Course:
PHP from scratch

by Sergey Podgornyy

Object-oriented programming

WEB ACADEMY

PROGRAMMING COURSES

About me

Sergey Podgornyy

Full-Stack Web Developer

WebbyLab

and

ignite software outsourcing

zend CERTIFIED PHP ENGINEER

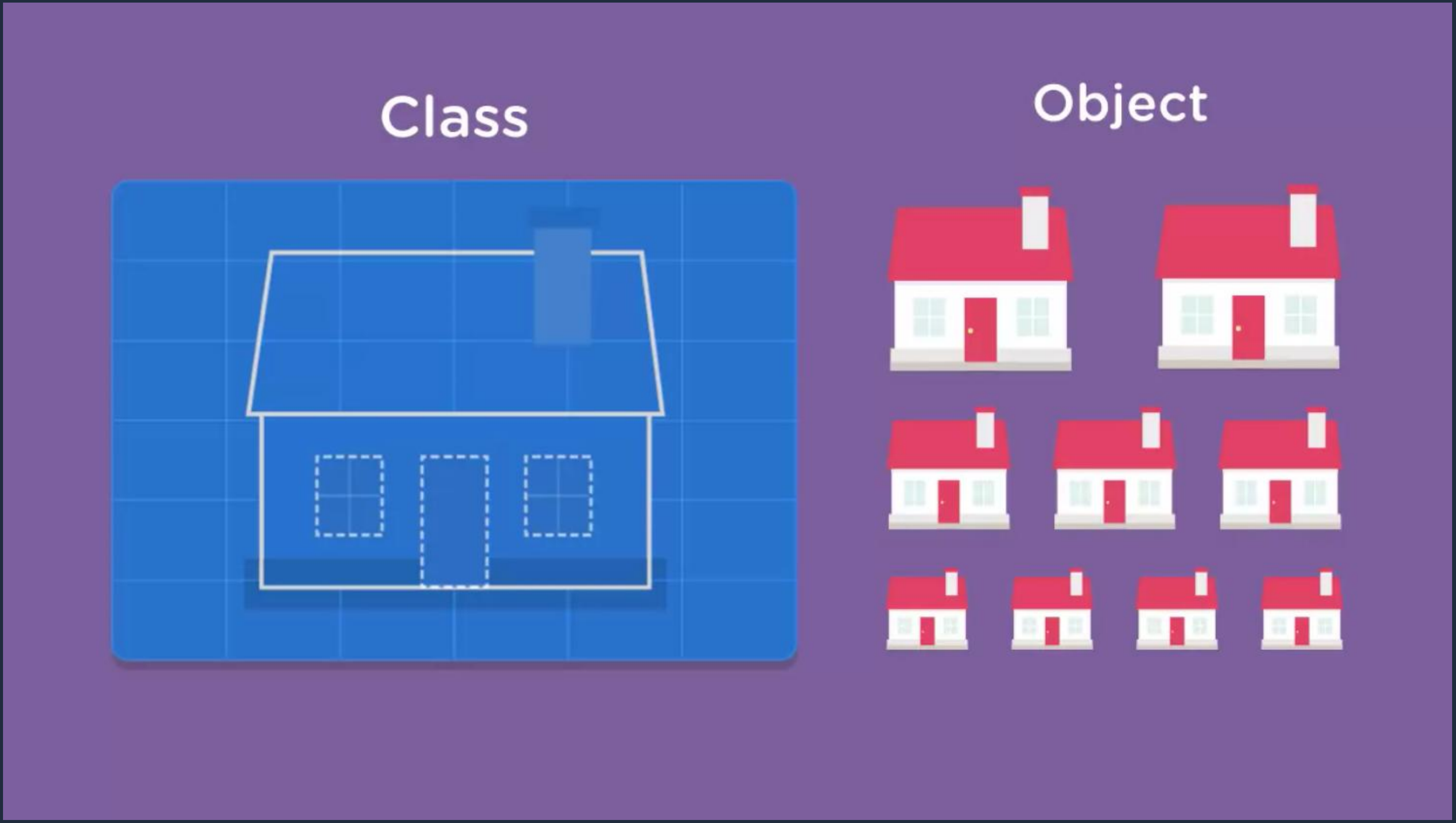
Linux Professional Institute LPIC-1

Overview

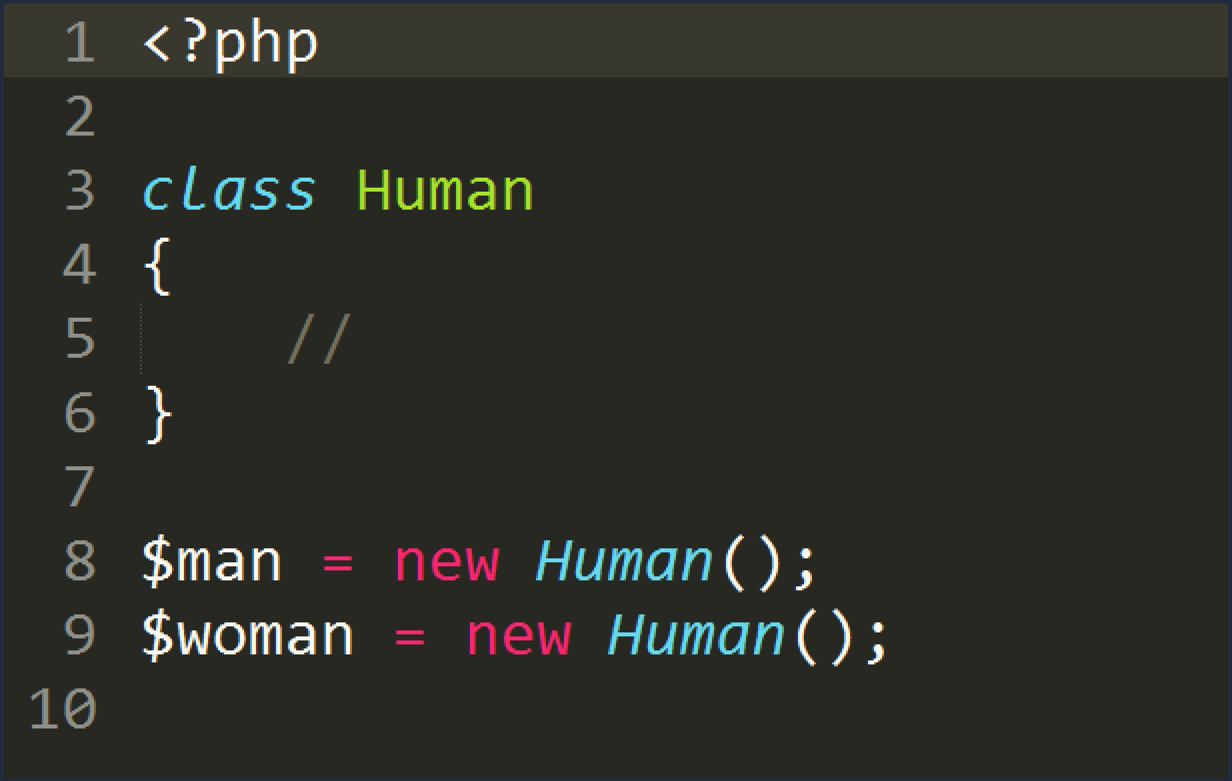
- Classes and objects
- Setting properties and working with methods
- Visibility
- Getters and setters
- Static variables
- **`__construct`**, **`__destruct`**, and other magic methods
- Arguments and types
- Object Oriented Paradigm
- **`final`**
- Cloning objects
- Abstract classes and interfaces
- Late static binding, **`static`** keyword
- Exceptions
- Namespaces and traits
- composer

Object-oriented programming

Classes and objects



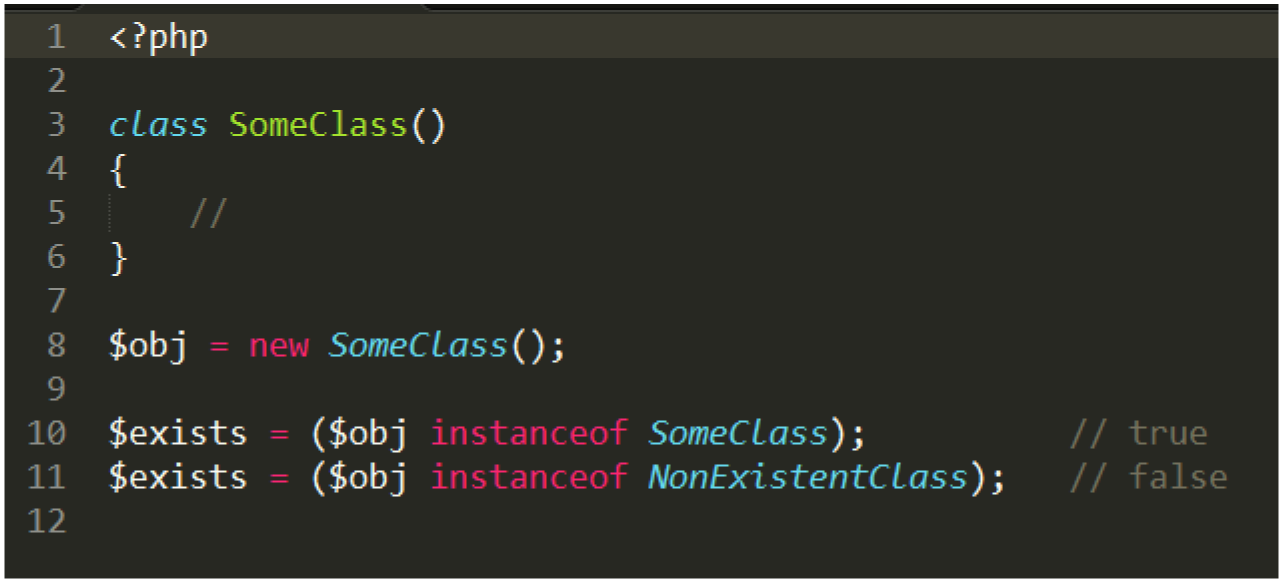
Classes and objects in PHP



```php
<?php

class Human
{
    //
}

$man = new Human();
$woman = new Human();

```

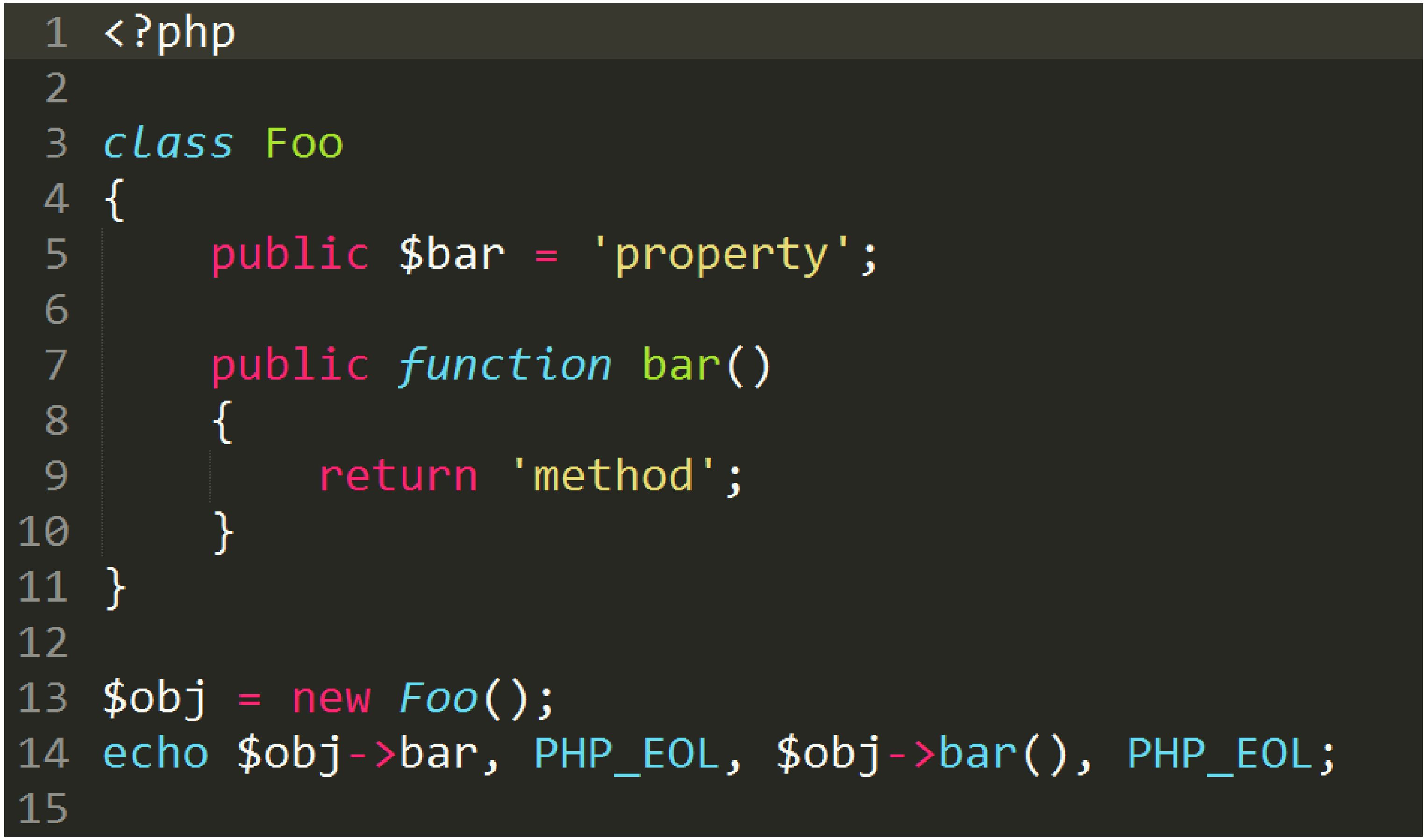
instanceof

used to determine whether a PHP variable is an instantiated object of a certain class



```php
<?php

class SomeClass()
{
    //
}

$obj = new SomeClass();

$exists = ($obj instanceof SomeClass);          // true
$exists = ($obj instanceof NonExistentClass);   // false

```

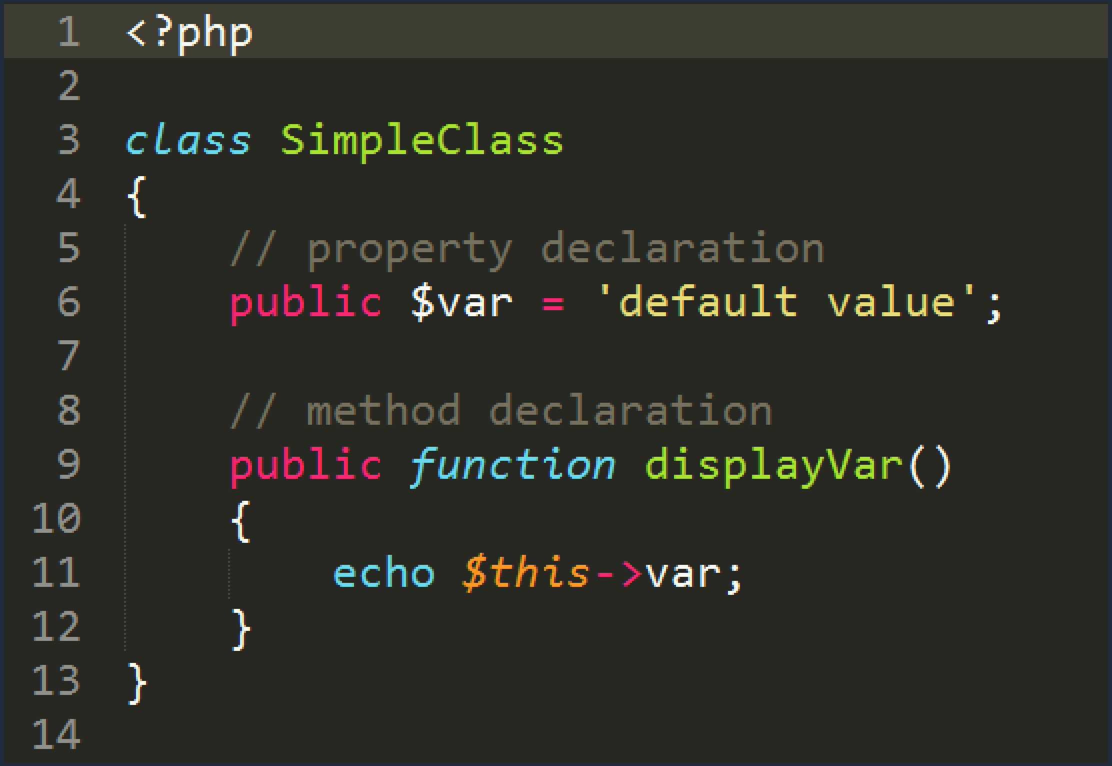
Properties and methods



```php
<?php

class Foo
{
    public $bar = 'property';

    public function bar()
    {
        return 'method';
    }
}

$obj = new Foo();
echo $obj->bar, PHP_EOL, $obj->bar(), PHP_EOL;

```

Pseudo-variable `$this`

`$this` is used to access a class object from *within the class*



```php
<?php

class SimpleClass
{
    // property declaration
    public $var = 'default value';

    // method declaration
    public function displayVar()
    {
        echo $this->var;
    }
}

```

Object Oriented Paradigms

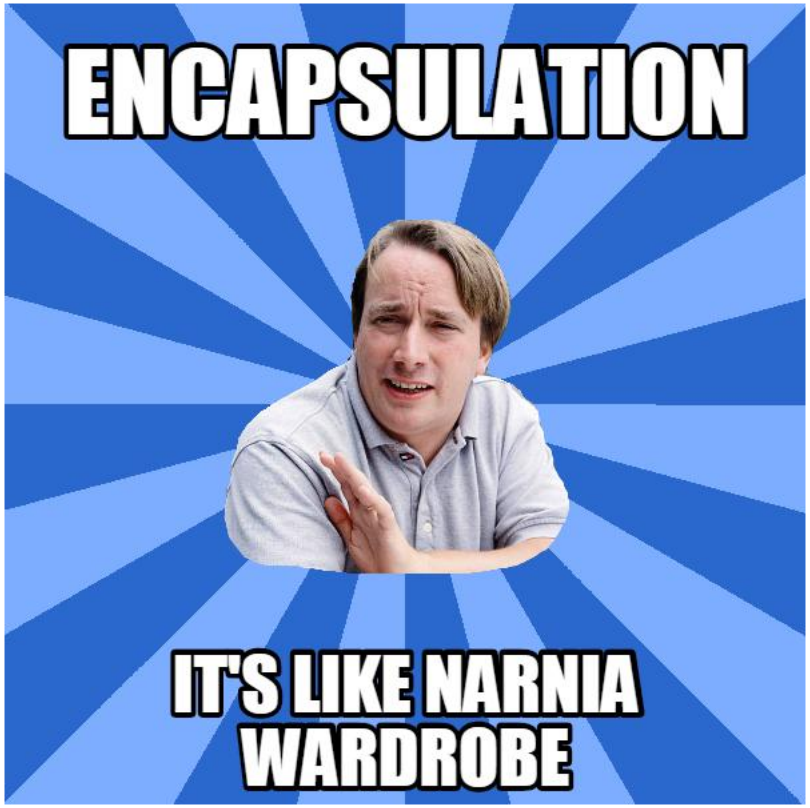
1. **Encapsulation**
2. **Inheritance**
3. **Polymorphism**
4. **Abstraction**
5. *Sending messages*
6. *Reusage*

Object Oriented Paradigms

Oh darling, all I want to hear are those three little words...
Encapsulation Inheritance Polymorphism

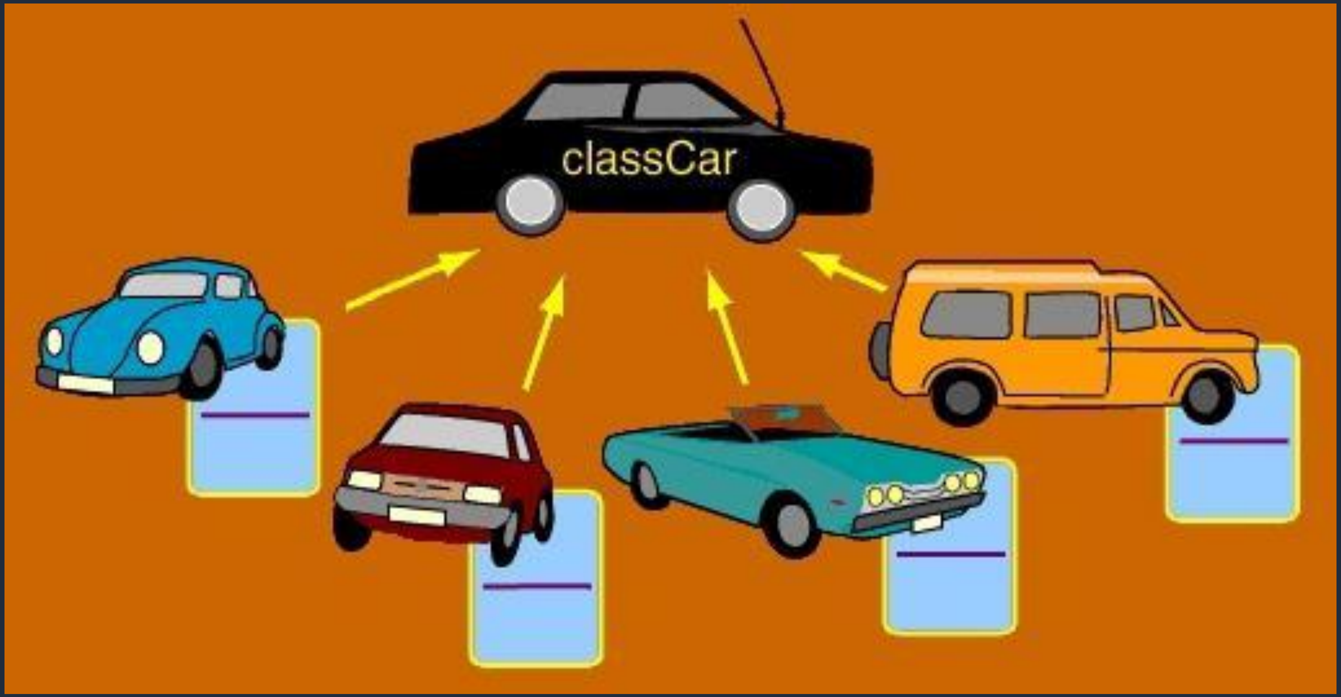
1. Encapsulation

Concealing parts of software systems



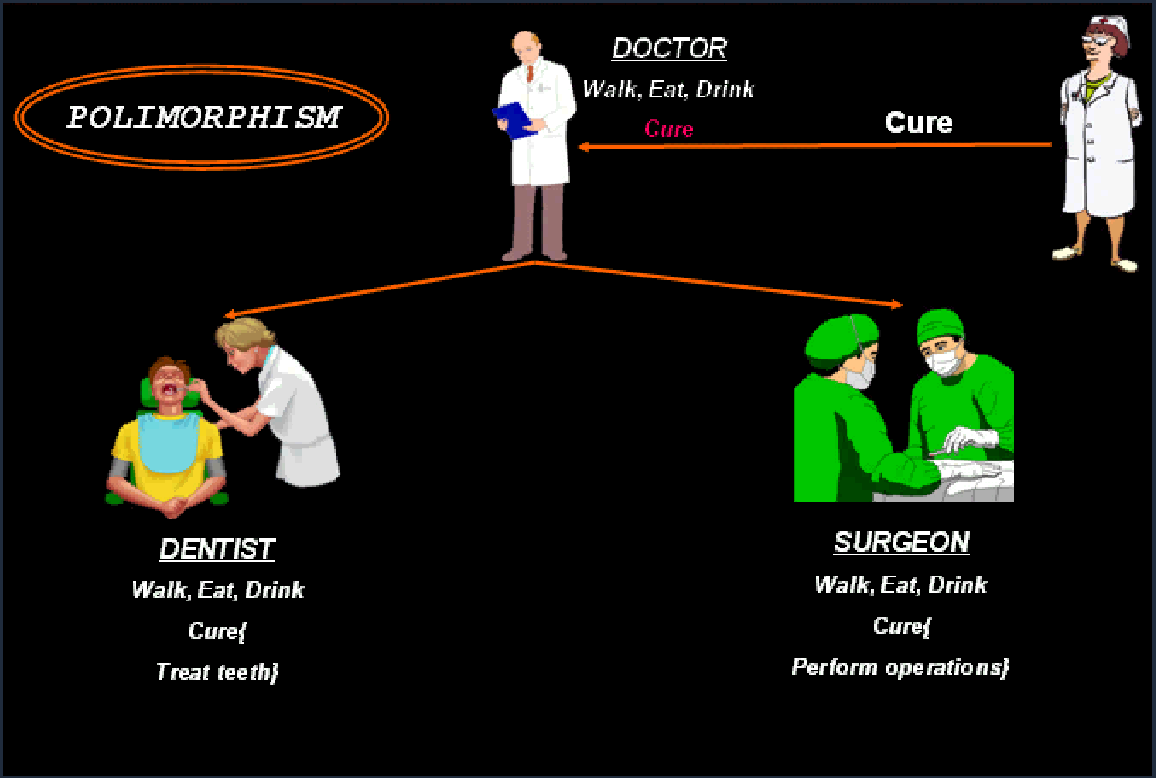
2. Inheritance

Create a new object based on the old one



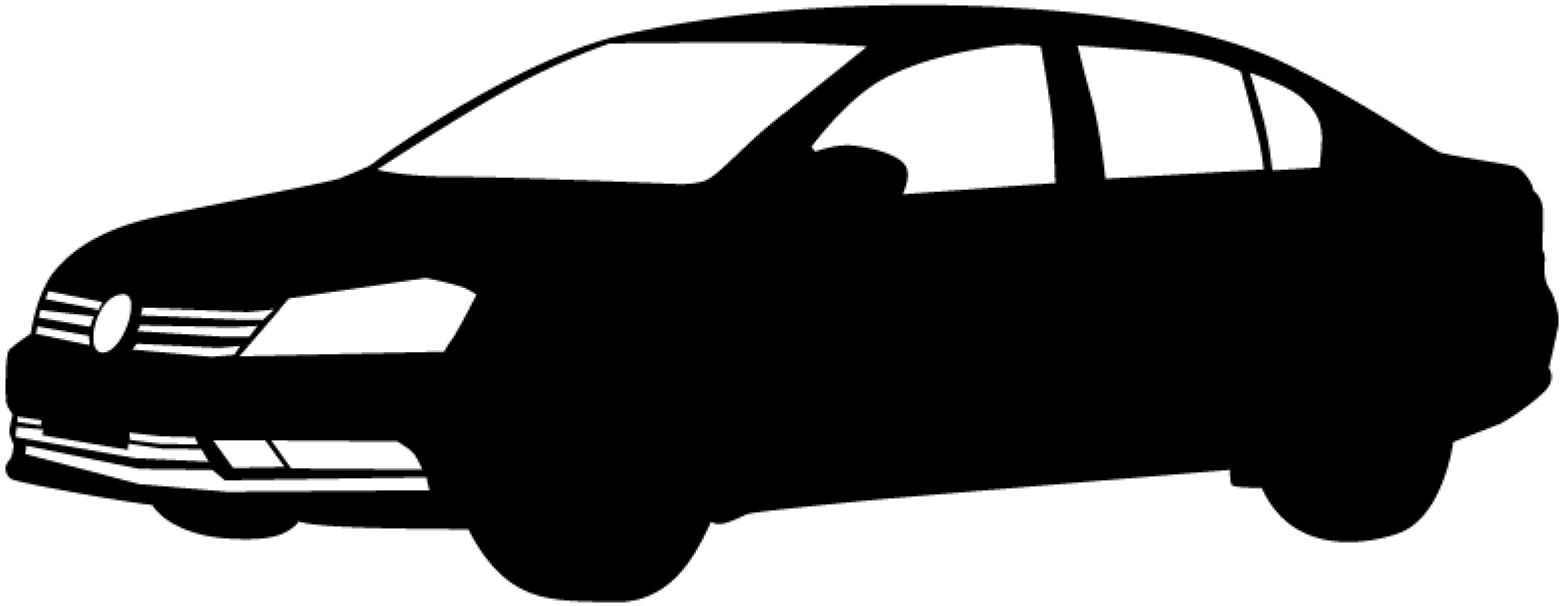
3. Polymorphism

The ability of an object to appear in different forms, in different cases



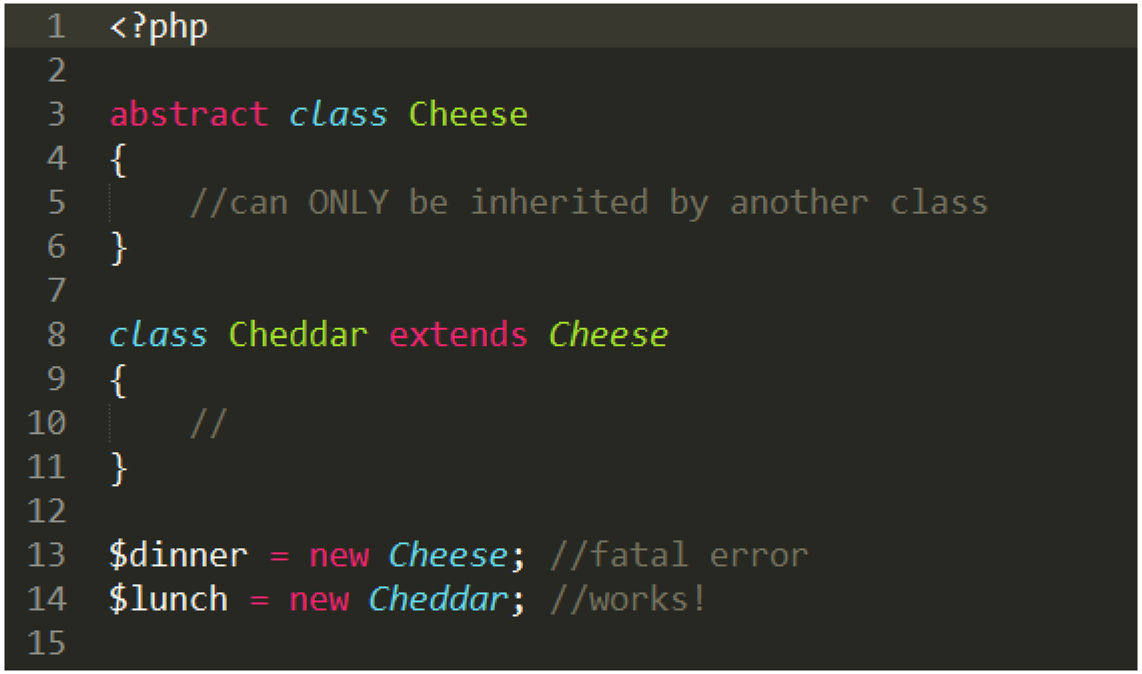
4. Abstraction

Giving an object characteristics that distinguish it from all other objects and clearly define its conceptual boundaries



4. Abstraction

Giving an object characteristics that distinguish it from all other objects and clearly define its conceptual boundaries

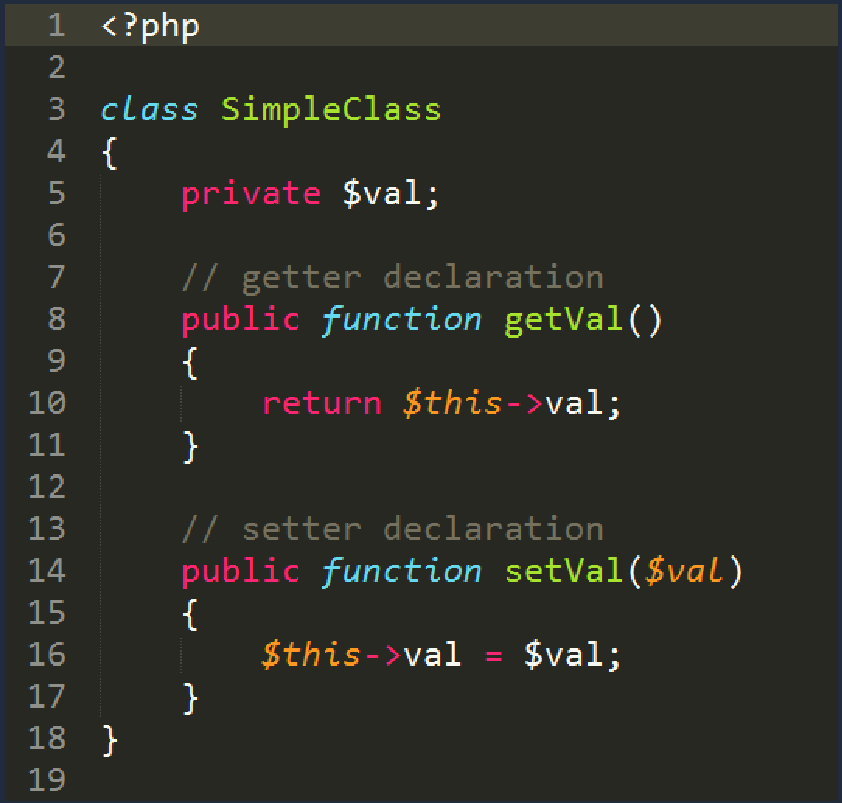


```php
<?php

abstract class Cheese
{
    //can ONLY be inherited by another class
}

class Cheddar extends Cheese
{
    //
}

$dinner = new Cheese; //fatal error
$lunch = new Cheddar; //works!

```

Visibility Modifiers

getter & setter

To work with **private** and **protected** properties, special methods are often created from the context of the object - *getters* and *setters*, which return the value of the closed property and set the value of the closed property respectively



```php
<?php

class SimpleClass
{
    private $val;

    // getter declaration
    public function getVal()
    {
        return $this->val;
    }

    // setter declaration
    public function setVal($val)
    {
        $this->val = $val;
    }
}

```

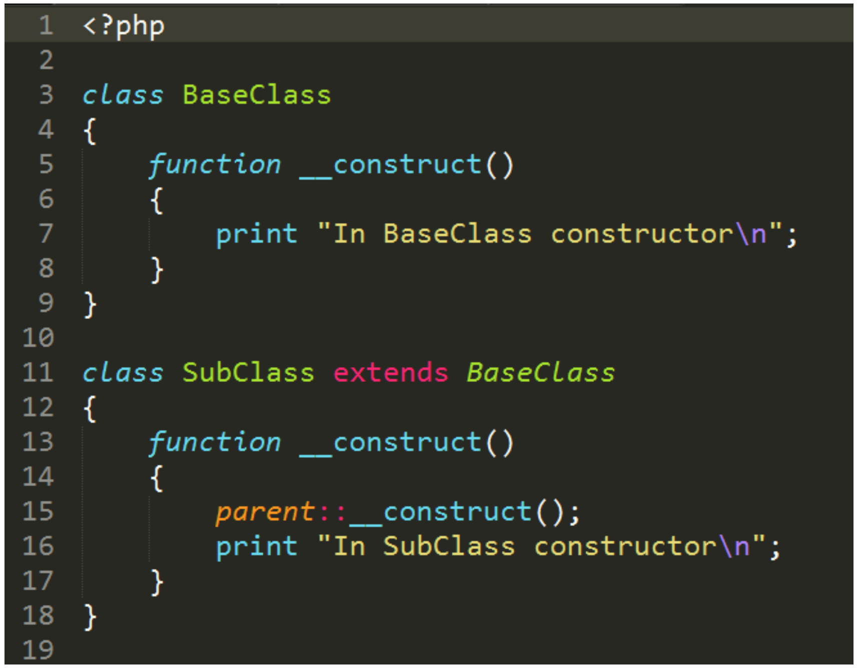
__set() & __get()

__get() method is utilized for reading data from inaccessible properties

__set() method is run when writing data to inaccessible properties

Constructors and Destructors

Classes which have a constructor method call this method on each newly-created object, so it is suitable for any initialization that the object may need before it is used

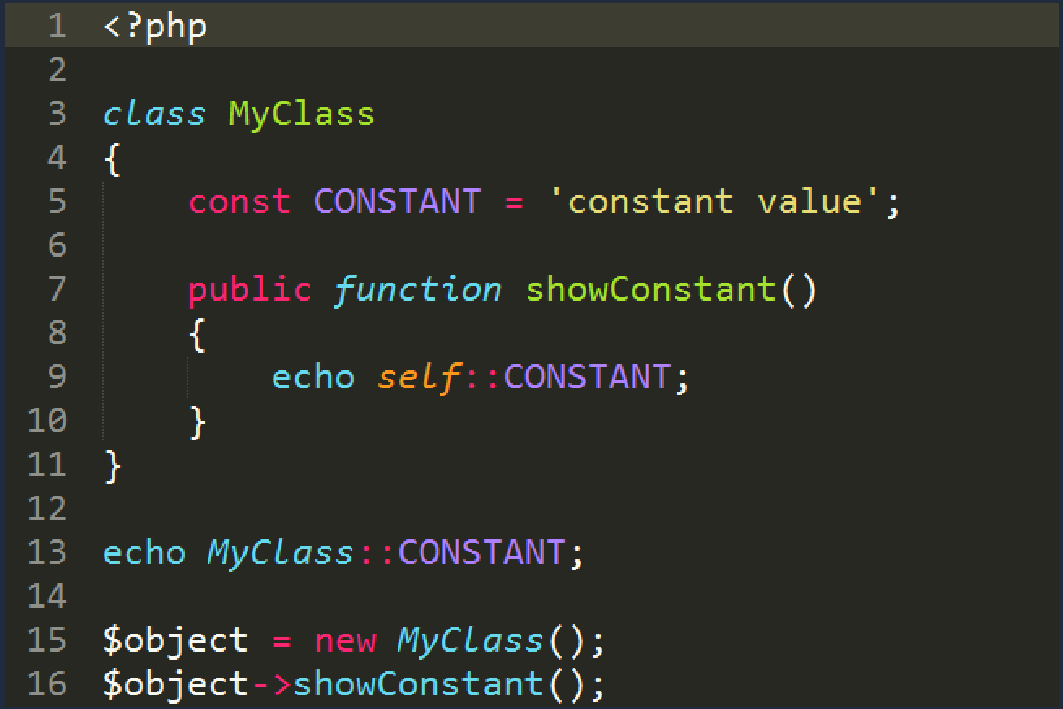


```php
<?php

class BaseClass
{
    function __construct()
    {
        print "In BaseClass constructor\n";
    }
}

class SubClass extends BaseClass
{
    function __construct()
    {
        parent::__construct();
        print "In SubClass constructor\n";
    }
}

```

In order to run a parent constructor, a call to `parent::__construct()` within the child constructor is required

Constants

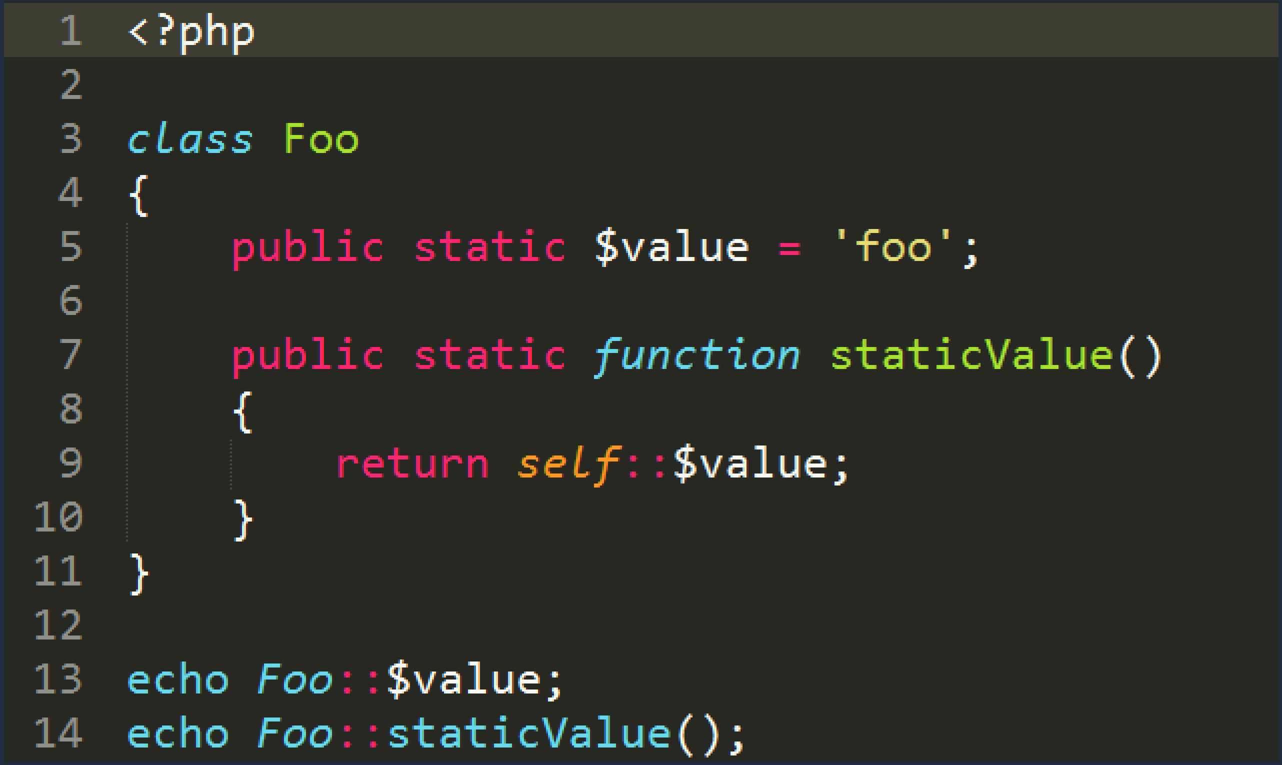
The value must be a constant expression, not (for example) a variable, a property, or a function call



```php
<?php

class MyClass
{
    const CONSTANT = 'constant value';

    public function showConstant()
    {
        echo self::CONSTANT;
    }
}

echo MyClass::CONSTANT;

$object = new MyClass();
$object->showConstant();
```

The default visibility of class constants is *public*

Static properties and functions

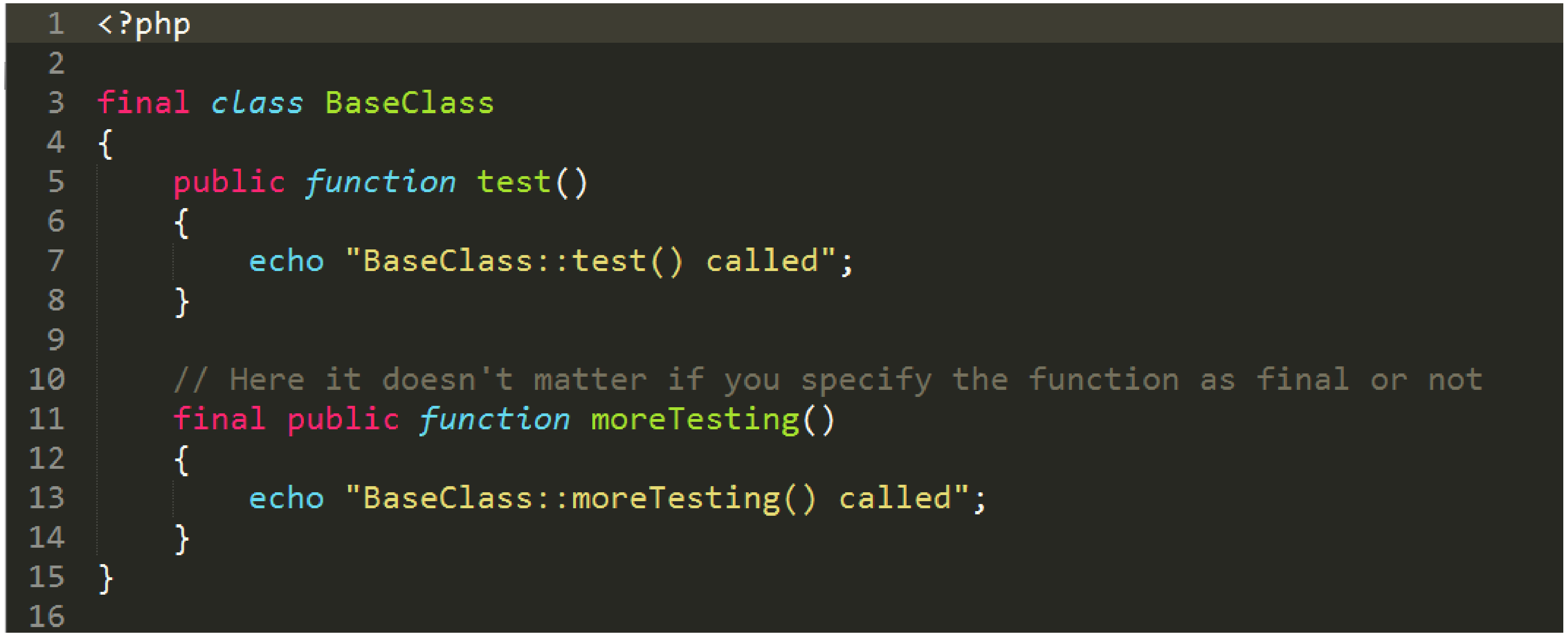


```php
<?php

class Foo
{
    public static $value = 'foo';

    public static function staticValue()
    {
        return self::$value;
    }
}

echo Foo::$value;
echo Foo::staticValue();
```

Declaring class properties or methods as static makes them accessible without needing an instantiation of the class

final keyword

Prevents child classes from overriding a method by prefixing the definition with **final**

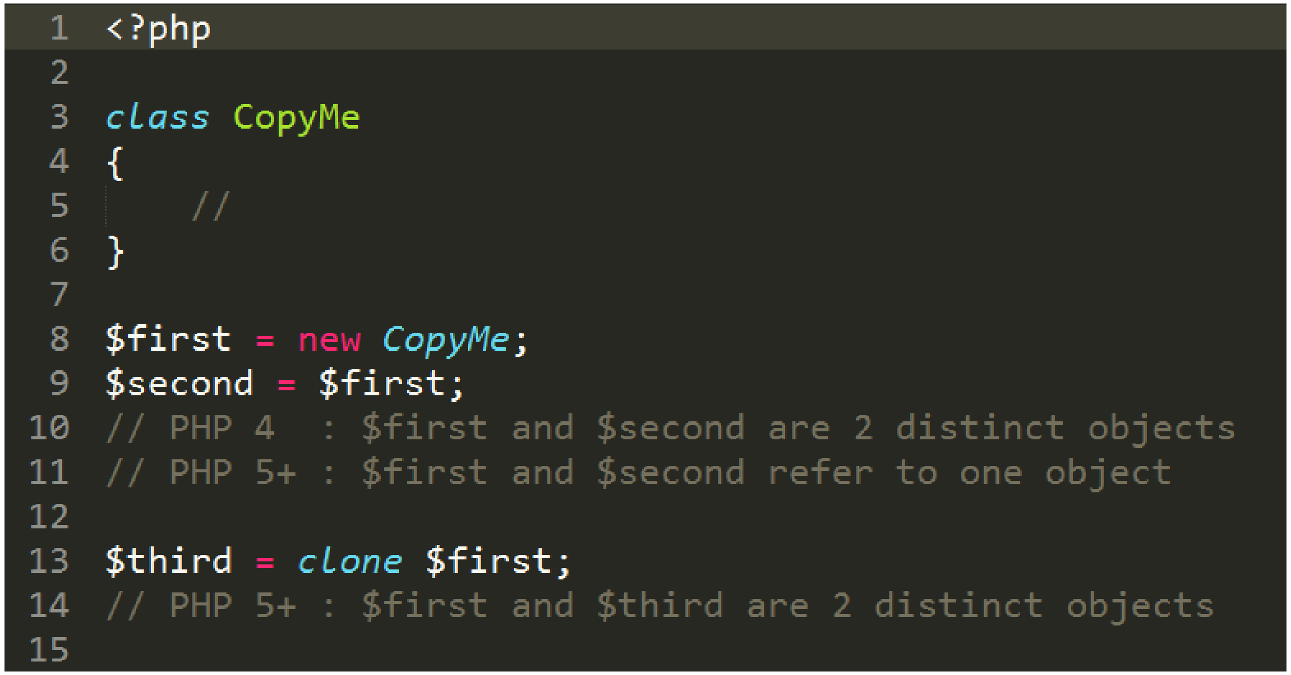


```php
<?php

final class BaseClass
{
    public function test()
    {
        echo "BaseClass::test() called";
    }

    // Here it doesn't matter if you specify the function as final or not
    final public function moreTesting()
    {
        echo "BaseClass::moreTesting() called";
    }
}

```

If the class itself is being defined **final** then it cannot be extended

Object Cloning

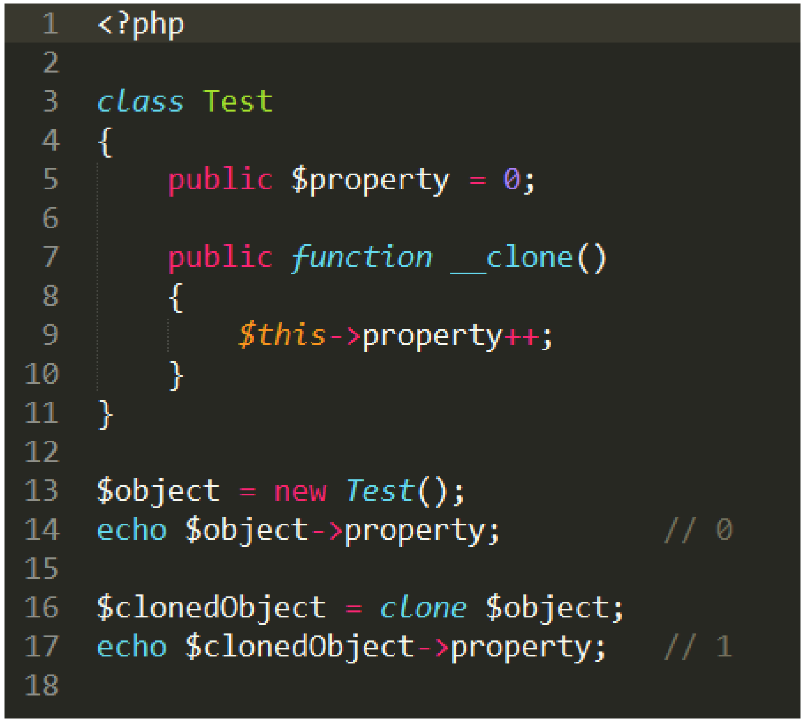
Since PHP 5, objects are always assigned and passed around by references



```php
<?php

class CopyMe
{
    //
}

$first = new CopyMe;
$second = $first;
// PHP 4  : $first and $second are 2 distinct objects
// PHP 5+ : $first and $second refer to one object

$third = clone $first;
// PHP 5+ : $first and $third are 2 distinct objects

```

Object Cloning

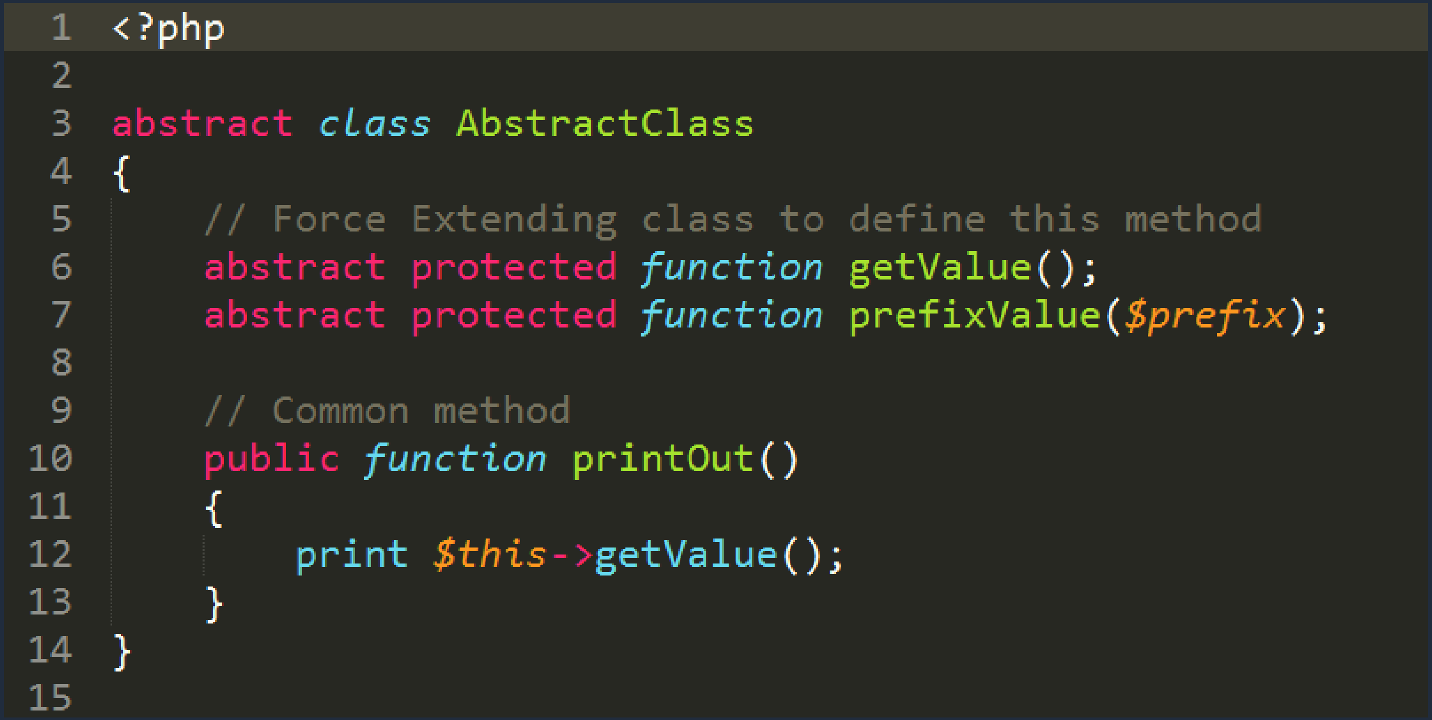
Once the cloning is complete, if a `__clone()` method is defined, then the newly created object's `__clone()` method will be called



```php
<?php

class Test
{
    public $property = 0;

    public function __clone()
    {
        $this->property++;
    }
}

$object = new Test();
echo $object->property;         // 0

$clonedObject = clone $object;
echo $clonedObject->property;   // 1

```

Class Abstraction

Classes defined as abstract may not be instantiated

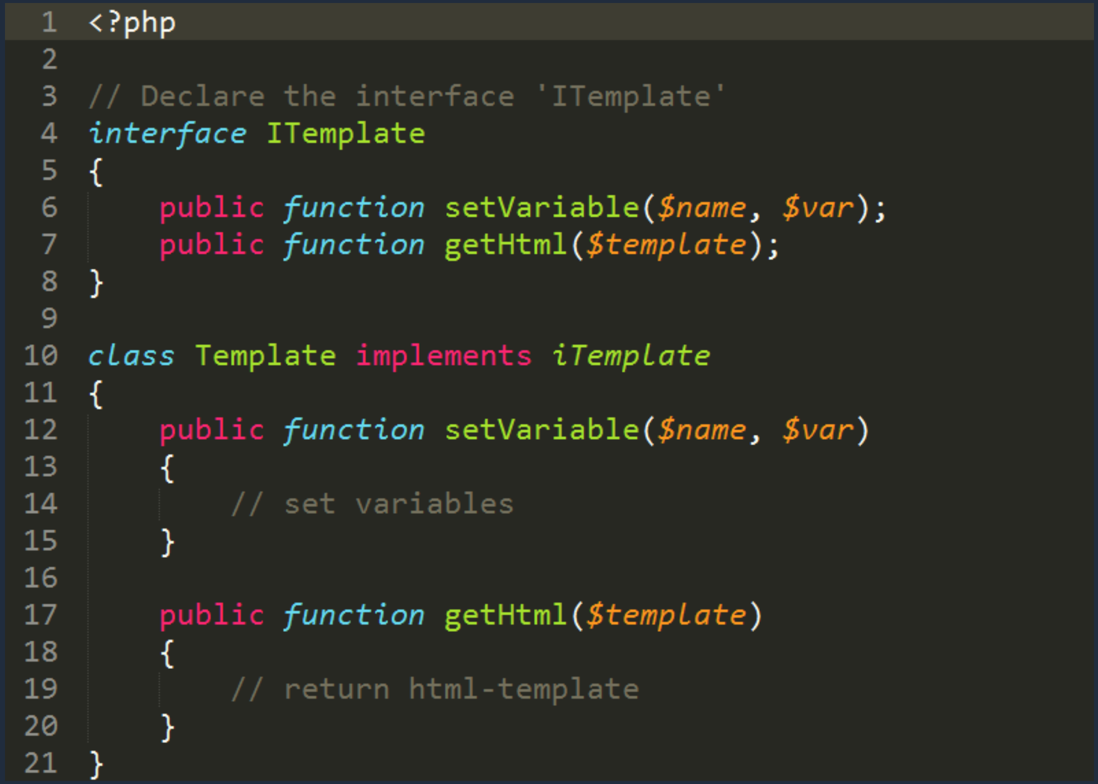


```php
<?php

abstract class AbstractClass
{
    // Force Extending class to define this method
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // Common method
    public function printOut()
    {
        print $this->getValue();
    }
}

```

Any class that contains at least one abstract method must also be abstract

Object Interfaces

specifies which methods a class must implement, without having to define how these methods are handled

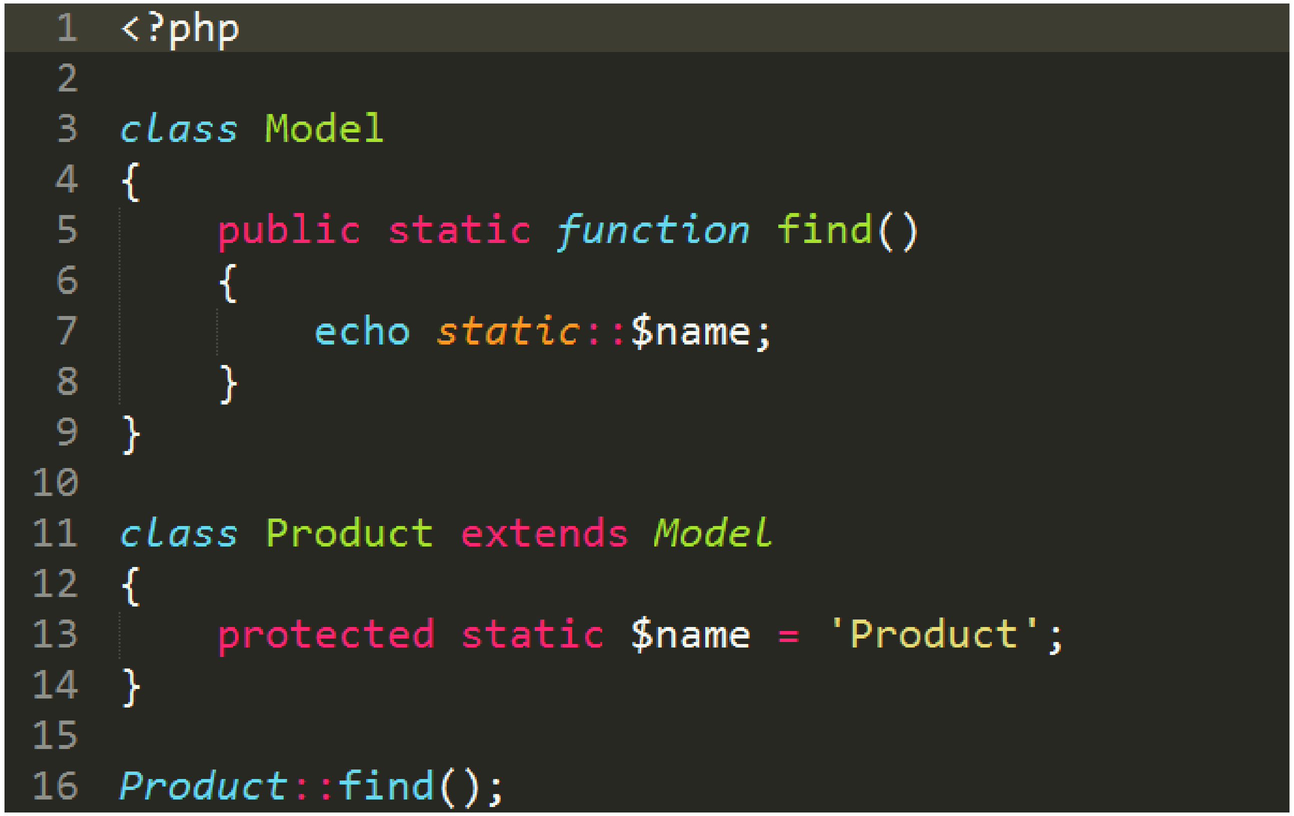


```php
<?php

// Declare the interface 'ITemplate'
interface ITemplate
{
    public function setVariable($name, $var);
    public function getHtml($template);
}

class Template implements iTemplate
{
    public function setVariable($name, $var)
    {
        // set variables
    }

    public function getHtml($template)
    {
        // return html-template
    }
}
```

Classes may implement more than one interface if desired by separating each interface with a comma

Late Static Bindings

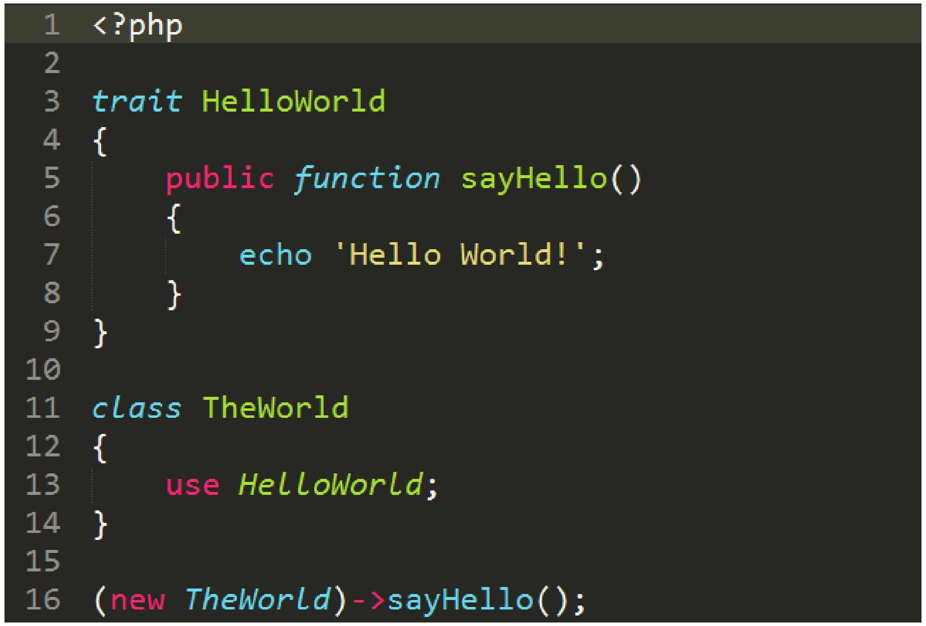
Can be used to reference the called class in a context of static inheritance



```php
<?php

class Model
{
    public static function find()
    {
        echo static::$name;
    }
}

class Product extends Model
{
    protected static $name = 'Product';
}

Product::find();
```

Traits

Traits are a mechanism for code reuse in single inheritance languages

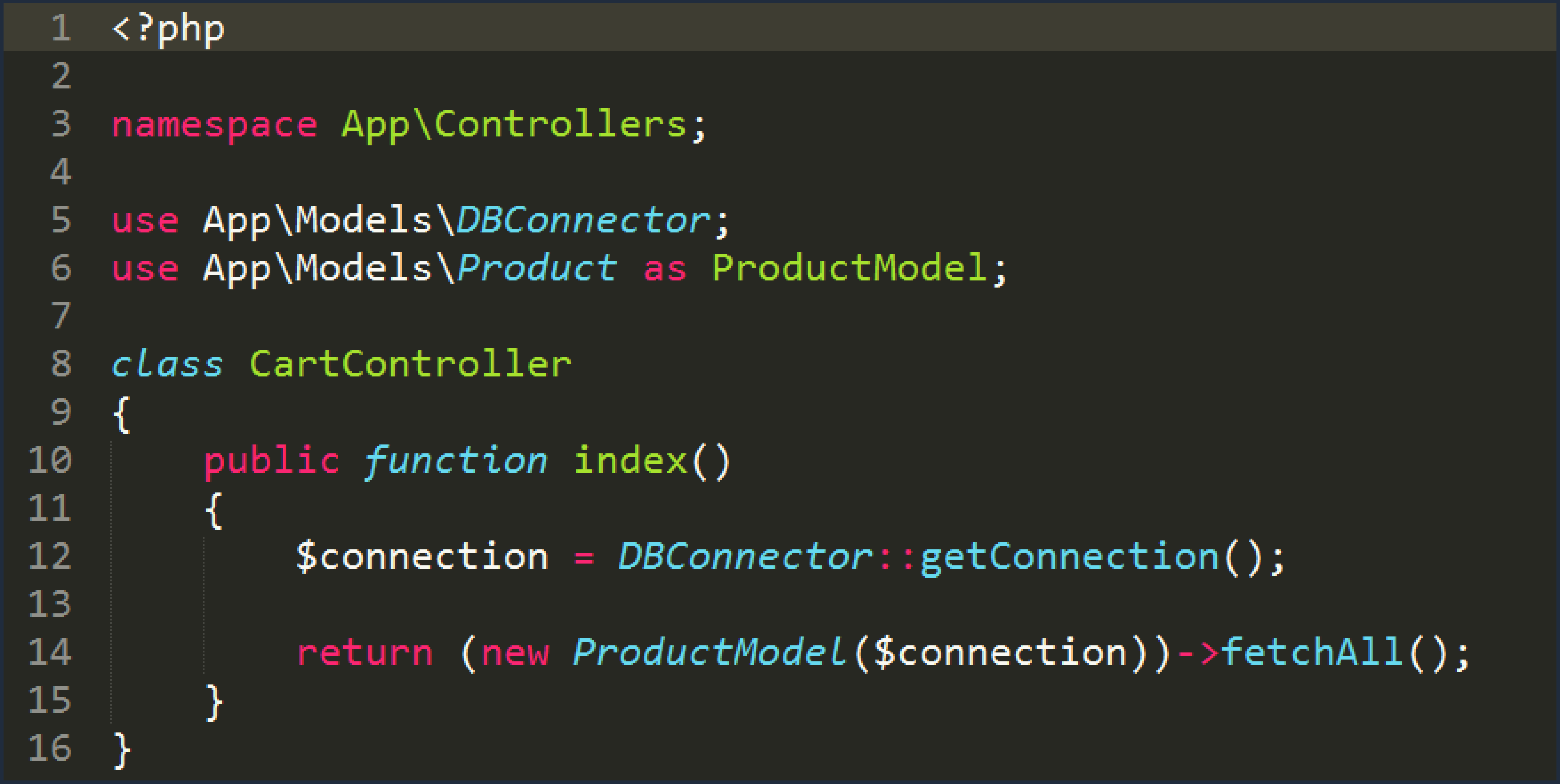


```php
<?php

trait HelloWorld
{
    public function sayHello()
    {
        echo 'Hello World!';
    }
}

class TheWorld
{
    use HelloWorld;
}

(new TheWorld)->sayHello();
```

A Trait is similar to a class, but only intended to group functionality in a fine-grained and consistent way

Namespaces

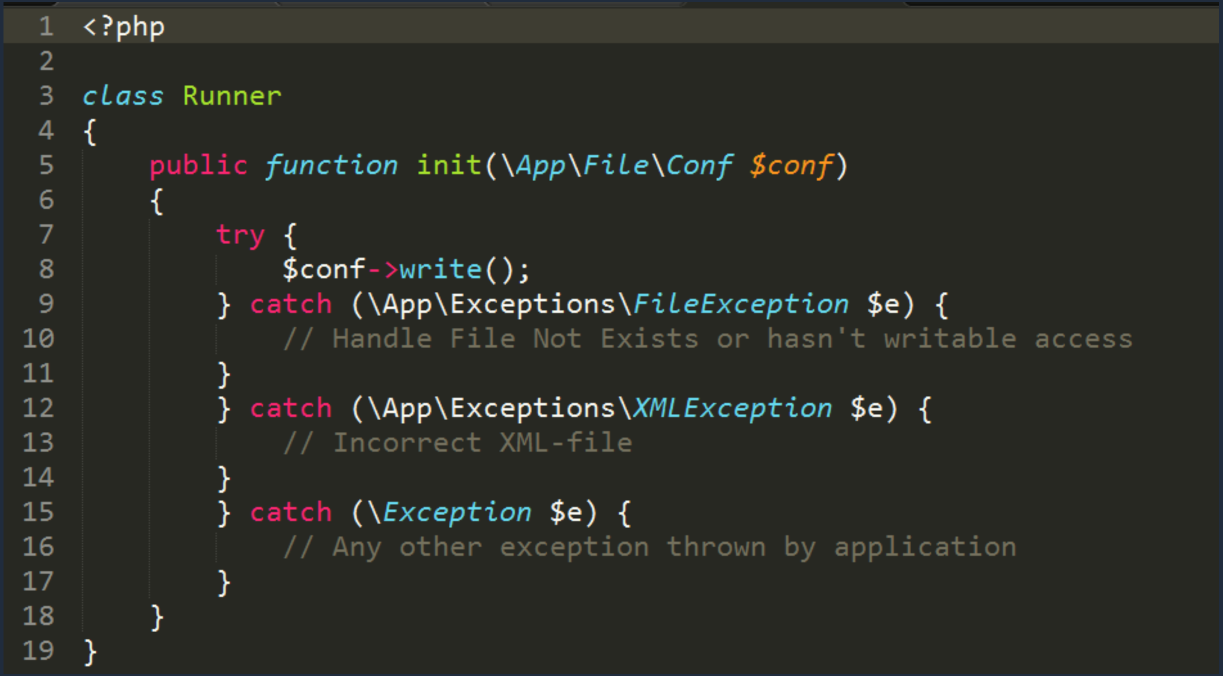
namespaces are a way of encapsulating items



```php
<?php

namespace App\Controllers;

use App\Models\DBConnector;
use App\Models\Product as ProductModel;

class CartController
{
    public function index()
    {
        $connection = DBConnector::getConnection();

        return (new ProductModel($connection))->fetchAll();
    }
}
```

Exceptions

An exception can be **throw**n, and caught ("**catch**ed") within PHP



```php
<?php

class Runner
{
    public function init(\App\File\Conf $conf)
    {
        try {
            $conf->write();
        } catch (\App\Exceptions\FileException $e) {
            // Handle File Not Exists or hasn't writable access
        }
        } catch (\App\Exceptions\XMLException $e) {
            // Incorrect XML-file
        }
        } catch (\Exception $e) {
            // Any other exception thrown by application
        }
    }
}
```

Multiple `catch` blocks can be used to catch different classes of exceptions

Code within the **finally** block will always be executed after the `try` and `catch` blocks

Composer

```
composer update  # Update all remote repositories
composer install # Install updates from composer.phar
```

Useful resources

- Visibility
- Object Oriented Paradigm (RU)
- Object Oriented Paradigm
- Introduction to OOP in PHP
- OOP basics
- Namespaces
- PHP OOP at Devionity (RU)
- Composer

Thanks for your attention

Q & A

Let’s stay in touch

LinkedIn™

YAHOO!® MAIL

larapulse
Never stop learning

twitter

Instagram

www.web-academy.com.ua