



IPL

escola superior
de tecnologia e gestão
instituto politécnico
de leiria

PHP - OOP

Object-Oriented Programming in PHP

Marco Monteiro



Contributors

2

► Author(s):

- Vitor Carreira (vitor.carreira@ipleiria.pt)
- Marco Monteiro (marco.monteiro@ipleiria.pt)

► Contributor(s):

- Fernando Silva(Fernando.silva@ipleiria.pt)



Summary

3

1. Basic OOP
2. Autoload
3. Namespaces
4. Exceptions
5. References



1 – BASIC OOP



Class Syntax

5

► Example of class definition and object creation/usage

```
class Person {  
    private $canDance = true;  
  
    public function __construct($name)  
    {  
        $this->name = $name;  
    }  
  
    public function dance()  
    {  
        return $this->canDance ?  
            "I'm dancing!" : "I can't dance!";  
    }  
}  
  
$me = new Person("John");  
echo $me->dance();
```



Class Syntax

6

- ▶ Class names should be declared in StudlyCaps
- ▶ Method names should be declared in camelCase
- ▶ By default, class members are public
 - ▶ Visibility options: **public** | **protected** | **private**
- ▶ Constructor: `__construct` method
- ▶ **\$this** - reference to the current object
- ▶ **new** – creates an instance (object) of a class



PSR-2 (Coding Style Guide)

Properties

- Visibility MUST be declared on all properties.
- The var keyword MUST NOT be used to declare a property.
- There MUST NOT be more than one property declared per statement.
- Property names SHOULD NOT be prefixed with a single underscore to indicate protected or private visibility.

Methods

- Visibility MUST be declared on all methods.
- Method names SHOULD NOT be prefixed with a single underscore to indicate protected or private visibility.
- Method names MUST NOT be declared with a space after the method name. The opening brace MUST go on its own line, and the closing brace MUST go on the next line following the body. There MUST NOT be a space after the opening parenthesis, and there MUST NOT be a space before the closing parenthesis.



Access class members

8

- > is the object operator – to access an instance member (attribute or method)

```
$object->method();
```

```
$object->attribute
```

- :: is the Scope Resolution Operator – to access static, constant, and overridden properties or methods of a class

```
ClassName::staticMethod()
```




Constants

9

PSR-1 (Basic Coding Standard):

- **Class constants** MUST be declared in all upper case with underscore separators.

```
class Product {  
    private $price;  
    const VAT = 0.23;  
    . . .  
    public function priceWithVAT()  
    {  
        return $this->price *  
            (1 + self::VAT);  
    }  
}  
  
$p = new Product();  
echo $p->priceWithVAT();  
echo Product::VAT;
```



Inheritance

10

```
class DVD extends Product
{
    protected $year;
    public function __construct($title, $price, $year)
    {
        parent::__construct("dvd", $title, $price);
        $this->year = $year;
    }
    public function getYear()
    {
        return $this->year;
    }
    public function __toString()
    {
        return 'DVD['.parent::__toString().
            '; year='.$this->year. ']';
    }
}

$dvd1 = new DVD("47 Ronin", 25, 2013);
echo $dvd1;
```



Static and Final Members

11

```
class FooBar
{
    protected static $counter = 0;
    public $num;
    public function __construct()
    {
        self::$counter++;
        $this->num = self::$counter;
    }
    public static function convertLbToKg($pounds)
    {
        return $pounds * 0.4535923;
    }
    final public function instanceConvertLbToKg($pounds)
    {
        return self::convertLbToKg($pounds);
    }
}
echo FooBar::convertLbToKg(10);
```

PSR-2 (Coding Style Guide)

- When present, the **abstract** and **final** declarations MUST precede the visibility declaration.
- When present, the **static** declaration MUST come after the visibility declaration.



Abstract classes

12

```
abstract class Product
{
    protected $type;
    protected $title;
    abstract public function display();
}
```



Interfaces

13

```
interface IDownloads
{
    public function getFileLocation();
    public function createDownloadLink();
}

class Book extends Product implements IDownloads
{
    ...
    public function getFileLocation() {
        // details here
    }
    public function createDownloadLink() {
        // details here
    }
}
```

PSR-2 (Coding Style Guide)

The **extends** and **implements** keywords MUST be declared on the same line as the class name..



Object and class references

14

- ▶ **\$this** – current object
 - ▶ `$this->member` – object members
- ▶ **self::** – current class
 - ▶ `self::staticmember` – static members (class)
- ▶ **parent::** – base class (parent)



Operators and special methods

15

- ▶ **instanceof** operator – check if an object is an instance of a class
 - ▶ Related functions: `get_class()`; `get_parent_class()`, `is_a()`; `is_subclass_of()`;

```
$dvd1 = new DVD("Matrix", 25, 1999);  
echo get_class($dvd1);           // Outputs DVD  
echo get_parent_class($dvd1);    // Outputs Product  
echo ($dvd1 instanceof DVD);     // Outputs 1 (true)  
echo ($dvd1 instanceof Product); // Outputs 1 (true)
```

- ▶ **Special methods:**

`__construct()` | `__destruct()` | `__toString()` | `__clone()`
`($b = clone $a)` | `__get()` | `__set()` | `__call()` (to support
overloading `__autoload()`)



2 – AUTOLOAD



Auto loading classes

17

- ▶ PSR 0 / PSR 4 specifies that each class must be in a separate file by itself
- ▶ This would mean that it would be necessary to write long list of `require_once` (`include_once`) instructions at the beginning of the files (one for each class)
- ▶ To avoid that annoyance, PHP supports auto loading of classes or interfaces files



Auto loading classes

18

- ▶ spl_autoload_register function will automatically load the classes.
- ▶ It is executed automatically, when an undefined class is used

```
spl_autoload_register(function ($class_name) {  
    require_once "cls/" . $class_name . ".php";  
});
```

```
$dvd1 = new DVD("47 Ronin", 25, 2013);
```

- ▶ In the previous example, DVD class will be loaded from the file "cls/DVD.php"



Composer autoload

19

- ▶ Composer is a package manager for PHP that install and handles dependencies of php packages
- ▶ Composer includes a **built-in autoloader**, that is used to auto load the external packages, but it can also be used to autoload our own classes/files



Composer autoload

20

- Composer autoload example. File composer.json:

```
{  
    "autoload": {  
        "psr-4": {  
            "": "src"  
        },  
        "files": [  
            "src/functions/myfunc.php"  
        ]  
    }  
}
```



Composer autoload

21

- ▶ To install the composer autoload, execute on the shell (command prompt) “**composer install**”
- ▶ To update it, execute “**composer update**”
- ▶ To use the composer autoload it is also mandatory to include (on each file) the following code:

```
require 'vendor/autoload.php';
```

or

```
require_once 'vendor/autoload.php';
```



Composer autoload

22

- ▶ Previous example explained:
- ▶ **psr-4** section - defines a mapping from namespaces to paths
 - ▶ In the example `"" : "src"`
 - ▶ `""` - the namespace prefix
 - ▶ `"src"` - the root path
- ▶ **files** section - defines a set of files that are automatically included on every request.
Useful for files with PHP functions, that cannot be automatically loaded by PHP



3 – NAMESPACES



Namespaces

24

- ▶ One problem that might potentially occur in complex projects is the name collision
 - ▶ The same class (interface, function, etc.) name might be used on different packages.
- ▶ To solve this problem, PHP supports namespaces
- ▶ PHP Namespaces provide a way in which to group related classes, interfaces, functions and constants



► Namespace example:

```
namespace my\name;
```

```
class MyClass {  
    . . .  
}
```

```
use my\name\MyClass;  
$a = new MyClass;  
$c = new \my\name\MyClass;
```

PSR-2 (Coding Style Guide)

- When present, there MUST be one blank line after the namespace declaration.

\my\name\MyClass -> Fully Qualified Name



PSR4 Autoloader

26

- ▶ Namespaces and class MUST follow an “autoloading” PSR [PSR0; PSR4]
 - ▶ Preferably - PSR4 (PSR0 is deprecated)
- ▶ PSR 4 describes a specification for autoloading classes from file paths
 - ▶ Folders will translate to namespace or sub-namespaces
 - ▶ PHP files will translate to classes or interfaces, traits, and other similar structures



PSR4 Autoloader

27

1. Term “class” refers to classes, interfaces, traits, and other similar structures.
2. Fully qualified class name has the following form:

`\<NamespaceName> (\<SubNamespaceNames>) * \<ClassName>`

1. The top level namespace = vendor namespace
2. One or more sub-namespaces
3. Always terminate with a class name
4. Underscore have no special meaning
5. Alphabetic characters MAY be any lower and upper case
6. All class names MUST be referenced in a case-sensitive fashion.



3. When loading a file that corresponds to a fully qualified class name:
 1. The **namespace prefix** (with one more namespaces or sub-namespaces) corresponds to at least one **base directory**
 2. The sub-namespaces names after the namespace prefix will correspond to a sub-directory within the base directory (name of directory MUST match the case of sub-namespaces)
 3. Class name corresponds to a file name ending in .php. File name MUST match the case of the class name.



► Examples of PSR4 mappings

Fully Qualified Class Name	Namespace Prefix	Base Directory	Resulting File Path
\meu\produto\BlueRay	meu	./meu/	./meu/produto/BlueRay.php
\Acme\Log\Writer\File_Writer	Acme\Log\Writer	./acme-log-writer/lib/	./acme-log-writer/lib/File_Writer.php
\Aura\Web\Response>Status	Aura\Web	/path/to/aura-web/src/	/path/to/aura-web/src/Response/Status.php



Using Namespaces: Aliasing/Importing

- ▶ The ability to refer to an external fully qualified name with an alias, or importing
- ▶ In PHP, aliasing is accomplished with the **use** operator

```
namespace Vendor\Controller;
```

```
use Vendor\Model\Product;
```

```
class ProductController { . . . }
```

PSR-2 (Coding Style Guide)

- When present, all use declarations MUST go after the namespace declaration.
- There MUST be one use keyword per declaration.
- There MUST be one blank line after the use block.



Aliasing / Importing

31

```
use My\Full\Classname as Another;  
...  
$obj = new Another;  
// OR:  
$obj = new \My\Full\Classname;
```

```
use my\name\Something;
```

The same as:

```
use my\name\Something as Something;
```



Aliasing / Importing

32

```
use My\Full\Classname as Another;  
...  
$obj = new Another;      // My\Full\Classname  
$obj = new \Another;     // Another
```

```
use My\Full\NSname;  
// same as: use My\Full\NSname as NSname  
...  
$obj1 = new NSname\SomeClass;  
$obj2 = new NSname\AnotherNS\OtherClass;
```




4 – EXCEPTIONS



Exceptions

34

- ▶ Base class for all exceptions: class **Exception**
- ▶ Built-in methods
 - ▶ **getCode()** - Returns the error code as passed to the constructor
 - ▶ **getMessage()** - Returns the error message as passed to the constructor
 - ▶ **getFile(), getLine(), getTrace(), getTraceAsString()**



Exception handling

35

► try/catch example:

```
<?php
    try {
        . . .
    } catch (Exception $e) {
        echo "Exception " . $e->getCode() .
            ": " . $e->getMessage() . "<br>" .
            " in " . $e->getFile() .
            " on line " . $e->getLine() . "<br>";
    }
?>
```



Exception handling

36

► try/catch/finally:

```
try {  
    . . .  
} catch (Exception $e) {  
    . . .  
} finally {  
    . . .  
}
```

```
try {  
    . . .  
} finally {  
    . . .  
}
```

Throw

37

- To create (throw) an exception (error)

```
throw new Exception("An error has occurred", 42);
```

Error Message

Error Code



Custom Exceptions

38

```
class MyCustomException extends Exception
{
    // Exception details here
}
```

```
throw new MyCustomException( . . . );
```



5 – REFERENCES



References

40

- ▶ Official (PHP)
 - ▶ <http://www.php.net/>
 - ▶ <http://php.net/manual/en/>
 - ▶ http://php.net/manual/pt_BR/

- ▶ PSR - PHP Standard Recommendations
 - ▶ <https://www.php-fig.org>
 - ▶ <https://www.php-fig.org/psr/psr-4/>

- ▶ PHP and MySQL Web Development (4th Edition)
 - ▶ Luke Welling and Laura Thomson, Addison-Wesley 2009
- ▶ PHP Objects, Patterns, and Practice (2nd Edition)
 - ▶ Matt Zandstra, APress 2008
- ▶ Object Oriented PHP Concepts Techniques and Code
 - ▶ Peter Lavin, No Starch Press 2006