Meeting 19 - Encapsulating Effects

Bor-Yuh Evan Chang

Saturday, October 26, 2024

Meeting 19 - Encapsulating Effects

What questions does your neighbor have?

Links

In-Class Slides

In-Class Jupyter

Book Chapter

Announcements

- Lab 4 due ~~Monday~~ Tuesday 6pm
  - Implement static typing (from last Tuesday's chapter and lecture)
  - Apply higher-order methods for `List` and `Map` from HW4
  - For immutable objects and multi-parameter functions, reading semantics
- Review 3-4 Thursday and Friday
  - Sign up on Canvas – needed to get assignment points
  - Scored on the quality of your feedback for your partner, not how you were assessed by your partner
  - Submit on Gradescope as a group — to get each other's feedback
  - 0 for no-shows
- Exam 3-4 next Tuesday 11/5
  - 50 minutes in class

Today

- Triage Your Questions
  - Static Type Checking?
  - Higher-Order Functions?
- Encapsulating Effects

Questions?

- Review:
  - What is static type checking and what is the benefit?

Questions?

Error Effects

Exceptions

```
1  def toDoubleException(s: String): Double = s.toDouble
2
3  toDoubleException("1")
4  toDoubleException("4.2")
```

```
defined function toDoubleException
res0_1: Double = 1.0
res0_2: Double = 4.2
```

Option

```
1  def toDoubleOption(s: String): Option[Double] = ???
```

```
defined function toDoubleOption
```

```
1  def toDoubleNoNaNOption(s: String): Option[Double] = ???
```

```
defined function toDoubleNoNaNOption
```

Option as a Collection

```
1  def toDoubleNoNaNOption(s: String): Option[Double] = ???
```

```
defined function toDoubleNoNaNOption
```

```
1  def addToDoubleOption(s1: String, s2: String): Option[Double] = ???
```

```
defined function addToDoubleOption
```

Map, Filter, FlatMap

Exercise 1 Implement `map` for `Option[A]`s:

```
1  def map[A,B](opt: Option[A])(f: A => B): Option[B] = ???
```

```
defined function map
```

Exercise 2 Implement `filter` for `Option[A]`s:

```
1  def filter[A](opt: Option[A])(f: A => Boolean): Option[A] = ???
```

```
defined function filter
```

Exercise 3 Implement `flatMap` for `Option[A]`s:

```
1  def flatMap[A,B](opt: Option[A])(f: A => Option[B]): Option[B] = ???
```

```
defined function flatMap
```

Comprehensions

```
1  def addToDoubleOption(s1: String, s2: String): Option[Double] = ???
```

```
defined function addToDoubleOption
```

Either and Try

```
def toDoubleEither(s: String): Either[NumberFormatException, Double] = ???

def addToDoubleEither(s1: String, s2: String): Either[NumberFormatException
  for {
    d1 <- toDoubleEither(s1)
    d2 <- toDoubleEither(s2)
  } yield d1 + d2
```

```
defined function toDoubleEither
defined function addToDoubleEither
```

```
import scala.util.Try
def toDoubleTry(s: String): Try[Double] = ???

def addToDoubleTry(s1: String, s2: String): Try[Double] =
  for {
    d1 <- toDoubleTry(s1)
    d2 <- toDoubleTry(s2)
  } yield d1 + d2
```

```
import scala.util.Try
defined function toDoubleTry
defined function addToDoubleTry
```

Mutation Effects

```
1  def freshVarImperative: String = ???
```

```
defined function freshVarImperative
```

```
1  def freshVar: Int => (Int, String) = ???
```

```
defined function freshVar
```

Encapsulating Mutation Effects

```
1  type DoWith[S, A] = S => (S, A)
```

```
defined type DoWith
```

```
1  freshVar: DoWith[Int, String]
```

```
res25: Int => (Int, String) = ammonite.$sess.cmd22$Helper$$Lambda$2244/0x000000
```

```
1  def map[S,A,B](doer: DoWith[S,A])(f: A => B): DoWith[S,B] = ???
```

```
defined function map
```

```
1  def flatMap[S,A,B](doer: DoWith[S,A])(f: A => DoWith[S,B]): DoWith[S,B] = ?
```

```
defined function flatMap
```

```
1  val counter = 0
```

```
counter: Int = 0
```

DoWith

```scala
sealed class DoWith[S,A] private (doer: S => (S, A)) {
  def map[B](f: A => B): DoWith[S,B] = new DoWith[S,B]({
    (s: S) => {
      val (s_, a) = doer(s)
      (s_, f(a))
    }
  })

  def flatMap[B](f: A => DoWith[S,B]): DoWith[S,B] = new DoWith[S,B]({
    (s: S) => {
      val (s_, a) = doer(s)
      f(a)(s_)
    }
  })

  def apply(s: S): (S, A) = doer(s)
}

```

```
defined class DoWith
defined object DoWith
import DoWith._
```

```scala
def freshVar: DoWith[Int,String] = ???
```

```
defined function freshVar
```

```
1  val counter = 0
```

```
counter: Int = 0
```