Type Classes in Scala

What? Why? How?

Daniel Reigada

IST - Programming Languages

August 15, 2018

Monad error

A typeclass defined that allows one to abstract over error-handling monads.

Monad error

A typeclass defined that allows one to abstract over error-handling monads.

But what does that mean?

Monad error

It allows the creation of generic code, that would otherwise return errors wrapped in a specific monad.

Monad error

It allows the creation of generic code, that would otherwise return errors wrapped in a specific monad.

OKAY! But what does that mean?!?

Monad error - example

```
def divide(num: Int, denom: Int): Int = num / denom
```

Monad error - example

What if *denom is 0 ?*

```
def divide(num: Int, denom: Int): Int = num / denom
```

Monad error - example

We can improve this using Option...

```
def divide(num: Int, denom: Int): Option[Int] =
  if(denom == 0) None else Some(num / denom)
```

Monad error - example

...or Try...

```
def divideTry(num: Int, denom: Int): Try[Int] =
  if (denom == 0) Failure(new Throwable("Division by 0"))
  else Success(num / denom)
```

Monad error - example

...or Future...

```
def divideFuture(num: Int, denom: Int): Future[Int] =
  if (denom == 0) Future.failed(new Throwable("Division by 0"))
  else Future.successful(num / denom)
```

Monad error - example

...or Either...

```
def divideEither(num: Int, denom: Int): Either[String, Int] =
  if (denom == 0) Left("Division by 0")
  else Right(num / denom)
```

Monad error - example

...or a custom result type (i.e. foundation Response)...

```
def divideEither(num: Int, denom: Int): Result[Int] =
  if (denom == 0) Result.error("Division by 0")
  else Result.success(num / denom)
```

Monad error - example

... you get the idea.

What if you are trying to write generic code (i.e. library)?
We should be able to abstract the method

```
def divide???(num: Int, denom: Int): F[Int] =
  if (denom == 0) ERROR_CASE
  else SUCCESS_CASE
```

Monad error - example

By the power of the Monad!!!

Monad error - example

```
def divideF[F[_]](num: Int, denom: Int)(
    implicit M: MonadError[F, Throwable]): F[Int] = {
  if (denom == 0) M.raiseError(new Throwable("Division by 0"))
  else M.pure(num / denom)
}
```

Monad error - example

```
def getNum: Try[Int] = ???
def getDenom: Try[Int] = ???

for {
  num <- getNum
  denom <- getDenom
  result <- divideF[Try](num, denom)
} yield result
```

Monad error - example