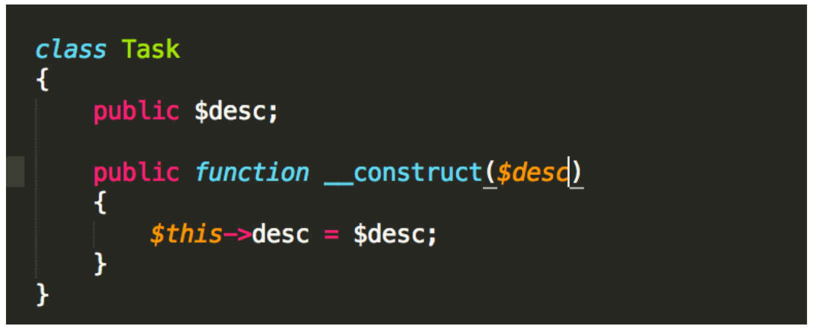
OOP PHP Understanding From Laracast

Documented By HlaingTinHtun

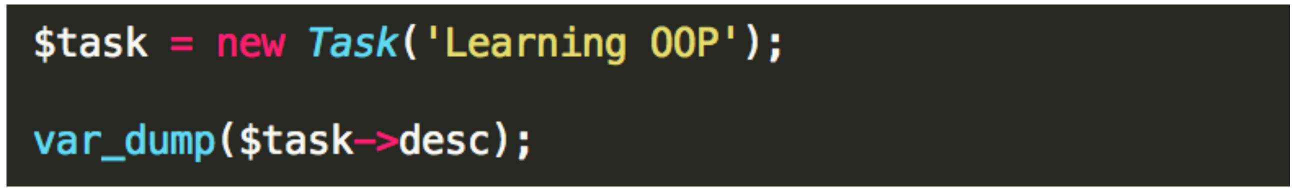
Classes

- Make the class and build a constructor within this class
- set the property within that class and assign these variables



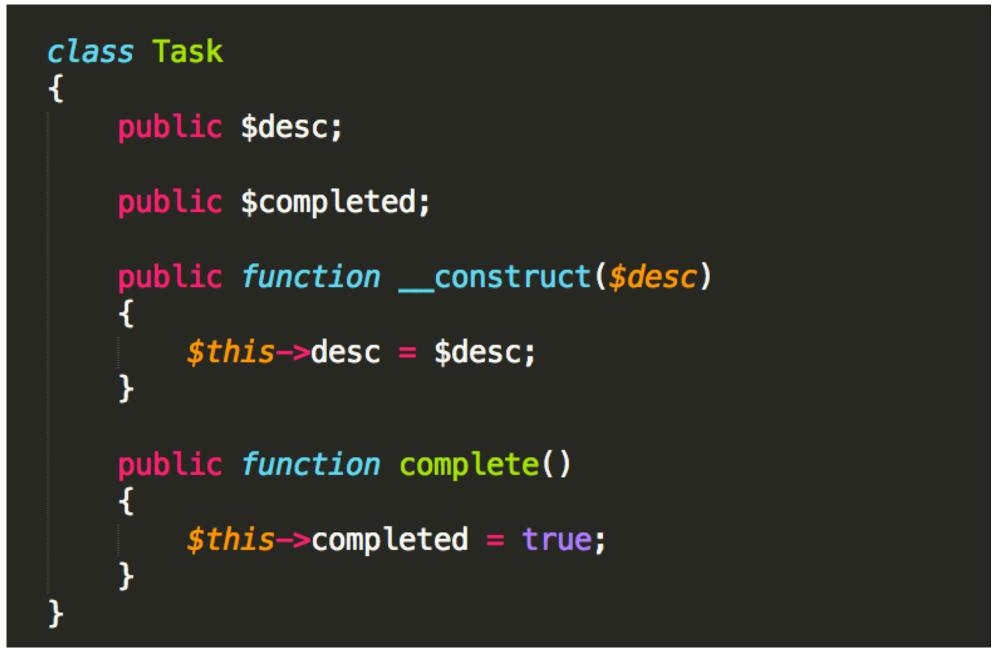
```php
class Task
{
    public $desc;

    public function __construct($desc)
    {
        $this->desc = $desc;
    }
}
```

- Then you can call you can call simply by requesting class name

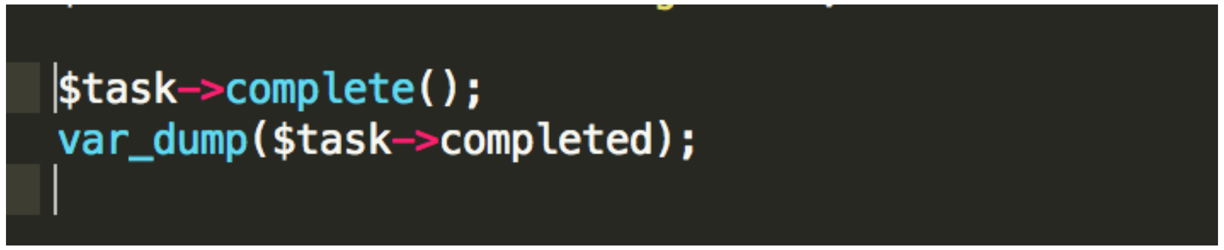


```php
$task = new Task('Learning OOP');

var_dump($task->desc);
```

-You can also call by function name directly



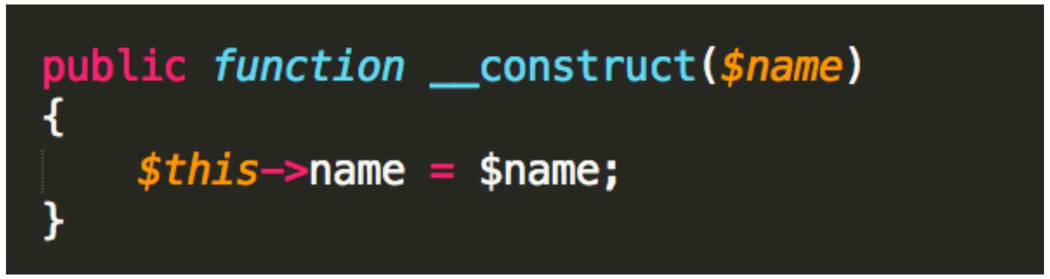
```php
class Task
{
    public $desc;

    public $completed;

    public function __construct($desc)
    {
        $this->desc = $desc;
    }

    public function complete()
    {
        $this->completed = true;
    }
}
```



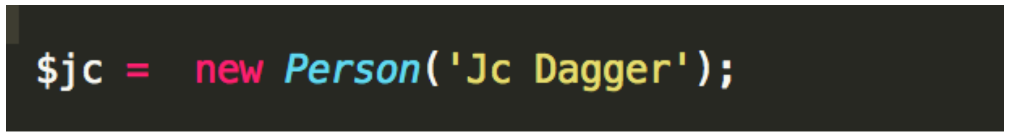
```php
$task->complete();
var_dump($task->completed);
```

Getters and Setters

Building the constructor within Person class

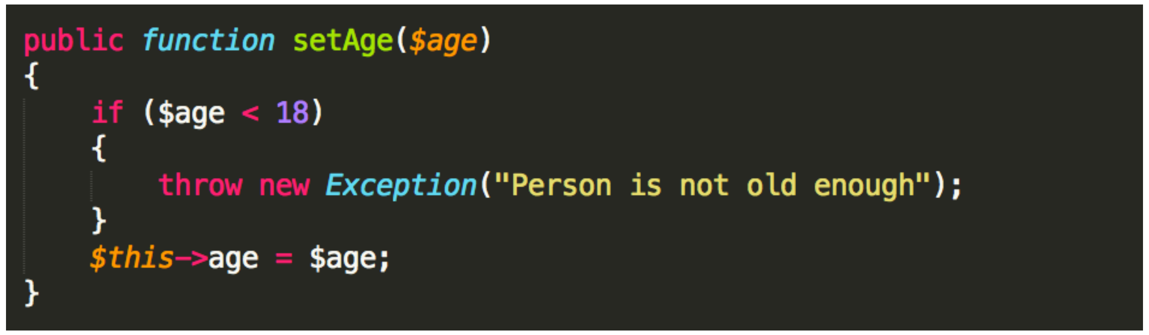


```
public function __construct($name)
{
    $this->name = $name;
}
```

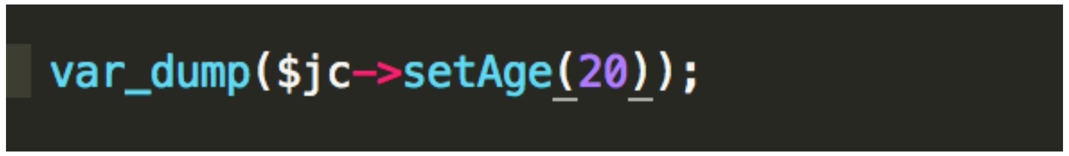


```
$jc =  new Person('Jc Dagger');
```

- Setting up the data in class and calling back.



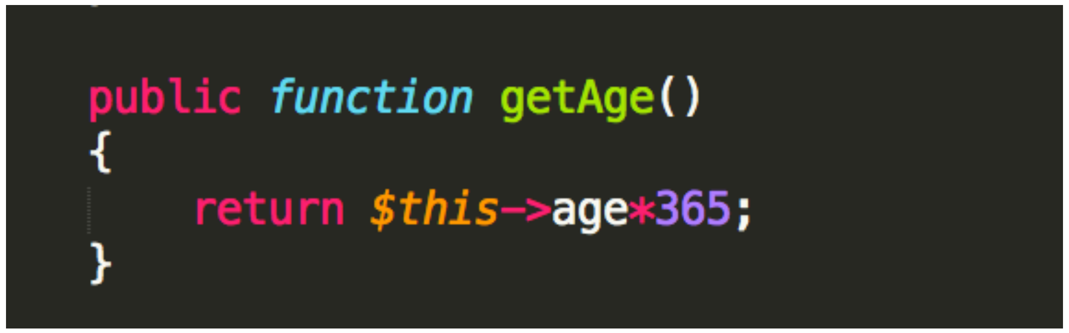
```
public function setAge($age)
{
    if ($age < 18)
    {
        throw new Exception("Person is not old enough");
    }
    $this->age = $age;
}
```



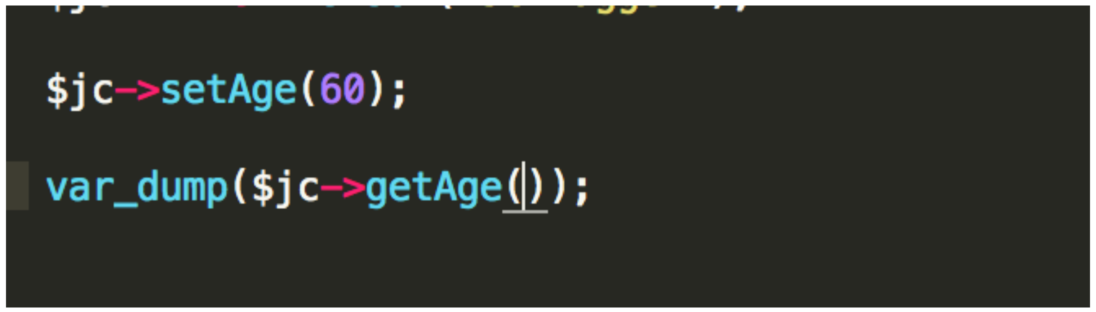
```
var_dump($jc->setAge(20));
```

- Getting Up the data from getter

- Then calling Back



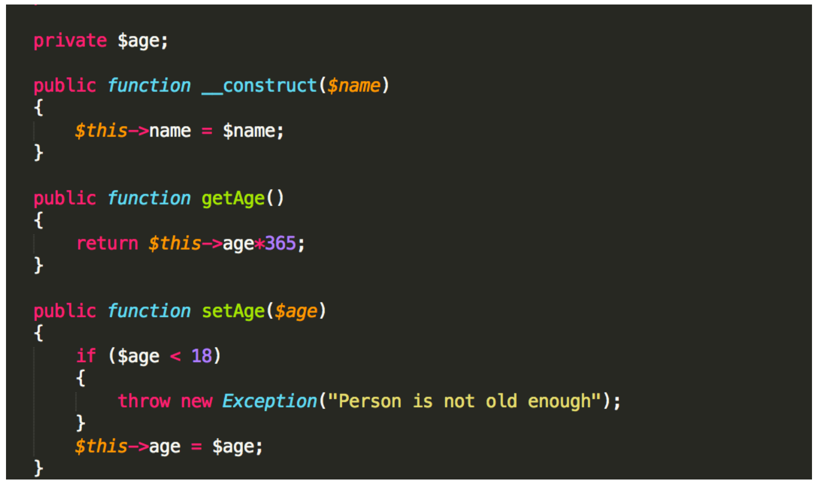
```
public function getAge()
{
    return $this->age*365;
}
```



```
$jc->setAge(60);
var_dump($jc->getAge());
```

Encapsulation

- Difference between public, protected and private.
- We can call public function from outside of the class directly
- But we can't call protected and private method outside of the class directly.
- So We can protect properties or function that we want to hide.
- For Example, In last getter and setter section, lets say we want to protect persons' age . we can do like this



```php
private $age;

public function __construct($name)
{
    $this->name = $name;
}

public function getAge()
{
    return $this->age*365;
}

public function setAge($age)
{
    if ($age < 18)
    {
        throw new Exception("Person is not old enough");
    }
    $this->age = $age;
}
```

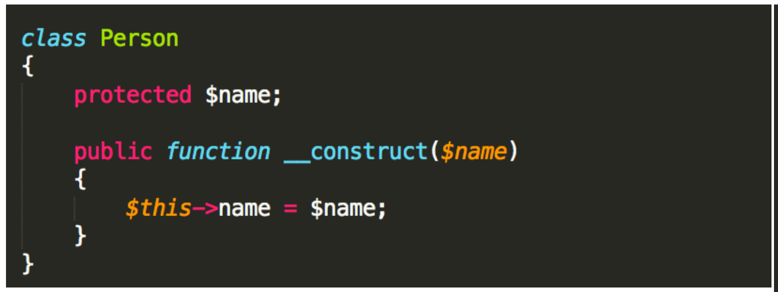
Inheritance

- Lets say two classes, Senior class and Junior class. Junior class can extend the data from the class of senior.
- Also the junior class can be easily overwritten although this class extend senior class.
- Advantage of using is that we want to dry our codes
- When we have two subclasses that need to do same functionality , we can keep it in main class and sub class extends the main class.
- Then they can access the functionality.
- These are the overall view of inheriteance.

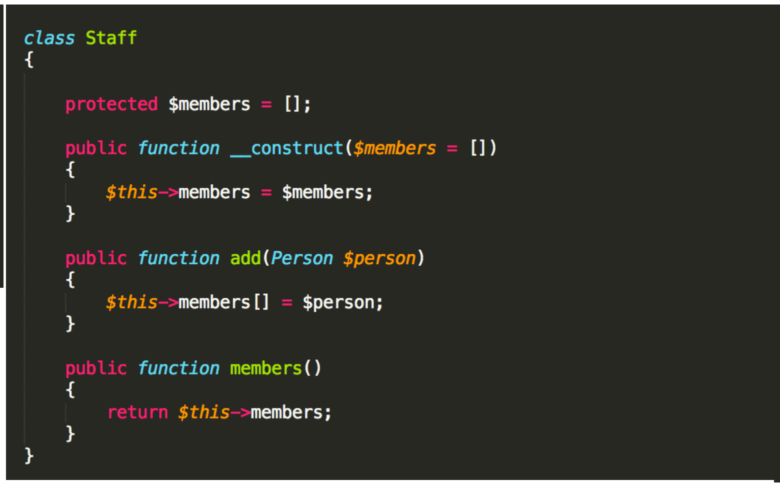
Messages

- Core component of OOP
- In most a little bit huge projects, there can be alot of classes and objects
- There can be complex understanding of code . Thus why we use messages.
- This can help us to connect other subclasses in their specific class and can know how objects

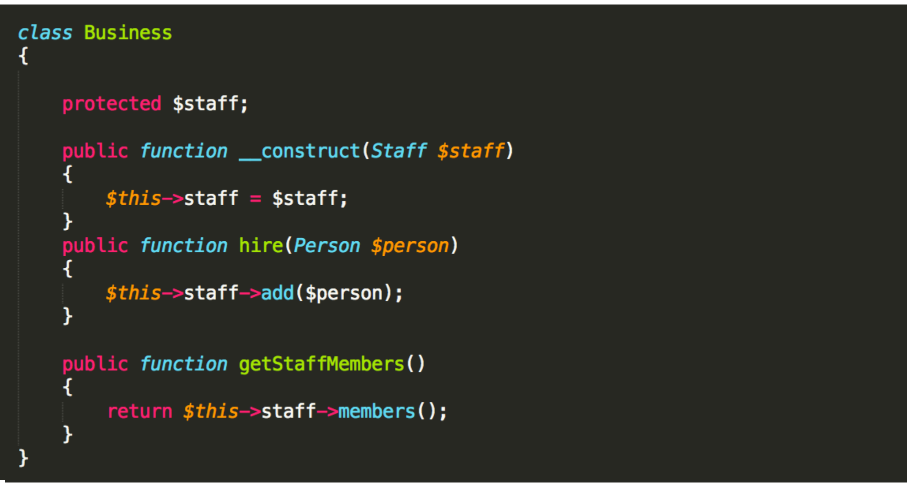
  communicate each other
- See Examples Below



```php
class Person
{
    protected $name;

    public function __construct($name)
    {
        $this->name = $name;
    }
}
```

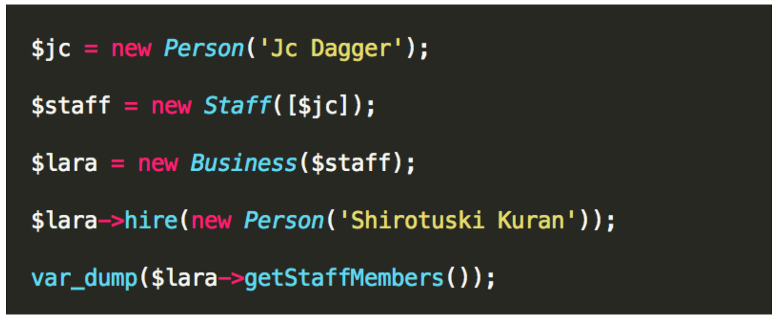


```php
class Staff
{

    protected $members = [];

    public function __construct($members = [])
    {
        $this->members = $members;
    }

    public function add(Person $person)
    {
        $this->members[] = $person;
    }

    public function members()
    {
        return $this->members;
    }
}
```



```php
class Business
{

    protected $staff;

    public function __construct(Staff $staff)
    {
        $this->staff = $staff;
    }
    public function hire(Person $person)
    {
        $this->staff->add($person);
    }

    public function getStaffMembers()
    {
        return $this->staff->members();
    }
}
```

Communicate and print out



```php
$jc = new Person('Jc Dagger');

$staff = new Staff([$jc]);

$lara = new Business($staff);

$lara->hire(new Person('Shirotuski Kuran'));

var_dump($lara->getStaffMembers());
```

Namespacing-Autoloading-and-PSR4

- When we need to load the other class file in a php file, we call with require functions
- In this time, we can make better than before with the help of composer
- Make composer.json file and add psr-4 autoloading that need to declare the key and specific root folder namespace
- Then you can generate the autoload classes
- Finally you just need to call autoloader class in you index file
- After that put the file namespaecs and can connect each file

Statics and Constants

- Statics methods are shared, they are bounded to any specific objects.
- So it can be riskful when you use in dynamic objects. It can print out incorrect result
- And static method can be accessible by scope resolution operator (::)
- A property declared as static cannot be accessed with an instantiated class object (though a static method can).
- Constants are like variables except that once they are defined they cannot be changed or undefined.

Interfaces

- Programs to an interface not a implemenatation
- No Logics are used in interface
- SubClasses that are similar functionality implements the interface
- Why we use interface is that we need to be less busy when code customisation came across
- We don’t need to change the all class anymore
- Just need to add a new concrete class or modify that specific class
- Advantages of using interfaces is to check back or review back easily for the complicated code project or may be your old projects XD.

Interface-versus-Abstract

Difference between Abstract Class and Interface

Abstract Classes

- An abstract class can provide some functionality and leave the rest for derived class.
- The derived class may or may not override the concrete functions defined in base class.
- The child class extended from an abstract class should logically be related

Interface

- An interface cannot contain any functionality. It only contains definitions of the methods.
- The derived class MUST provide code for all the methods defined in the interface.
- Completely different and non-related classes can be logically be grouped together using an interface

Scope-and-Context

- Just making sure that we need to make scopes and contextes before we make the project
- It can be include most injections like constructor injection and method injections
- Then we cover in last lesson like messages to specific objects
- And name spacing and autoloading, interfaces
- They are the scopes and contextes that we need to make in your projects that we want to be long term robust projects.

Thanks You For Reading

- I just make that pdf for whom are need to know about oop
- And who wanna brush up or review back for oop concept
- This can be also suitable for whom are going to learn laravel, they gonna need to know some basic steps of oop concepts