Functional Programming Principles in PHP - Functors

Kai Sassnowski - kai-sassnowski.com - @warsh33p

Disclaimer

Do not try this at home

Things we know

[]

array_map

```php
array_map(function ($x) {
    return $x + 1;
}, [1, 2, 3, 4]);

// [2, 3, 4, 5]
```

Things we might not know

Mapping is not specific to lists

`array_map` *lifts* a function

```php
$add1 = function ($x) { $x + 1; };
```

- Takes a single value and returns a single value.

```php
$add1 = function ($x) { $x + 1; };
```

- Takes a single value and returns a single value.
- Does *not* operate on lists

```php
$add1 = function ($x) { $x + 1; };
```

- Takes a single value and returns a single value.
- Does *not* operate on lists
- `array_map` makes it work on lists

```php
$add1 = function ($x) { $x + 1; };
```

- Takes a single value and returns a single value.
- Does *not* operate on lists
- `array_map` makes it work on lists

```php
array_map($add1, [1, 2, 3, 4]);

// [2, 3, 4, 5]
```

Some Notation

Single argument

```haskell
add1 :: Int -> Int
```

Single argument

```haskell
add1 :: Int -> Int
```

Multiple arguments

```haskell
add :: Int -> Int -> Int
```

Single argument

```
add1 :: Int -> Int
```

Multiple arguments

```
add :: Int -> Int -> Int
```

Higher Order function

```
map :: (a -> b) -> [a] -> [b]
```

Functors

```haskell
class Functor f where
    fmap :: (a -> b) -> f a -> f b
```

Applied to Lists

```
fmap :: (a -> b) -> [a] -> [b]
fmap :: callable -> array -> array
```

```php
interface Functor
{
    public function fmap(callable $fn): Functor;
}
```

Instead of

```
fmap($function, $functor);
```

we say

```
$functor->fmap($function);
```

```php
class Arr implements Functor
{
    private $array;

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

    public function fmap(callable $fn): Functor
    {
        return new self(array_map($fn, $this->array));
    }
}
```

```php
$add1 = function ($x) { return $x + 1; };

$myArr = new Arr([1, 2, 3, 4]);

$mapped = $myArr->fmap($add1);

// $mapped is now Arr([2, 3, 4, 5])
```

Neat

```
abstract class Result implements Functor { }
```

```php
abstract class Result implements Functor { }
```

```php
class Success extends Result
{
    private $value;

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

    public function fmap(callable $fn): Functor
    {
        return new static($fn($this->value));
    }
}
```

```php
abstract class Result implements Functor { }
```

```php
class Failure extends Result
{
    public function fmap(callable $fn): Functor
    {
        return $this;
    }
}
```

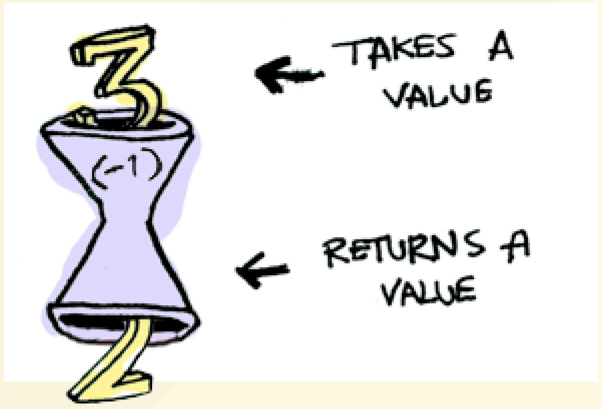
```php
$add1 = function ($x) { return $x + 1; };

$success = new Success(2);
$success->fmap($add1);
// Success(3)
```

```php
$add1 = function ($x) { return $x + 1; };

$success = new Success(2);
$success->fmap($add1);
// Success(3)
```

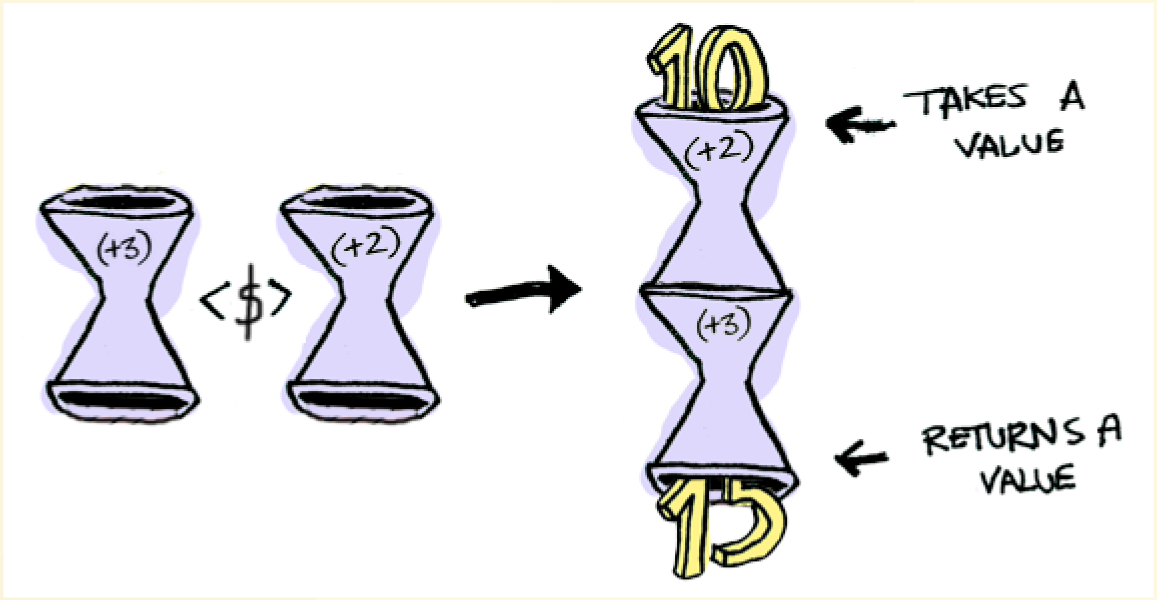
```php
$add1 = function ($x) { return $x + 1; };

$failure = new Failure();
$failure->fmap($add1);
// Failure()
```

`add1` never even got called.

Functions are Functors too!



(http://adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html)



(http://adit.io/posts/2013-04-17-functors,_applicatives,_and_monads_in_pictures.html)

It's function composition!

```php
class Fn implements Functor
{
    private $fn;

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

    public function fmap(callable $fn): Functor
    {
        return new self(function ($arg) use ($fn) {
            return call_user_func($this->fn, $fn($arg));
        });
    }

    public function __invoke($arg)
    {
        return call_user_func($this->fn, $arg);
    }
}
```

```php
$add1 = function ($x) { return $x + 1; };

$double = new Fn(function ($x) { return $x * 2; });

$add1AndThenDouble = $double->fmap($add1);
// Fn(function ...)

$add1AndThenDouble(5);
// 12
```

NO ONE MAN SHOULD HAVE ALL THAT POWER

```php
$add1 = function ($x) { return $x + 1; };
```

```php
$arr = new Arr([1, 2, 3, 4]);
$arr->fmap($add1);
// Arr([2, 3, 4, 5])
```

```php
$add1 = function ($x) { return $x + 1; };
```

```php
$arr = new Arr([1, 2, 3, 4]);
$arr->fmap($add1);
// Arr([2, 3, 4, 5])
```

```php
$success = new Success(5);
$success->fmap($add1);
// Success(6)
```

```php
$add1 = function ($x) { return $x + 1; };
```

```php
$arr = new Arr([1, 2, 3, 4]);
$arr->fmap($add1);
// Arr([2, 3, 4, 5])
```

```php
$success = new Success(5);
$success->fmap($add1);
// Success(6)
```

```php
$double = new Fn(function ($x) { return $x * 2; });
$double->fmap($add1);
// Fn($double($add1($x)))
```

```php
$add1 = function ($x) { return $x + 1; };
```

```php
$arr = new Arr([1, 2, 3, 4]);
$arr->fmap($add1);
// Arr([2, 3, 4, 5])
```

```php
$success = new Success(5);
$success->fmap($add1);
// Success(6)
```

```php
$double = new Fn(function ($x) { return $x * 2; });
$double->fmap($add1);
// Fn($double($add1($x)))
```

```php
$arr->fmap($double->fmap($add1));
// new Arr([4, 6, 8, 10])
```

Now what?

Thanks for listening!