Notes:

Best Practices:

1. Class name and file name should be same where class name in PascalCase and file name in lower case format and method in Hungarian (name with underscore) style.
2. Avoid closing tag of php when define a class because of avoiding white space.
3. Avoid magic number in code due to magic number will be reduced code readability.
4. Name should be meaningful in all cases.
5. In object oriented data must be private never be public.

General Notes:

1. Modern programming languages usually support or even require an object-oriented approach to software development. Object-oriented development attempts to use the classifications, relationships, and properties of the objects in the system to aid in program development and code reuse.
2. A classes method is used to manipulate its' own data / properties.
3. “Object-oriented programming is a style of coding that allows developers to group similar tasks into classes.”

This helps keep code following the tenet "don't repeat yourself" (DRY) and easy-to-maintain.

1. Objects

In the context of OO software, an object can be almost any item or concept—a physical object such as a desk or a customer; or a conceptual object that exists only in software, such as a text input area or a file.

Object-oriented software is designed and built as a set of self-contained objects with both attributes and operations. *Attributes* are properties or variables that relate to the object. *Operations* are methods, actions, or functions that the object can perform to modify it or perform for some external effect.

Object-oriented software’s central advantage is its capability to support and encourage *encapsulation*—also known as *data hiding*. Essentially, access to the data within an object is available only via the object’s operations, known as the *interface* of the object.

1. Classes form the structure of data and actions and use that information to build objects.
2. "The power of OOP becomes apparent when using multiple instances of the same class."
3. One of the major benefits of DRY programming is that, if a piece of information changes in your program, usually only one change is required to update the code.
4. The getter and setter names, match the associated property name.

Example:

class person{

var $name;

function set\_name($new\_name){

$this->name = $new\_name;

}

function get\_name(){

return $this->name;

}

}

This way, when other PHP programmers want to use your objects, they will know that if you have a method/function called 'set\_name()', there will be a property/variable called 'name'.

We use getter function when if we want to see the data of the field or properties value.

1. The '$this' variable :

The $this is a built-in variable (built into all objects) which points to the current object. Or in other words, $this is a special self-referencing variable. You use $this to access properties and to call other methods of the current class.

If we want to use the class properties or field in method then we use this operator.

function get\_name() {

return $this->name;

}

1. Include your class in your main PHP page:

It is always best practice to create separate php pages that only contain your classes. Then you would access your php objects/classes by including them in your main php pages with either a php 'include' or 'require'.

Example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<?php include("class\_lib.php"); ?>

</head>

<body>

</body>

</html>

1. Instantiate/create your object:

Instantiation is the process of creating an instance of an object in memory. What memory? The server's memory of course!

<?php include("class\_lib.php"); ?>

</head>

<body>

<?php

$stefan = new person();

?>

</body>

</html>

The variable $stefan becomes a handle/reference to our newly created person object. I call $stefan a 'handle', because we will use $stefan to control and use the person object.

1. The 'new' keyword:

To create an object out of a class, you need to use the 'new' keyword.

1. Set an objects properties:

Now that we've created/instantiated our two separate 'person' objects, we can set their properties using the methods (the setters) we created.

Please keep in mind that though both our person objects ($stefan and $nick) are based on the same 'person' class, as far as php is concerned, they are totally different objects.

<?php include("class\_lib.php"); ?>

</head>

<body>

<?php

$stefan = new person();

$jimmy = new person();

$stefan->set\_name("Stefan Mischook");

$jimmy->set\_name("Nick Waddles");

?>

</body>

</html>

1. Accessing an object's data:

We use the getter methods to access the data held in our objects … this is the same data we inserted into our objects using the setter methods.

<?php include("class\_lib.php"); ?>

</head>

<body>

<?php

$stefan = new person();

$jimmy = new person();

$stefan->set\_name("Stefan Mischook");

$jimmy->set\_name("Nick Waddles");

echo "Stefan's full name: " . $stefan->get\_name();

echo "Nick's full name: " . $jimmy->get\_name();

?>

</body>

</html>

When accessing methods and properties of a class, you use the arrow (->) operator.

1. Directly accessing properties - don't do it! :

You don't have to use methods to access objects properties; you can directly get to them using the arrow operator (->) and the name of the variable.

For example: with the property $name (in object $stefan,) you could get its' value like so: $stefan->name.

1. Constructors:

All objects can have a special built-in method called a 'constructor'. Constructors allow you to initialize your object's properties (translation: give your properties values,) when you instantiate (create) an object.

Note: If you create a \_\_construct() function (it is your choice,) PHP will automatically call the \_\_construct() method/function when you create an object from your class. The 'construct' method starts with two underscores (\_\_) and the word 'construct'. You 'feed' the constructor method by providing a list of arguments (like a function) after the class name.

1. Create an object with a constructor:

Now that we've created a constructor method, we can provide a value for the $name property when we create our person objects.

For example: $stefan = new person("Stefan Mischook");

This saves us from having to call the set\_name() method reducing the amount of code. Constructors are common and are used often in PHP, Java etc.

Less code means fewer bugs.

1. Restricting access to properties using 'access modifiers':

One of the fundamental principles in OOP is 'encapsulation'. The idea is that you

Create cleaner better code, if you restrict access to the data structures (properties) in your objects. You restrict access to class properties using something called 'access modifiers'.

There are 3 access modifiers:

• Public

• Private

• protected

**Note**: When you declare a property with the 'var' keyword, it is considered 'public'.

1. Restricting access to properties: Part 2

When you declare a property as 'private', only the same class can access the property.

When a property is declared 'protected', only the same class and classes derived from that class can access the property - this has to do with inheritance …more on that later.

Properties declared as 'public' have no access restrictions, meaning anyone can access them.

1. Restricting access to methods:

Like properties, you can control access to methods using one of the three access modifiers:

• Public

• protected

• Private

Why do we have access modifiers?

The reason for access modifiers comes down to control: it is makes sense to control how people use classes.

1. Reusing code the OOP way: inheritance:

Inheritance is a fundamental capability/construct in OOP where you can use one class, as the base/basis for another class … or many other classes.

Doing this allows you to efficiently reuse the code found in your base class.

The actual code being reused, can be reused in many (unlimited) classes but it is only typed out in one place … conceptually, this is sort-of like PHP includes().

'extends' is the keyword that enables inheritance.

1. Inheritence: It is the process by which one object acquires the properties of another object. Inheritence provides the idea or reusability.
2. Polymorphism: means the ability to take more than one form. Allows us to use an entity in multiple forms. It allows us to have more than one function with the same in a program.
3. Scope Resolution Operator (::)

It is a token that allows access to static, constant, and overridden properties or methods of a class.

When referencing these items from outside the class definition, use the name of the class.

It's possible to reference the class using a variable. The variable's value cannot be a keyword (e.g. self, parent and static).

Example #1 :: from outside the class definition

<?php

class MyClass {

const CONST\_VALUE = 'A constant value';

}

$classname = 'MyClass';

echo $classname::CONST\_VALUE; // As of PHP 5.3.0

echo MyClass::CONST\_VALUE;

?>

Three special keywords self, parent and static are used to access properties or methods from inside the class definition.

Example #2 :: from inside the class definition

<?php

class OtherClass extends MyClass

{

public static $my\_static = 'static var';

public static function doubleColon() {

echo parent::CONST\_VALUE . "\n";

echo self::$my\_static . "\n";

}

}

$classname = 'OtherClass';

echo $classname::doubleColon(); // As of PHP 5.3.0

OtherClass::doubleColon();

?>

Example #3 calling a parent's method

<?php

class MyClass

{

protected function myFunc() {

echo "MyClass::myFunc()\n";

}

}

class OtherClass extends MyClass

{

// Override parent's definition

public function myFunc()

{

// But still call the parent function

parent::myFunc();

echo "OtherClass::myFunc()\n";

}

}

$class = new OtherClass();

$class->myFunc();

?>

1. Magic Methods in OOP

To make the use of objects easier, PHP also provides a number of magic methods, or special methods that are called when certain common actions occur within objects.

The "magic" methods are ones with special names, starting with two underscores, which denote methods which will be triggered in response to particular PHP events.

These are Magic Methods that allow you to react to certain events when using these particular objects. This means when certain things happen to your object, you can define how it should react in that instance.

Only magical methods can use the magical constants.

Converting to a String :

To avoid an error if a script attempts to output MyClass as a string, another magic method is used called \_\_toString().

Without \_\_toString(), attempting to output the object as a string results in a fatal error. Attempt to use echo to output the object without a magic method in place:

The function names \_\_construct(), \_\_destruct(), \_\_call(), \_\_callStatic(), \_\_get(), \_\_set(), \_\_isset(), \_\_unset(), \_\_sleep(), \_\_wakeup(), \_\_toString(), \_\_invoke(), \_\_set\_state(), \_\_clone() and \_\_debugInfo() are magical in PHP classes. You cannot have functions with these names in any of your classes unless you want the magic functionality associated with them.

Using Constructors and Destructors:

When an object is instantiated, it's often desirable to set a few things right off the bat. To handle this, PHP provides the magic method \_\_construct(), which is called automatically whenever a new object is created.

Constructor function is called at the very beginning of the objects and Destructor function is called at the very last of the scope of the objects.

\_\_CLASS\_\_ returns the name of the class in which it is called; this is what is known as a magic constant. There are several available magic constants, which you can read more about in the PHP manual.

To call a function when the object is destroyed, the \_\_destruct() magic method is available. This is useful for class cleanup (closing a database connection, for instance).

Destructor function is automatically call have been unset all of the references.

The destruct method will allow you to cleanup anything that shouldn’t be around once the object has been destroyed. For example this could be a connection to an external service or database.

To be honest you don’t really see much of the \_\_destruct() method. PHP is not really the type of language that you would be using for long running processes so I think for the most part you don’t really need to ever clean up anything using the \_\_destruct() method. The lifecycle of a PHP request is so short, actually implementing the \_\_destruct() method is probably more hassle than it’s worth.

"When the end of a file is reached, PHP automatically releases all resources."

To explicitly trigger the destructor, you can destroy the object using the function unset().

1. Abstract Class:

* A class that represents a generalization and provides functionality but is only intended to be extended and not instantiated
* An Abstract Class is basically a blueprint that should only ever be extended by child classes and not used directly. By extending the blueprint, the developer can ensure that the basic functionality is present in the child class in order for it to be functional.
* Abstract class is treat as a super class and may be extended but never ever be instantiated.
* The word abstract must before declare a class as an abstract class.

1. Interface

* A completely abstract class that defines a protocol for object interactions.
* An interface is a collection of method declarations.
* An interface is a class-like concept.
* An interface has no variable declarations or method bodies.
* Describes a set of methods that a class can be forced to implement.
* Interfaces can be used to implement multiple inheritance like hierarchies.
* Interfaces are described to provide a common function names to the implementers
* Interface only contains function prototype( i.e function declaration)
* We can implement the interface using ‘implements’ keyword. And then, we can define the functions according to our requirement.
* Multiple inheritance is achieved through interfaces

Difference Abstract Class and Interface

|  |  |
| --- | --- |
| Abstract Class | Interface |
| A class can extend only one abstract class. | A class can implement several interfaces. |
| The members of abstract class can be private as well as protected. | An interface can only have public members. |
| Abstract class should have subclass. | Interfaces must have implementations by classes. |
| There can be a constructor for abstract class. | Interface does not have constructor. |

1. Difference between overriding and overloading

* Overriding involves the creation of two or more methods with the same name and same signature (arguments, parameters or data type) in different classes (one of them should be parent class and other should be child.)
* Overloading is a concept of using a method at different places with same and different signatures (arguments) within the same class. There are two types of overloading: function overloading and operator overloading.
* A class cannot inherit the constructor of its base class.
* Overriding of functions occurs when one class is inherited from another class. Overloading can occur without inheritance.
* Overloading is static binding whereas overriding is dynamic binding.

1. If we use word finally that means the function is not override again.
2. Whenever a class is defined without any name is called anonymous class.
3. What is the difference between Abstract Class and Inheritance?

Abstract classes usually represent an abstract concept or an entity with partial or no implementation. Inheritance allows new classes to extend other classes. Because, Abstract classes cannot be instantiated, you need to use the concept of inheritance to make use of Abstract classes. Otherwise, an Abstract class has no use. Abstract classes may contain Abstract methods and when the class is extended, all methods (Abstract and concrete) are inherited. The inherited class can implement any or all the methods. If all the Abstract methods are not implemented, then that class also becomes an Abstract class. A class cannot inherit from more than one Abstract class (this is not a quality of Abstract classes per se, but rather a restriction of inheritance).