Step 1

Modes of Composition in functional Scala programming

Ben Hutchison

Lambdajam Online 2020

Step 1: Decoding an Order Message

- Decoding an Order Message will introduce
  - the effect type F
  - typeclasses on the effect type, the so called “tagless final” style
  - Strategies for error representations in effects
  - running unit tests

Zooming in on the problem

```
object OrderProcessor {

 def decodeMsg[F[_]: ApplicativeError[*[_], Throwable]](msg: Array[Byte]): F[OrderMsg] =
???

}
```

Zooming in on the problem

```
object OrderProcessor {

 def decodeMsg[F[_]: ApplicativeError[*[_], Throwable]](msg: Array[Byte]): F[OrderMsg] =
???

}
```

Method name

Zooming in on the problem

```
object OrderProcessor {

 def decodeMsg[F[_]: ApplicativeError[*[_], Throwable]](msg: Array[Byte]): F[OrderMsg] =
???

}
```

methods defined on scala objects are “static”; i.e. can be called from anywhere, and have a fixed, concrete implementation

Zooming in on the problem

```
object OrderProcessor {

 def decodeMsg[F[_]: ApplicativeError[*[_], Throwable]](msg: Array[Byte]): F[OrderMsg] =
???

}
```

the section in **[..]** are the *type parameters* set at each call-site during compile (comma separated) and any *typeclass constraints* (aka “context bounds”) on type-parameters, each following a colon.

here there is one type parameter **F**, and it has one typeclass **ApplicativeError[*[_], Throwable]** (unfortunately for a first example, this is quite a complex typeclass signature, but we’ll look into its meaning soon).

Zooming in on the problem

```
object OrderProcessor {

 def decodeMsg[F[_]: ApplicativeError[*[_], Throwable]](msg: Array[Byte]): F[OrderMsg] =
???

}
```

F[_] is the *effect type*. The **[_]** signals that its a higher-kinded (“container”) type. Called **F** by convention

Zooming in on the problem

```
object OrderProcessor {

 def decodeMsg[F[_]: ApplicativeError[*[_], Throwable]](msg: Array[Byte]): F[OrderMsg] =
???

}
```

The method returns an **OrderMsg** *payload type*, wrapped in the effect type **F[_]**.

Zooming in on the problem

```scala
object OrderProcessor {

 def decodeMsg[F[_]: ApplicativeError[*[_], Throwable]](msg: Array[Byte]): F[OrderMsg] =
???

}
```

F has one typeclass constraint **ApplicativeError[*[_], Throwable]**. This says that F’s effects must include the ability to raise and handle errors of type **Throwable**.

Practically, the methods of **ApplicativeError** become available in the body of this method

Zooming in on the problem

```
object OrderProcessor {

 def decodeMsg[F[_]: ApplicativeError[*[_], Throwable]](msg: Array[Byte]): F[OrderMsg] =
???

}
```

Finally, value parameters passed at runtime go in round brackets. In this case the array of bytes we must decode.

Quiz: Check your understanding

```
object OrderProcessor {

 def decodeMsg[F[_]: ApplicativeError[*[_], Throwable]](msg: Array[Byte]): F[OrderMsg] =
???

}
```

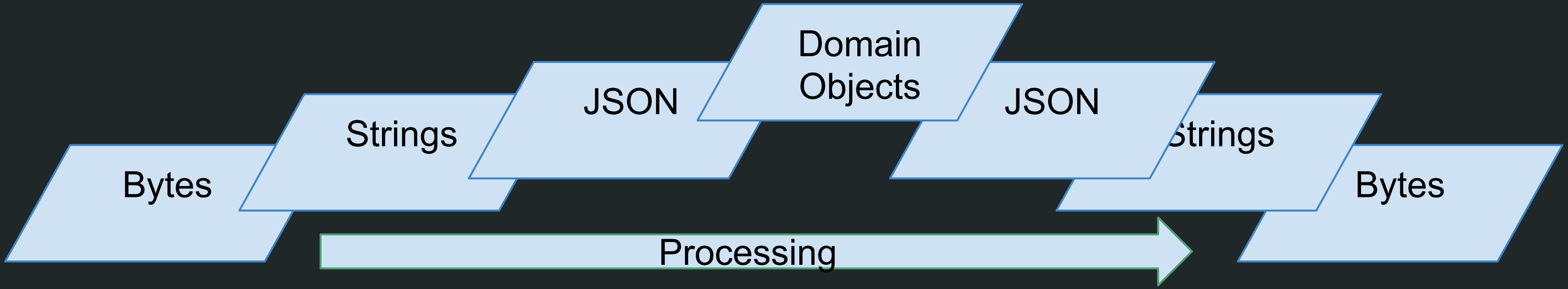
What effects can be used in the body of decodeMsg?

❑ No effects - json parsing must be “pure”
❑ Any side-effects
❑ Depends what F-type passed in
❑ Raising errors
❑ Handling errors

Modes of Composition

- An effectful value is internally composed of layers
  - Different rules or invariants apply in each layer

- The typical structure sees weakly typed layers on the outer rings of an application and one to many more strongly typed interior layers
  - Data representation is different in each layer



Building Blocks: Character Decoding

```
new String(msg)
```

Legacy JVM string constructor decodes bytes using platform default encoding (we'll consider that good enough)

Throw exception if the bytes can't be decoded

Building Blocks: Json Decoding

```
parser.decode[A](jsonString: String): Either[Error, A]
```

Circe library parses and then decodes a String to an structured type

Uses compile-time reflection over the specified type **A** to know what json fields to read

returns either an exception or the parsed A

Building Blocks: Throwing Errors

```
errorValueFromEither[F: ApplicativeError[*[_], Throwable], E, A](e: =>Either[E, A]): F[A]
```

errorValueFromEither is a provided combinator that accepts an Either[E, A] and “eats” the left-side. ie If the Either is a Left error, it uses the error effect to lift the error into the F effect.

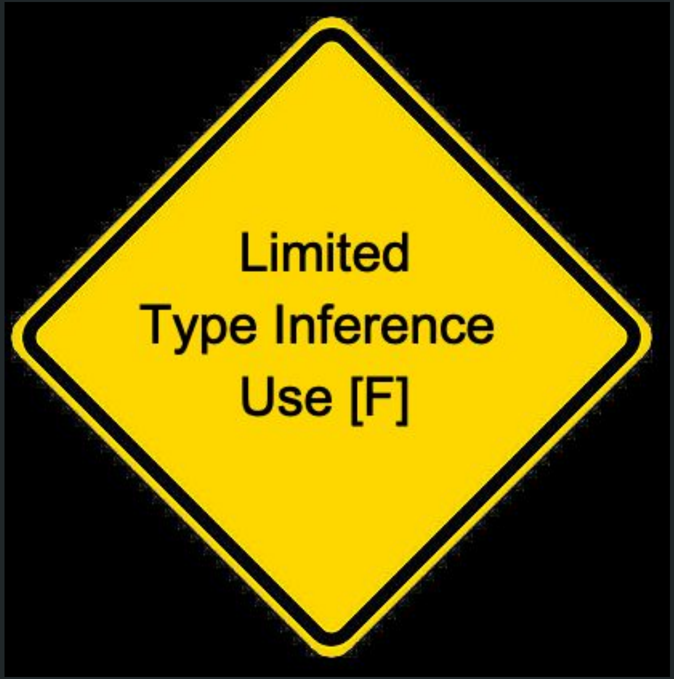
It also traps any exceptions thrown in evaluating the Either parameter and lifts them into the F effect

Have a go

- Try to combine the building blocks to implement **decodeMsg**
- Run the unit tests to check your work
  - step1/test

Advice:

- Scala’s Type Inference is limited
  - Methods that take the effect type F as a type parameter will likely need to have F specified
  - If not, likely to see compiler error messages about diverging or ambiguous implicits



Advice: Where are the imports?

- Almost are the required imports are provided at a project level
  - Since Scala 2.13.x, SBT can specify a set of package imports that are automatically available
  - See **`build.sbt`** file for details