

Class & Object

- Class allow developers to encapsulate related data and functions into a single entity.
- Making it easier to manage and extend code
- An object is an instance of a class. You create an object of a class using the new keyword.

```
class Car {  
    public $color = "red";  
    public function drive() {  
        echo "Car is driving!";  
    }  
}  
  
$myCar = new Car();  
echo $myCar->color;  
$myCar->drive();
```

Constructor

- Method that gets executed whenever an object is instantiated from a class
- The constructor method has a magic name: `__construct`

```
class Car {  
    public function __construct() {  
        $num1=10;  
        $num2=20;  
        echo $num1+$num2;  
    }  
}  
  
$myCar = new Car();
```

Constructor Parameters

- Pass parameters to the constructor just like you would with any other function or method.
- Constructor can assign value to class properties

```
class Car {  
    public function __construct($num1,$num2) {  
        echo $num1+$num2;  
    }  
}
```

```
}  
$myCar = new Car(2,3);
```

```
class Car {  
    public $num1;  
    public $num2;  
    public function __construct($num1,$num2) {  
        $this->num1 = $num1;  
        $this->num2 = $num2;  
    }  
    function AddTwoNum(){  
        echo $this->num1+$this->num2;  
    }  
}  
$myCar = new Car(2,3);  
$myCar->AddTwoNum();
```

```
class Car {  
  
    public $num1;  
    public $num2;  
    public function __construct($num1,$num2) {  
        $this->num1 = $num1;  
        $this->num2 = $num2;  
    }  
    function AddTwoNum($a,$b){  
        echo $a+$b;  
    }  
}  
  
$myCar = new Car(2,3);  
$myCar->AddTwoNum(6,8);
```

Inheritance

- Inheritance sets up a "like parent, like child" relationship between classes.
- Instead of rewriting code, the child class can reuse or change what it gets from the parent.
- One class (the child) can use everything from another class (the parent).

```
class Father {  
    public function print100() {  
        for($i=0;$i<=100;$i++){
```

```

        echo "$i <br/>";
    }
}

class Son extends Father {

}

$SonObject = new Son();
$SonObject->print100();

```

Overriding Methods

- Subclasses can override inherited methods from the superclass.

```

class Father {
    public function print100() {
        for($i=0;$i<=100;$i++){
            echo "$i <br/>";
        }
    }
}

class Son extends Father {
    public function print100() {
        for($i=0;$i<=80;$i++){
            echo "$i <br/>";
        }
    }
}

$SonObject = new Son();
$SonObject->print100();

```

Parent Keyword

- You can call the parent class's method using the parent keyword.

```

class Father {
    public function print100() {
        for($i=0;$i<=100;$i++){
            echo "$i <br/>";
        }
    }
}

```

```

    }
}

class Son extends Father {
    public function CallFromFather() {
        parent::print100();
    }
}

$SonObject = new Son();
$SonObject->CallFromFather();

```

Abstract Classes

- Abstract classes cannot be instantiated on their own but can be subclassed

```

abstract class Father {
    public function print100() {
        for($i=0;$i<=100;$i++){
            echo "$i <br/>";
        }
    }
}

class Son extends Father {

}

$SonObject = new Son();
$SonObject->print100();

```

Final Keyword

- If you declare a class as final, it means it cannot be extended (inherited).
- If you declare a method as final, it means it cannot be overridden by a subclass.

```

final class Father {
    final public function print100() {
        for($i=0;$i<=100;$i++){
            echo "$i <br/>";
        }
    }
}

```

```

class Son extends Father {
    public function print100() {
        for($i=0;$i<=80;$i++){
            echo "$i <br/>";
        }
    }
}

```

Constructors and Inheritance

- If a child class has its own constructor, the parent class's constructor will not be automatically called.
- Use `parent::__construct()` if you want to explicitly call the base class's constructor.

```

class Father {
    public function __construct() {
        echo "Father constructor";
    }
}

class Son extends Father {
    public function __construct() {
        parent::__construct();
        echo " and Son constructor";
    }
}

$newObj = new Son();

```

Static Properties

- Static properties are tied to the class, not an instance of the class.
- They can be accessed without creating an instance of the class.

```

class MyClass {
    public static $staticProperty = "Static Property";
}

echo MyClass::$staticProperty; // Outputs: Static Property

```

Static Methods

- Just like static properties, static methods are accessed without creating an instance of the class
- They are often used as utility functions that do not rely on any instance-specific data

```
class MyClass {  
    public static function staticMethod() {  
        echo "Static Method";  
    }  
}  
  
MyClass::staticMethod();
```

Accessing Static Properties Inside Class Methods

- Within class methods, static properties and methods are accessed using the self keyword followed by the scope resolution operator

```
class MyClass {  
    public static $value = "Static Value";  
    public static function showValue() {  
        echo self::$value;  
    }  
}  
  
MyClass::showValue();
```

Access modifiers

Access modifiers control the visibility of class properties and methods

- **public** – accessible everywhere
- **protected** – accessible within the class and its subclasses (inheritance)
- **private** – accessible only within the class itself

```
class Fruit {  
  
    public $color;           // Can be accessed anywhere  
  
    protected $taste;       // Can be accessed within this class and derived  
    classes  
  
    private $origin;        // Can be accessed only within this class
```

```

public function setTaste($taste) {

    $this->taste = $taste;

}

public function setOrigin($origin) {

    $this->origin = $origin;

}

public function describe() {

    echo "This fruit is " . $this->color . " and tastes " . $this->taste . "
from " . $this->origin . ".\n";

}

}

class Apple extends Fruit {

    public function revealTaste() {

        return $this->taste; // Allowed because $taste is protected

    }

    // This function will generate an error if uncommented

    // public function revealOrigin() {

    //     return $this->origin; // Error, as $origin is private to Fruit

    // }

```

```
}
```

```
$apple = new Apple();
```

```
$apple->color = "red";
```

```
$apple->setTaste("sweet");
```

```
$apple->setOrigin("Washington");
```

```
$apple->describe(); // This fruit is red and tastes sweet from Washington.
```

```
echo $apple->revealTaste(); // Outputs: sweet
```

```
// Uncommenting the following line would produce an error
```

```
// echo $apple->revealOrigin();
```

```
#php
```