Unlocking magic of Monads with Java 8

Oleg Šelajev
@shelajev
ZeroTurnaround

Who am I

ZEROTURNAROUND

REBELLABS

1632 UNIVERSITY OF TARTU

What do we want here?

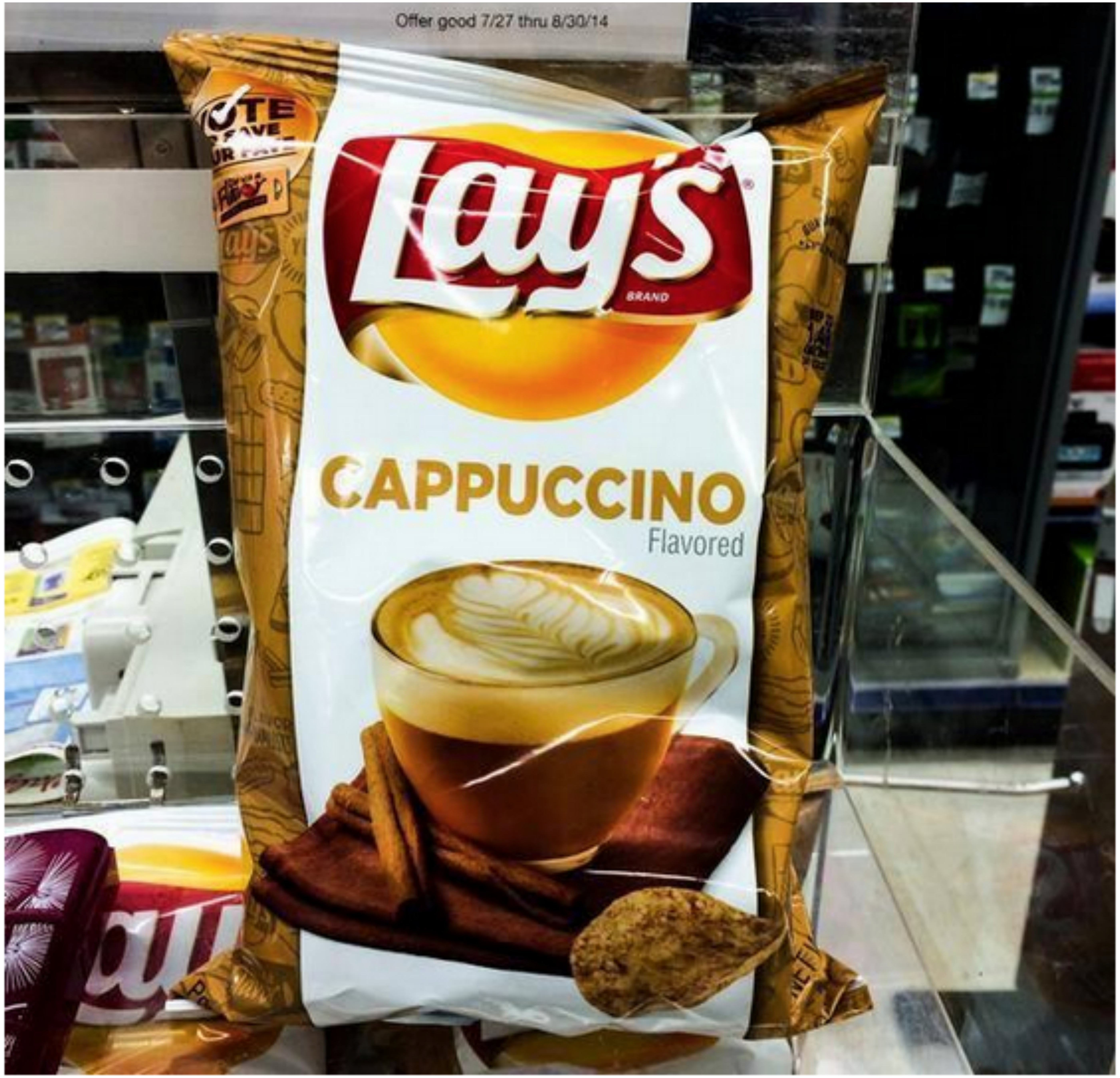
> have fun while hopefully learning stuff

> understand the concept of **Monad**

> devise generic constructs for Monads

> solve a real-world problem

> ignore lack of “ad hoc” polymorphism and other slightly relevant facts


Offer good 7/27 thru 8/30/14
VOTE TO SAVE YOUR FAVE
Lay's
BRAND
CAPPUCCINO
Flavored

Java 8: lambda recap

```java
@FunctionalInterface
public interface Function<T, R> {
  R apply(T t);
}
```

Java 8: lambda recap

```java
Function<String, Integer> f =
                           Integer::valueOf;
```

Java 8: lambda recap

```java
String prefix = "J1: “;

Function<String, Integer> f = (str) -> {
  System.out.println(prefix + str);
  return str.hashCode();
};
```

Death by 1000 tutorials

Problem statement

```
$("#button").fadeIn("slow",
    function() {
        console.log("hello world");
    }
);
```

Type: async result

- `java.util.concurrent.Future<V>`

- `boolean isDone();`
- `V get() …`
- `V get(long timeout, TimeUnit unit)`

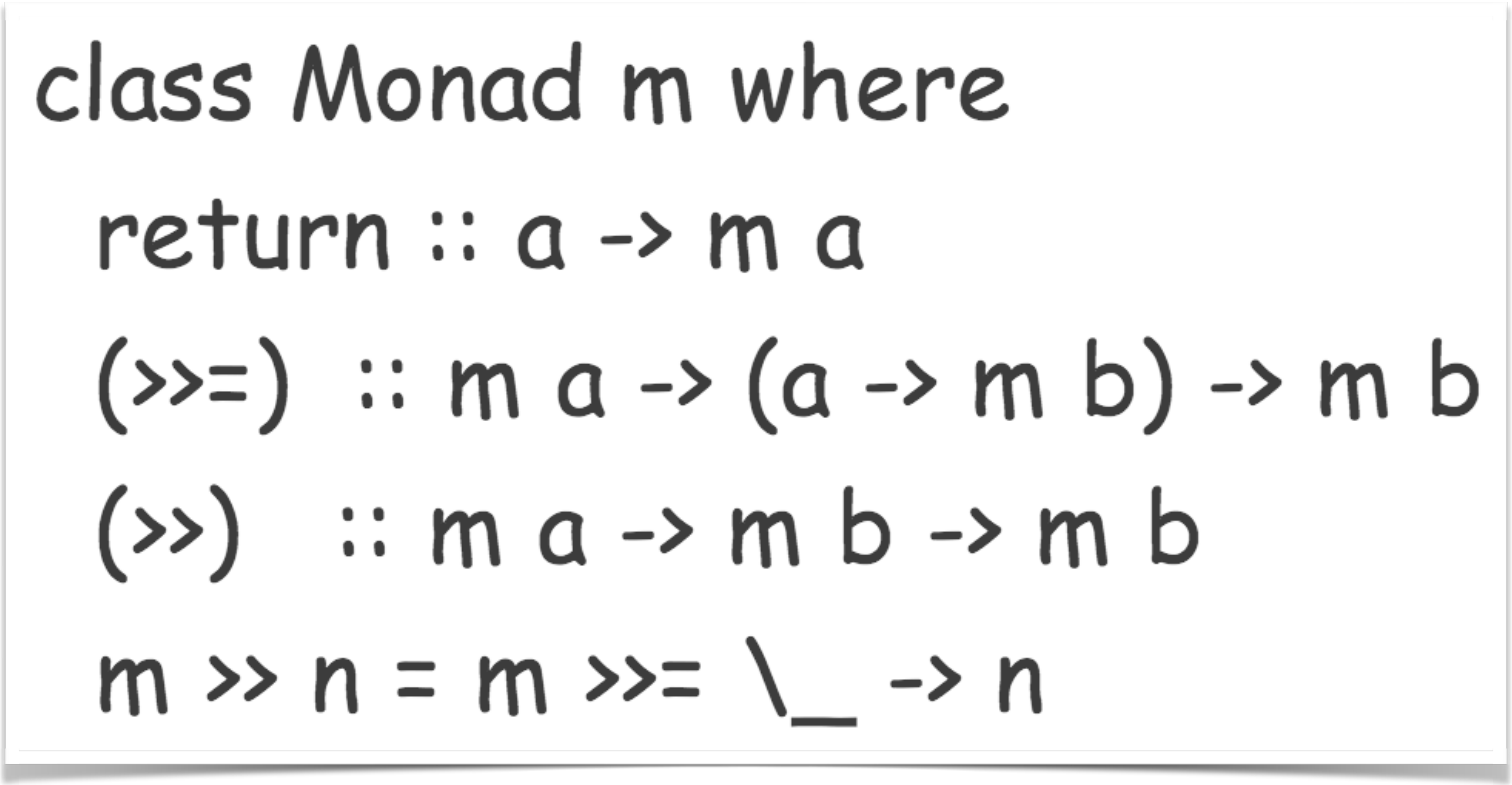
Type: async result

- java.util.concurrent.Future<V>

- boolean isDone();
- V get();
- V get(long timeout, TimeUnit unit)

Can we do better?

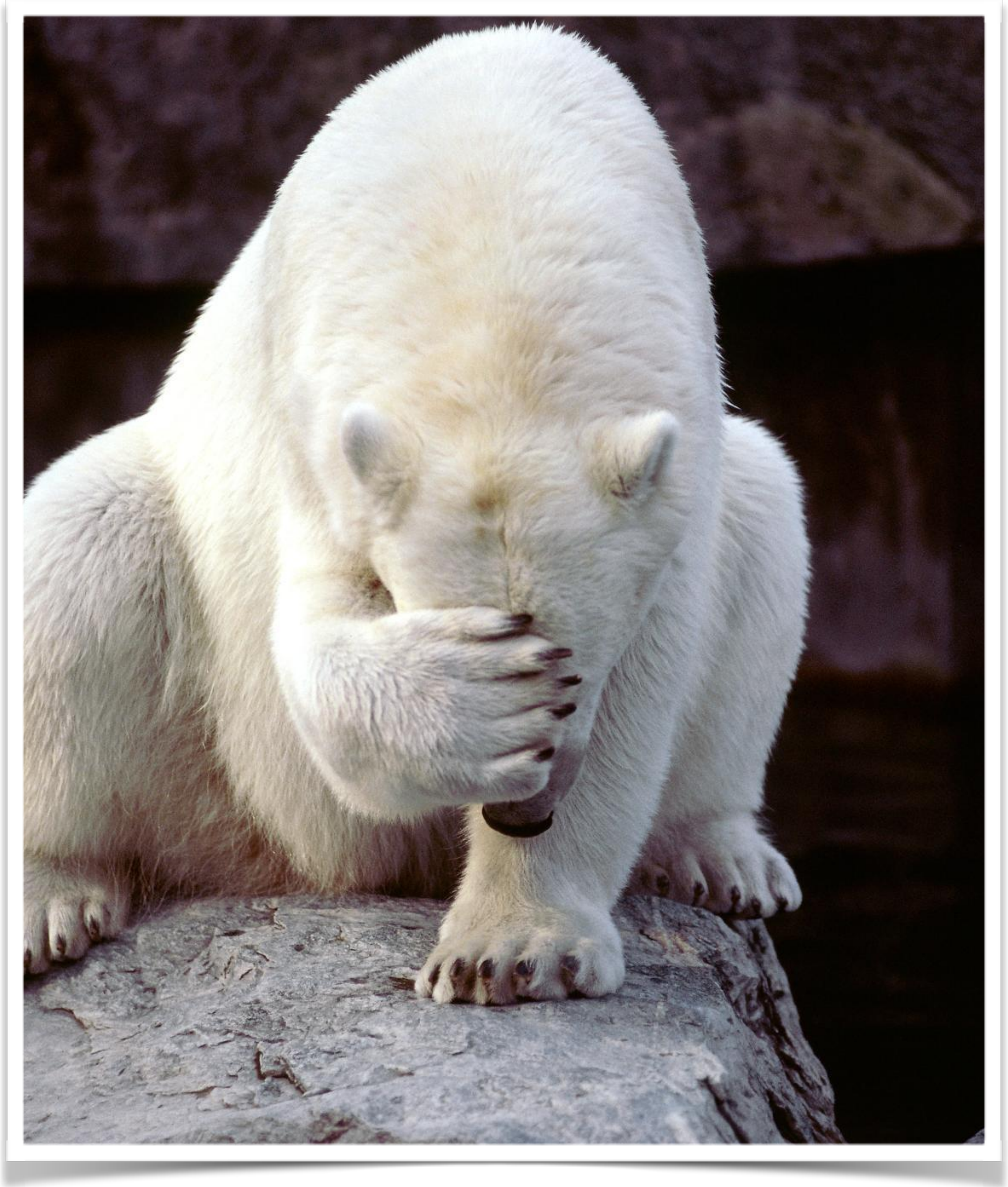
Monads to the rescue



```
class Monad m where
  return :: a -> m a
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)   :: m a -> m b -> m b
  m >> n = m >>= \_ -> n
```

Oh my…

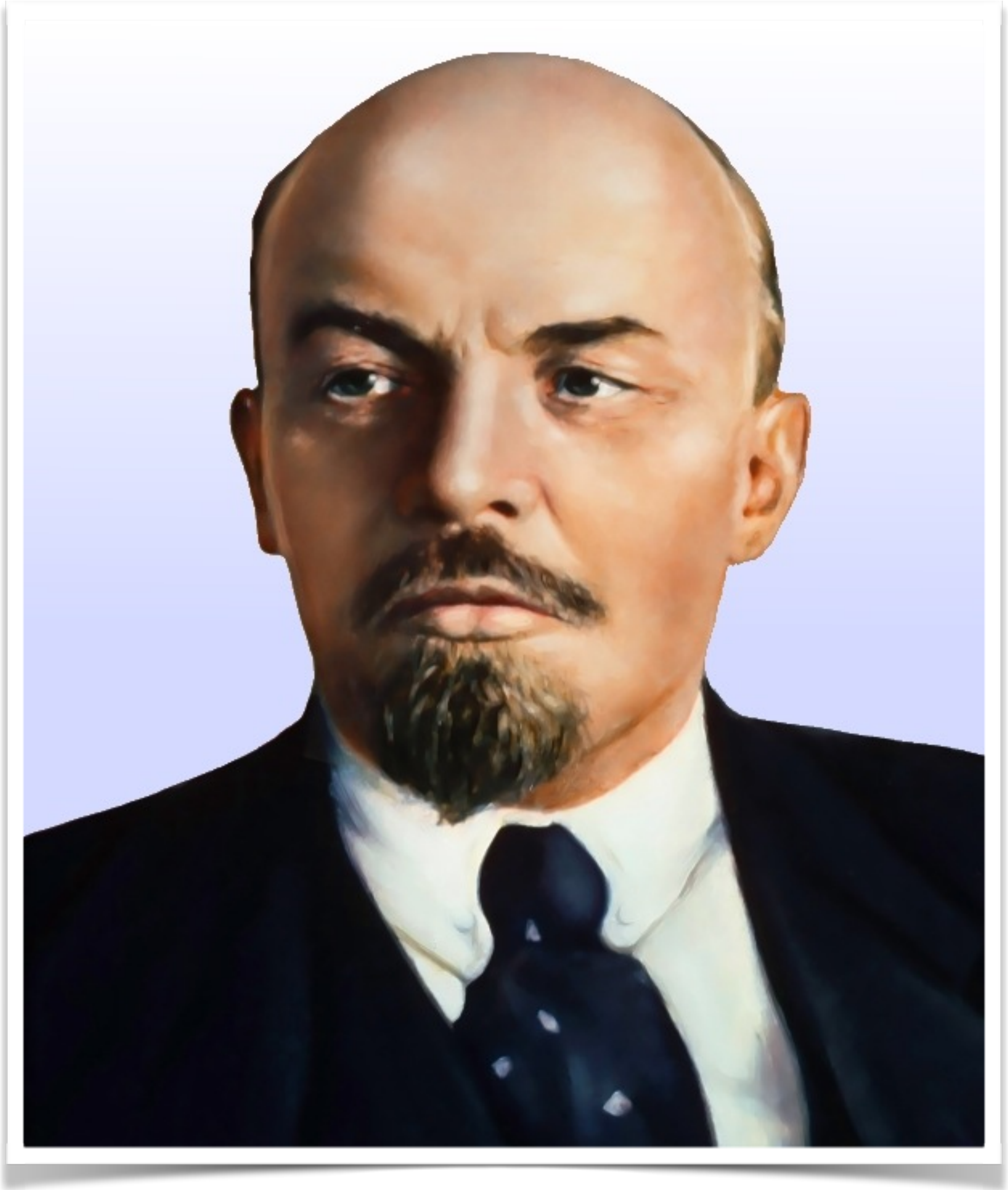
- a monad in X is just a monoid in the category of endofunctors of X, with product × replaced by composition of endofunctors and unit set by the identity endofunctor.

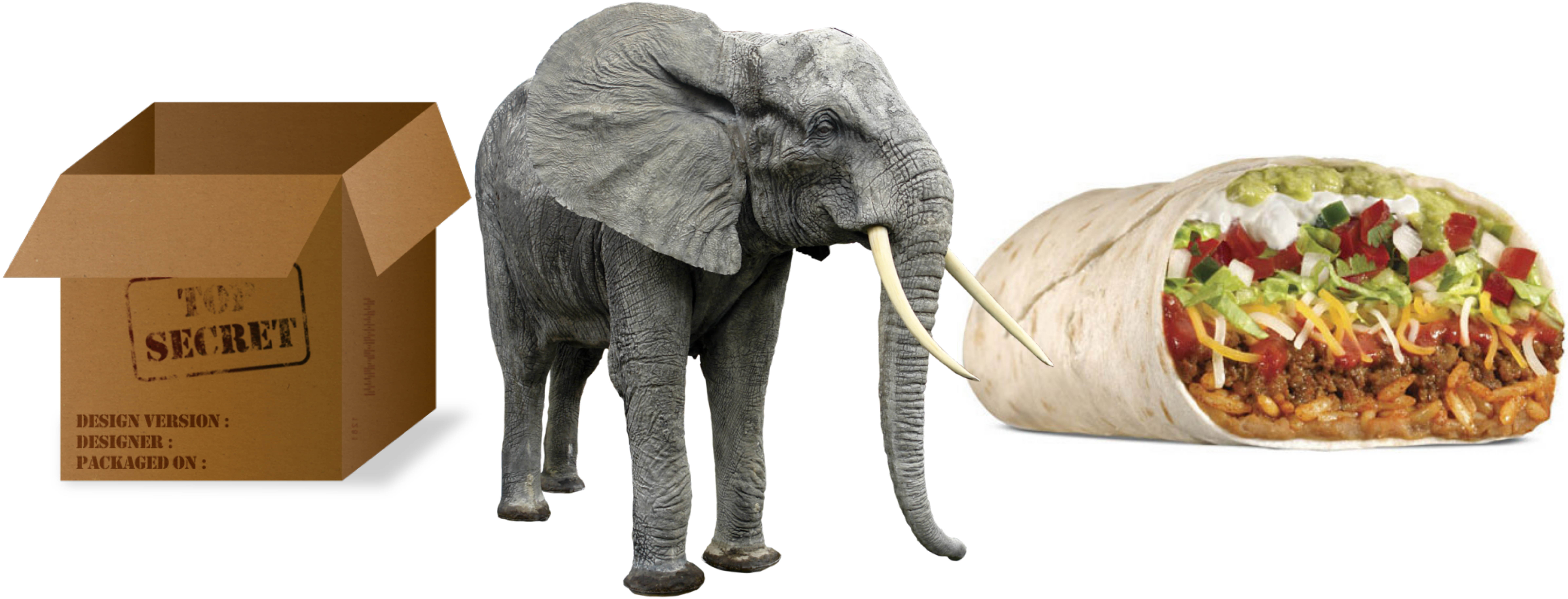


It is known

Every cook can understand, compile and use monads...

V. Lenin (1923)




TOP
SECRET
DESIGN VERSION :
DESIGNER :
PACKAGED ON :

Monads: intuition

> **wrapping** things

> **chaining** functions on those things

> monad is a **type**

Wrapping: return / pure

- Take *instance of “a”, return: “m a”*
- **Constructor** / Factory method

Pure in Java

```
public interface Monad<V> {
  Monad<V> pure(V value);
}
```

Chaining: bind / (>>=)

- take:
  - *monad: “m a”*
  - *function: “a => m b”*
- *return: monad “m b”*

Bind in Java

```java
public interface Monad<V> {
  Monad<V> pure(V v);
  <R> Monad<R> bind(Function<V, Monad<R>> f);
}
```

Hacking time

- Promise<V> - result of async computation
- Kinda like Future<V>
- supports chaining functions: **bind**

Promise<V>

```
> p.invokeWithException(Throwable t);
> p.invoke(V v);
> p.onRedeem(Action<Promise<V>> callback);
```

Promise<V>: pure

```
public static <V> Promise<V> pure (final V v)
{
  Promise<V> p = new Promise<>();
  p.invoke(v);
  return p;
}
```

Promise<V>: bind

```java
public <R> Promise<R> bind(final Function<V, Promise<R>>
function) {
  Promise<R> result = new Promise<>();
  this.onRedeem(callback -> {
    V v = callback.get();
    Promise<R> applicationResult = function.apply(v);
    applicationResult.onRedeem(c -> {
      R r = c.get();
      result.invoke(r);
    });
  return result;
}
```

Promise<V>: get

```
public V get() throws InterruptedException,
ExecutionException {
    taskLock.await();
    if (exception != null) {
        throw new ExecutionException(exception);
    }
    return result;
}
```

Example

```
Promise<String> p = Async.submit(() -> {
  return  "hello world";
});

Promise<Integer> result = p.bind(string ->
Promise.pure(Integer.valueOf(string.hashCode())));

System.out.println("HashCode = " + result.get());
```

Checkpoint

> Promise - represents async computation

> Values and exceptions handling

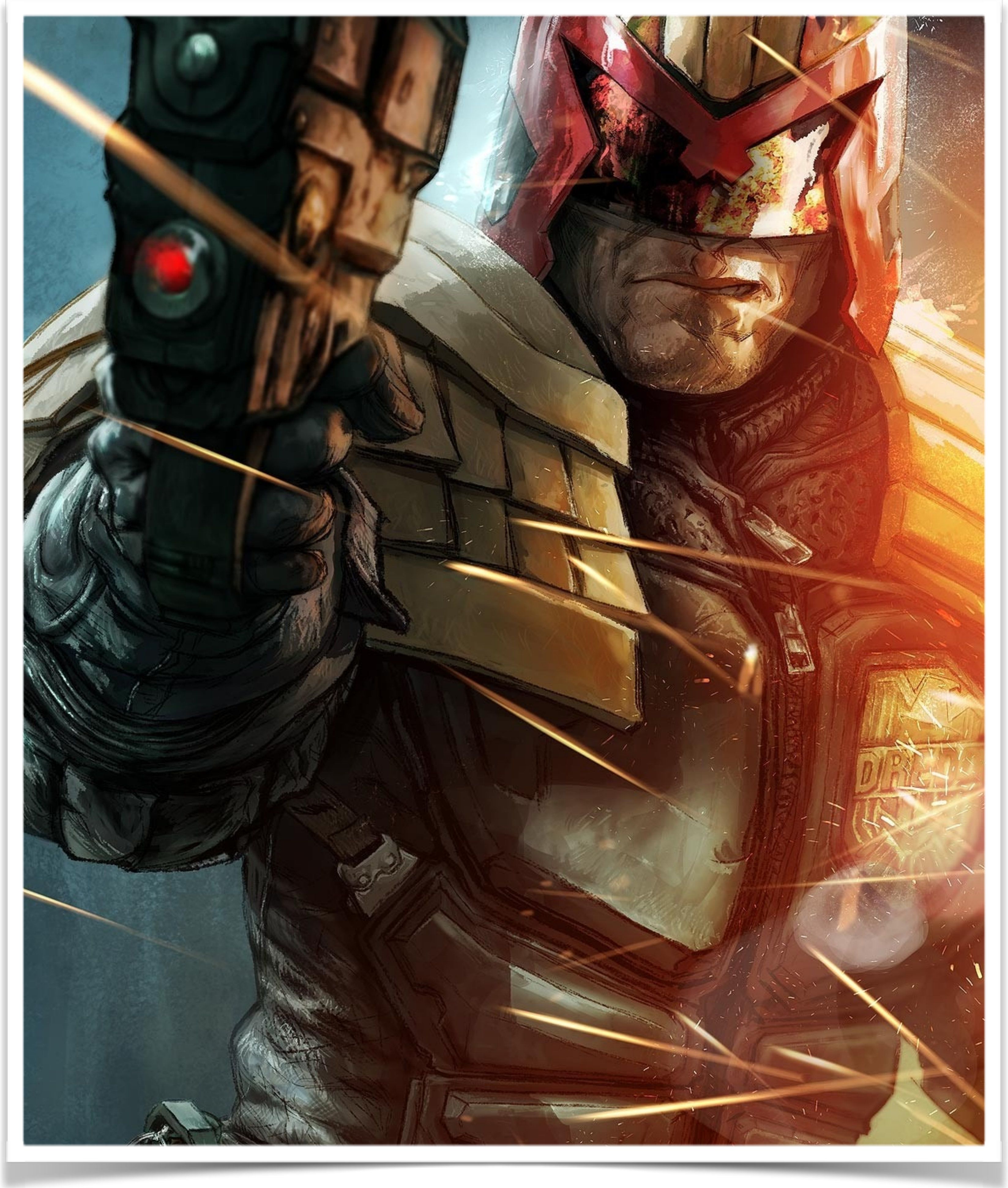
> Chaining of functions

Wait, is that it?

> **Monad** vs. **Instance of monad**

Typeclass? Higher functions?

- Common interface
- Generic functions over all monads



Greatness

- Common operations for Monads
  - sequence, zip
- Limited under parametrised polymorphism in Java

Some languages have it easier than others

> m >> n = m >>= _ -> n

> f x a = … a

> Implicit get()

Monad in Java

```
public interface Monad<V> {
  Monad<V> pure(V v);
  <R> Monad<R> bind(Function<V, Monad<R> f);

  V get();
}
```

> One does not simply call itself a monad!

Laws

- return a >>= f ≡ f a
- m >>= return ≡ m
- (m >>= f) >>= g ≡ m >>= (\x -> f x >>= g)

Left identity

> pure(v).bind(f) ≡ f.apply(v)

Right identity

> m.bind(m::pure) ≡ m

Associativity

```
> m.bind(f).bind(g) ≡ m.bind(
                      (v) -> f.apply(v).bind(g))
```

Some have it easy

- Referential transparency
- Partial application
- ≡ is easy

Mortal platforms

- No referential transparency
- f.apply(v) != f.apply(v)
- equals() + hashcode()

Defining ≡ for Java

> Side effects are similar

> m.get() observes the same values

> values or exceptions

Promise: left identity

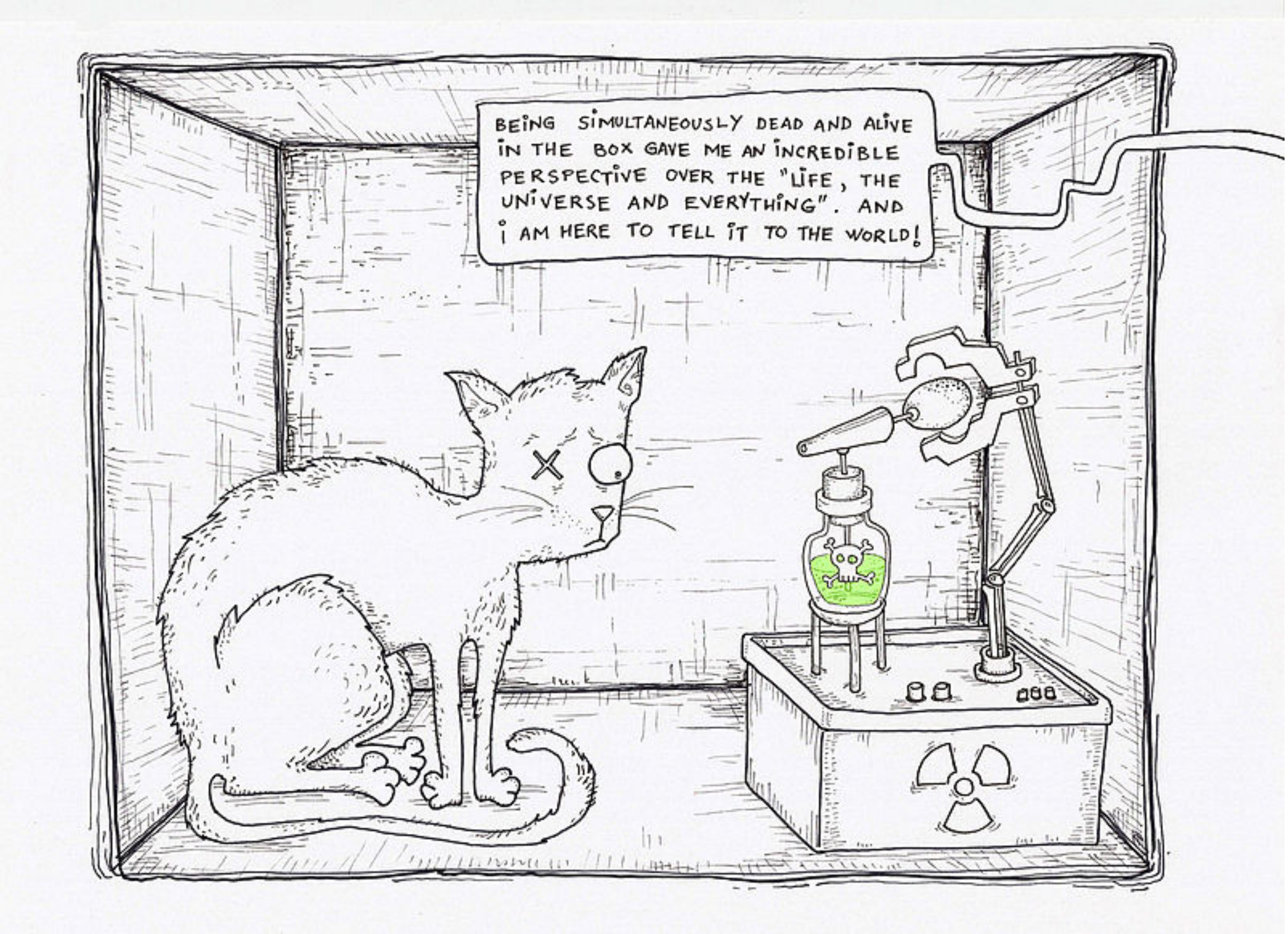
```
Function<Integer, Promise<Boolean>> f =
(x) -> {
  return submit(() -> x % 2 == 0);
};
Integer val = new Integer(100);

assertEquals(Promise.pure(val).bind(f),
             f.apply(val).get());
```

Promise: right identity

```
Integer val = new Integer(100000);
Promise<Integer> p =
Promise.pure(val).bind(Promise::pure);

assertEquals(val, p.get());
assertEquals(identityHashCode(val),
             identityHashCode(p.get()));
```

Quality software

- java.util.concurrent.**CompletableFuture**
- thenApply(Function / Consumer / etc)
- thenApplyAsync(Function / etc)
- Async => FJP.common()


BEING SIMULTANEOUSLY DEAD AND ALIVE IN THE BOX GAVE ME AN INCREDIBLE PERSPECTIVE OVER THE "LIFE, THE UNIVERSE AND EVERYTHING". AND I AM HERE TO TELL IT TO THE WORLD!

Optional pure

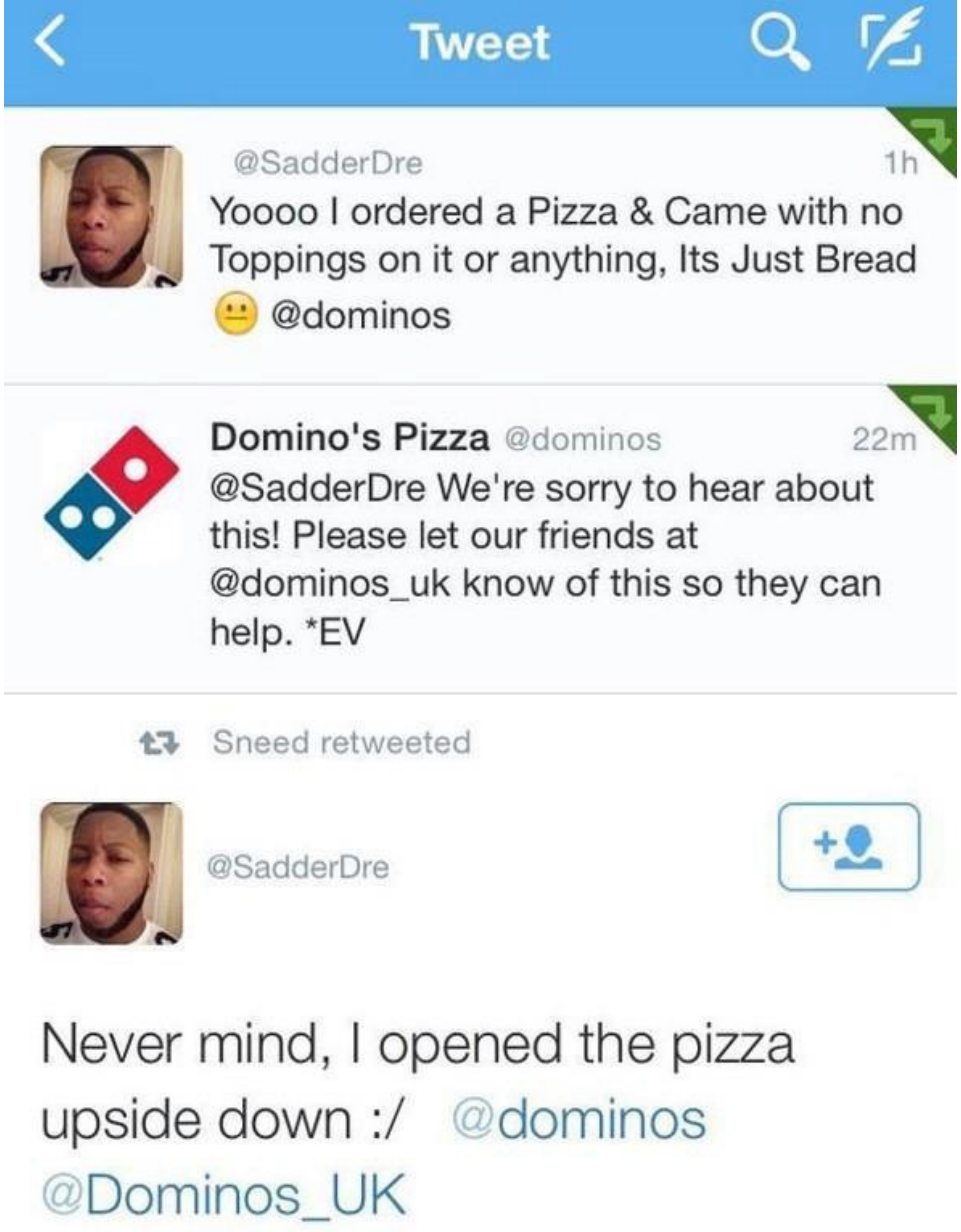
```
static <T> Optional<T> of(T value) {
    return new Optional<>(value);
}

static <T> Optional<T> ofNullable(T value) {
    return value == null ?
             empty() :
             of(value);
}
```

Optional bind

```
public<U> Optional<U> flatMap(Function<T,
                              Optional<U>> mapper) {
  Objects.requireNonNull(mapper);
  if (!isPresent())
    return empty();
  else {
    return Objects.requireNonNull(mapper.apply(value));
    }
}
```

Optional bind

```
public<U> Optional<U> map(Function<T, U> mapper) {
  Objects.requireNonNull(mapper);
  if (!isPresent())
    return empty();
  else {
    return
        Optional.ofNullable(mapper.apply(value));
    }
}
```



@SadderDre 1h

Yoooo I ordered a Pizza & Came with no Toppings on it or anything, Its Just Bread 😐 @dominos

Domino's Pizza @dominos 22m

@SadderDre We're sorry to hear about this! Please let our friends at @dominos_uk know of this so they can help. *EV

Sneed retweeted

@SadderDre

Never mind, I opened the pizza upside down :/ @dominos @Dominos_UK

Feedback!

Contact me

ZEROTURNAROUND

oleg@zeroturnaround.com

@shelajev

github.com/shelajev/promises

Minority report

- Alternative definition of Monad:
  - fmap :: (a -> b) -> f a -> f b
  - join :: m (m a) -> m a

Exercises

- Implement (>>=) in terms of fmap and join.
- Now implement join and fmap in terms of (>>=) and return.