Previous episodes recap

```
object wordCountFolder extends zio.App {
  def readFileAsString(path: String): Task[String] = {
    var string = ""
    var strLine: String = null

    Task(new BufferedReader(new InputStreamReader(new FileInputStream(path))))
      .bracket(inputStream => UIO(println("closing")) *> UIO(inputStream.close)) {
        br =>
          {
            while ({
              ({ strLine = br.readLine; strLine }) != null
            }) { // Print the content on the console
              string += strLine
            }
            Task(string)
          }
      }
  }
}
```

```scala
def run(args: List[String]) =
   myAppLogic.exitCode

 val myAppLogic =
   for {
     _ <- putStrLn(
       "Hello! What folder path do you want to word count? please enter full folder path")
     fullPath <- getStrLn
     files <- getFolderFiles(fullPath)
     count <- ZIO.collectAllParN(3)(files.map(countPerFile(_)))
     _ <- putStrLn(s"found ${count.sum} in all files")
   } yield count.sum

 def getFolderFiles(path: String): Task[List[String]] =
   for {
     file <- ZIO.effect(new File(path))
   } yield file.listFiles().filter(_.isFile).map(_.getAbsolutePath).toList

def countPerFile(path: String) =
  for {
    contents <- readFileAsString(path)
    count <- countWords(contents)
    _ <- putStrLn(s"found ${count} words")
  } yield count

def countWords(str: String): UIO[Int] = UIO(str.split(" ").length)
```

Recap

- ZIO modeling using UIO, Task
- Bracket
- Zip Right
- collectAll, CollectAllParN combinators
- ZIO.effect
-

Environments!

Testability via zio environments. (Recpie!)

```
import zio.{Has, ZLayer}

type FileRepo = Has[FileRepo.Service]

object FileRepo {
  trait Service {
    def readFileAsString(path: String): Task[String]
  }
}
```

Count words again

```
def countWords(str: String): ZIO[FileRepo, Throwable, Int] = ???
```

Count words

```
def countWords(str: String): ZIO[FileRepo, Throwable, Int] =
  for {
    content <- ZIO.accessM[FileRepo](_.get.readFileAsString(str))
    count <- UIO(content.split(" ").size)
  } yield count
```

Let's compile and see what happens

Main app

```
val live: Layer[Nothing, FileRepo] = ZLayer.succeed(new FileRepo.Service {} )
```

```scala
def run(args: List[String]) =
    myAppLogic.provideSomeLayer(FileRepo.live ++ zio.console.Console.live).exitCode
```

Demo

What was the trouble for?

ZIO test

```scala
import zio.test.Assertion._
import zio.test._

object WordCountFolderSpec extends DefaultRunnableSpec {
  override def spec =
    suite("WordCountSpec")(
      test("count words properly") {
        assert(1)(equalTo(2))
      }
    )
}
```

Lets test count Words

```scala
import wordCount.wordCountFolderAsZioEnv.FileRepo
import zio.{Layer, Task, UIO, ZLayer}
import zio.test.Assertion._
import zio.test._

object WordCountFolderSpec extends DefaultRunnableSpec {

  val test: Layer[Nothing, FileRepo] = ZLayer.succeed(new FileRepo.Service {
    override def readText(path: String): Task[String] = {
      Task(path match {
        case "/tmp/dor" => "hello world"
        case _          => ""
      })
    }
  })
  override def spec =
    suite("WordCountSpec")(
      testM("count words properly") {
        for {
          count <- wordCount.wordCountFolderAsZioEnv.countWords("/tmp/dor")
        } yield assert(count)(equalTo(2))
      }
    ).provideSomeLayer(test)
}
```

Recap

- Provide Some Layer
- The Service Recpie
- ZIO test for effects
- Mocking