OOP

1. Classes(এটা হচ্ছে BluePrint)

- **Definition**: A class is a blueprint for creating objects. It defines a set of attributes (properties) and behaviors (methods) that the objects created from it will have.

2. Objects(এটা হচ্ছে real object)

- **Definition**: An object is an instance of a class. When a class is defined, objects can be created based on that class.

```php
<?php
    class Car {
        public $color;
        public $model;

        public function __construct($color, $model) {
            $this->color = $color;
            $this->model = $model;
        }

        public function drive() {
            echo "The $this->model with color $this->color is driving.";
        }
    }
    $my_car=new Car("red","BMW");
    $my_car->drive();
?>
```

3. Methods

- **Definition**: Methods are functions defined within a class that represent the behaviors of an object.

4. Properties

- **Definition**: Properties are variables that hold the data of an object and define its characteristics.

```php
<?php
class Car
{
    public $fuelLevel = 0; // Property with a default value

    // Public method to add fuel
    public function addFuel($amount)
    {
        $this->fuelLevel += $amount;
    }

    // Public method to drive, using the private method
    public function drive()
    {
        echo "Car is driving...Fuel remaining $this->fuelLevel:";
    }
}

$myCar = new Car();
$myCar->addFuel(20); // Adds fuel
$myCar->drive(); // Calls the drive method which internally checks fuel level

?>
```

5. Public and Private Methods and Properties of Classes

- **Public**: Accessible from outside the class.
- **Private**: Only accessible within the class itself.

```php
<?php
class BankAccount {
    // Private property to store the balance, not accessible outside the class
    private $balance;

    // Constructor to initialize the balance
    public function __construct($initialBalance) {
        $this->balance = $initialBalance;
    }

    // Public method to deposit money, accessible from outside
    public function deposit($amount) {
        if ($amount > 0) {
            $this->balance += $amount;
            echo "Deposited: $amount. New balance is: $this->balance\n";
        } else {
            echo "Deposit amount must be positive.\n";
        }
    }

    // Public method to withdraw money, accessible from outside
    public function withdraw($amount) {
        // Using a private method to check if sufficient funds are available
        if ($this->hasSufficientBalance($amount)) {
            $this->balance -= $amount;
            echo "Withdrew: $amount. New balance is: $this->balance\n";
        } else {
            echo "Insufficient funds for withdrawal.\n";
```

```
        }
    }

    // Public method to get the current balance, accessible from outside
    public function getBalance() {
        return $this->balance;
    }

    // Private method to check if there is enough balance for a withdrawal
    private function hasSufficientBalance($amount) {
        return $this->balance >= $amount;
    }
}

// Example usage
$account = new BankAccount(100); // Initial balance of 100
$account->deposit(50);           // Deposits 50, new balance: 150
$account->withdraw(30);          // Withdraws 30, new balance: 120
echo "Current Balance: " . $account->getBalance() . "\n"; // Outputs balance
$account->withdraw(200);         // Attempts to withdraw 200, shows insufficient funds

?>
```

7. Inheritance

- **Definition**: Inheritance allows a class to inherit properties and methods from another class.

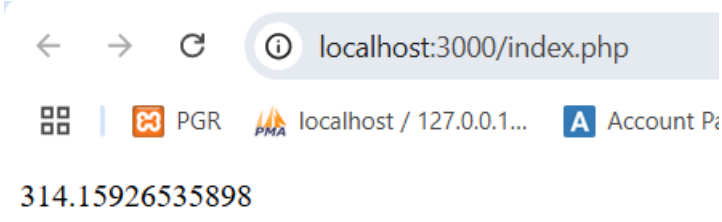
```
<?php
// Base class
class Vehicle {
    public $speed;

    // Constructor to initialize speed
    public function __construct($speed) {
        $this->speed = $speed;
    }

    // Public method to start the engine
    public function startEngine() {
        echo "Engine started at speed {$this->speed} km/h.\n";
    }
}

// Derived class
class Car extends Vehicle {
    public $model;

    // Constructor to initialize speed and model
    public function __construct($speed, $model) {
        // Call the parent constructor to set speed
        parent::__construct($speed);
        $this->model = $model;
    }

    // Public method specific to Car
    public function honk() {
        echo "Honk! This is a {$this->model} car.\n";
    }
}

// Create an object of the Car class
$myCar = new Car(100, "Toyota");
```

```php
// Access inherited method from Vehicle
$myCar->startEngine(); // Output: Engine started at speed 100 km/h.

// Access method specific to Car
$myCar->honk(); // Output: Honk! This is a Toyota car.
?>
```

8. Abstract(যে ক্লাস এর অবজেক্ট তৈরি করা যাবে না।যে inherit করবে তাকে override করতে হবে।)

- **Definition**: An abstract class is a class that cannot be instantiated and is designed to be extended. Abstract methods in an abstract class must be implemented in any subclass.

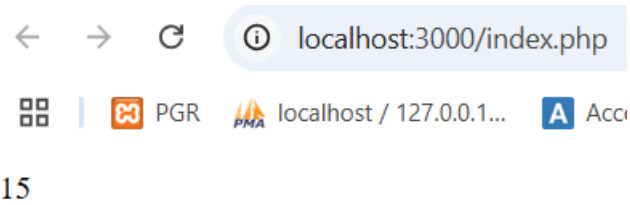
```php
<?php
abstract class Shape {
    abstract public function area();
}

class Circle extends Shape {
    private $radius;
    public function __construct($radius) {
        $this->radius = $radius;
    }
    public function area() {
        return pi() * $this->radius ** 2;
    }
}
$my_circle=new Circle(10);
$my_circle->area();
?>
```



9. Static(ক্লাস নেইম দিয়ে access করা যাবে।)

- **Definition**: A static property or method belongs to the class itself rather than to any specific object instance and can be accessed without creating an instance of the class.

```php
<?php
class MathHelper {
    public static function add($a, $b) {
        return $a + $b;
    }
}

echo MathHelper::add(5, 10);  // Output: 15

?>
```



10. Final(Final class কে ইনহেরিট করা যায় না , Final method কে override করা যায় না।)

- **Definition**: A final class cannot be extended, and a final method cannot be overridden in a subclass.

```php
<?php
final class DatabaseConnection {
    final public function connect() {
        echo "Connecting to database...";
    }
}

?>
```

INTERFACE

In PHP, an interface is a contract that defines a set of methods that a class must implement. Unlike classes, interfaces cannot contain any concrete code (i.e., no method bodies) but only method signatures. This is useful in enforcing consistency across different classes, ensuring that they follow a certain structure.

(এর মাধ্যমে ক্লাস এ ইন্টারফেসকে ইমপ্লিমেন্ট করলে ক্লাসকে ফোর্স করা হয় যাতে ইন্টারফেসর মেথডকে ইমপ্লিমেন্ট করতে।)

```php
<?php
    interface Printer{
        public function print();
    }
    class PaperPrinter implements Printer{
        public function print(){
            echo"I am from PaperPinter<br>";
        }
    }
    class PdfPrinter implements Printer{
        public function print(){
            echo "I am from PdfPrinter<br>";
        }
    }
    function PrintDocument(Printer $printer){
        $printer->print();
    }

    $obj_paper=new PaperPrinter();
    $obj_paper->print();

    $obj_paper1=new PdfPrinter();
    $obj_paper1->print();
?>
```