Handling Failures

Lesson Objectives

- After completing this lesson, you should be able to:
  - Describe the usage and importance of a Try
  - Describe how to pattern match on a Try
  - Outline how to use higher order functions on a Try
  - Illustrate how to use for comprehensions to work with Try

JVM Exceptions

- They represent runtime failures for various reasons
  - NullPointerException (Runtime)
  - ClassCastException (Runtime)
  - IOException (Checked)
  - When one occurs, control is “thrown” back within a thread stack to whomever “catches” it

Typesafe

Catching an Exception

```
def toInt(s: String): Int =
  try {
    s.toInt
  } catch {
    case _: NumberFormatException => 0
  }
```

Typesafe

Idiomatic Scala and Exceptions

- In Scala, we do not believe in this approach, as it represents a possible “side effect”
  - We want everything in our code to be pure
  - When we interact with libraries or services that may fail, we “wrap” the call in a Try to capture the failure

Wrapping a Call in Try

```
scala> import scala.util.{Try, Success, Failure}
import scala.util.{Try, Success, Failure}

scala> Try("100".toInt)
res0: scala.util.Try[Int] = Success(100)

scala> Try("Martin".toInt)
res1: scala.util.Try[Int] =
  Failure(java.lang.NumberFormatException:
    For input string: "Martin")
```

Typesafe

Pattern Matching on Try

```
scala> import scala.util.{Try, Success, Failure}
import scala.util.{Try, Success, Failure}

scala> def makeInt(s: String): Int = Try(s.toInt) match {
     |   case Success(n) => n
     |   case Failure(_) => 0
     | }
makeInt: (s: String)Int

scala> makeInt("35")
res2: Int = 35

scala> makeInt("James")
res3: Int = 0
```

Higher Order Functions and Try

```
scala> import scala.util._
import scala.util._

scala> def getScala: Try[String] = Success("Scala")
getScala: scala.util.Try[String]

scala> val scala = getScala
scala: scala.util.Try[String] = Success(Scala)

scala> scala.map(s => s.reverse)
res0: scala.util.Try[String] = Success(alacS)
```

Higher Order Functions and Try

```
scala> import scala.util._
import scala.util._

scala> def getOuch: Try[String] =
  Failure(new Exception("Ouch"))
getOuch: scala.util.Try[String]

scala> val ouch = getOuch
ouch: scala.util.Try[String] =
  Failure(java.lang.Exception: Ouch)

scala> ouch.map(s => s.reverse)
res0: scala.util.Try[String] =
  Failure(java.lang.Exception: Ouch)
```

For Expressions and Try

```
scala> Success("Scala").map(_.reverse)
res0: scala.util.Try[String] = Success(alacS)

scala> for {
     |   language <- Success("Scala")
     |   behavior <- Success("rocks")
     | } yield s"$language $behavior"
res1: scala.util.Try[String] = Success(Scala rocks)
```

Typesafe

Lesson Summary

- Having completing this lesson, you should be able to:
  - Describe the usage and importance of a Try
  - Describe how to pattern match on a Try
  - Outline how to use higher order functions on a Try
  - Illustrate how to use for comprehensions to work with Try