EFF MONAD

ONE MONAD TO CONTROL THEM ALL

JAMES CARRAGHER

- Software Engineer at Paidy
- Functional Programing enthusiast

AGENDA

- What are effects?
- How do I use one effect?
- How do I use many effects?
- Eff Monad
- The problems Eff solves
- How Eff works
- How you can use it
- Conclusion

EFFECTS

- What are effects?

EFFECTS

- **Option[A]**: may or may not contain a value **A.**
- **Either[A, B]**: either contains a value **A** or a value **B**.
- **Future[A]**: may return a value of type **A** in the future.

HOW DO I USE ONE EFFECT?

OF THE SAME TYPE: OPTION[A]

```
for {
  person1 <- Some("James")
  person2 <- Some("Mary")
  person3 <- Some("Bob")
} yield s"Hello $person1, $person2, and $person3"

// Some("Hello James, Mary, and Bob")
```

```
for {
  person1 <- Some("James")
  person2 <- None
  person3 <- Some("Bob")
} yield s"Hello $person1, $person2, and $person3"

// None
```

HOW DO I USE ONE EFFECT?

OF THE SAME TYPE (FUTURE[A])

```
for {
  person1 <- Future.successful("James")
  person2 <- Future.successful("Mary")
  person3 <- Future.successful("Bob")
} yield s"Hello $person1, $person2, and $person3"

// Success("Hello James, Mary, and Bob")
```

```
for {
  person1 <- Future.successful("James")
  person2 <- Future.failed(new Exception("Person2 not found"))
  person3 <- Future.successful("Bob")
} yield s"Hello $person1, $person2, and $person3"

// Failure(Exception("Person2 not found"))
```

HOW DO I USE TWO EFFECTS?

EITHER[ERROR, _] + FUTURE[_]

```scala
case class User(email: String)
case class Error(message: String)

def isValidEmail(email: String): Either[Error, String] =
  Either.cond(email.contains("@neopets.com"), email, Error("Lame email address"))

def getUserByEmail(email: String): Future[Either[Error, User]] =
  Future.successful(Right(User(email)))
```

```scala
def getUser(email: String): Future[Either[Error, User]] = for {
  validEmail <- isValidEmail(email)
  user <- getUserByEmail(validEmail)
} yield user

getUser("mary@neopets.com") // Success(Right(User("mary@neopets.com")))
```

NOT POSSIBLE

A FEW SOLUTIONS

- Monad transformers
- Free Monad

EFF!

THE EFF MONAD

HISTORY

Introduced in the paper: Freer monads, more extensible effects

Introduced to scala by: atnos-org/eff

THE PROBLEMS IT SOLVES

- Combining multiple effects (which can be painful as we've seen)
- Declaratively specifying what your program *does*
  - By separating your program's *description* from its *interpretation*
  - With effect handers

HOW EFF WORKS

You define a type-level list of "effect constructors"

```scala
import org.atnos.eff._
import org.atnos.eff.all._
import org.atnos.eff.syntax.all._
import monix.eval.Task // we use Task to wrap Futures

type EitherError[A] = Either[Error, A] // A monad is a single arity type constructor

type Stack = Fx.fx2[EitherError, Task] // Your effect stack

type _eitherError[R] = EitherError |= R // EitherError is a member of R
```

USING EFF

```scala
import org.atnos.eff.addon.monix.task._

def getUser[R : _eitherError : _task](email: String): Eff[R, Either[Error, User]] = for
  email <- fromEither(isValidEmail(email))
  user <- fromTask(Task.fromFuture(getUserByEmail(email)))
} yield user
```

```scala
val prog = getUser[Stack]("mary@neopets.com") // Eff.ImpureAp(...) - description

val task = prog.runEither.runAsync // Task[Either[Error, User]] - interpretation

Await.result(task.runAsync, 10.seconds) // Right(User("mary@neopets")) - result
```

WAT.

- Where did `fromEither` etc. come from?
- Eff provides these functions to lift your monad into `Eff`
- There are similar lifting functions for many commonly used types:
  - State
  - Reader
  - cats.effect.IO
  - etc.
- You can define your own effects!

EASY MOCKING

- We can pass parameters to our interpreters
- A notorious problem in testing is how do you pass time into your app?
  - Using `Time.now()` in your app prevents you from mocking time in tests

```
import io.pjan.effx.time._

type Stack = Fx.fx2[EitherError, Time]

def emailAndTime[R : _eitherError : _time](email: String): Eff[R, Either[Error, (String
  email <- fromEither(isValidEmail(email))
  time <- timestamp
} yield (email, time)

val prog = getUser[Stack]("mary@neopets.com")
```

DEFINING TIME

Define what *time* means in your app

In production

```
val liveResult = prog.runEither.runTime(Clock.live) // Pass the interpreter a live Cloc
```

In tests

```
val newYearTime = OffsetDateTime.of(2018, 1, 1, 0, 0, 0, 0, ZoneOffset.UTC)

val frozenResult = prog.runEither.runTime(Clock.frozen(newYearTime)) // Pass it a froze
```

CONCLUSION

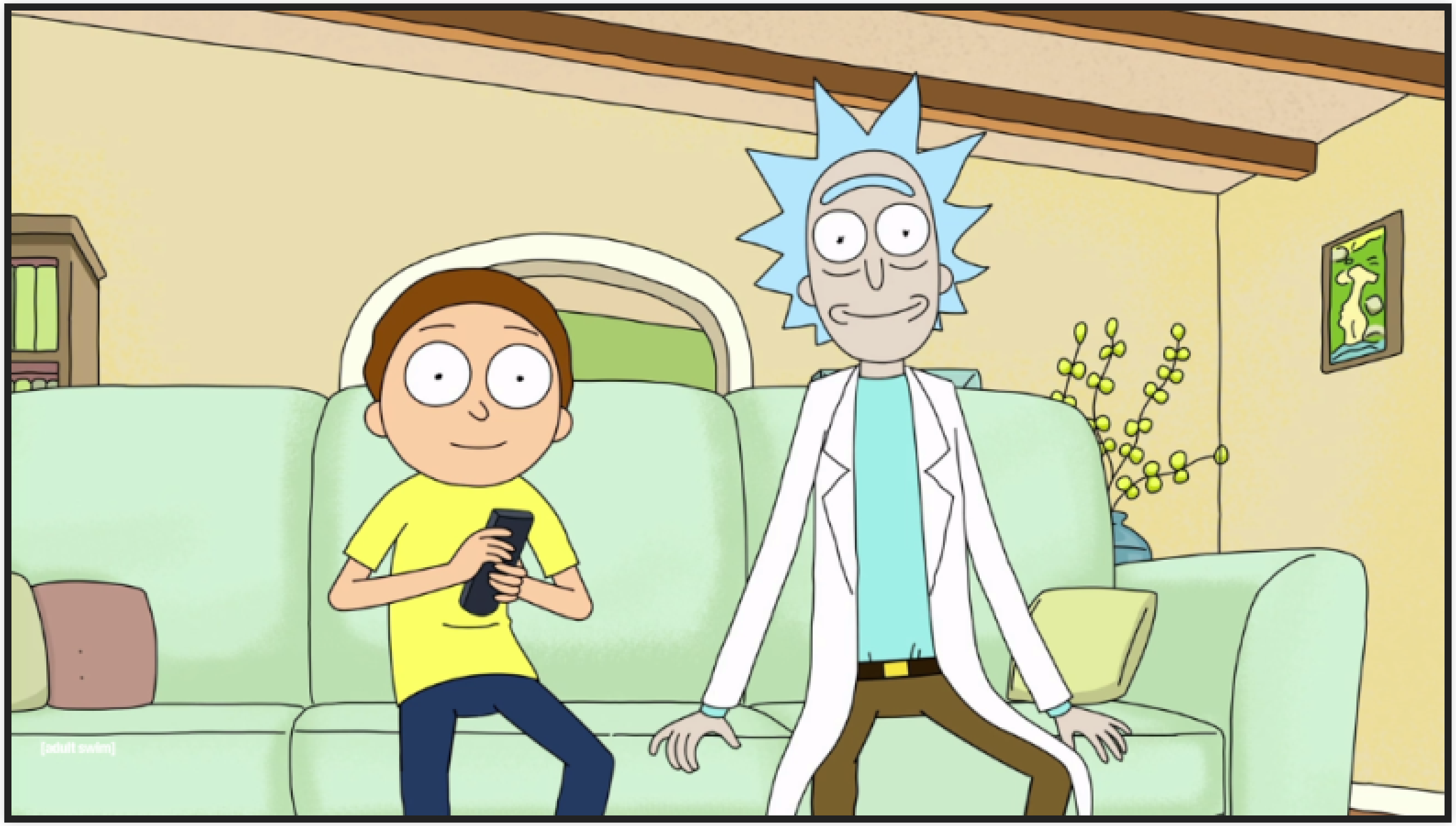
Eff provides you fine grained control over how your code behaves

Great documentation, go check it out!

Overhead, this stuff is not easy

BUT IT'S WORTH IT!

QUESTIONS?



paidy

カードのいらないカンタン決済

WE ARE HIRING!

https://engineering.paidy.com/