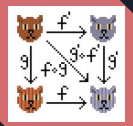

Scala with Cats
&
Kotlin with Arrow

….yet another one 😪

Style

[A] vs <A>

Interoperability

- Kotin was built to interoperate with Java at Jetbrains, as they were looking for a better Java, but they noted the slow compilation time for Scala for a deal-breaker for them (Note: This is prior to Dotty/Scala 3)
- Although Scala does interoperate with Java just fine, there are a few quirks

```java
// Java
interface Api {  // Assume implementation is class MyApi
    void setNumbers(List<Integer> numbers);
}

//Kotlin
val api = MyApi()
api.addNumbers(listOf(1, 2, 3))

//Scala
import scala.jdk.CollectionConverters._
val api = new MyApi()
api.setNumbers(List(1, 2, 3).map(Integer.valueOf).asJava)
```

Pattern Matching - Part I

- Both options provide some form or pattern matching, however the Scala way is unmatched, as Kotlin when does not possible to destructure branches

```scala
private def patternMatchMulti(value: Thing): Unit = {
  value match {
    case Action(_, _) => patternMatchSimple(value)
    case Button(_, 3) => println("its button 3")
    case _ => println("it's a dud")
  }
}
```

```kotlin
private fun patternMatchMulti(value: Thing) = when {
    value is Action -> patternMatchSimple(value)
    value is Button && value.no == 3 -> println("its button 3")
    else -> println("it's a dud")
}
```

Pattern Matching - Enums/ADT's

- Kotlin has Enumerations to provide existing matching over ADT's, while the current version of Scala we use does not(Scala 3 has them 😄)

```kotlin
enum class Color {
    Red,
    Green,
    Yellow
}

// The following will not compile if it's missing a branch
private fun patternMatchEnum(color: Color) = when(color) {
    Red -> println("It's red")
    Green -> println("It's green")
    Yellow -> println("It's yellow")
}
```

```scala
sealed trait Color
case object Red extends Color
case object Green extends Color
case object Yellow extends Color
// The following will compile even if missing a branch
private def patternMatchEnum(color: Color): Unit = {
  color match {
    case Red => println("It's red")
    case Green => println("It's green")
    case Yellow => println("It's yellow")
  }
}
```

Kotlin Coroutines

- Generalize subroutines for cooperative multitasking, which can be suspended and resumed
- In kotlin land, a coroutine is a compiled suspend -> A
- Much like IO, a coroutine can be deferred, suspended, run in parallel or in sync

Why and why not try Kotlin with Arrow?

- ✅ Kotlin is 3x as popular compared Scala according to the 2021 StackOverflow survey. Although popularity might not be a selling point for many, more users = more resources.
- ✅ Excellent Java interpretability, excellent tooling(thanks Jetbrains)
- ✅ Some highlights
  - Null safety with nullables
  - Lazy properties
  - Typesafe DSL builders

- ❌ Arrow, although a very rich solution, not as battle tested as the Type Level ecosystem
- ❌ As Kotlin does not aim to the mostly-functional replacement, much of the ecosystem is not plug and play with Arrow
- ❌ Arrow has had to change it's API many times in order to address current limitations of the Kotlin compiler, like removing HKT's.
- ❌ Arrow is not yet 1.0, getting very close

Effects

Arrow

- Effects managed via suspend() -> A

```
suspend fun number(): Int = 1

suspend fun triple(): Triple<Int, Int, Int> =
  Triple(number(), number(), number())
```

Cats Effect

- Effects managed via IO[A]

```
def number(): IO[Int] = IO(1)

def triple(): IO[(Int, Int, Int)] = {
  for {
    a <- number()
    b <- number()
    c <- number()
  } yield (a, b, c)
}
```

suspend() -> A over IO[A]/F[_]

- An effect can be described as suspend() -> A
- We use IO to encapsulate nasty side effects in a safe and referential transparent manner
- IO provides powerful concurrent and cancellation operators.
- suspend() -> A provides that same benefits with the bonus added they are native to the language (one less thing to worry about)
- We always have to work with the IO/F type to access the value we care about within. In cats this is solved using Transformers

```
suspend fun ioProgram(): Either<PersistenceError, ProcessedUser> =
    either {
        val user = fetchUser().bind()
        val processed = user.process().bind()
        processed
    }
```

```
def ioProgram(): IO[Either[PersistenceError, ProcessedUser]] = {
  val result = for {
    user <- EitherT(fetchUser())
    processed <- EitherT(user.process())
  } yield processed
  result.value
}
```

Higher Kinded Types - Part I

- A HKT is a type that abstracts over a type, which in turn abstracts over another type. Below is defined a trait which takes E as a parameter, which in E in turn takes a parameter _

```
trait FirstOnlyCollection[E[_]] {
  def first[A](b: E[A]): A
}
```

```
val list: FirstOnlyCollection[List] = new FirstOnlyCollection[List] {
  override def first[A](b: List[A]): A = b.head
}
```

😞 Neither Kotlin nor Arrow provides such a feature

😞 Arrow did in fact have the Kind<*> type, but was phased out, as the kotlin compiler does not have proper support for plugins. Arrow in fact does have a product called Arrow-Meta, for writing compiler plugins, however they have decided to leave it out of the API, until the Kotlin IDEA plugin support them, in order to finalize the API for Arrow 1.0

Higher Kinded Types - Part II

- We usually use HKT's for effects, and those can be emulated sufficiently via suspend functions and concrete types. For example most normal applications use can get by with using IO = suspend and Either for modeling errors.
- HKT's are not limited to effects only, but it is the most common use case

HKT use cases

- Abstracting over containers of values, i.e trait Iterable[A, Container[_]]
  - This is not available in Kotlin/Arrow
  - This might become available as Kotlin address issues in it's compiler, but as of now, no.
- Stacking monads for effects
  - This is replaced in arrow with suspend functions and comprehensions, but not as good as Scala. This again might be addressed in the future

Examples

Refer to code

All done! 👏

- Next time you building something simple in your free time, maybe give Arrow and Kotlin a go, learning something new is fun, you never know what you end up liking
- Kotlin and Scala are cool!
- FP is hard
- Feedback always welcome!