

# an **object**;

## is **defined**;

An 'object' is an instance of a 'class'. You define a class using the code below.

```
class parentClass
{
    //code in here
}
```

## is **instantiated**;

```
$parentObject = new parentClass;
```

Optionally, you can pass variables (or properties) into the class which are used by the constructor method.

```
$parentObject = new parentClass($var1, $var2);
```

## has a **constructor**;

This is a 'magic method' which is executed when the object is instantiated. If no constructor is defined, the constructor from the parent class (if applicable) is used.

```
function __construct()
{
    //code in here
}
```

If you are passing variables into the object then you will need to use the following code:

```
function __construct($var1, $var2)
{
    //code in here
}
```

You can use the parent constructor instead. By defining the constructor in the child class you can run some processing or pass in additional arguments into the parent constructor. For example,

```
function __construct($var1, $var2)
{
    parent::__construct($var1, $var2, $var3);
}
```

## has a **destructor**;

This 'magic method' is executed when the object has finished, usually once the script has finished processing.

```
function __destruct()  
{  
    //code in here  
}
```

## can **inherit**;

A class can inherit from another class and access all of the parent methods and properties contained within. This is one of the key principles of OOP; **Inheritance**.

```
class childClass extends parentClass  
{  
    //code in here  
}
```

## can be **abstract**;

This class cannot be instantiated directly and must be extended from. This is one of the key principles of OOP; **Abstraction**.

```
abstract class parentClass  
{  
    //code in here  
}
```

## can be **final**;

This is the last class in the hierarchy. You cannot extend from this class.

```
final class childClass extends parentClass  
{  
    //code in here  
}
```

## can be **copied**;

An object (an instance of a class) can be copied. This creates an entirely new instance of the object containing the data at the point the object was copied.

```
$childObject = new childClass;  
$copy_of_childObject = clone $childObject;
```

## can be **autoloaded**;

Auto loading is a 'magic method' which is called once a new object is instantiated. It is commonly used to include the class PHP file so you don't need to include all the class files in your application. For example;

```
function __autoload($class)
{
    include_once(CLASS_INCLUDE_PATH.'class.'.$class.'.php');
}
```

## using **instanceof**;

This is used to determine whether a variable is an instantiated object.

```
$childObject = new childClass;
if ($childObject instanceof childClass)
{
    //this is true
}
```

## using **\_\_call()**;

This is a 'magic method' which is run when an inaccessible or undefined method is called. The function name and arguments must be passed in as arguments.

```
function __call($function, $args)
{
    echo 'Call to inaccessible or undefined method.<br>';
    echo 'Method: '.$function.'<br>';
    echo implode(' ', $args);
    exit;
}
```

# a **method**;

## is **defined**;

A method is a function within a class.

```
function childMethod()
{
    echo 'This is a method.';
}
```

## is **accessed**;

Use methods in the exact way as you would normally use a function. You can pass variables into the method if you wish. To call the method from within the class itself use;

```
$this->childMethod();
```

To call the method from outside the class use;

```
$childObject = new childClass;
$result = $childObject->childMethod();
```

You can also call methods from parent classes in the exact same way. You reference them from within the class and outside of the class using;

```
$childObject = new childClass;
$result = $childObject->parentMethod();
```

Alternatively, from within the child class you can use the following. This is normally only used when referencing static methods or properties.

```
$result = parent::parentMethod();
```

### can be **abstract**;

Making a method abstract means it must be defined in all child classes. If arguments are given, they must also be included in the child classes. See examples below from the parent class;

```
abstract function requiredMethod($var1, $var2);
abstract function anotherRequiredMethod();
```

### can be **overridden**;

You can override a method in child classes, providing they are not set as final (see below). For example, the following method can be defined in the child class. This is one of the key principles of OOP; **Polymorphism**.

```
function parentMethod()
{
    echo 'This is the same as the parent method.';
}
```

### can be **final**;

Marking a method as final means it cannot be overridden in subsequent child classes.

```
final function parentMethod()
{
    echo 'You cannot define another method with this name';
}
```

## can be **public**;

By default, all methods are public. However, it is best practice to add the public keyword. This is one of the key principles of OOP; **Encapsulation**.

```
public function parentMethod()
{
    //code in here
}
```

## can be **private**;

If a method is set as private it can only be accessed from within the class itself.

```
private function parentMethod()
{
    //code in here
}
```

## can be **static**;

A static method allows you to access static properties within a class, and nothing else. You do not need to instantiate an object to access the static method. To define a static method use;

```
//define a static constant
static public $static = 10;

static function staticMethod()
{
    //to access a static property use the self keyword
    echo self::$static;
}
```

Then to access the method from outside the class without instantiating the object (where *childClass* is the name of the object) use;

```
echo childClass::staticMethod();
```

## type hinting;

Type hinting allows you to specify the data type of arguments being passed into a method. For example, you may only wish an array or an instance of an object to be passed in.

```
public function parentMethod(childClass $childObject)
{
    echo 'The variable must be an instance of childClass';
}
```

You can also specify that an argument is an array using;

```
public function parentMethod(array $array)
{
    echo 'The variable must be an array';
}
```

## using `func_num_args()`;

This is a function which can be used within a method to check the number of arguments passed into the method. For example;

```
public function parentMethod($var1, $var2)
{
    if (func_num_args() != 2)
    {
        return false;
    }
    else
    {
        //code in here
    }
}
```

## a **property**;

### is **defined**;

A property is a variable within a class. To define a property use;

```
$childVar = True;
```

### is **accessed**;

To reference a property or set a property value from within the class itself use;

```
//to reference a property
print $this->childVar;

//to set a property value
$this->childVar = False;
```

To reference a property or set a property value from outside of the class use;

```
//to reference a property
print $childObject->childVar;

//to set a property value
$childObject->childVar = False;
```

can be **public**;

By default, all properties are public. This means they can be accessed and manipulated from anywhere. However, it is best practice to add the public keyword;

```
public $childVar = True;
public $anotherVar;
```

can be **private**;

A private property can only be accessed from within the class itself, not even in child classes.

```
private $childVar = True;
private $anotherVar;
```

can be **protected**;

A protected property can only be accessed from within the class itself and any child classes. It cannot be accessed from outside.

```
protected $childVar = True;
protected $anotherVar;
```

can be **static**;

You do not need to instantiate an object to access a static property. To define a static property use;

```
static public $static = 10;
```

To access the property from outside the class without instantiating the object (where *childClass* is the name of the object) use;

```
echo childClass::$static;
```

can be **constant**;

Constant properties will never change under any circumstances and are typically used as references for passing in as arguments into a method. For example, rather than passing in a value of 1, pass in the constant value *STATUS\_PENDING* which has meaning. To define a constant use;

```
const STATUS_PENDING = 1;
```

To access the constant you use the class name in the same way as access static properties.

```
echo childClass::STATUS_PENDING;
```

# an **interface**;

## is **defined**;

An interface is an 'instruction manual' or 'template' on how classes must be used. It is similar to defining abstract methods in parent classes but allows for greater flexibility across multiple classes. It is typically used for high-level tasks such as logging or exception handling.

```
interface template
{
    //code in here
}
```

## can contain **methods**;

This means that the defined methods and their signatures must be used somewhere in the hierarchy that uses this interface.

```
interface template
{
    public function parseXML($xml);
}
```

## can contain **constants**;

You can declare interface constants which can be used within the classes and also from outside. They cannot be overridden any class which uses the interface.

```
interface template
{
    const name = 'Hello';
}
```

To access these constants use;

```
echo template::name;
```

## is **implemented**;

When defining a class, you use the implements keyword to say the class must use the interface template. More than one interface can be implemented if desired.

```
class parentClass implements template
{
    //code in here
}
```

can be **extended**;

Interface can extend from other interfaces and inherit the same methods and constants from the parent interfaces.

```
interface template extends mainTemplate
{
    //code in here
}
```

## misc;

**php5 oop**;

<http://www.php.net/manual/en/oop5.intro.php>