

Effective Scala

J. Suereth

Who am I?

- Software Engineer
- Blogger
- Author
- Big Nerd



Who am I?

- Software Engineer
- Blogger
- Author
- Big Nerd
- Unicorn Expert (Klout)



What is Effective Scala?

Optimising your use of the **Scala** programming language to solve real world problems without **explosions**, **broken thumbs** or **bullet wounds**.



The Basics

Write **expressions**,
not statements

Statements

```
def errMsg(errorCode: Int): String = {  
    var result: String = _  
    errorCode match {  
        case 1 => result = "Network Failure"  
        case 2 => result = "I/O Failure"  
        case _ => result = "Unknown Error"  
    }  
    return result;  
}
```

State

```
def errMs = {  
  var re  
  errorC  
    case  
    case  
    case  
  }  
  return  
}
```



Expression!

```
def errMsg(errorCode: Int): String =  
  errorCode match {  
    case 1 => "Network Failure"  
    case 2 => "I/O Failure"  
    case _ => "Unknown Error"  
  }
```


Be expressive

```
def findPeopleIn(city: String,  
    people: Seq[People]): Set[People] =  
    val found = new mutable.HashSet[People]  
    for(person <- people) {  
        for(address <- person.addresses) {  
            if(address.city == city)  
                found.put(person)  
        }  
    }  
    return found  
}
```

Be ex

```
def findPeople(people):  
    people = sorted(people)  
    val found = false  
    for (person in people) {  
        for (address in person.addresses) {  
            if (address == "123 Main St") {  
                found = true  
            }  
        }  
    }  
    return found  
}
```



Be Expressive

```
def findPeopleIn(city: String,  
    people: Seq[People]): Set[People] =  
    for {  
        person <- people.toSet[People]  
        address <- person.addresses  
        if address.city == city  
    } yield person
```

The Basics

Use the **REPL**

The Basics

Stay **Immutable**

Immutability

- Safe to share across **threads**
 - **No locking**
- Safe to **hash** on attributes
- Easier **equality**
- Safe to **share internal state** with other objects
- **Co/Contra-variance**

Using Immutability

Doesn't mean lack of **mutation**.

```
def foo: Seq[A] = {  
    val a = new ArrayBuffer[Int]  
    fillArray(a)  
    a.toSeq  
}
```

The Basics

Use Option

Option

```
def authenticateSession(  
  session: HttpSession,  
  username: Option[String],  
  password: Option[Array[Char]]) =  
  for {  
    u <- username  
    p <- password  
    if canAuthenticate(u, p)  
    privileges <- privilegesFor.get(u)  
  } injectPrivs(session, privileges)
```

Options

```
def authenticate(session, username, password) =  
  session: HttpSession  
  username: Option[String]  
  password: Option[String]  
  for {  
    u <- username  
    p <- password  
    if canAuthenticate(u, p)  
    privileges <- privilegesFor.get(u)  
  } injectPrivs(session, privileges)
```

Style

You know it when you got it

Scala ain't Java
Scala ain't Ruby
Scala ain't Haskell

Object Orientation

Use **def** for abstract members

abstract defs

```
trait Foo {  
  def bar: String  
}
```

```
class NewFoo extends Foo {  
  override val bar = "ZOMG"  
}
```

00

Annotate non-trivial return
types for public methods.

Annotate Return Types

```
object Foo {  
  def name: Option[String] = ...  
}
```

00

Composition can use
Inheritance

Composition + Inheritance

```
trait Logger {  
    ...  
}  
trait HasLogger {  
    def logger: Logger  
}  
trait HasAwesomeLogger {  
    lazy val logger = new AwesomeLogger  
}
```

Implicit

Limit the **scope** of implicit

What are implicits?

```
implicit val pool: Executor =  
    Executors.newCachedThreadPool()
```

```
def determinant(m: Matrix) (implicit ctx:  
Executor): Double = ...
```

determinant(m)

vs.

determinant(m) (pool)

Implicit Scope

- First look in current scope
 - Implicits defined in current scope (1)
 - Explicit imports (2)
 - wildcard imports (3)
- Parts of the type of implicit value being looked up and their companion objects
 - Companion objects of the type
 - Companion objects of type arguments of types
 - Outer objects for nested types
 - Other dimensions

Implicit Scope (Parts)

```
trait Logger { ... }  
object Logger {  
  implicit object DefaultLogger  
    extends Logger { ... }  
  
  def log(msg: String) (implicit l: Logger) =  
    l.log(msg)  
}  
  
Logger.log("Foo")
```

Imp



IMPLICIT CAT

DISAPPROVES OF YOUR VIEWS

Implicits

Use for **type constraints** and
type traits

Implicit type constraints

```
import java.nio.ByteBuffer

class Buffer[T] {
  def toJByteBuffer(
    implicit ev: T <:: Byte): ByteBuffer
}
```


Type traits

```
trait Encodable[T] {  
  def encode(t: T): Array[Byte]  
  def decode(buf: ByteBuffer): T  
}
```

```
object Encodable {  
  def encode[T: Encodable](t: T) =  
    implicitly[Encodable[T]].encode(t)  
}
```

Type traits, default implementation

```
object Encodable {  
  implicit object IntEncodable  
    extends Encodable[Int] { ... }  
  
  implicit def tupleEncodable[A,B] (  
    implicit ea: Encodable[A],  
    eb: Encodable[B]  
  ) : Encodable[(A,B)] = ...  
  
}
```

Type Traits - external impls

```
trait TehAwesome { /* ucanthandlethis */ }
```

```
object TehAwesome {  
  implicit object encoder  
    extends Encodable[TehAwesome] {  
    ...  
  }  
}
```

Type traits - Benefits

- External to class hierarchy
 - **monkey patch** on existing classes **you don't control**
- Overridable at **call** site
- **Separate** Abstractions
 - One class can have **two** implementations
 - Similar type-traits **don't fight** for **method names**.
- Can separate method arguments into **roles**

```
def synchronize[  
  F: Source, T: Sink] (  
  from: F, to: T): (F, T) = ...
```

Type System

Preserve **specific** types

Preserve Specific Types

```
def foo(s: Seq[A]) : Seq[A] = ?
```

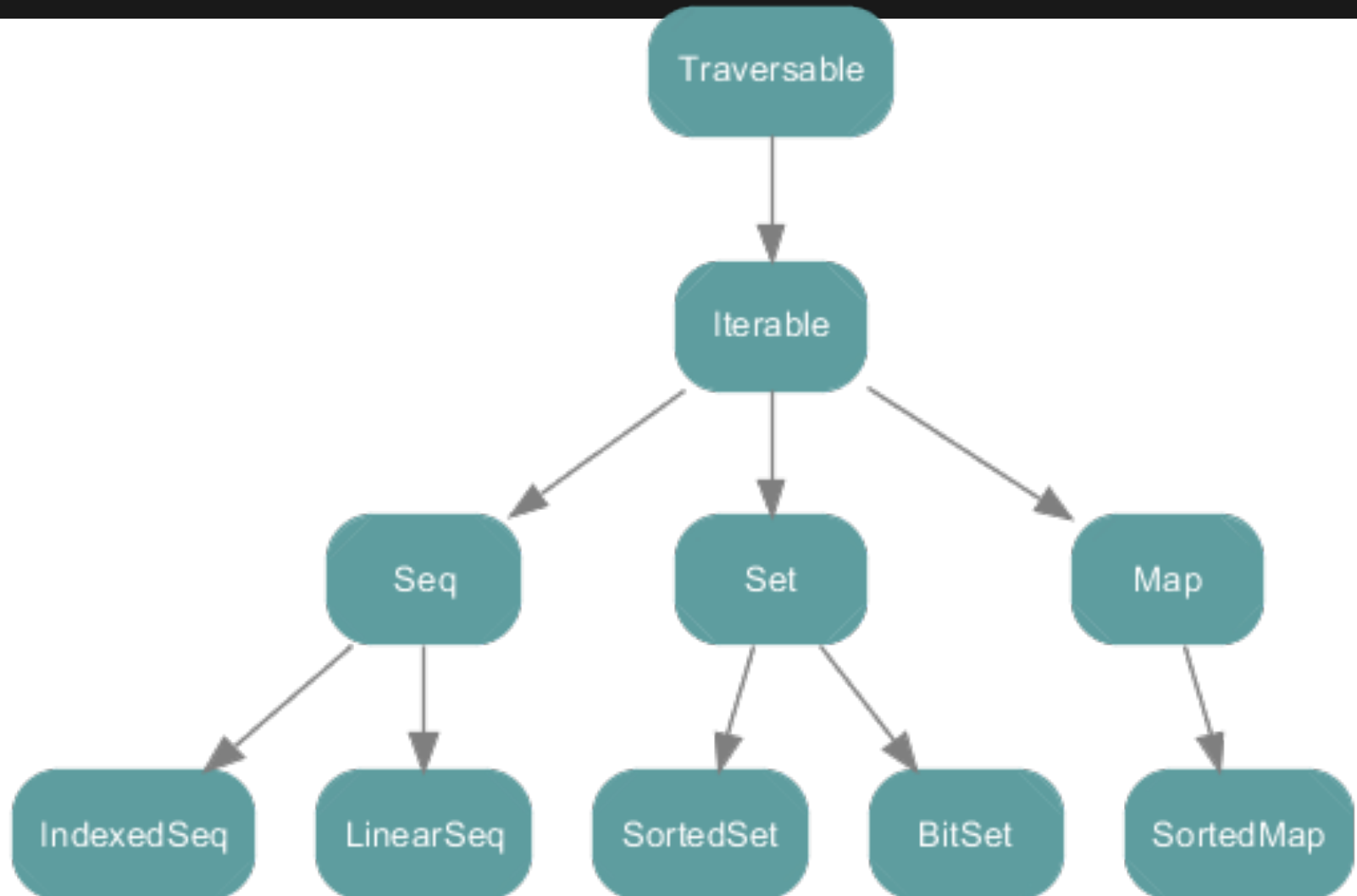
vs.

```
def foo[T <: Seq[A]] (s: T) : T = ?
```

Collections

Know your **collections**

Know your collections



Know your collection API

seq, companion, seq, flatten, transpose, toString, isEmpty, map, exists, find, init, last, head, filter, slice, tail, ++, ++, headOption, drop, filterNot, flatMap, takeWhile, repr, newBuilder, forall, foreach, thisCollection, toCollection, parCombiner, view, view, copyToArray, hasDefiniteSize, ++:, ++:, collect, partition, groupBy, scan, scanLeft, scanRight, lastOption, sliceWithKnownDelta, sliceWithKnownBound, tails, inits, toTraversable, toIterator, withFilter, take, splitAt, dropWhile, span, stringPrefix, toStream, min, max, count, size, toArray, seq, sum, toList, mkString, mkString, mkString, toSet, foldLeft, foldRight, reduceLeft, reduceRight, toSeq, toIterable, copyToArray, copyToArray, reversed, nonEmpty, collectFirst, /:, :\\, reduceLeftOption, reduceRightOption, reduce, reduceOption, fold, aggregate, product, maxBy, minBy, copyToBuffer, toIndexedSeq, toBuffer, toMap, addString, addString, addString, toSet, toSeq, toIterable, toTraversable, isTraversableAgain, toMap, /:\\, size, groupBy, isTraversableAgain, min, max, count, toArray, seq, sum, toList, mkString, mkString, mkString, foldLeft, foldRight, reduceRight, copyToArray, copyToArray, nonEmpty, /:, :\\, reduceLeftOption, reduceRightOption, reduce, reduceOption, fold, aggregate, product, maxBy, minBy, toIndexedSeq, toBuffer, seq, par, map, head, filter, slice, tail, ++, drop, filterNot, flatMap, takeWhile, repr, foreach, collect, partition, scan, scanLeft, scanRight, take, splitAt, dropWhile, span, stringPrefix, isEmpty, exists, find, forall, copyToArray, hasDefiniteSize, toIterator, toStream, parCombiner, size, foreach, isEmpty, head, flatten, newBuilder, foreach, transpose, genericBuilder, unzip, unzip3, isEmpty, exists, find, forall, foreach, copyToArray, hasDefiniteSize, toTraversable, isEmpty, iterator, zip, head, sameElements, zipAll, zipWithIndex, seq, isEmpty, first, iterator, exists, find, zip, zip, elements, head, slice, drop, takeWhile, forall, foreach, canEqual, sameElements, sameElements, foldRight, reduceRight, dropRight, thisCollection, toCollection, view, view, projection, toIterable, grouped, sliding, sliding, copyToArray, zipAll, zipAll, zipWithIndex, firstOption, take, takeRight, toStream, equals

Collections

Use **Vector**

Questions?

www.manning.com
40% books/epub/MEAP
code: **devnex40**

Scala IN DEPTH

Joshua D. Suereth

 MANNING

