Hands-on Arrow

(How we filled Kotlin standard library gaps without converting code to Scala)

Karin-Aleksandra Monoid
GDG Nuremberg, 2021

Agenda

- What is Arrow?
- NonEmptyList
- Either
- Monad Comprehensions

What is Arrow?

What is Arrow?

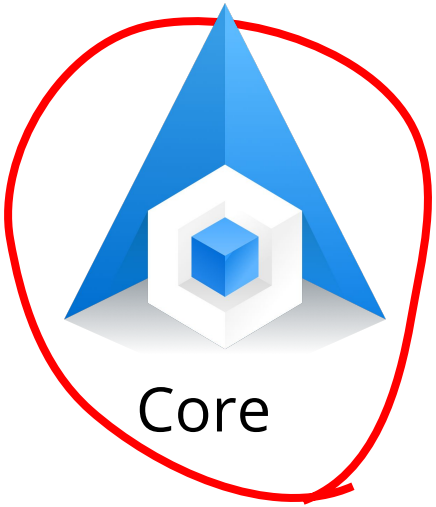
Core

FX

Optics

Meta

What is Arrow?



Core

FX

Optics

Meta

NonEmptyList

Non-empty list

Payments	List of items (Phone, Reference, Amount)		
Phone	...	String(100)	Required
Reference	...	String(100)	Required
Amount	...	Decimal	Required

Is JSON `{“payments”: []}` ok?

Non-empty list

```
NonEmptyList.of() // does not compile

NonEmptyList.of(1, 2, 3, 4, 5) // NonEmptyList<Int>
```

Non-empty list

```
NonEmptyList.of() // does not compile

NonEmptyList.of(1, 2, 3, 4, 5) // NonEmptyList<Int>
```

Either

Java Exceptions

```
if (somethingWrong)
   throw new SomeException(...)
```

Java Exceptions - “Handling” in signature

```
void testFun() throws SomeException
```

Java Exceptions - Handling with `try`

```
try {
    testFun()
} catch (SomeException e) {
    ...
}
```

Java Exceptions

```
IntStream.range(0, 10)
         .forEach(i -> testFun())
```

Java Exceptions

```
IntStream.range(0, 10)
         .forEach(i -> testFun())
```

Java Exceptions

```
IntStream.range(0, 10)
    .forEach(hideException(i -> testFun()));
```

Java Exceptions

```
unhideSomeException(() ->
    IntStream.range(0, 10)
        .forEach(
            hideException(i ->
                testFun()
));
```

Kotlin Exceptions

- No more checked exceptions (like Scala, Groovy and everyone else)

Kotlin Exceptions

```
(1 until 10).forEach { testFun() }
```

Null

```
fun String.toIntOrNull(): Int?
```

- Not expressive
- ?????
- null-checks

Result Type

```
inline class Result<out T> : Serializable
```

Result Type

```
fun deserialize(): Something {
   return try {
     doSomething()
   } catch (_: Exception) {
     try {
       doSomethingElse()
     } catch(_: Exception) {
       doSomethingElseAgain()
}}...
```

```
fun deserialize(): Something {
   return runCatching {
       doSomething()
   }.recoverCatching {
       doSomethingElse()
   }.getOrElse {
       doSomethingElseAgain()
   }
}
```

Sealed Type

```
sealed class Result

class Success : Result()

class FullMoonFailure : Result()

class ElonMuskTweetedFailure : Result()
```

Either

```
sealed class Either<out A, out B>
```

Validated

```
sealed class Validated<out A, out B>
```

Validated vs Either

- Either stops at first fail, validated collects all:

```
User(phone,  name,        age)
User(“test”, XÆA-12 Musk, -10)
```

- Either: invalid phone
- Validated: invalid phone, name*, age

* https://www.kalzumeus.com/2010/06/17/falsehoods-programmers-believe-about-names/

Monad comprehensions

Require

```
requireNotNull(request.userId)
requireNotNull(request.transaction.id)
```

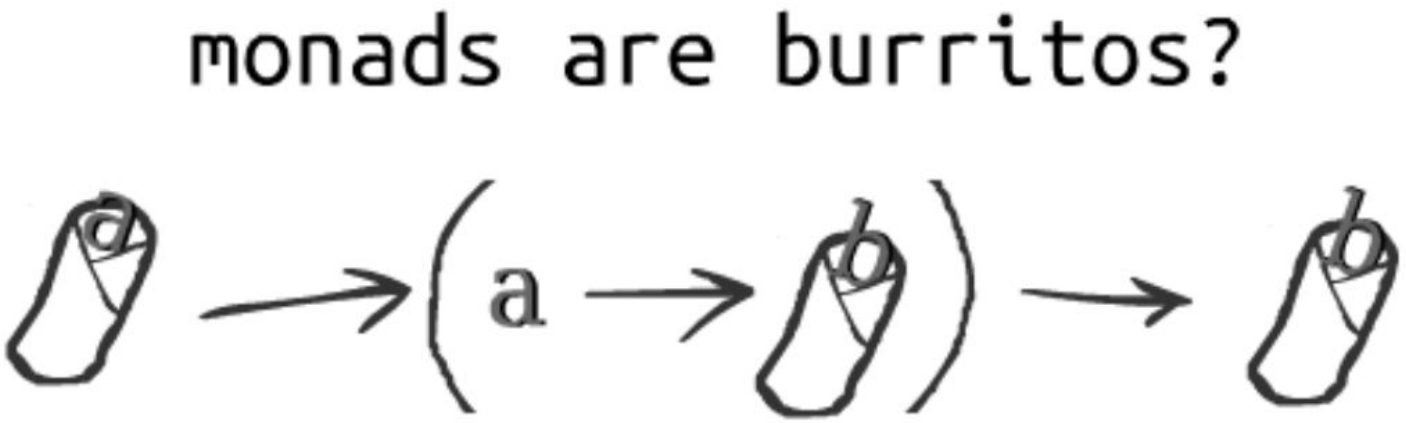
Require

```
public inline fun <T : Any>
   requireNotNull (value: T?): T {
      contract {
         returns() implies (value != null)
   }
      return requireNotNull(value) {
        "Required value was null."
   }
}
```

if-else-if-else-if-else

```
if(request.userId == null)
  return buildExceptionResponse(...)
if(request.transaction.id == null)
  return buildExceptionResponse(...)
...
```

Arrow Monads



https://chrisdone.com/posts/monads-are-burritos/

Arrow Monads

```
inline fun <T, R> Iterable<T>.flatMap(
    transform: (T) -> Iterable<R>
): List<R>
```

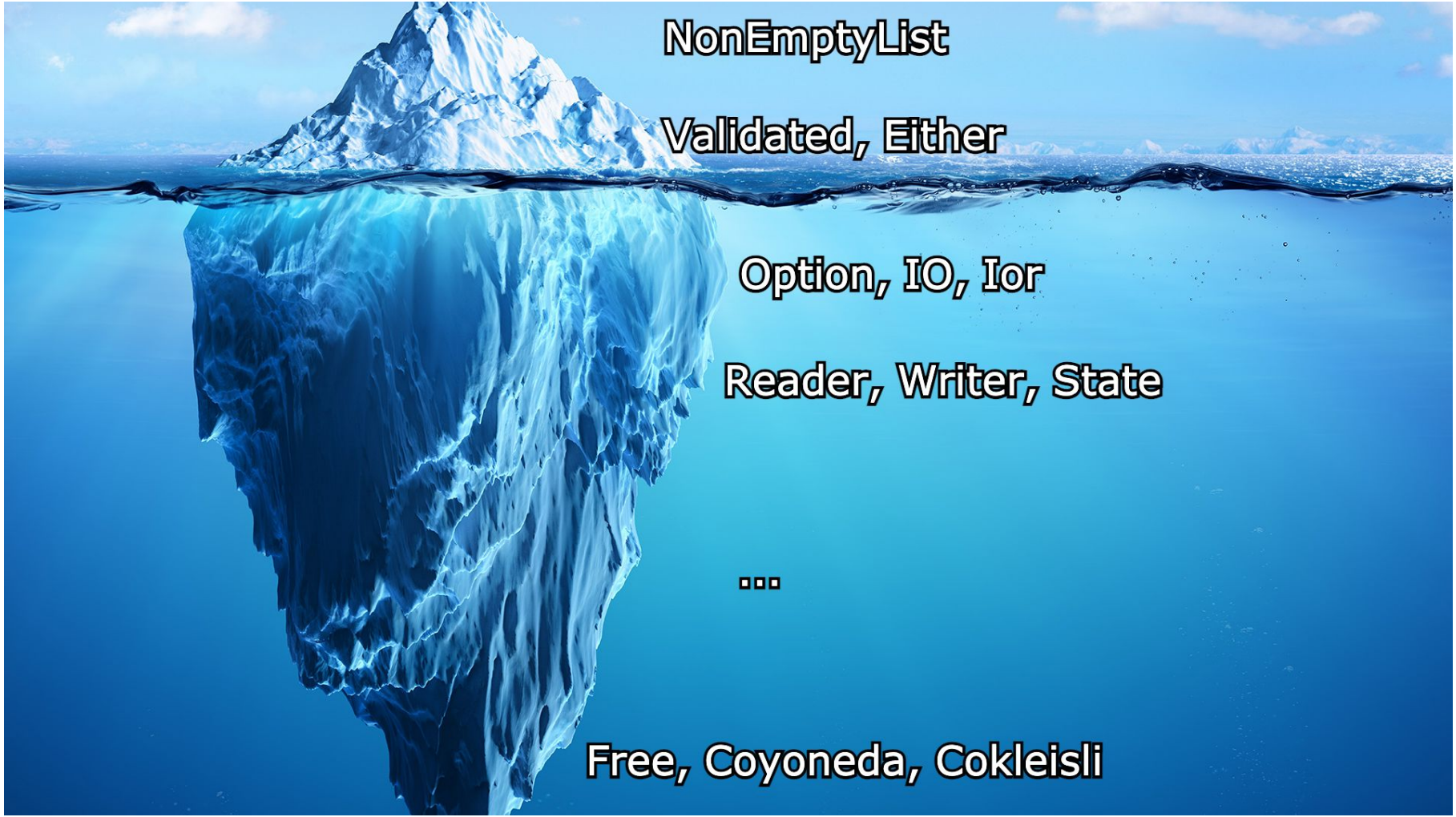
Chaining - flatMap

```
val response = checkForNull(...)
  .flatMap { userId ->
      checkForNull(...).flatMap { paymentId ->
         checkForNull(...).flatMap { items ->
            checkForNull(...).flatMap { status ->
               checkForNull(...).map {
( … )}}}}}}}}}
```

Chaining - monad comprehensions

```
val response = Either.fx {
    val userId = checkForNull(request.userId).bind()
    val paymentId = checkForNull(request.transaction.id).bind()
    val items = checkForNull(request.transaction.items).bind()
    val status = client.checkPaymentStatus(userId,...) // ... }
return response.getOrHandle { it }

private fun <T : Any> checkForNull(param: T?, paramName: String):
     Either<ErrorResponse, T>
```


NonEmptyList
Validated, Either
Option, IO, Ior
Reader, Writer, State
...
Free, Coyoneda, Cokleisli