بمحدد الوصول protected
    abstract protected function calculateArea($length, $width);
}

class Rectangle extends Shape {
    // إعادة تعريف الدالة بنفس الاسم ونفس أو مستوى أقل تقييدًا (public هنا)
    public function calculateArea($length, $width, $unit = "cm") {
        $area = $length * $width;
        echo "مساحة المستطيل: $area $unit²\n";
    }
}

class Square extends Shape {
    // إعادة تعريف الدالة بنفس الاسم ومحدد الوصول
    protected function calculateArea($side, $unused = null) {
        $area = $side * $side;
        echo "مساحة المربع: $area cm²\n";
    }
}

// إنشاء كائن من الصف Rectangle
$rect = new Rectangle();
$rect->calculateArea(5, 10); // يطبع: مساحة المستطيل: 50 cm²

// إنشاء كائن من الصف Square
$sq = new Square();
$sq->calculateArea(4); // يطبع: مساحة المربع: 16 cm²
```

❖ الواجهات (Interfaces) :

الواجهات تتيح لك تحديد الدوال التي يجب أن يقوم الصف بتنفيذها.

الواجهات تجعل من السهل استخدام عدة صفوف مختلفة بنفس الطريقة. عندما تستخدم صفوف متعددة نفس الواجهة، يُطلق على ذلك اسم "التعددية الشكلية(Polymorphism) "

يتم تعريف الواجهات باستخدام الكلمة المفتاحية interface

لتنفيذ واجهة، يجب على الصف استخدام الكلمة المفتاحية implements.

```php
// تعريف الواجهة
interface Animal {
    public function makeSound();
    public function eat();
}

class Dog implements Animal {
    public function makeSound() {
        echo "الكلب ينبح: هو هو!\n";
    }

    public function eat() {
        echo "الكلب يأكل.\n";
    }
}

class Cat implements Animal {
    public function makeSound() {
        echo "القطة تموء: مياو!\n";
    }

    public function eat() {
        echo "القطة تأكل.\n";
    }
}

$dog = new Dog();
$cat = new Cat();

$dog->makeSound();  // يطبع: الكلب ينبح: هو هو!
$dog->eat();        // يطبع: الكلب يأكل.

$cat->makeSound();  // يطبع: القطة تموء: مياو!
$cat->eat();        // يطبع: القطة تأكل.
```

❖ الترايت(Traits) :

PHPيدعم الوراثة الفردية فقط أي أن الصف الابن يمكنه الوراثة فقط من صف أب واحد.

فماذا لو كان الصف بحاجة إلى وراثة سلوكيات متعددة؟ هنا تأتي الترايت (Traits) لحل هذه المشكلة في البرمجة الكائنية

الترايت تستخدم لإعلان دوال يمكن استخدامها في عدة صفوف. يمكن أن تحتوي الترايت على دوال ودوال مجردة يمكن استخدامها في صفوف متعددة، كما أن الدوال يمكن أن تحتوي على أي محدد وصول (عام، خاص، أو محمي).

يتم تعريف الترايت باستخدام الكلمة المفتاحية trait

```php
// تعريف الترايت
trait Logger {
    public function log($message) {
        echo "السجل: " . $message . "\n";
    }
}

trait Database {
    public function connect() {
        echo "تم الاتصال بقاعدة البيانات.\n";
    }
}

// الصف الذي يستخدم الترايت
class User {
    use Logger, Database; // استخدام الترايتين Logger و Database

    public function createUser($name) {
        $this->log("إنشاء المستخدم: $name");
        $this->connect();
        echo "تم إنشاء المستخدم $name بنجاح.\n";
    }
}

// إنشاء كائن من الصف User واستخدام الترايت
$user = new User();
$user->createUser("أحمد");
```

❖ **الدوال الثابتة(Static Methods) :**

يمكن استدعاء الدوال الساكنة مباشرةً ـ دون الحاجة إلى إنشاء مثيل (كائن) من الصف أولاً.

يتم تعريف الدوال الساكنة باستخدام الكلمة المفتاحية static

للوصول إلى دالة ثابتة(Static Method) ، يتم استخدام اسم الصف متبوعًا بعامل النطاق المزدوج (::)، ثم اسم الدالة

يمكن أن يحتوي الصف على دوال ساكنة (Static) ودوال غير ساكنة (Non-Static) معًا. يمكن الوصول إلى الدالة الساكنة من داخل دالة في نفس الصف باستخدام الكلمة المفتاحية self متبوعة بعامل النطاق المزدوج:(::)

```php
class Calculator {
    // تعريف دالة ساكنة
    public static function add($a, $b) {
        return $a + $b;
    }

    // تعريف دالة غير ساكنة
    public function calculateSum($x, $y) {
        // استدعاء الدالة الساكنة باستخدام self
        $sum = self::add($x, $y);
        echo "مجموع $x و $y هو: $sum\n";
    }
}

// استدعاء الدالة الساكنة مباشرة باستخدام اسم الصف
echo "المجموع (دالة ساكنة): " . Calculator::add(5, 10) . "\n";

// استدعاء الدالة غير الساكنة التي تستدعي الدالة الساكنة
$calc = new Calculator();
$calc->calculateSum(15, 20);
```

❖ **الخصائص الساكنة(Static Properties)**

يمكن استدعاء الخصائص الساكنة مباشرةً ـ دون الحاجة إلى إنشاء مثيل (كائن) من الصف.

يتم تعريف الخصائص الساكنة باستخدام الكلمة المفتاحية static

للوصول إلى خاصية ساكنة(Static Property) ، يتم استخدام اسم الصف متبوعًا بعامل النطاق المزدوج (::)، ثم اسم الخاصية:

يمكن أن يحتوي الصف على خصائص ساكنة (Static) وغير ساكنة (Non-Static) معًا. يمكن الوصول إلى الخاصية الساكنة من داخل دالة في نفس الصف باستخدام الكلمة المفتاحية self متبوعة بعامل النطاق المزدوج:(::)

```php
class Counter {
    // تعريف خاصية ساكنة
    public static $count = 0;

    // تعريف خاصية غير ساكنة
    public $name;

    public static function increment() {
        self::$count++;
    }

    // دالة غير ساكنة
    public function setName($name) {
        $this->name = $name;
        self::increment();
    }

    public function display() {
        echo "الاسم: {$this->name}, عدد الكائنات: " . self::$count . "\n";
    }
}

$obj1 = new Counter();
$obj1->setName("الكائن الأول");
$obj1->display();

$obj2 = new Counter();
$obj2->setName("الكائن الثاني");
$obj2->display();

echo "(عبر الخاصية الساكنة مباشرة) عدد الكائنات: " . Counter::$count . "\n";
```

❖ المساحات الاسمية(Namespaces)

المساحات الاسمية هي محددات تُستخدم لحل مشكلتين مختلفتين:

1. **تتيح تنظيمًا أفضل من خلال تجميع الصفوف التي تعمل معًا لأداء مهمة معينة.**
2. **تسمح باستخدام نفس الاسم لأكثر من صف واحد.**

يتم تعريف المساحات الاسمية (Namespaces) في بداية الملف باستخدام الكلمة المفتاحية namespace

يمكن إنشاء كائنات من الصفوف التابعة لمساحة الاسم دون الحاجة إلى أي محددات للوصول إلى الصفوف من خارج مساحة الاسم، يجب إرفاق اسم مساحة الاسم بالصف.

قد يكون من المفيد إعطاء مساحة اسم أو صف اسمًا مستعارًا لتسهيل الكتابة يتم ذلك باستخدام الكلمة المفتاحية use

```php
// تعريف مساحة الاسم الأولى Html
namespace Html;

class Table {
    public function draw() {
        echo "رسم جدول HTML.\n";
    }
}
class Row {
    public function draw() {
        echo "رسم صف في جدول HTML.\n";
    }
}

// تعريف مساحة الاسم الثانية Furniture
namespace Furniture;

class Table {
    public function build() {
        echo "بناء طاولة أثاث.\n";
    }
}
class Chair {
    public function build() {
        echo "بناء كرسي.\n";
    }
}
```

```php
namespace Main;

use Html\Table as HtmlTable;
use Furniture\Table as FurnitureTable;

$htmlTable = new HtmlTable(); // إنشاء كائن من جدول HTML
$htmlTable->draw();

$furnitureTable = new FurnitureTable(); // إنشاء كائن من طاولة الأثاث
$furnitureTable->build();
```

❖ القابلية للتكرار(Iterable):

القابلية للتكرار هي أي قيمة يمكن التكرار عبرها باستخدام حلقة .foreach()
تم تقديم النوع الزائف iterable في PHP 7.1 ، ويمكن استخدامه كنوع بيانات لوسائط الدوال وقيم العودة من الدوال.
يمكن استخدام الكلمة المفتاحية iterable كنمط بيانات لوسيط دالة أو كنمط عودة لدالة.

```php
// دالة تأخذ iterable كوسيط وتطبع كل عنصر في المجموعة
function printItems(iterable $items) {
    foreach ($items as $item) {
        echo $item . "\n";
    }
}

// دالة ترجع iterable
function getNumbers(): iterable {
    return [1, 2, 3, 4, 5];
}

// استخدام الدالة التي تأخذ iterable
$numbers = getNumbers();
printItems($numbers);
```

❖ خاتمة:

في ختام هذا البحث، نجد أن البرمجة الكائنية (OOP)في PHP توفر طريقة مرنة وقوية لتطوير التطبيقات المعقدة. من خلال استخدام مفاهيم مثل الصفوف، الكائنات، الوراثة، التجريد، التعددية، والواجهات، يمكن للمطورين تنظيم الكود بشكل أكثر فعالية، مما يسهل صيانته وتطويره.

ما تتيح PHP للمطورين استخدام المساحات الاسمية (Namespaces)لتنظيم الكود بشكل أفضل، وحل مشكلات تعارض الأسماء بين الصفوف المختلفة. وتقنيات مثل الخصائص الساكنة والدوال الساكنة توفر مزيدًا من المرونة عند التعامل مع بيانات ثابتة في الصفوف.

بالإضافة إلى ذلك، تعتبر الترايتات (Traits)وسيلة رائعة لإعادة استخدام الكود عبر الصفوف المختلفة دون الحاجة للوراثة المتعددة، مما يساهم في تحسين بنية التطبيق. كما أن الأنواع القابلة للتكرار (Iterable)تقدم إمكانيات قوية في التعامل مع المجموعات من خلال دوال مرنة.

بناءً على ما تم استعراضه في هذا البحث، يظهر أن البرمجة الكائنية في PHP لا تقتصر على تنظيم الكود فحسب، بل توفر أيضًا تقنيات قوية تساهم في كتابة تطبيقات قابلة للتوسع والصيانة بسهولة. إن فهم هذه المبادئ واستخدامها بالشكل الأمثل يعتبر خطوة مهمة لكل مطور يعمل في بيئةPHP ، حيث تساهم في تحسين جودة الكود وتسهيل تطوير البرمجيات على المدى الطويل.