Simplify Scala with Scalaz

And I'm not Joking

Devoxx Fr

Anwar Rizal @anrizal

http://voidmainargs.blogspot.com

Amadeus, Nice

Scalaz in Scala Ecosystem

- Scalaz in Scala Ecosystem
- Having Fun with Scalaz
- Monoid, Functor, Applicative
- Validation

Scalaz-Camel

- https://github.com/krasserm/scalaz-camel

SJson-App

- https://github.com/krasserm/scalaz-camel

Scalaz-Akka

- https://github.com/derekjw/akka-scalaz

Specs 2

- http://etorreborre.github.com/specs2/

Having Fun with Scalaz

Getting Started

```
scala> import scalaz._
import scalaz._

scala> import Scalaz._
import Scalaz._
```

Having Fun with |+|

```
scala> val xOpt:Option[Int] = Some(4)
xOpt: Option[Int] = Some(4)

scala> val yOpt:Option[Int] = Some(3)
yOpt: Option[Int] = Some(3)

scala> val zOpt:Option[Int] = None
zOpt: Option[Int] = None
```

Having Fun with |+|

```
scala> Some(xOpt.get +  yOpt.get)
res2: Some[Int] = Some(7)

scala> Some(xOpt.get +  zOpt.get)
java.util.NoSuchElementException: None.get
        at scala.None$.get(Option.scala:275)
        at scala.None$.get(Option.scala:273)
```

Having Fun with |+|

```
def sumOpt(x:Option[Int], y:Option[Int]) =
  (x,y) match  {
    case ( Some(a), Some(b) ) => Some(a +  b)
    case ( Some(_), None)     => x
    case ( None, Some(_) )    => y
    case ( None,  None )      => None
  }
```

```
sumOpt(xOpt, zOpt)          // Some(4)
```

Having Fun |+|

```
 scala> xOpt |+| yOpt
res11: Option[Int] = Some(7)
```

Having Fun |+|

```
 scala> xOpt |+| yOpt
res11: Option[Int] = Some(7)

scala> xOpt |+| zOpt
res12: Option[Int] = Some(4)
```

Having Fun |+|

```
 scala> xOpt |+| yOpt
res11: Option[Int] = Some(7)

scala> xOpt |+| zOpt
res12: Option[Int] = Some(4)

scala> xOpt |+| yOpt |+| zOpt
res13: Option[Int] = Some(7)
```

Having Fun |+|

```
 scala> val xs = List(some(2), some(6), some(7), none,
some(10))
xs: List[Option[Int]] = List(Some(2), Some(6), Some(7), None,
Some(10))

scala> xs.reduce(_ |+| _ )
res15: Option[Int] = Some(25)
```

Having Fun |+|

```
scala> val t1 = (4, 5, 20)
t1: (Int, Int, Int) = (4,5,20)

scala> val t2 = (10, 3, 4)
t2: (Int, Int, Int) = (10,3,4)
```

Having Fun |+|

```
scala> val t1 = (4, 5, 20)
t1: (Int, Int, Int) = (4,5,20)

scala> val t2 = (10, 3, 4)
t2: (Int, Int, Int) = (10,3,4)

scala> t1 |+| t2
res16: (Int, Int, Int) = (14,8,24)
```

Having Fun with |+|

```
scala> val t1 = (4, "Hello", some(5))


scala> val t2 = (10, "World", some(10))
```

Having Fun with |+|

```
scala> val t1 = (4, "Hello", some(5))


scala> val t2 = (10, "World", some(10))

scala> t1 |+| t2
res20: (Int, java.lang.String, Option[Int]) =
(14,HelloWorld,Some(15))
```

Having Fun with |@|

```
scala> val mul:(Int,Int)=>Int = _ * _
mul: (Int, Int) => Int = <function2>

scala> val sum:(Int,Int)=>Int = _ + _
sum: (Int, Int) => Int = <function2>
```

Having Fun with |@|

```
scala> val x = some(6)
x: Option[Int] = Some(6)

scala> val y = some(7)
y: Option[Int] = Some(7)

scala> val z = none
z: Option[Nothing] = None
```

Having Fun with |@|

```
scala> val mul:(Int,Int)=>Int = _ * _
mul: (Int, Int) => Int = <function2>

scala> val sum:(Int,Int)=>Int = _ + _
sum: (Int, Int) => Int = <function2>
```

Having Fun with |@|

```
scala> val mul:(Int,Int)=>Int = _ * _
mul: (Int, Int) => Int = <function2>

scala> val sum:(Int,Int)=>Int = _ + _
sum: (Int, Int) => Int = <function2>

scala> (x |@| y)(add)
res2: Option[Int] = Some(13)

scala> (x |@| z)(add)
res3: Option[Int] = None
```

Having Fun with |@|

```
scala> (x |@| y)(mul)
res4: Option[Int] = Some(42)

scala> (x |@| z)(mul)
res5: Option[Int] = None
```

Having Fun with |@|

```
scala> val xs = List(1, 4, 4)
xs: List[Int] = List(1, 4, 4)

scala> val ys = List(2, 5, 7)
ys: List[Int] = List(2, 5, 7)
```

Having Fun with |@|

```
scala> val xs = List(1, 4, 4)
xs: List[Int] = List(1, 4, 4)

scala> val ys = List(2, 5, 7)
ys: List[Int] = List(2, 5, 7)

scala> (xs |@| ys)(add)
res9: List[Int] = List(3, 6, 8, 6, 9, 11, 6, 9, 11)

scala> (xs |@| ys)(mul)
res10: List[Int] = List(2, 5, 7, 8, 20, 28, 8, 20, 28)
```

Having Fun with |@|

```
scala> val x:Either[String,Int] = Right(4)
x: Either[String,Int] = Right(4)

scala> val y:Either[String,Int] = Right(6)
y: Either[String,Int] = Right(6)

scala> val z:Either[String,Int] = Left("Error")
z: Either[String,Int] = Left(Error)
```

Having Fun with |@|

```
scala> val x:Either[String,Int] = Right(4)
x: Either[String,Int] = Right(4)

scala> val y:Either[String,Int] = Right(6)
y: Either[String,Int] = Right(6)

scala> val z:Either[String,Int] = Left("Error")
z: Either[String,Int] = Left(Error)

scala> (x |@| y)(add)
res11: Either[String,Int] = Right(10)

scala> (x |@| y)(mul)
res12: Either[String,Int] = Right(24)

scala> (x |@| z)(mul)
res13: Either[String,Int] = Left(Error)
```

Type Class Is the Key

Sort

```
scala> val xs = List(1, 4, 5, 2, 8, 6)
xs: List[Int] = List(1, 4, 5, 2, 8, 6)
```

Sort

```
scala> val xs = List(1, 4, 5, 2, 8, 6)
xs: List[Int] = List(1, 4, 5, 2, 8, 6)

scala> xs.sorted
res0: List[Int] = List(1, 2, 4, 5, 6, 8)
```

Sort

```
scala> val ts = List( (1, "A"), (5, "Today"), (3, "Try"), (4, "Dump"), (5, "Hello"))

ts: List[(Int, java.lang.String)] = List((1,A), (5,Today), (3,Try), (4,Dump), (5,Hello))
```

Sort

```
scala> val ts = List( (1, "A"), (5, "Today"), (3, "Try"), (4, "Dump"), (5, "Hello"))

ts: List[(Int, java.lang.String)] = List((1,A), (5,Today), (3,Try), (4,Dump), (5,Hello))

scala> ts.sorted

res1: List[(Int, java.lang.String)] = List((1,A), (3,Try), (4,Dump), (5,Hello), (5,Today))
```

Sort

- List[A]

```
sorted [B >: A] (implicit ord: Ordering[B])

sortBy[B](f: A=>B)(implicit ord: Ordering [B])
```

Sort

```
trait Ordering[T] {
    def compare(x: T, y: T): Int
}
```

Instance of Ordering

```
// Ordering.scala
trait IntOrdering extends Ordering[Int] {
   def compare(x: Int, y: Int) =
        if (x < y) -1
        else if (x == y) 0
        else 1
}
  implicit object Int extends IntOrdering
```

Type Class

- A trait (e.g. Ordering trait)
- Implicit parameter using the trait (e.g sorted [B >: A] (implicit ord: Ordering[B])
- Instance of the trait available through implicit

Scalaz Type Class: Show ?

Functor, Applicative, and Monoid Scare You?

What if we change their names to appendable, mappable, contextual mappable ?

Monoid

- A type class with two functions:
  - zero
  - append

Monoid

```
// simplified version
trait Monoid[A] {
    val zero: A
    def append(s1 : A, s2 : => A)
}
```

Monoid Contract

- zero append x = x
- x append zero = x
- (x append y) append z =

  x append (y append z)

Monoid Instances

- Int, Short, BigInt
- Boolean, BooleanConjunction
- List, Stream
- String
- Either.LeftProjection, Either.RightProjection
- ...

Monoid Instances

- Option[A] , when A is monoid
- Tuple[A, B, C, D] , when A, B, C, and D are monoid
- Map[A,B], when B is monoid
- …

My Monoid

```
case class FareAmount(
  baseFare: Double,
  taxes:Map[String, Double],
  surcharges:Option[Double],
  fees: Option[Double])
```

My Monoid

```
object FareAmountMonoids {
  implicit def FareAmountMonoid = new Monoid[FareAmount] {
    val zero = FareAmount(0, Map.empty, none, none)

    def append(s1 : FareAmount, s2 : => FareAmount) = {
      FareAmount(
        s1.baseFare +  s2.baseFare,
        s1.taxes |+| s2.taxes,
        s1.surcharges |+| s2.surcharges,
        s1.fees |+| s2.surcharges)
    }
  }
}
```

My Monoid

```
val amount1 =
      FareAmount(200.0,
                 Map("US" -> 14.0,
                     "YA" -> 20.3,
                     "UO" -> 40.0),
                 Some(39.5),
                  None)
val amount2 =
      FareAmount(220.0,
                 Map("US" -> 14.0,
                     "UO" -> 40.0),
                 Some(40.5),
                 Some(5))
```

My Monoid

```
val amount1 =
      FareAmount(200.0,
                 Map("US" -> 14.0,
                     "YA" -> 20.3,
                     "UO" -> 40.0),
                 Some(39.5),
                  None)
val amount2 =
      FareAmount(220.0,
                 Map("US" -> 14.0,
                     "UO" -> 40.0),
                 Some(40.5),
                 Some(5))


amount1 |+| amount2
// FareAmount(420.0,Map(US -> 28.0, YA -> 20.3, UO ->
80.0),Some(80.0),Some(40.5))
```

My Monoid

```
val fares:List[FareAmount] = List(amount1, amount2)

fares.sum
```

Functor

Applicative

Applicative-based Validation