Lean and Functional Domain Modelling

Marcello Duarte

@_md

Summer 2017

WEB SUMMER CAMP

marcello duarte

INVIQA

WINNER 2014 AGILE UK AWARDS

@PhunkiePhp

#websc

@_md

What will I learn

- Functional thinking
- Algebraic data types
- Tactical modelling techniques
- Domain objects life cycle in FP

Further topics recommended

Strategical Domain modelling
Messaging/Events/Service between BCs
Reactive programming

Clarify

- Is PHP functional?
- Is the future functional?
- A word on preprocessors
- FP Complexity

Functional Programming
Algebraic Data Types
Domain modelling
Domain blocks life cycle

Agenda

Making the most

- Silence comms
- Tweet loads #websc
- CoC
- Be on Time
- Participate!

#websc

@_md

Course Progress

Lean and Functional Domain Modelling

Lesson 1 – Short Intro to FP

Lesson 2 – PHP Functional Features

Lesson 3 – Algebraic Data Types

Lesson 4 – Lead Domain Modelling

Lesson 5 - Domain blocks life cycle

what is functional programming?

programming with functions.

what is a function?

$$f(x) = y$$

pure functions

an expression involving a pure function is

referentially transparent

referentially transparent expression

{

can be replaced by its value
referentially transparent expression
without modifying the results

```php
$x = 42;

$y = $x + 3; // $y == 45
```

```php
$x = 42;

$y = $x + 3; // $y == 45

$y = 42 + 3; // $y == 45
```

```php
$x = 42;

$y = $x + 3; // $y == 45

$y = 42 + 3; // $y == 45
```

referentially transparent

a function is pure when it can always be replaced by the same value

pure functions

pure functions

#1 for every x, always same y

#2 no side effects

Exercise

```php
function square(float $x): float
{
    return $x * $x;
}
```

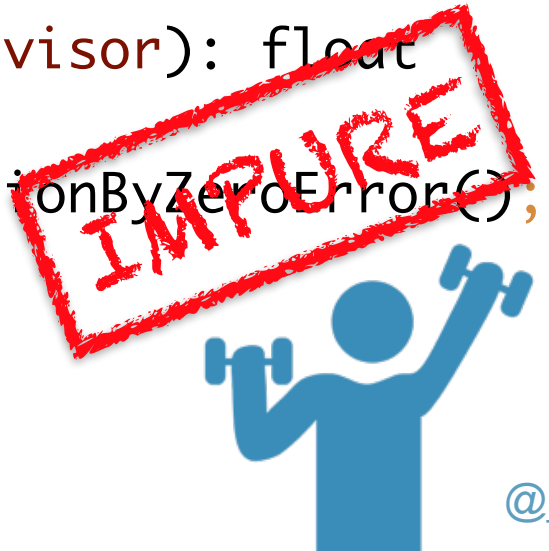
```
function square(float $x): float
{
    return $x * $x;
}

function zero(int $x): int
{
    return 0;
}
```

```php
function greaterThan(int $name): int
{
    return rand($name, PHP_INT_MAX);
}
```

```php
function greaterThan(int $name): int
{
    return rand($name, PHP_INT_MAX);
}

function divide(int $dividend, int $divisor): float
{
    if ($divisor == 0) throw new DivisionByZeroError();
    return $dividend / $divisor;
}
```



What questions do I have

Course Progress

Lean and Functional Domain Modelling

Lesson 1 – Short Intro to FP

Lesson 2 – PHP Functional Features

Lesson 3 – Algebraic Data Types

Lesson 4 – Lead Domain Modelling

Lesson 5 - Domain blocks life cycle

Anonymous Functions

```php
$square = function(float $x): float
{
    return $x * $x;
};
```

functions can return functions

```php
function squarer(): callable
{
    return function(float $x): float {
        return $x * $x;
    };
}

$squarer = squarer();
$y = $squarer(2.0); // $y == 4.0
```

functions can receive functions as arguments

```php
function pow0f16(callable $multiplier): float
{
    return $multiplier(16);
}

$squarer = squarer();
$y = pow0f16($squarer); // $y == 16 * 16
```

can receive functions as arguments

return functions

higher order functions

Closure

```php
function divBy(int $divisor): Closure
{
    return function(int $dividend) use ($divisor): float {
        return $dividend / $divisor;
    };
}
```

Currying

Currying

Converting a function with many arguments into a function with one argument.

```php
function position(int $i): callable
{



}

$firstCharIn = position(0);
echo $firstCharIn("Hello"); // H
```

```php
function position(int $i): callable
{



}

$firstCharIn = position(0);
echo $firstCharIn("Hello"); // H

$secondCharIn = position(1);
echo $secondCharIn("Hello"); // e
```

```php
function position(int $i): callable
{
    return function() {

    };
}

$firstCharIn = position(0);
echo $firstCharIn("Hello"); // H

$secondCharIn = position(1);
echo $secondCharIn("Hello"); // e
```

```php
function position(int $i): callable
{
    return function(string $s): string {

    };
}

$firstCharIn = position(0);
echo $firstCharIn("Hello"); // H

$secondCharIn = position(1);
echo $secondCharIn("Hello"); // e
```

```php
function position(int $i): callable
{
    return function(string $s) use ($i): string {
        return $s[$i];
    };
}

$firstCharIn = position(0);
echo $firstCharIn("Hello"); // H

$secondCharIn = position(1);
echo $secondCharIn("Hello"); // e
```

Invoke magic method

```php
class DivBy
{
    private $divisor;
    function __construct(int $div) { $this->divisor = $div; }

    function __invoke(int $dividend): float
    {
        return $dividend / $this->divisor;
    }
}
```

```php
class DivBy
{
    private $divisor;
    function __construct(int $div) { $this->divisor = $div; }

    function __invoke(int $dividend): float
    {
        return $dividend / $this->divisor;
    }
}
```

```php
class DivBy
{
    private $divisor;
    function __construct(int $div) { $this->divisor = $div; }

    function __invoke(int $dividend): float
    {
        return $dividend / $this->divisor;
    }
}

$divBy2 = new DivBy(2);
```

```php
class DivBy
{
    private $divisor;
    function __construct(int $div) { $this->divisor = $div; }

    function __invoke(int $dividend): float
    {
        return $dividend / $this->divisor;
    }
}

$divBy2 = new DivBy(2);
$y = $divBy2(4); // $y == 2
```

Wrapping a callable in a class

```php
class ArithmeticOperation
{
    private $operation;

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

    public function run(... $operands): float
    {
        return ($this->operation)(... $operands);
    }
}
```

Namespace your functions. Add a constant to the signature.

```
namespace Acme;

const myFunction = "Acme\\myfunction";
function myFunction() {...}
```

Many ways to pass callable in PHP

```php
call_user_func_array("strpos", ["Rovinj", "t"]);

call_user_func_array(function($x, $y) { return strpos($x, $y); },
   ["Rovinj", "t"]);

call_user_func($invokable, 42);

call_user_func([Some::class, "somePublicStaticMethod"], 42);

call_user_func([$obj, "somePublicMethod"], 42);
```

```php
use function Acme\someOtherFunction;
use const Acme\myFunction;
use Some\Name\Space\{ Some, Classes, const sum, function diff };

call_user_func_array(myFunction, ["Rovinj", "t"]);
```

Exercise

Exercise

Name 4 ways you can pass callables in PHP?

Exercise

What are higher order functions?

Exercise

What is a closure?

How is type hinting Closure different to callable?

Exercise

What is currying?

What questions do I have

Course Progress

Lean and Functional Domain Modelling

Lesson 1 – Short Intro to FP

Lesson 2 – PHP Functional Features

Lesson 3 – Algebraic Data Types

Lesson 4 – Lead Domain Modelling

Lesson 5 - Domain blocks life cycle

ADTs

{

Sum Types (model data that can be A or B)

Product Types (model data that is A and B and N)

Structural Recursion (decomposition of ADTs)

Type Constructors (type that creates types)

Sum Types

A is a B or a C

```
abstract class A {…}

final class B extends A {…}
final class C extends A {…}
```

A is a B or a C

Sum types

Transition of states. Statuses of processes.

Variations of the same concept.

Product Types

A is a B and a C

```php
class A
{
    function __construct(B $b, C $c)
    {…}
}
```

Product types

Composite types

State represented by a combination of values from other types

Structural Recursion

polymorphism or pattern matching

with

Polymorphism

```
abstract class Account {…}

final class Green extends Account {…}
final class Gold extends Account {…}
final class Closed extends Account {…}
```

```
abstract class Account {
    abstract function claim(); }

final class Green extends Account {
    function claim() {
        $this->assertMinimumStars();
        return new Green($this->stars - 15);
    }
}
final class Gold extends Account {
    function claim() {
        $this->assertMinimumStars();
```

```
        function claim() {
            $this->assertMinimumStars();
            return new Green($this->stars - 15);
        }
    }
    final class Gold extends Account {
        function claim() {
            $this->assertMinimumStars();
            return new Gold($this->stars - 10);
        }
    }
```

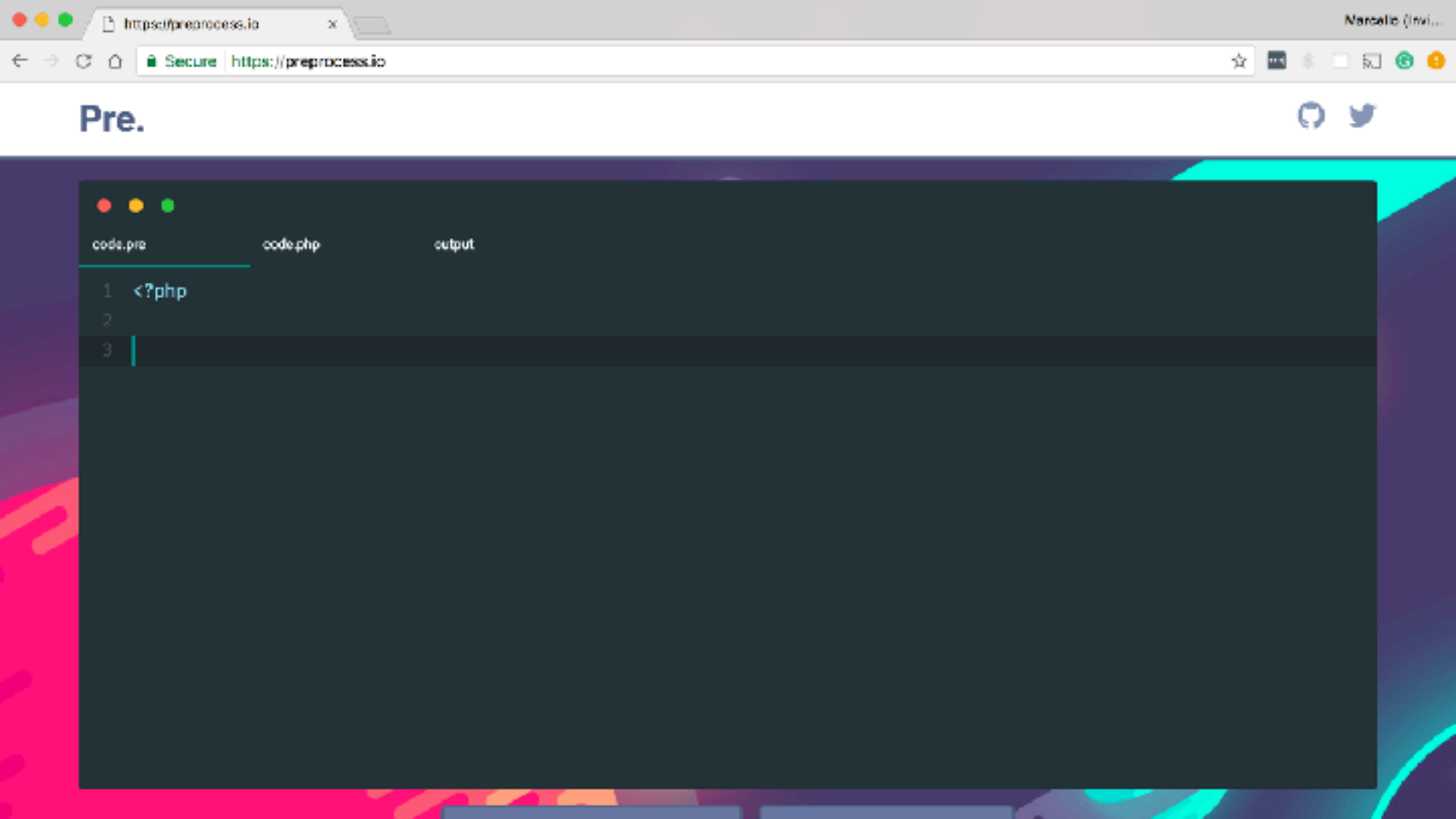
with

Pattern Matching

github.com/phunkie/phunkie

```
$on = match($account); switch (true) {
    case $on(Green($a)):
        return Green($a - 15);
    case $on(Gold($a)):
        return Gold($a - 15);
    case $on(Closed()):
        return Closed();
}
```

Pre


https://preprocess.io
Secure
https://preprocess.io
Pre.
code.pre
code.php
output
1 <?php
2
3

```
return match($account) {
    case Green($a) => Green($a - 15)
    case Gold($a) => Gold($a - 10)
    case Closed() => Closed()
};
```

```
if minStars($account) {
    return match($account) {
        case Green($a) => Green($a - 15)
        case Gold($a) => Gold($a - 10)
        case Closed() => Closed()
    };
}
```

```php
$on = match($account); switch (true) {
    case $on(Green($a)): return Green($a - 15);
    case $on(Gold($a)): return Gold($a - 15);
    case $on(Closed()): return Closed();
}
```

```php
return match($account) {
    case Green($a) => Green($a - 15)
    case Gold($a) => Gold($a - 10)
    case Closed() => Closed()
};
```

Type Constructors

```
abstract class Result {…}
final class Good extends Result {…}
final class Bad extends Result {…}
```

A Result is a Good or Bad

```
abstract class Result {…}
final class Good extends Result {
    function __construct($result) {…}
}
final class Bad extends Result {…}
```

A Result of ? is a Good of ? or Bad

```
function deduct($stars): Result {
    if ($stars >= 15) {
        return new Good($stars - 15);
    }
    return new Bad();
}
```

Option

```
abstract class Option {…}
final class Some extends Option {…}
final class None extends Option {…}
```

```
abstract class Option {…}
final class Some extends Option {…}
final class None extends Option {…}
```

An Option of ? is a Some of ? or a None

Exercise - 3a

Lesson3/Exercise3a.php

Write a function `maybeDivBy` that makes sure division by zero produces a `new None` and other divisors produce a `new Some($result)`.

```
$ composer test 3a
```

Solution - 3a

```
function maybeDivBy(int $divisor): \Closure
{
    return function(int $dividend) use ($divisor): Option {
        return $divisor ?
                new Some($dividend / $divisor) :
                new None();
    };
}
```

```
return match(deduct($stars)) {
    case Some($r) => "Free Coffee! ☕"
    case None => "Not enough stars :("
};
```

Either

```
abstract class Either {…}
final class Left extends Either {…}
final class Right extends Either {…}
```

An Either of E,V is a Left of E or a Right of V

```
function deduct($stars): Either {
    if ($stars >= 15) {
        return Right($stars - 15);
    }
    return Left("Not enough stars");
}
```

Exercise - 3b

Lesson3/Exercise3b.php

Write a method `eitherDivBy` that makes sure division by zero produces a new `Left("Division by zero")` and other divisors produce a `new Right($result)`.

```
$ composer test 3b
```

Solution - 3b

```
function eitherDivBy(int $divisor): \Closure
{
    return function(int $dividend) use ($divisor): Either {
        return $divisor ?
                new Right($dividend / $divisor) :
                new Left("Division by zero");
    };
}
```

Either is right-biased

Disjunction, a.k.a. Validation

```
abstract class Validation {…}
final class Success extends Validation {…}
final class Failure extends Validation {…}
```

A Validation of E,V is a Failure of E or Success of V

```
function deduct($stars): Validation {
  if ($stars >= 15) {
    return Success($stars - 15);
  }
  return Failure("Not enough stars");
}
```

ADTs

{

Sum Types (model data that can be A or B)

Product Types (model data that is A and B and N)

Structural Recursion (decomposition of ADTs)

Type Constructors (type that creates types)

What questions do I have

Break

Composition

Combinators

```
if (null !== $user) {
    $card = $user->getCard();

    if (null !== $card) {
        $amount = $card->getAmount();

        if (null !== $amount) {
            return needsTopUp($user, $amount);
        };
    };
};
```

map

Apply a function to the value inside the ADT

```
public function map(callable $f): Option
{
    // Apply $f to the value in the Some
    // context and return a Some with
    // the new result,
    // or None if the Option is None
}
```

map

```
Some($user)->map(function($u){
    return $u->getCard();
}); // Some($card)
```

map

```
None()->map(function($u){
    return $u->getCard();
}); // None
```

```
Some($user)->map(function($u){
    return $u->getCard();
})->map(function($c){
    return $c->getAmount();
});
```

```
Some($user)->map(function($u){
    return $u->getCard();
})->map(function($c){
    return $c->getAmount();
});
```

```
Some(
    $card
)->map(function($c){
    return $c->getAmount();
});
```

```
final class User {
    public function getCard():Option//<Card>
    {…}
}

final class Card {
    public function getAmount():Option//<Amount>
    {…}
}
```

```
Some(
    Some($card)
)->map(function($c){
    return $c->getAmount();
});
```

Exercise - 3c

Lesson3/Exercise3c.php

Write a `map` combinator for the `Option` ADT.

```
$ composer test 3c
```

Solution - 3c

```
public function map(callable $f): Option
{
    if ($this instanceof Some) {
        return new Some($f($this->get()));
    }
    return new None();
}
```

flatmap

maps a function that returns a ADT and then flattens it

```php
public function flatMap(callable $f): Option
{
    // Apply $f to the value in the Some
    // context and return the new result,
    // or None if the Option is None.
}
```

$f must return an Option

flatmap

```
new Some($user)->flatMap(function($u){
    return $u->getCard();
}); // Some($card)
```

```
if (null !== $user) {
    $card = $user->getCard();

    if (null !== $card) {
        $amount = $card->getAmount();

        if (null !== $amount) {
            return needsTopUp($user, $amount);
        };
    };
};
```

```
Some($user)->flatMap(function($u) {
    return $u->getCard()->flatMap(function($c) use ($u) {
        return $c->getAmount()->map(function($a) use ($u) {
            return needsTopUp($u, $a);
        });
    });
});
```

Pre

github.com/MarcelloDuarte/for-comprehension-preprocessor

```
Some($user)->flatMap(function($u) {
    return $u->getCard()->flatMap(function($c) use ($u) {
        return $c->getAmount()->map(function($a) use ($u) {
            return needsTopUp($u, $a);
        });
    });
});
```

```
for {
    $u <- Some($user)
    $c <- $u->getCard()
    $a <- $c->getAmount()
} yield needsTopUp($u, $a)
```

```
for {
    $u <- Some($user)
    $c <- $u->getCard()
    $a <- $c->getAmount()
} yield needsTopUp($u, $a)
```

```
Some($user)->flatMap(function($u) {
    return $u->getCard()->flatMap(function($c) use ($u) {
        return $c->getAmount()->map(function($a) use ($u) {
            return needsTopUp($u, $a);
        });
    });
});
```

What questions do I have

Course Progress

Lean and Functional Domain Modelling

Lesson 1 – Short Intro to FP

Lesson 2 – PHP Functional Features

Lesson 3 – Algebraic Data Types

Lesson 4 – Lead Domain Modelling

Lesson 5 - Domain blocks life cycle

Domain model

A model is a map.

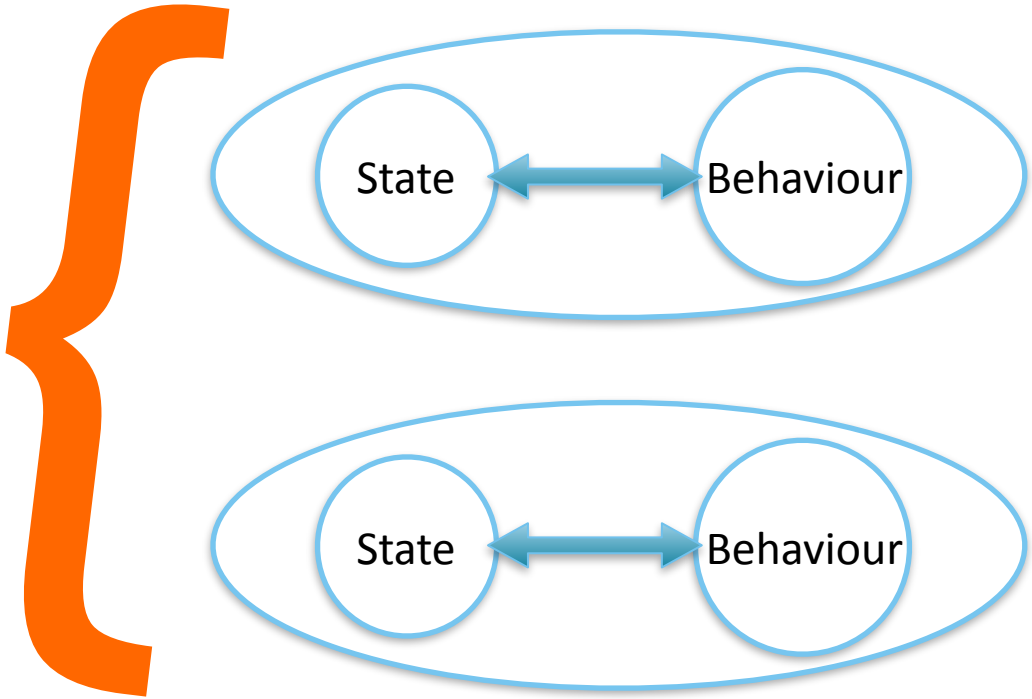
All we have are maps of reality.

So it is not about who is right,
but who's map of reality is more useful
for my experience of reality.

"A model is a system of abstractions that describes selected aspects in the [problem] domain".

– Eric Evans

Rich models are useful

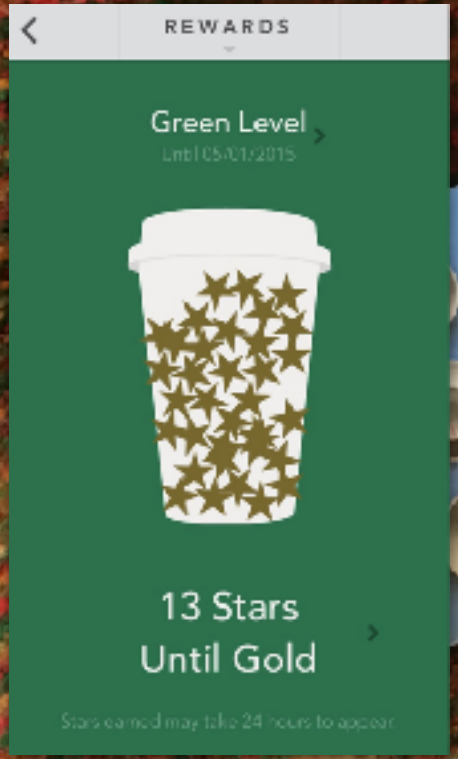


Rich domain models

Rich domain models

- Approach programming functionally
- Algebraic data types
- Compatible tactic modelling techniques
- Domain objects life cycle in FP

Modelling rewards with rich domains


REWARDS
Green Level
13 Stars
Until Gold


CAFFÈ
NERO
PLANET ORGANIC
GET ONE FREE

#websc

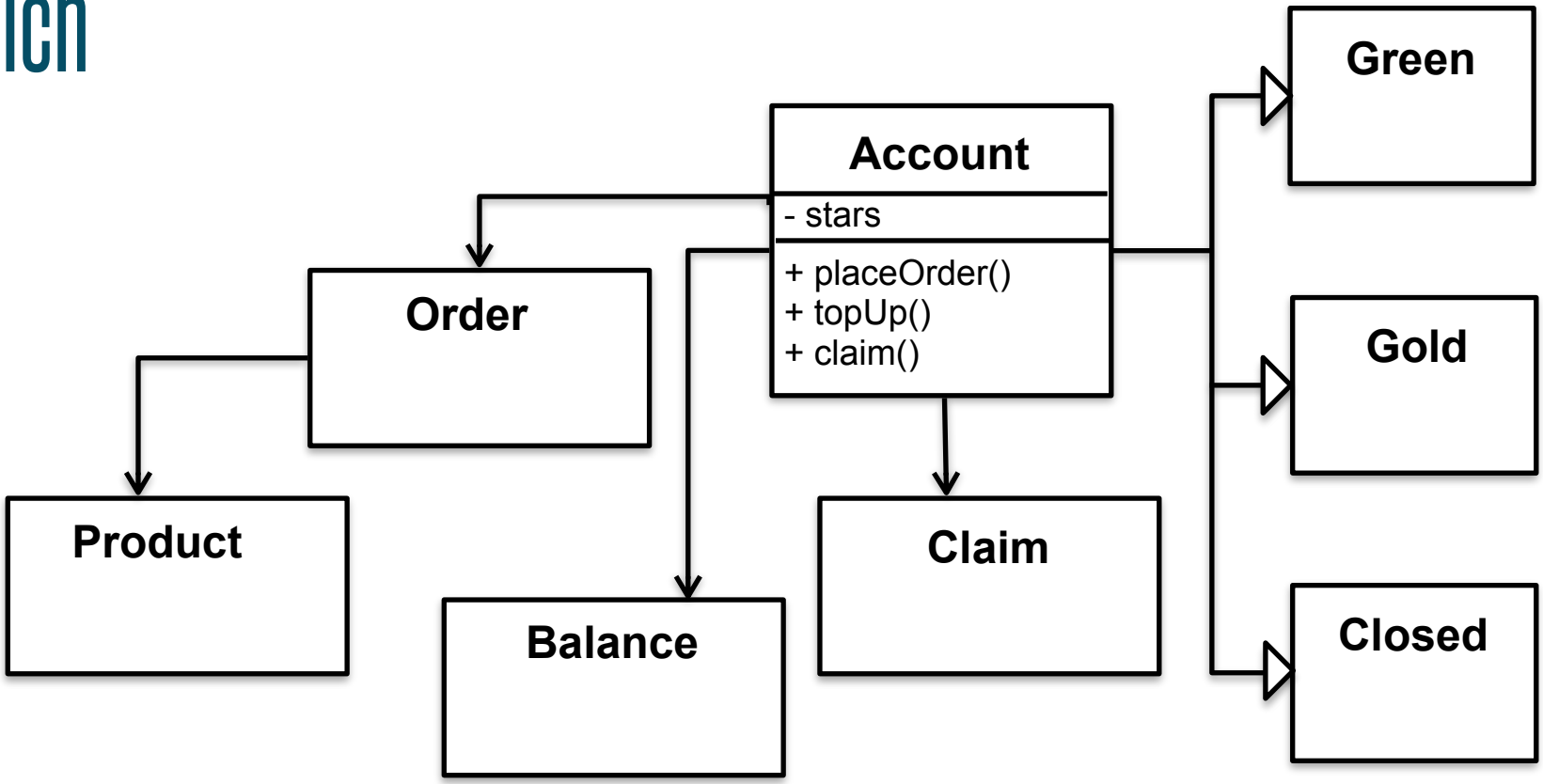
@_md

scenario

Problem Domain: `Rewarding loyal accounts`.

Brief: Accounts can be Green, Gold or Closed. An account can top up, order and return items at the cafe. When they order drinks, among other products, they earn a star. When they earn 15 stars, they can claim a drink for free. If they earn 50 stars in one year they are upgraded to Gold. Gold accounts only need 10 stars to get a free drink. You need to earn 50 stars in a year to keep your Gold account.

rich

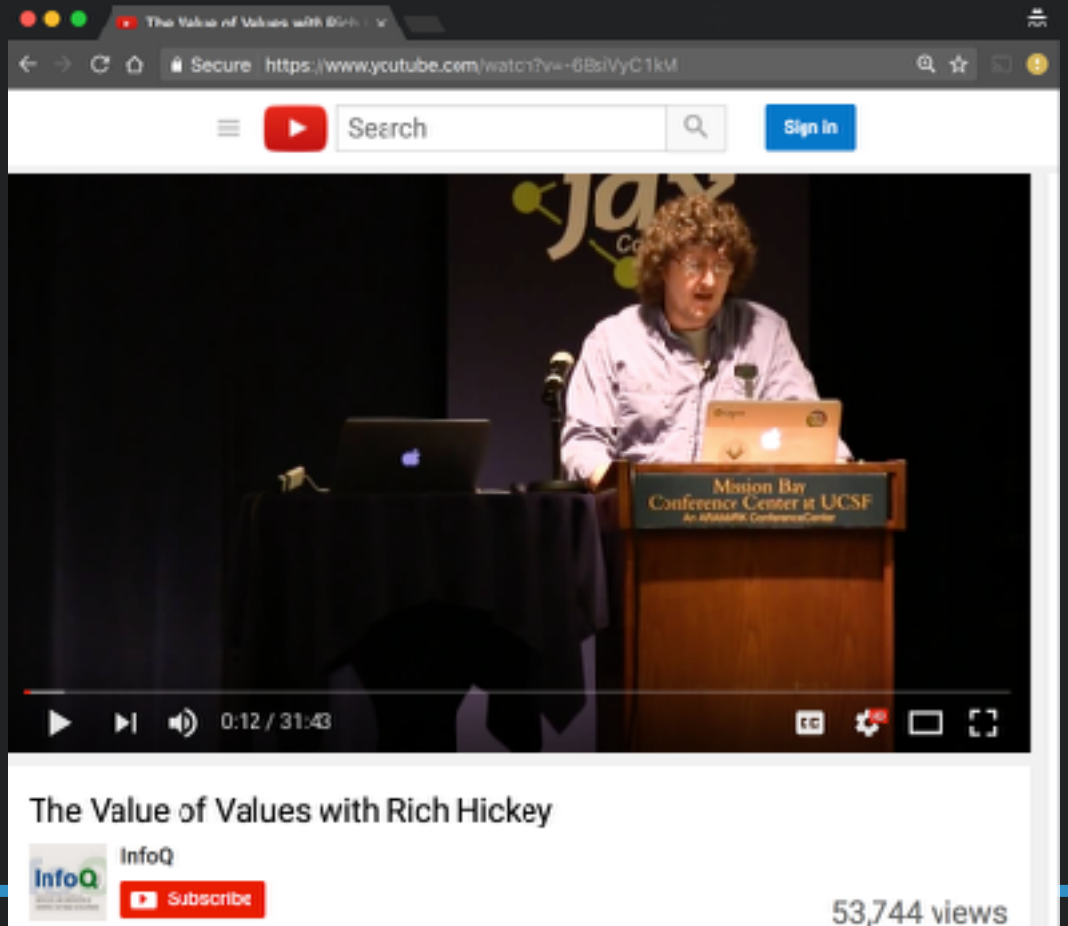

Account
- stars
+ placeOrder()
+ topUp()
+ claim()
Order
Product
Balance
Claim
Green
Gold
Closed

```php
// instantiate an user entity
$this->stars =  $starsFromDb;

// buy coffee
$this->stars +=  1;

// claim
$this->assertMinimumStars();
$this->stars -=  15;
```

The value of values


Secure https://www.ycutube.com/watch?v=-6BsiVyC1kM
Search
Sign in
Mission Bay
Conference Center at UCSF
0:12 / 31:43
The Value of Values with Rich Hickey
InfoQ
Subscribe
53,744 views

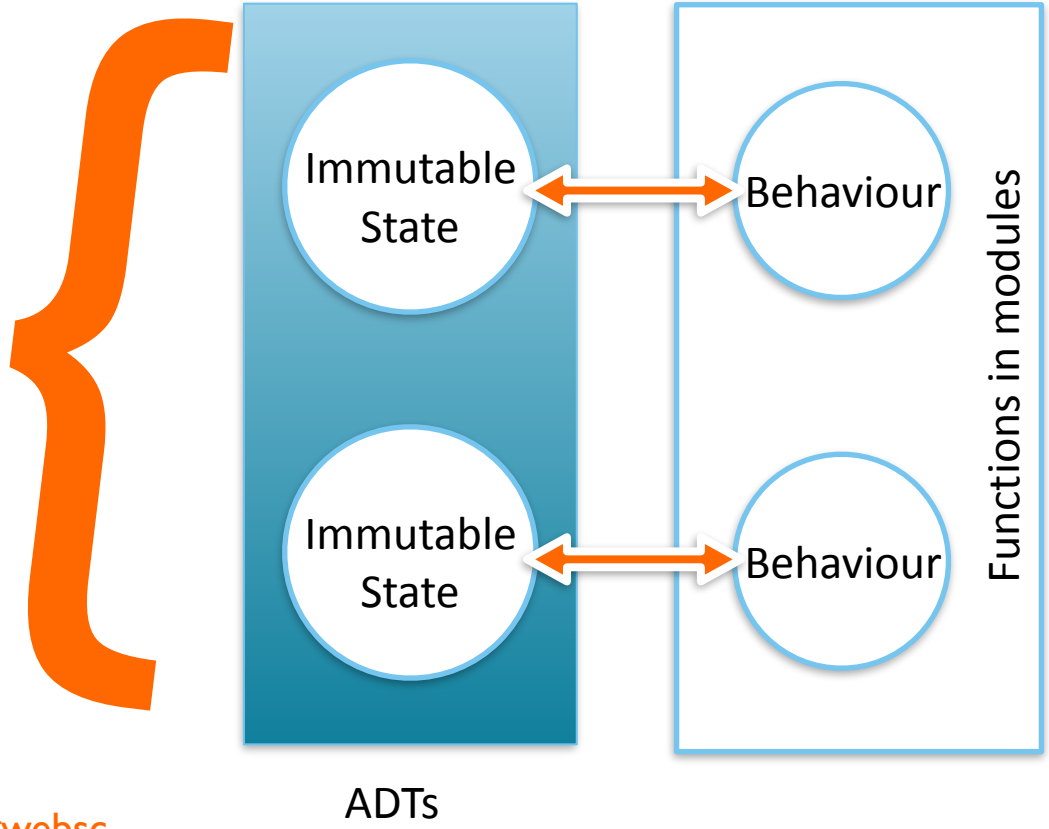
#websc

@_md

Purity

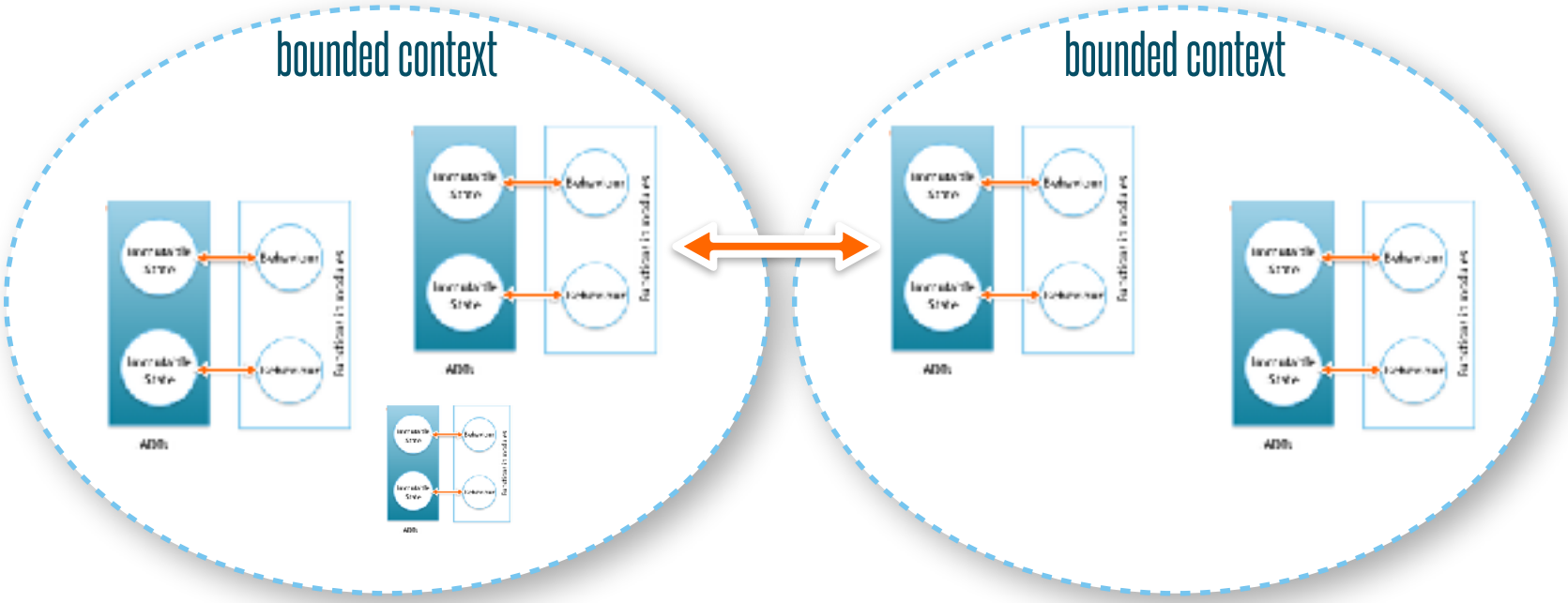
A => B => C

Introducing lean models



Lean domain models

A domain model is a collection of models



Start with the behaviour

scenario

Problem Domain: `Rewarding loyal accounts`.

Brief: Accounts can be Green, Gold or Closed. An account can top up, order and return items at the cafe. When they order drinks, among other products, they earn a star. When they earn 15 stars, they can claim a drink for free. If they earn 50 stars in one year they are upgraded to Gold. Gold accounts only need 10 stars to get a free drink. You need to earn 50 stars in a year to keep your Gold account.

Service

scenario

Problem Domain: Rewarding loyal accounts.

Brief: Accounts can be Green, Gold or Closed. An account can top up, order and return items at the cafe. When they order drinks, among other products, they earn a star. When they earn 15 stars, they can claim a drink for free. If they earn 50 stars in one year they are upgraded to Gold. Gold accounts only need 10 stars to get a free drink. You need to earn 50 stars in a year to keep your Gold account.

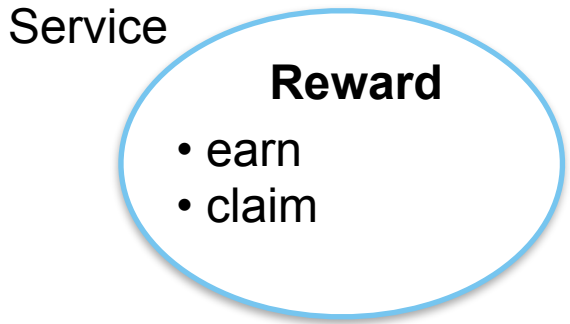
scenario

Problem Domain: Rewarding loyal accounts.

Brief: Accounts can be Green, Gold or Closed. An account can top up, order and return items at a cafe. When they order drinks, among other products, they earn a star. When they earn 15 stars, they can claim a drink for free. If they earn 50 stars in one year they are upgraded to Gold. Gold accounts only need 10 stars to get a free drink. You need to earn 50 stars in a year to keep your Gold account.

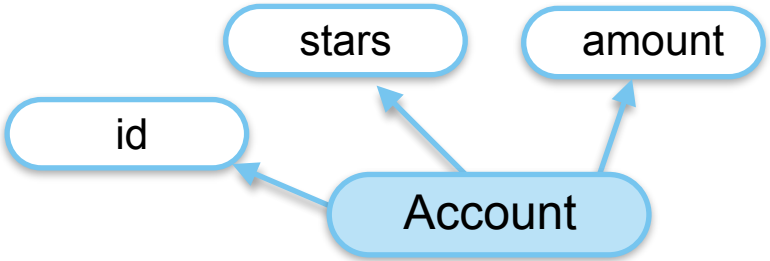
Rewarding

```
interface RewardsService {
    function earn();
    function claim();
}
```



Product types

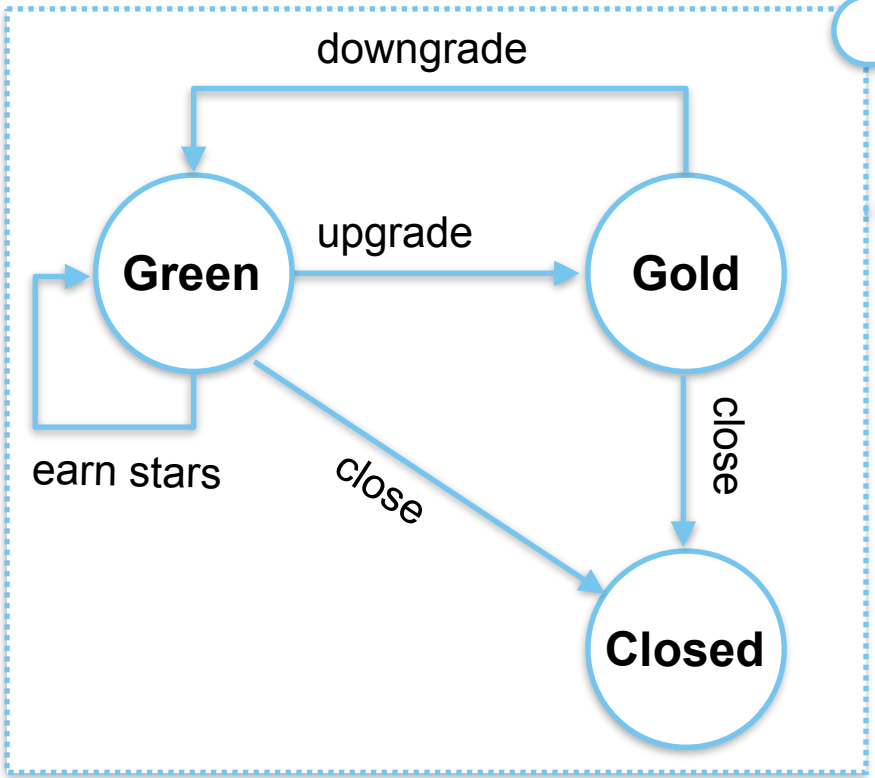
```
class Account
{
    function __construct(Guid $id, Stars $stars, Money $amount)
    {…}
}
```

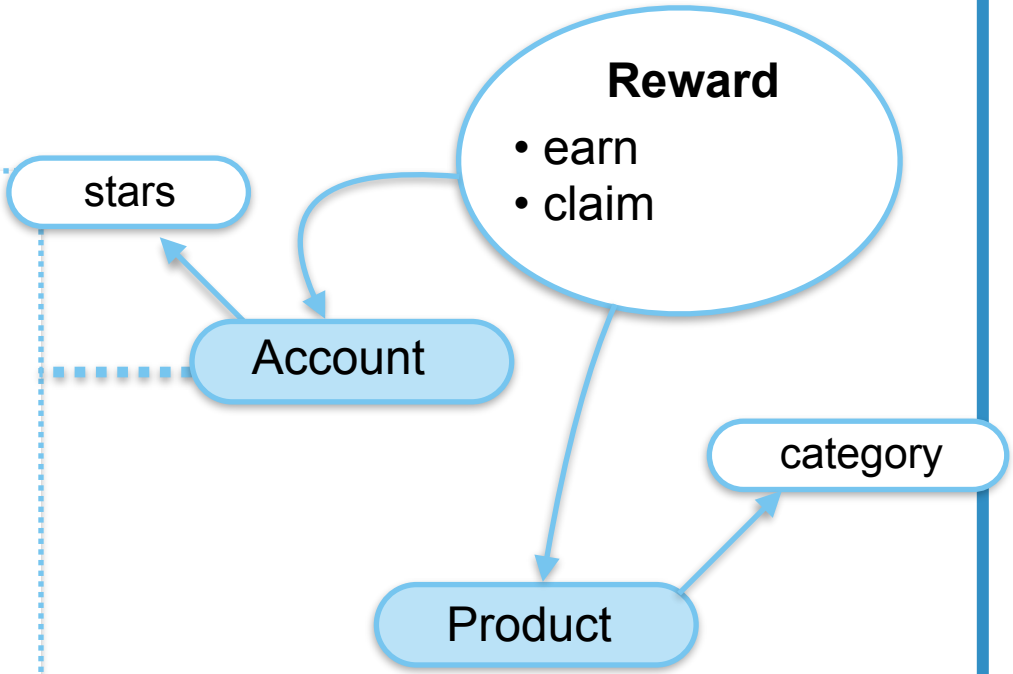


```
interface RewardsService {
    function earn();
    function claim(Product $drink);
}
```

Sum types

sum types


downgrade
Green
upgrade
Gold
earn stars
close
close
Closed

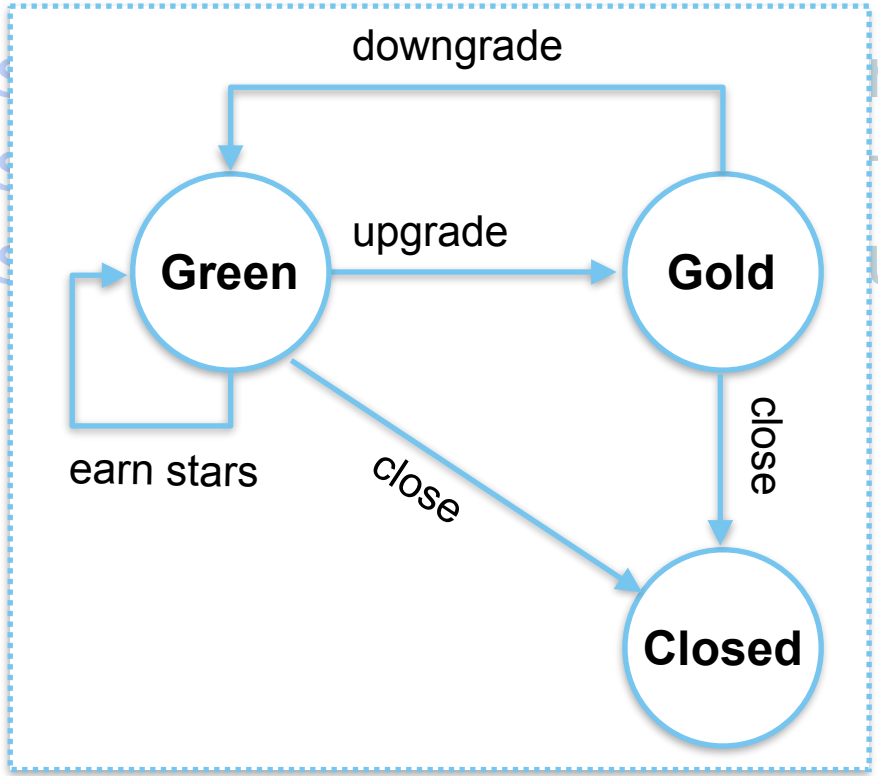

stars
Reward
• earn
• claim
Account
category
Product

```
abstract class Account {…}

final class Green extends Account {…}
final class Gold extends Account {…}
final class Closed extends Account {…}
```

```
abstract class Ac
final class
final class
final class
```

Account



What questions do I have

Let's model it together - 4a

Lesson4/Exercise4a

Problem Domain: `Claim work expenses`.

Brief: Expenses can be of Food, Travel, Accommodation or Other. They have a cost and date.
Staff can claim an Expense Sheet (list of expenses). The Claim can be pending, approved, rejected or paid. Staff can add expense to a sheet, to claim later. Only approved/unpaid claim can be paid.

Course Progress

Lean and Functional Domain Modelling

Lesson 1 – Short Intro to FP

Lesson 2 – PHP Functional Features

Lesson 3 – Algebraic Data Types

Lesson 4 – Lead Domain Modelling

Lesson 5 - Domain blocks life cycle

Domain object life time

Smart Constructor Idiom

```
abstract class RewardAccount {}
final class Green extends RewardAccount {}
final class Gold extends RewardAccount {}
```

```
abstract class RewardAccount {
    static function greenCard(): Green {}
    static function goldCard(): Gold {}
}
final class Green extends RewardAccount {}
final class Gold extends RewardAccount {}
```

```
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (                                          ): ???
{




}
```

```php
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): ???
{









}
```

```php
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): Validation
{









}
```

```php
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): Validation
{
   if (isMoreThanOneYear($anniversary) &&
       $starsCollected < 50) {



   }

}
```

```
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): Validation
{
   if (isMoreThanOneYear($anniversary) &&
       $starsCollected < 50) {
        return Failure(new DisqualifiedException(
            "Client does not qualify for the Gold
             account anymore. Downgraded."));
   }

}
```

```php
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): Validation
{
   if (isMoreThanOneYear($anniversary) &&
       $starsCollected < 50) {
        return Failure(new DisqualifiedException(
            "Client does not qualify for the Gold
             account anymore. Downgraded."));
   }
   return Success(Pair($anniversary, $starsCollected));
}
```

```
abstract class RewardAccount {
    static function greenCard(): Green {}
    static function goldCard(): Gold {}
}
final class Green extends RewardAccount {}
final class Gold extends RewardAccount {}

function validateStillQualified(…) {}
function validateStars(…) {}
function validateAmount(…) {}
```

```
abstract class RewardAccount {
    static function greenCard(…): Green
    {
        return applyValidations (
            validateStillQualified(…),
            validateStars(…),
            validateAmount(…)
        )(function(…){return new Green(…));
    }
    static function goldCard(…): Gold {…}
}
```

```
abstract class RewardAccount {
    static function greenCard(…): Green
    {
        return applyValidations (
            validateStillQualified(…),
            validateStars(…),
            validateAmount(…)
        )(function(…){return new Green(…));
    }
    static function goldCard(…): Gold {…}
}
```

```
abstract class RewardAccount {}
final class Green extends RewardAccount {}
final class Gold extends RewardAccount {}
final class Closed extends RewardAccount {}
```

```
abstract class RewardAccount {
    static function greenCard() {}
    static function goldCard() {}
    static function closedCard() {}
}
final class Green extends RewardAccount {}
final class Gold extends RewardAccount {}
final class Closed extends RewardAccount {}
```

```
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (                                          ): ???
{




}
```

```
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): ???
{









}
```

```
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): Validation
{









}
```

```
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): Validation
{
   if (isMoreThanOneYear($anniversary) &&
       $starsCollected < 50) {



   }

}
```

```
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): Validation
{
   if (isMoreThanOneYear($anniversary) &&
       $starsCollected < 50) {
        return Failure(new DisqualifiedException(
            "Client does not qualify for the Gold
             account anymore. Downgraded."));
   }

}
```

```php
/**
 * @return Validation<DisqualifiedException,(Date,int)>
 */
function validateStillQualified
    (Date $anniversary, int $starsCollected): Validation
{
   if (isMoreThanOneYear($anniversary) &&
       $starsCollected < 50) {
        return Failure(new DisqualifiedException(
            "Client does not qualify for the Gold
             account anymore. Downgraded."));
   }
   return Success(Pair($anniversary, $starsCollected));
}
```

```
abstract class RewardAccount {
    static function greenCard() {}
    static function goldCard() {}
    static function closedCard() {}
}
final class Green extends RewardAccount {}
final class Gold extends RewardAccount {}
final class Closed extends RewardAccount {}

function validateStillQualified(…) {}
function validateStars(…) {}
function validateAmount(…) {}
```

```
abstract class RewardAccount {
    static function greenCard(…): Validation
    {
        return applyValidations (
            validateStillQualified(…),
            validateStars(…),
            validateAmount(…)
        )(function(…){return new Green(…));
    }
    static function goldCard(…): Validation {…}
}
```

```
abstract class RewardAccount {
    static function greenCard(…): Validation
    {
        return applyValidations (
            validateStillQualified(…),
            validateStars(…),
            validateAmount(…)
        )(function(…){return new Green(…));
    }
    static function goldCard(…): Validation {…}
}
```

Exercise - 5a

Lesson5/Exercise5a

Complete the smart constructor for the Expense of Food.

You can only expense food up to 25 euros (you can assume the system only works in Euro). That's only true of food, other expenses can have a higher amount.

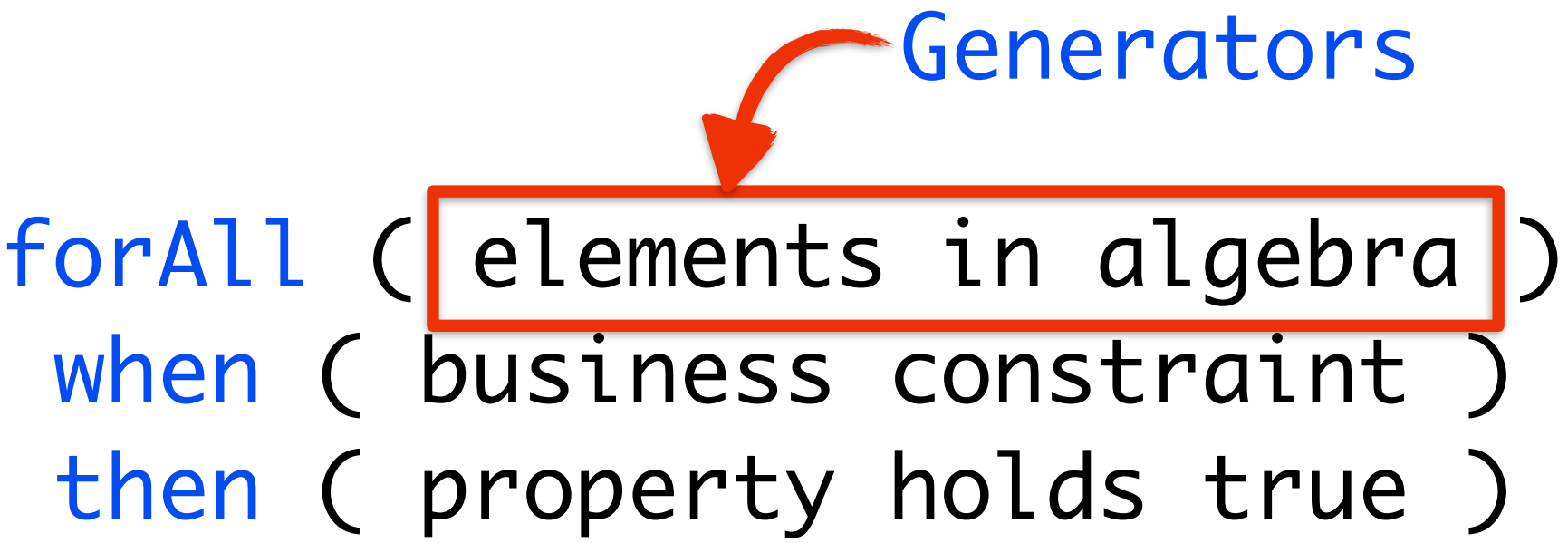
All expenses need a date earlier than today.

What questions do I have

Property Based Testing

```
forAll ( elements in algebra )
 when ( business constraint )
 then ( property holds true )
```

github.com/giorgiosironi/eris


Generators
forAll (elements in algebra)
when (business constraint)
then (property holds true)

```
forAll ( elements in algebra )
 when ( business constraint )
 then ( property holds true )
```

```php
use Eris\{ Generator, TestTrait };

class ShrinkingTest extends \PHPUnit_Framework_TestCase
{
    use TestTrait;

    public function testShrinkingAString()
    {
        $this->forAll(
            Generator\string()
        )
            ->then(function ($string) {
               $this->assertNotContains('B', $string);
            });
    }
}
```

```
1) ShrinkingTest::testShrinkingAString
Failed asserting that ':e|$dI,[B' does not contain "B".

/home/mduarte/code/eris/examples/ShrinkingTest.php:16
/home/mduarte/code/eris/src/Eris/Quantifier/Evaluation.php:51
/home/mduarte/code/eris/src/Eris/Shrinker/Random.php:68
/home/mduarte/code/eris/src/Eris/Quantifier/ForAll.php:128
/home/mduarte/code/eris/src/Eris/Quantifier/Evaluation.php:53
/home/mduarte/code/eris/src/Eris/Quantifier/ForAll.php:130
/home/mduarte/code/eris/src/Eris/Quantifier/ForAll.php:158
/home/mduarte/code/eris/examples/ShrinkingTest.php:17
/home/mduarte/code/eris/examples/ShrinkingTest.php:17

FAILURES!
Tests: 1, Assertions: 119, Failures: 1.
```

Eris Runner

github.com/MarcelloDuarte/
eris-runner

```
property ("Gold card holders must collect 50 stars a year",
   function() {
     forAll (Gen::date(), Gen::integer())

     ->when (function ($date, $stars) {
          return $stars < 50 && isMoreThanOneYear($date)
     })

     ->then (function ($date, $stars) {
         Assert::true(RewardCard::goldCard($date, $stars)
             ->isRight());
     });
});
```

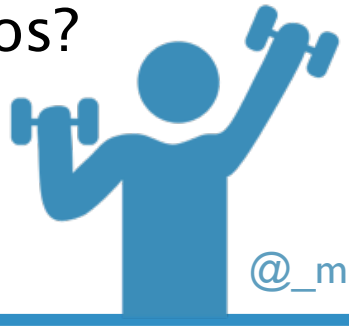
Home work - 5b

Lesson5/Exercise5b

Convert the example based tests from Exercise 5a into property based tests.

Things they may be uncovered:

- Have you considered a negative expense amount?
- What happens when amount is exactly 25 euros?



What questions do I have

Domain object life time

```
interface RewardsService {
    function order(ImmList $items);
    function top(Money $amount);
    function refund(Money $amount);
    function claim();
}
```

```
for {
    $o1 <- order($items1)
    $o2 <- order($items2)
    $r  <- refund($o2)
    $o3 <- order($items3)
} yield claim();
```

Domain object life time

Reader monad

```
class Reader
{
    function __construct(callable $run) {…}
    function run($x) { return ($this->run)($x); }
    function map(callable $f): Reader {…}
    function flatMap(callable $f): Reader {…}
    …
}
```

```
interface RewardsService {
    function order(ImmList $items): Reader;
    function top(Money $amount): Reader;
    function refund(Money $amount): Reader;
    function claim(): Reader;
}
```

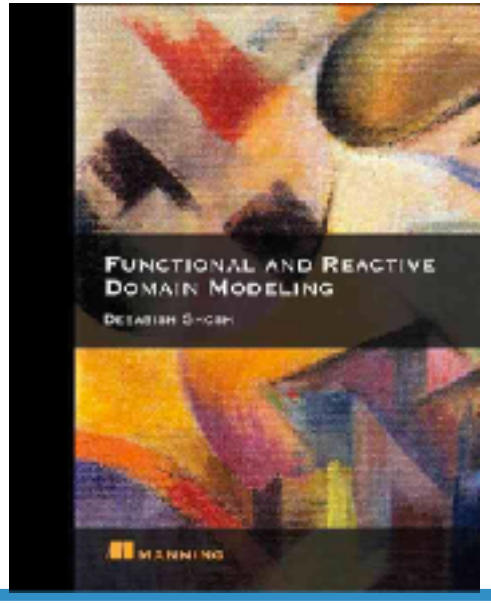
```
$reader = for {
    $o1 <- order($items1)
    $o2 <- order($items2)
    $r  <- refund($o2)
    $o3 <- order($items3)
} yield claim();

$reader->run($orderRepository)
```

What questions do I have

References

manning.com/books/functional-and-reactive-domain-modeling
github.com/MarcelloDuarte/for-comprehension-preprocessor
github.com/phunkie/phunkie
github.com/phunkie/adt
github.com/MarcelloDuarte/eris-runner
github.com/giorgiosironi/eris



Round Up

- FP intro
- Algebraic Data Types
- Composition
- Domain modelling
- Domain objects life cycle

vote for us :)

events.netgen.io/events

... and feedback

joind.in/talk/02ebb

Thank you

#websc

@_md