Manipulating `Try` Values

Effective Programming in Scala

Transformation Operations on `Try`

Often, you want to work with the successful case of a `Try` and to postpone error handling (via `recover` or pattern matching) to a later point in the program.

For instance, consider a situation where you want to read two dates from `String` values, and compute the duration between them. Parsing a date is an operation that may fail, but in your case you don't want to deal with the failures yet, you want to focus on the case where both dates were successfully parsed.

In this lesson, we will show you how you can deal with such situations by using the most common transformation operations on `Try`, namely `flatMap` and `map`.

Example: Parsing Two Dates

Consider the following problem: you want to parse two dates and compute the duration between them.

Here is the signature of an existing operation `parseDate`, which parses one date:

```
import java.time.LocalDate
def parseDate(str: String): Try[LocalDate]
```

Example: Parsing Two Dates

Consider the following problem: you want to parse two dates and compute the duration between them.

Here is the signature of an existing operation `parseDate`, which parses one date:

```
import java.time.LocalDate
def parseDate(str: String): Try[LocalDate]

import java.time.Period
def tryPeriod(str1: String, str2: String): Try[Period]
```

Example: Parsing Two Dates (Solution)

```
def tryPeriod(str1: String, str2: String): Try[Period] =
  parseDate(str1).flatMap { (date1: LocalDate) =>
    parseDate(str2).map { (date2: LocalDate) =>
      Period.between(date1, date2)
    }
  }
```

Alternatively:

```
def tryPeriod(str1: String, str2: String): Try[Period] =
  for
    date1 <- parseDate(str1)
    date2 <- parseDate(str2)
  yield
    Period.between(date1, date2)
```

Example: Parsing Two Dates (Usage)

```
tryPeriod("2020-07-27", "2020-12-25")
// : Try[Period] = Success(P4M28D)
tryPeriod("2020-19-27", "2020-12-25")
// : Try[Period] = Failure(Invalid value for MonthOfYear: 19)
tryPeriod("2020-07-27", "2020-22-25")
// : Try[Period] = Failure(Invalid value for MonthOfYear: 22)
tryPeriod("2020-19-27", "2020-22-25")
// : Try[Period] = Failure(Invalid value for MonthOfYear: 19)
```

Note: the program stops after the first failure.

Transformation Operations on Try

```
trait Try[A]:

  def map[B](f: A => B): Try[B]

  def flatMap[B](f: A => Try[B]): Try[B]

  def recover(f: PartialFunction[Throwable, A]): Try[A]

  def recoverWith(f: PartialFunction[Throwable, Try[A]]): Try[A]

end Try
```

Parsing a List of Dates

Harder problem: instead of parsing just two dates, you want to parse an arbitary number of dates, read from a file. Each date is written in a new line:

```
1970-01-01
2004-01-20
2016-07-15
```

Parsing a List of Dates

Harder problem: instead of parsing just two dates, you want to parse an arbitary number of dates, read from a file. Each date is written in a new line:

```
1970-01-01
2004-01-20
2016-07-15
```

```
def parseDates(fileName: String): Try[Seq[LocalDate]]
```

Parsing a List of Dates (Solution, Part 1)

```
import scala.io.Source

def readDateStrings(fileName: String): Try[Seq[String]] = Try {
  val source      = Source.fromFile(fileName)
  val dateStrings = source.getLines().toSeq
  source.close()
  dateStrings
}
```

Parsing a List of Dates (Solution, Part 2)

```
def parseDates(fileName: String): Try[Seq[LocalDate]] =
  readDateStrings(fileName).flatMap { dateStrings =>
    dateStrings.foldLeft[Try[Seq[LocalDate]]](Success(Vector.empty)) {
      (tryDates, dateString) =>
        for
          dates <- tryDates
          date  <- parseDate(dateString)
        yield
          dates :+ date
    }
  }
```

Resource Management

What happens if an exception is thrown after the file has been opened, but before closing it?

The operation `readDateStrings` catches the exception and returns it as a failed `Try` to its caller.

Problem: opening a file is an operation that allocates resources, but we never release these resources!

Resource Safe `readDateStrings`

```
import scala.util.Using

def readDateStrings(fileName: String): Try[Seq[String]] =
  Using(Source.fromFile(fileName)) { source =>
    source.getLines().toSeq
  }
```

The first parameter of `Using` is a computation that acquires a resource, its second parameter uses the acquired resource, and its result is a `Try`.

`Using` ensures that the resources we use are eventually released, whether our program fails or succeeds.

Summary

Leverage the operations `map` and `flatMap` to manipulate and combine `Try` values without dealing with the failure cases.

Leverage `Using` to mix in resource acquisition and release with operations that can fail.