OOP Summary

## PHP What is OOP?

- OOP stands for Object-Oriented Programming.

- Object-oriented programming is about creating objects that contain both data and functions.

## Why do I use OOP programming instead of procedural programming?

* OOP is faster and easier to execute.
* OOP provides a clear structure for the programs.
* OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug.
* OOP makes it possible to create full reusable applications with less code and shorter development time.

## **What is OOP Components?**

**1- Encapsulation ==> Hide Data**

## **2-** Abstraction ==> What Object Dose ?

## **3-** Inheritance ==> More reusable

## 4- **[Polymorphism](https://www.w3schools.com/python/python_polymorphism.asp" \t "https://www.w3schools.com/php/_blank)** ==> Flexibility

**1-Encapsulation :- (Setter , Getter)**

**Encapsulation is to hide the data and complexity of the class from the outside world, exposing only what is necessary through a public interface**

* **Key points about encapsulation:**

**- Data Hiding**

**- Public Interface**

**- Improved Maintainability**

**- Enhanced Security**

class employee{

           private string $empName;

             public function \_\_constractor( string $name){

                $this->$empName = $name;

             }

             //setter

             public function setName($name) :void{

                $this->empName = $name;

             }

            //getter

           public function getName() :string{

            return $this->empName;

           }

           public function showEmployee(){

            echo "Employee Id = " . $this->empId ."<br>" ."Employee Name = " . $this->empName ."<br>" ."Employee salary = " . $this->empSalary ;

           }

       }

       $emp = new employee(1 , "Waad" ,12000000);

        $emp->showEmployee();

**2-Abstraction :- ( What the object Dose?)**

- abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.

- Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.

* **Key points about encapsulation:**

-I can't take an object from abstract class

-Simplification

-Improved Design

-Single Inheritance

-Hides necessary Details

abstract class shape {

        protected string $color;

        public function \_\_construct(string $color) {

            $this->color = $color;

        }

        public function getColor() : string {

            return $this->color;

        }

        public function setColor(string $color) : void {

            $this->color = $color;

        }

        public abstract function getArea(): float;

        public abstract function getParimeter(): float;

        public abstract function display(): void;

    }

    class rectangle extends shape {

        private float $length;

        private float $width;

        public function \_\_construct(float $length, float $width, string $color) {

            parent::\_\_construct($color);

            $this->length = $length;

            $this->width = $width;

        }

        public function setLength(float $length): void {

            $this->length = $length;

        }

        public function setWidth(float $width): void {

            $this->width = $width;

        }

        public function getLength(): float {

            return $this->length;

        }

        public function getWidth(): float {

            return $this->width;

        }

        public function getArea(): float {

            return $this->length \* $this->width;

        }

        public function getParimeter(): float {

            return ($this->length + $this->width) \* 2;

        }

        public function display(): void {

            echo "rectangle color is : " . $this->getColor() . "<br>" ."rectangle Area is : " . $this->getArea() ."<br>" ."rectangle perimeter is : " . $this->getParimeter();

        }

    }

    $rect = new rectangle(10.5, 15.5, "red");

$rect->display();

**3-Inheritance :-**

- The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

- An inherited class is defined by using the extends keyword.

* **Key points about encapsulation:**

- Code Reusability

- Hierarchy

- Extensibility

class Person {

    protected string $name ;

    protected string $address;

    public function \_\_construct(string $name,string $address){

        $this->name = $name;

        $this->address = $address;

    }

    public function getName(): string{

        return $this->name;

    }

    public function setName(string $name): void{

        $this->name = $name;

    }

    public function getAddress(): string{

        return $this->address;

    }

    public function setAddress(string $address):void{

        $this->address = $address;

    }

    public function display():void{

        echo "Person "."<br>". "name :". $this->getName() ."<br>"."address :".$this->getAddress() ;

    }

}

$person = new Person("Yara","Cairo,Eygpt");

$person->display();

echo "<br>";

class Student extends Person{

    private string $program ;

    private int $year;

private float $fee;

    public function \_\_construct(string $name,string $address,string $program,int $year

,float $fee){

        parent::\_\_construct($name,$address);

        $this->program = $program;

        $this->year = $year;

        $this->fee = $fee;

    }

    public function getProgram(): string{

        return $this->program ;

    }

    public function setProgram(string $program):void{

        $this->program = $program;

    }

    public function getYear(): int{

        return $this->year;

    }

    public function setYear(int $year):void{

        $this->year = $year;

    }

    public function getFee():float{

        return $this->fee;

    }

    public function setFee(float $fee):void{

        $this->fee = $fee;

    }

    public function display():void{

        echo "Student "."<br>". "name :". $this->getName() ."<br>"."address :".$this->getAddress() ."<br>"."program :".$this->getProgram()."<br>"."year :".$this->getYear()."<br>"."fee :".$this->getFee();

    }

}

$std = new Student("waad","Cairo,Eygpt","Computer Science",2024,15000.00);

$std->display();

4-**[Polymorphism](https://www.w3schools.com/python/python_polymorphism.asp" \t "https://www.w3schools.com/php/_blank)** :- (Over ride , Over load)

- allows objects of different classes to be treated as objects of a common superclass. It enables the same method or function to behave differently based on the object it is acting upon.

* **Key points about encapsulation:**

- Method Overloading :-

-(Static Polymorphism / Compile-time Polymorphism/Early Binding).

-Method overloading allows multiple methods with the same name but different parameters within the same class.

class Calculator {

            public function calculate($x, $y) {

                return $x + $y;

            }

            public function calculate($x, $y, $z) {

                return $x + $y + $z;

            }

        }

- Method Overriding :-

- (Dynamic Polymorphism / Runtime Polymorphism/Late Binding).

- Method overriding occurs when a subclass provides a specific implementation of a method that is already defined in its superclass.

    class Shape {

            public function draw() {

                echo "Drawing a shape";

            }

        }

        class Circle extends Shape {

            public function draw() {

                echo "Drawing a circle";

            }

        }

1. **Interface :-**

- set of methods that a class must implement. It provides a way to achieve abstraction and multiple inheritance, allowing different classes to share a common set of methods without dictating how these methods are implemented.

-Interfaces are declared with the interface keyword:

* **Key points about encapsulation:**

- Method Declarations Only

- Multiple Inheritance

 interface Crud {

            public function create();

            public function getAll();

            public function getOne(int $id);

            public function update(int $id);

            public function delete(int $id);

            public function edit(int $id);

        }

        interface X {}

        class Post implements Crud, X {

            private $posts = [];

            public function create() {

                $this->posts[] = ['id' => count($this->posts) + 1, 'content' => 'New post content'];

                echo "Post created.\n";

            }

            public function getAll() {

                return $this->posts;

            }

            public function getOne(int $id) {

                for ($i = 0; $i < count($this->posts); $i++) {

                    if ($this->posts[$i]['id'] === $id) {

                        return $this->posts[$i];

                    }

                }

            }

            public function update(int $id) {

                for ($i = 0; $i < count($this->posts); $i++) {

                    if ($this->posts[$i]['id'] === $id) {

                        $this->posts[$i]['content'] = 'Updated post content';

                        echo "Post with ID $id updated.\n";

                        return;

                    }

                }

                echo "Post with ID $id not found.\n";

            }

            public function delete(int $id) {

                for ($i = 0; $i < count($this->posts); $i++) {

                    if ($this->posts[$i]['id'] === $id) {

                        unset($this->posts[$i]);

                        $this->posts = array\_values($this->posts);

                        echo "Post with ID $id deleted.\n";

                        return;

                    }

                }

                echo "Post with ID $id not found.\n";

            }

            public function edit(int $id) {

                for ($i = 0; $i < count($this->posts); $i++) {

                    if ($this->posts[$i]['id'] === $id) {

                        $this->posts[$i]['content'] = 'Edited post content';

                        echo "Post with ID $id edited.\n";

                        return;

                    }

                }

                echo "Post with ID $id not found.\n";

            }

        }

        $post = new Post();

        $post->create();

        print\_r($post->getAll());

        $post->update(1);

        print\_r($post->getOne(1));

        $post->edit(1);

        print\_r($post->getAll());

        $post->delete(1);

        print\_r($post->getAll());

## The \_\_construct Function

- A constructor allows you to initialize an object's properties upon creation of the object.

- If you create a \_\_construct() function, PHP will automatically call this function when you create an object from a class.

-The constructor is a special kind of method that doesn't have a return type and void.

class Student {

      public string $name;

      public function \_\_construct($name) {

        $this->name = $name;

      }

    }

$std = new Student("Waad");

## The \_\_destruct Function

-A destructor is called when the object is destructed or the script is stopped or exited.

-If you create a \_\_destruct() function, PHP will automatically call this function at the end of the script.

class Student {

      public string $name;

      public function \_\_construct($name) {

        $this->name = $name;

      }

      public function \_\_destruct() {

        echo "The Student is {$this->name}.";

      }

    }

$std = new Student("Waad");

## Interfaces vs. Abstract Classes

* class can implement multiple interfaces and class can extend only one abstract class
* Interfaces cannot have properties, while abstract classes can.
* All interface methods must be public, while abstract class methods is public or protected
* All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary

## Access Modifiers

There are three access modifiers:

* public - the property or method can be accessed from everywhere.

Public string $name;

* protected - the property or method can be accessed within the class and by classes derived from that class. )Inheritance(

Protected string $color;

* private - the property or method can ONLY be accessed within the class

  private string $address;