zalando

Functional resource safety management

Functional resource safety management with the `Bracket` typeclass and `Resource`

Javier Arrieta

23-07-2019

TABLE OF CONTENTS

Zalando

Javier

The problem

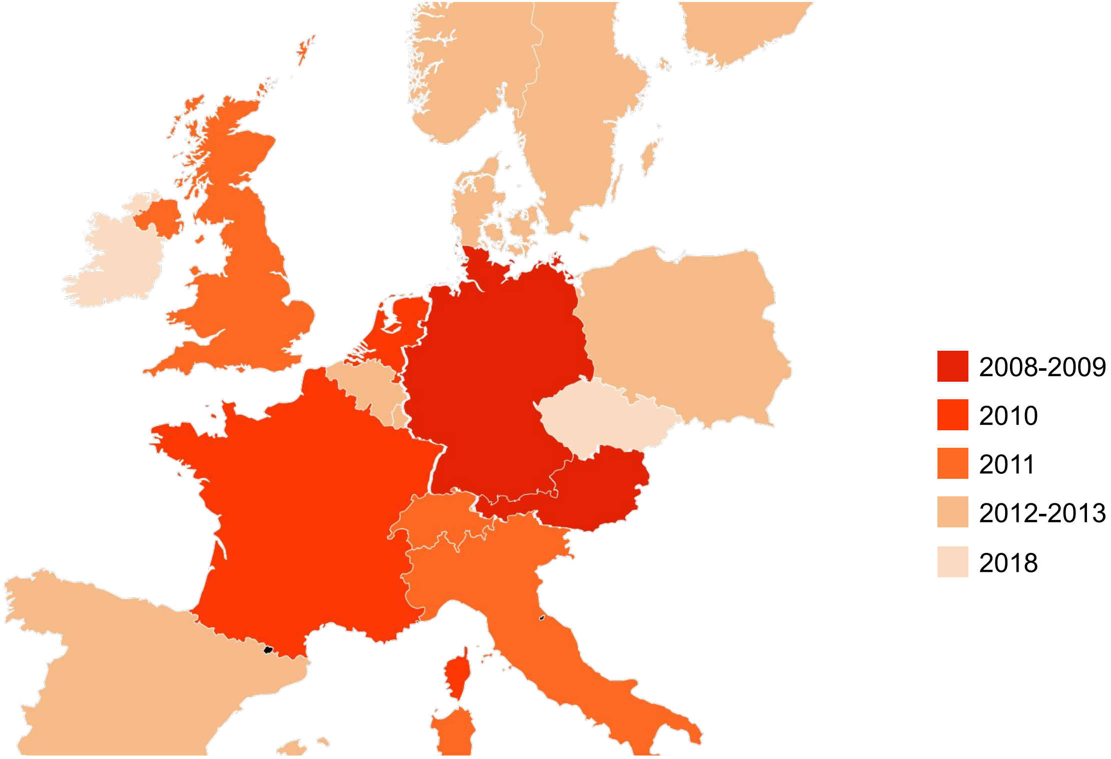
The bracket typeclass

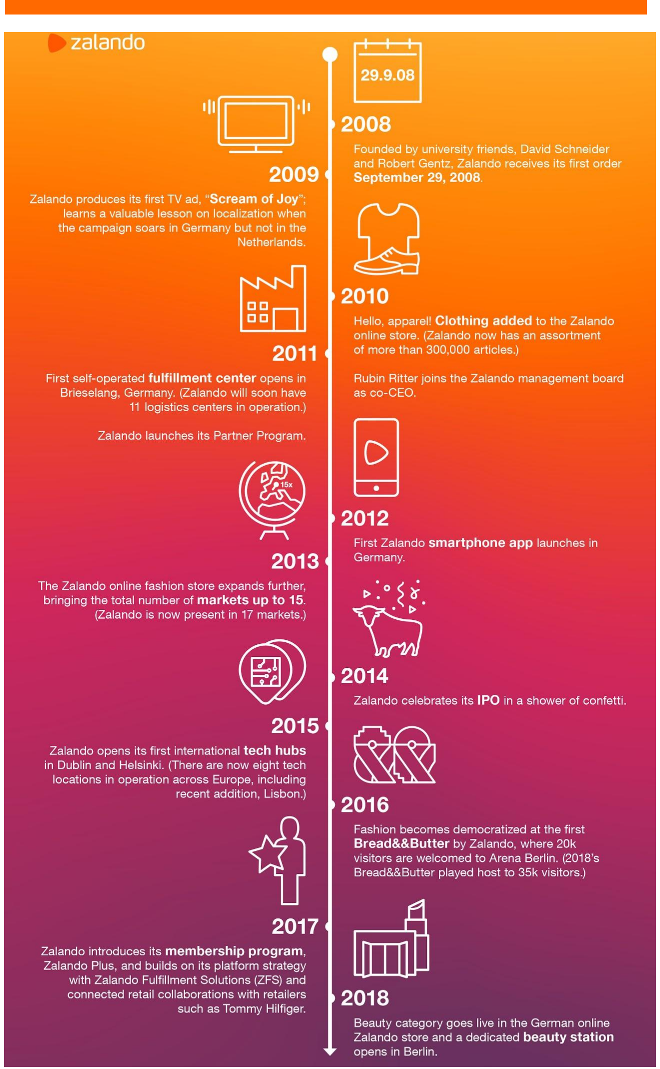
The Resource Monad

Coding

References

WE BRING FASHION TO PEOPLE IN 17 COUNTRIES





2008

Founded by university friends, David Schneider and Robert Gentz, Zalando receives its first order **September 29, 2008**.

2009

Zalando produces its first TV ad, "**Scream of Joy**"; learns a valuable lesson on localization when the campaign soars in Germany but not in the Netherlands.

2010

Hello, apparel! **Clothing added** to the Zalando online store. (Zalando now has an assortment of more than 300,000 articles.)

Rubin Ritter joins the Zalando management board as co-CEO.

2011

First self-operated **fulfillment center** opens in Brieselang, Germany. (Zalando will soon have 11 logistics centers in operation.)

Zalando launches its Partner Program.

2012

First Zalando **smartphone app** launches in Germany.

2013

The Zalando online fashion store expands further, bringing the total number of **markets up to 15**. (Zalando is now present in 17 markets.)

2014

Zalando celebrates its **IPO** in a shower of confetti.

2015

Zalando opens its first international **tech hubs** in Dublin and Helsinki. (There are now eight tech locations in operation across Europe, including recent addition, Lisbon.)

2016

Fashion becomes democratized at the first **Bread&&Butter** by Zalando, where 20k visitors are welcomed to Arena Berlin. (2018's Bread&&Butter played host to 35k visitors.)

2017

Zalando introduces its **membership program**, Zalando Plus, and builds on its platform strategy with Zalando Fulfillment Solutions (ZFS) and connected retail collaborations with retailers such as Tommy Hilfiger.

2018

Beauty category goes live in the German online Zalando store and a dedicated **beauty station** opens in Berlin.

Who am I

I am Javier, I work for Zalando as a software engineer.

I have been developing software for 40 years now, doing Scala the last 7 years.

@jarrieta

https://www.linkedin.com/in/jarrieta/

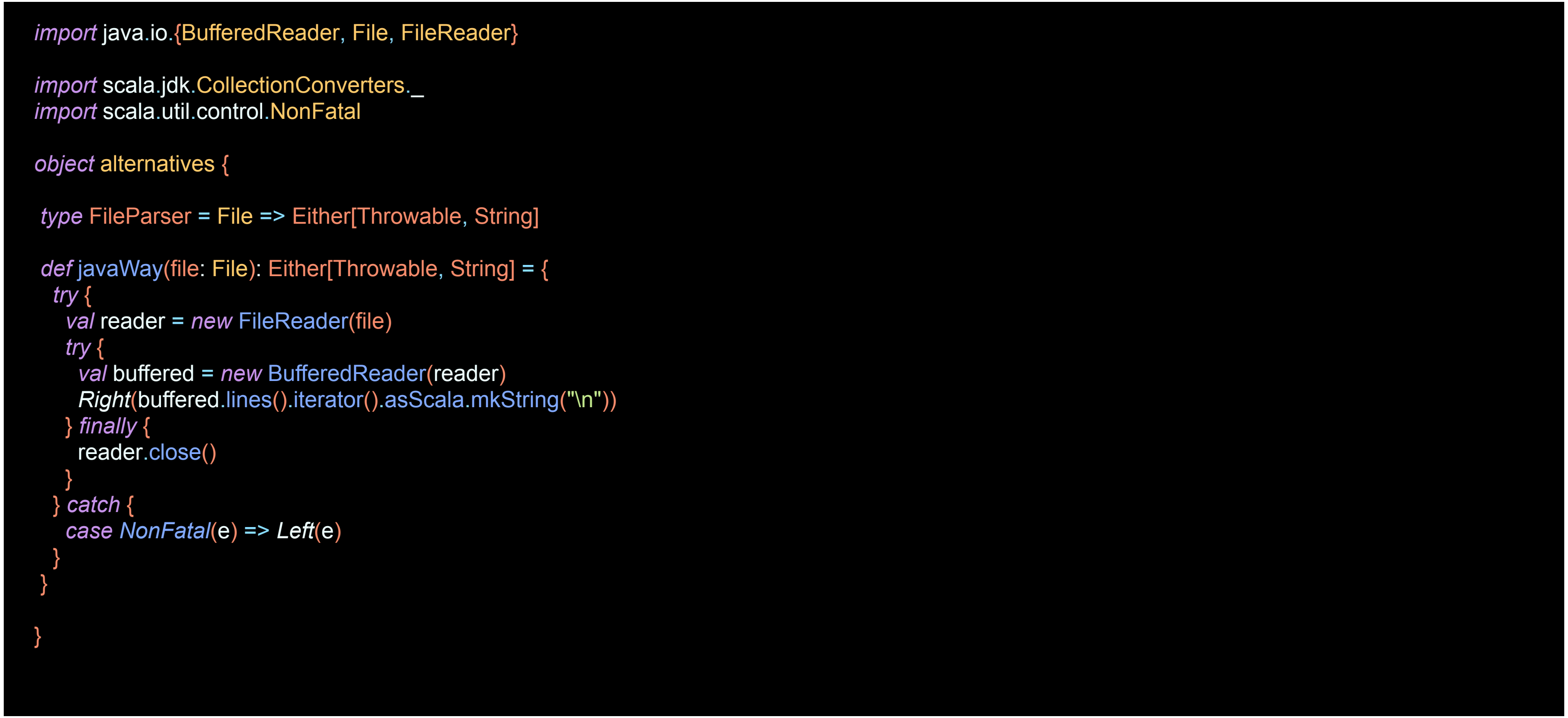
We are hiring: https://grnh.se/4f5514971

The Problem

How do we ensure we free up resources after usage?

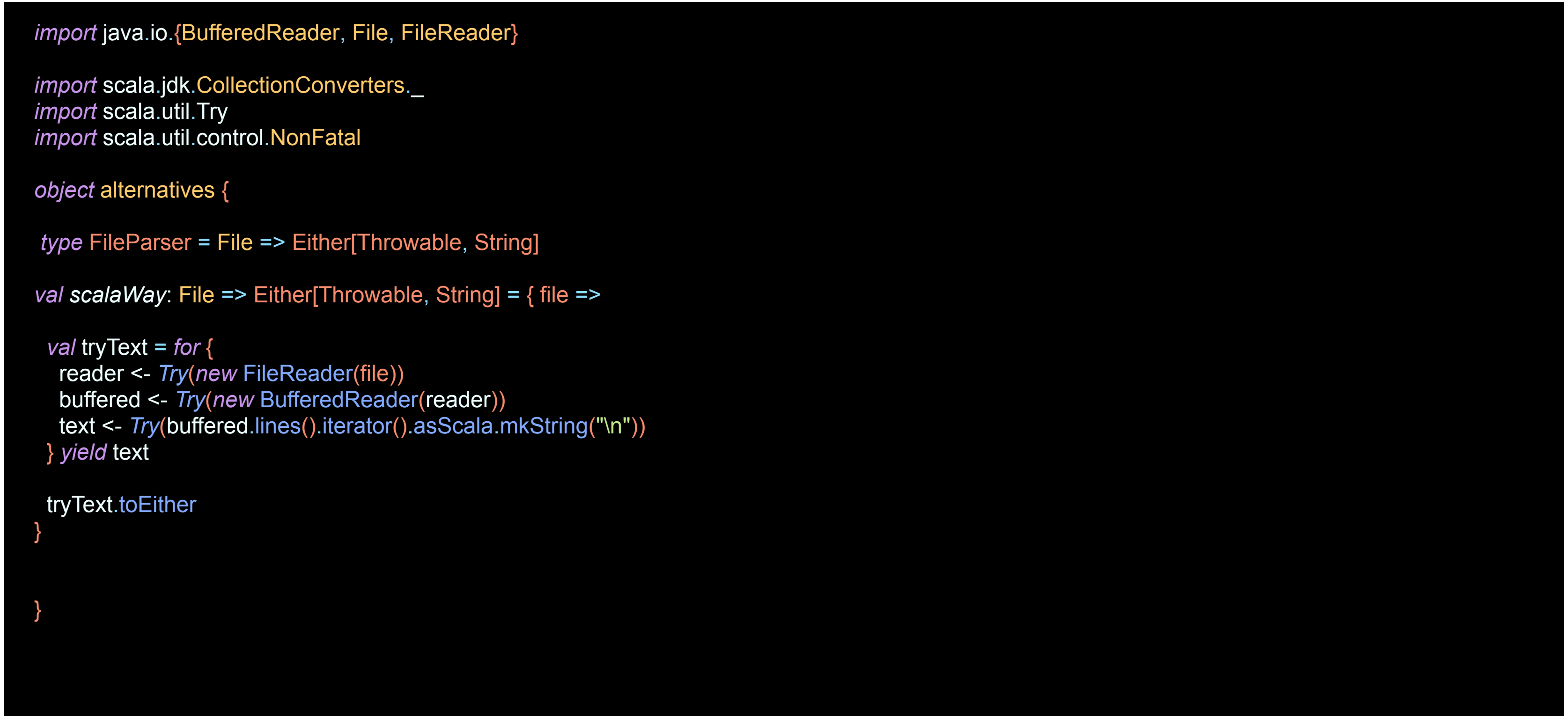
It usually involves a lot of boilerplate code

The Problem - Java way solution



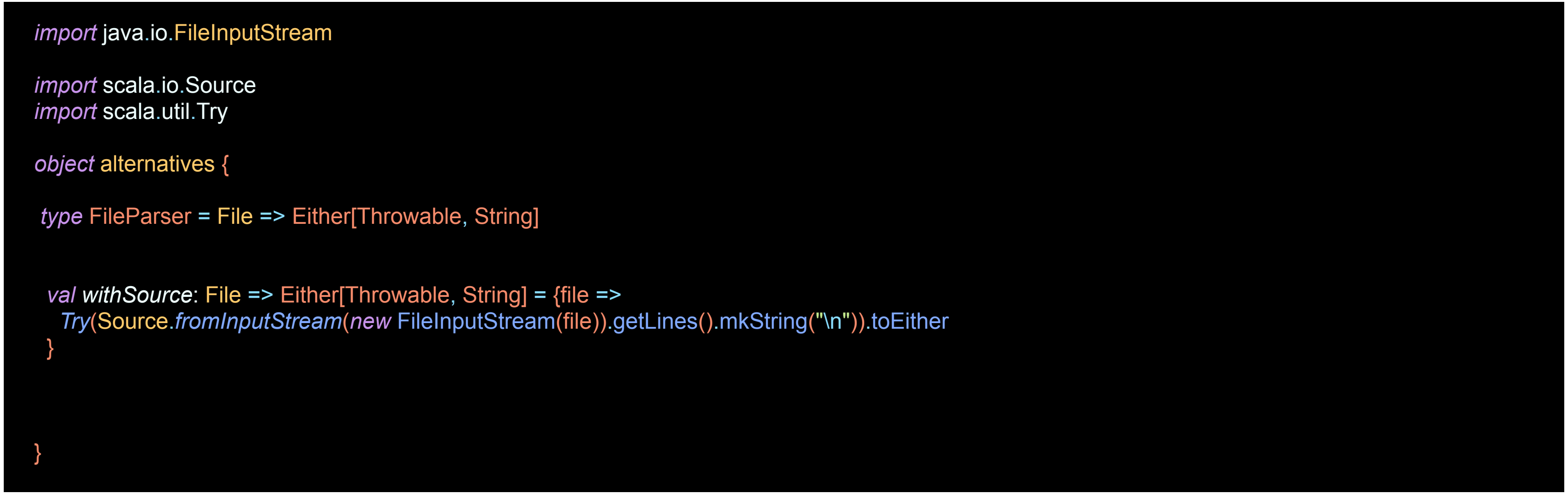
```
import java.io.{BufferedReader, File, FileReader}

import scala.jdk.CollectionConverters._
import scala.util.control.NonFatal

object alternatives {

 type FileParser = File => Either[Throwable, String]

 def javaWay(file: File): Either[Throwable, String] = {
  try {
    val reader = new FileReader(file)
    try {
      val buffered = new BufferedReader(reader)
      Right(buffered.lines().iterator().asScala.mkString("\n"))
    } finally {
      reader.close()
    }
  } catch {
    case NonFatal(e) => Left(e)
  }
 }

}
```

The Problem - More Scala way solution



```scala
import java.io.{BufferedReader, File, FileReader}

import scala.jdk.CollectionConverters._
import scala.util.Try
import scala.util.control.NonFatal

object alternatives {

 type FileParser = File => Either[Throwable, String]

val scalaWay: File => Either[Throwable, String] = { file =>

  val tryText = for {
    reader <- Try(new FileReader(file))
    buffered <- Try(new BufferedReader(reader))
    text <- Try(buffered.lines().iterator().asScala.mkString("\n"))
  } yield text

  tryText.toEither
}


}
```

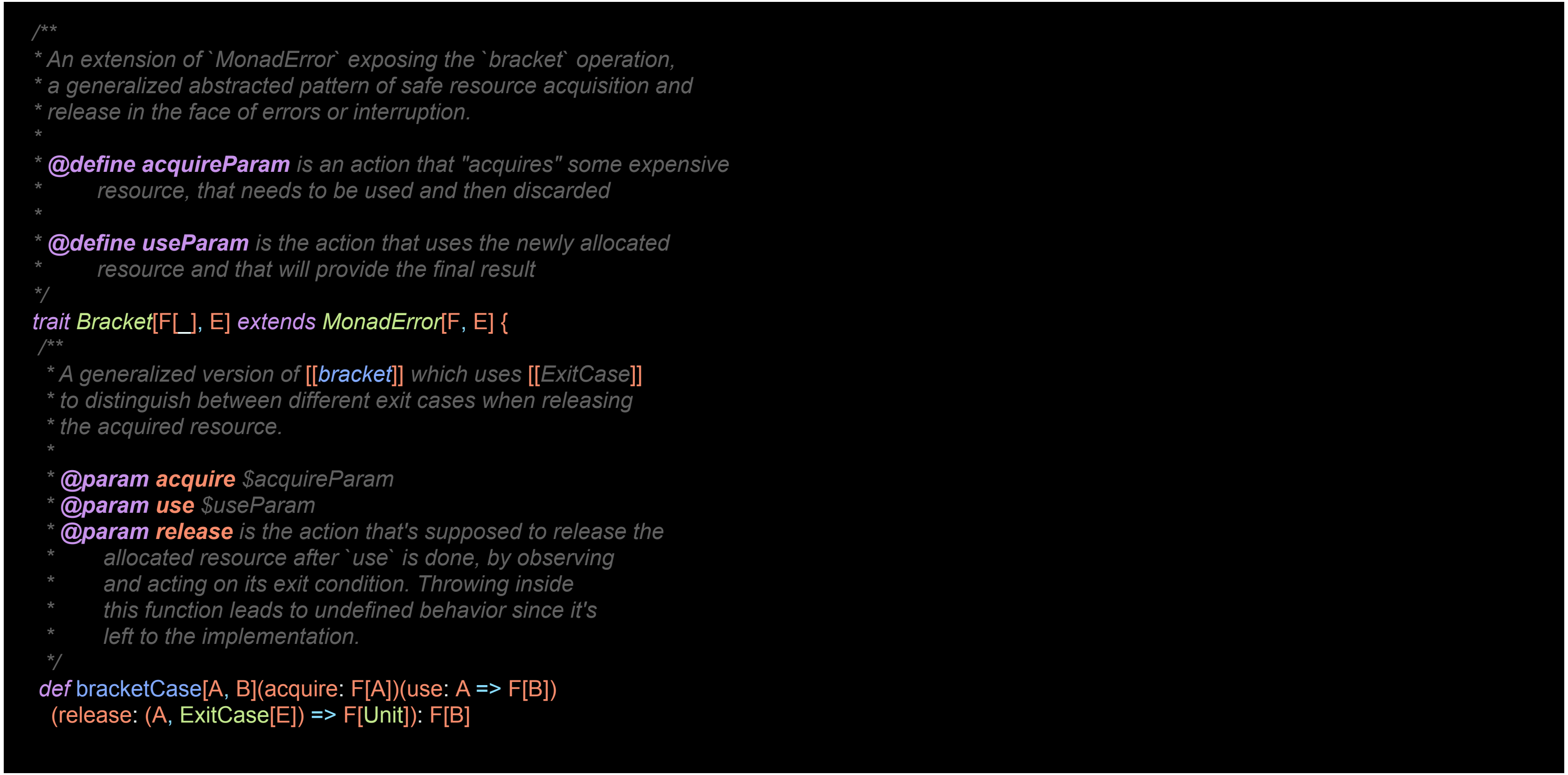
The Problem - Using `Source`



```
import java.io.FileInputStream

import scala.io.Source
import scala.util.Try

object alternatives {

 type FileParser = File => Either[Throwable, String]


  val withSource: File => Either[Throwable, String] = {file =>
    Try(Source.fromInputStream(new FileInputStream(file)).getLines().mkString("\n")).toEither
  }



}
```

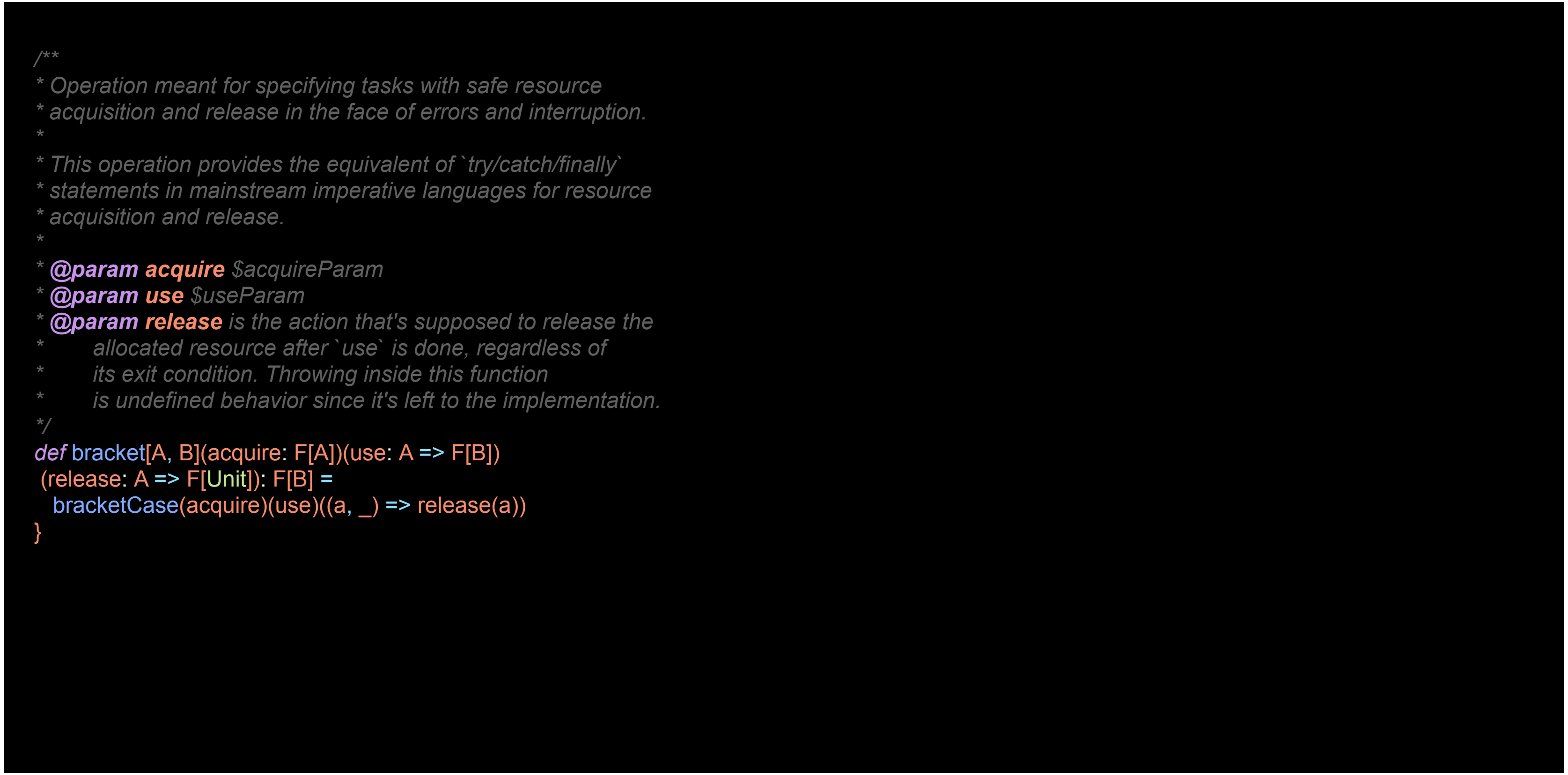
Only works for Input Streams ¯_(ツ)_/¯

The Functional Solution - Bracket Typeclass



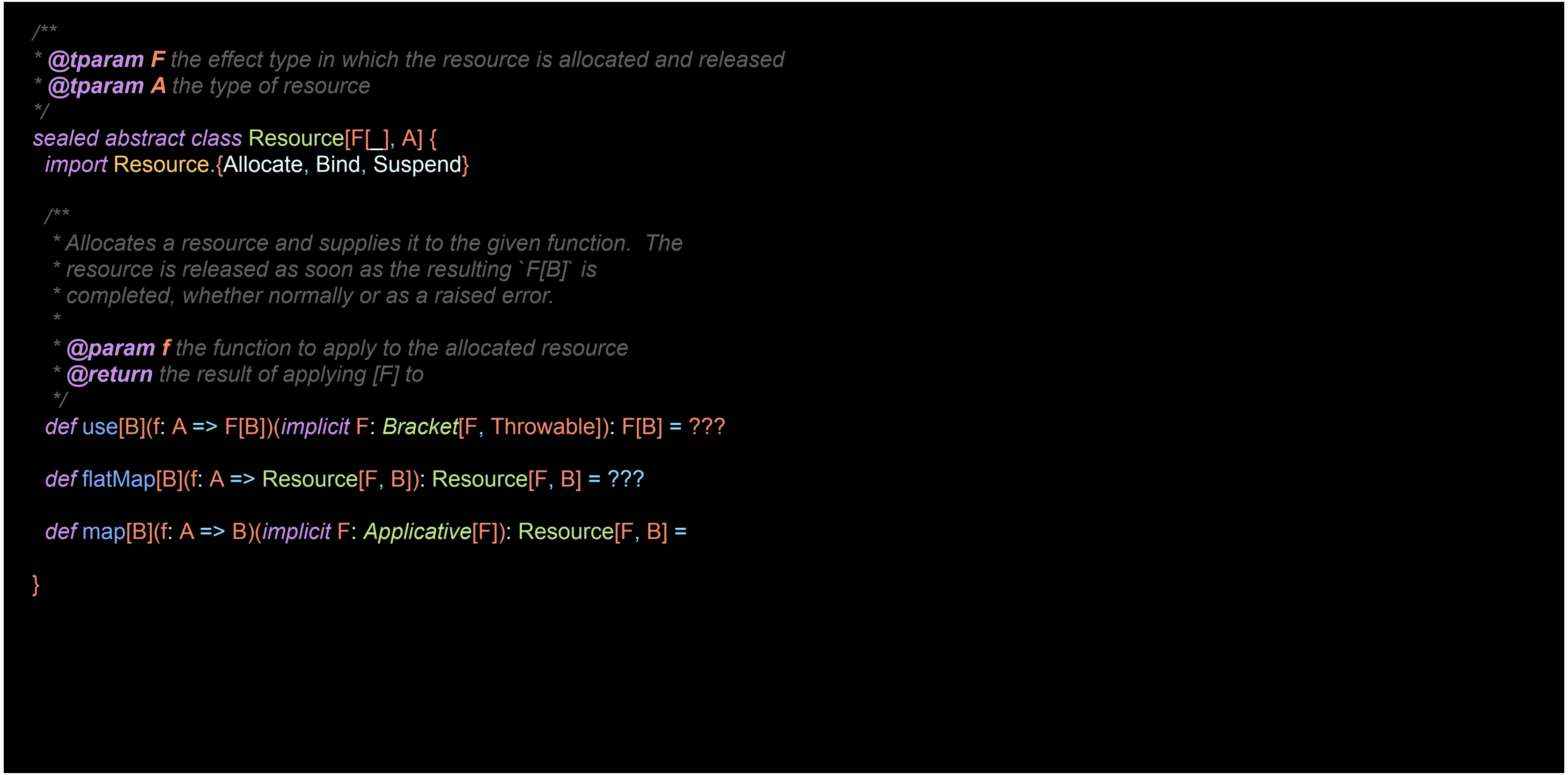
```
/**
* An extension of `MonadError` exposing the `bracket` operation,
* a generalized abstracted pattern of safe resource acquisition and
* release in the face of errors or interruption.
*
* @define acquireParam is an action that "acquires" some expensive
*        resource, that needs to be used and then discarded
*
* @define useParam is the action that uses the newly allocated
*        resource and that will provide the final result
*/
trait Bracket[F[_], E] extends MonadError[F, E] {
 /**
  * A generalized version of [[bracket]] which uses [[ExitCase]]
  * to distinguish between different exit cases when releasing
  * the acquired resource.
  *
  * @param acquire $acquireParam
  * @param use $useParam
  * @param release is the action that's supposed to release the
  *        allocated resource after `use` is done, by observing
  *        and acting on its exit condition. Throwing inside
  *        this function leads to undefined behavior since it's
  *        left to the implementation.
  */
 def bracketCase[A, B](acquire: F[A])(use: A => F[B])
  (release: (A, ExitCase[E]) => F[Unit]): F[B]
```

The Functional Solution - Bracket Typeclass - continued



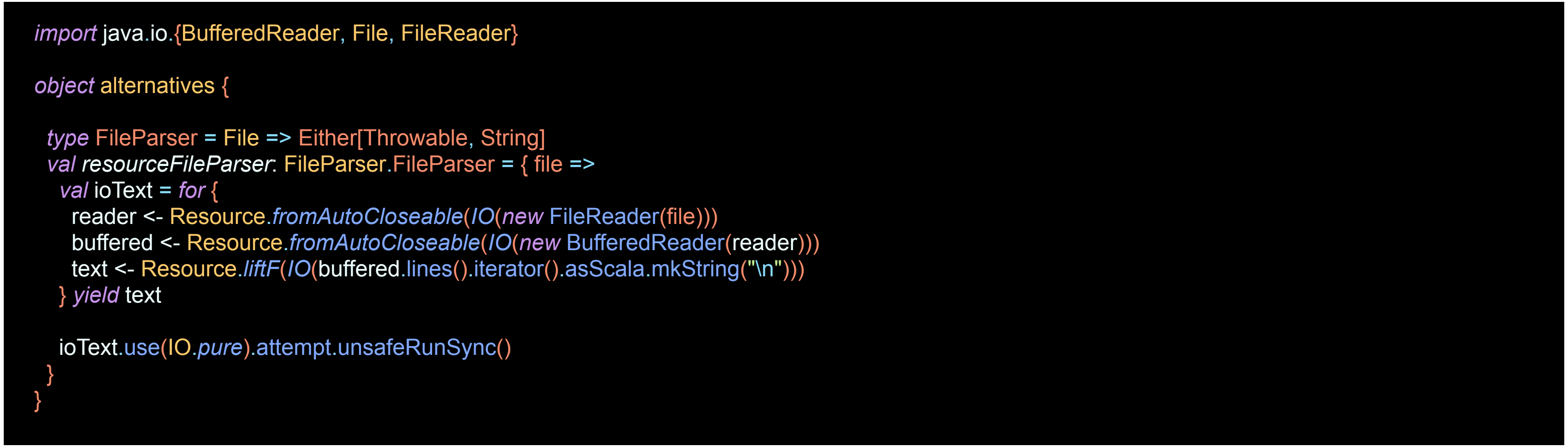
```
/**
* Operation meant for specifying tasks with safe resource
* acquisition and release in the face of errors and interruption.
*
* This operation provides the equivalent of `try/catch/finally`
* statements in mainstream imperative languages for resource
* acquisition and release.
*
* @param acquire $acquireParam
* @param use $useParam
* @param release is the action that's supposed to release the
*        allocated resource after `use` is done, regardless of
*        its exit condition. Throwing inside this function
*        is undefined behavior since it's left to the implementation.
*/
def bracket[A, B](acquire: F[A])(use: A => F[B])
 (release: A => F[Unit]): F[B] =
  bracketCase(acquire)(use)((a, _) => release(a))
}
```

The Functional Solution - Usage - The Resource class



```scala
/**
* @tparam F the effect type in which the resource is allocated and released
* @tparam A the type of resource
*/
sealed abstract class Resource[F[_], A] {
  import Resource.{Allocate, Bind, Suspend}

  /**
    * Allocates a resource and supplies it to the given function.  The
    * resource is released as soon as the resulting `F[B]` is
    * completed, whether normally or as a raised error.
    *
    * @param f the function to apply to the allocated resource
    * @return the result of applying [F] to
    */
  def use[B](f: A => F[B])(implicit F: Bracket[F, Throwable]): F[B] = ???

  def flatMap[B](f: A => Resource[F, B]): Resource[F, B] = ???

  def map[B](f: A => B)(implicit F: Applicative[F]): Resource[F, B] =

}
```

The Functional Solution - The example solution



```
import java.io.{BufferedReader, File, FileReader}

object alternatives {

  type FileParser = File => Either[Throwable, String]
  val resourceFileParser: FileParser.FileParser = { file =>
    val ioText = for {
      reader <- Resource.fromAutoCloseable(IO(new FileReader(file)))
      buffered <- Resource.fromAutoCloseable(IO(new BufferedReader(reader)))
      text <- Resource.liftF(IO(buffered.lines().iterator().asScala.mkString("\n")))
    } yield text

    ioText.use(IO.pure).attempt.unsafeRunSync()
  }
}
```

Live coding demo

References

Self	https://docs.google.com/presentation/d/1DEy_2j9dMoEbwiStHDKd4XUd4GxErlHP0LIDdDSoX64
Example	https://github.com/javierarrieta/bracket-scala-meetup
Cats-effect	https://typelevel.org/cats-effect
Http4s	https://http4s.org/v0.20/
ZIO	https://zio.dev/