

# EFFECTIVE SCALA

SOFTSHAKE 2013, GENEVA

MIRCO DOTTA  
TWITTER: @MIRCODOTTA



**GOLDEN RULE?**

**KEEP IT SIMPLE!**

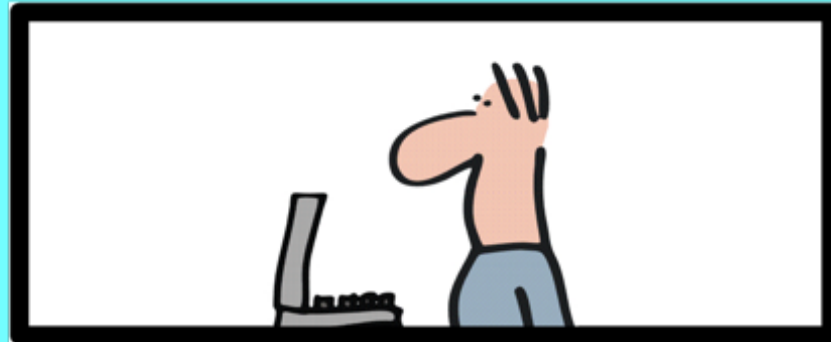






**IT IS NOT BECAUSE YOU  
CAN, THAT YOU SHOULD**





*ISN'T IT A GREAT  
FEELING WHEN YOU  
FINALLY GET YOUR OWN  
CODE, YOU'VE WRITTEN  
MONTHS BEFORE?*

**WHAT'S THIS TALK ABOUT?**

**OPTIMIZE YOUR USE OF  
SCALA TO SOLVE REAL  
WORLD PROBLEMS  
WITHOUT EXPLOSIONS,  
BROKEN THUMBS OR  
BULLET WOUNDS**

# Agenda

- *Style matters*
- *Mantras*
- *Collection-foo*
- *Implicit*



*Style matters*



# learn the Scala way

**YOU KNOW IT WHEN YOU GOT IT**

**SCALA AIN'T JAVA**

**NOR RUBY**

**NOR HASKELL**

**NOR <INSERT KNOWN PL>**

<http://docs.scala-lang.org/style/>

# Abstract members

```
trait Shape {  
  val edges: Int  
}
```

**NO**


Why?

```
class Triangle extends Shape {  
  override def edges: Int = 3  
}
```

Because this  
doesn't  
compile!

# Abstract members

**ALWAYS USE def FOR ABSTRACT MEMBERS!**




```
trait Shape {  
  def edges: Int  
}
```

# You've to override

```
trait Worker {  
  def run(): Unit  
}
```


```
abstract class MyWorker  
  extends Worker {  
  override def run(): Unit =  
    // ...  
}
```





# Don't leak your types!

```
trait Logger {  
  def log(m: String): Unit  
}  
  
object Logger {  
  class StdLogger extends Logger {  
    override def log(m: String): Unit =  
      println(m)  
  }  
  def apply = new StdLogger()  
}
```



**WHAT'S THE RETURN TYPE HERE?**

# Hide your secrets

```
object SauceFactory {  
  def makeSecretSauce(): Sauce = {  
    val ingredients = getMagicIngredients()  
    val formula = getSecretFormula()  
    SauceMaker.prepare(ingredients, formula)  
  }
```

*private!*

```
  def getMagicIngredients(): Ingredients = //...  
  def getSecretFormula(): Formula = //...  
}
```

# Visibility modifiers

- **SCALA HAS VERY EXPRESSIVE VISIBILITY MODIFIERS**
  - ACCESS MODIFIERS CAN BE **AUGMENTED** WITH QUALIFIERS

*Did you know that...*

- **NESTED PACKAGES HAVE ACCESS TO **PRIVATE** CLASSES**
- **COMPANION OBJECT/CLASSES HAVE ACCESS TO EACH OTHER **PRIVATE** MEMBERS!**

<b>PUBLIC</b>	(default)
<b>PACKAGE</b>	private[pkg]
<b>PROTECTED</b>	protected
<b>PRIVATE</b>	private


**EVERYTHING YOU HAVE IN JAVA, AND MUCH MORE**

# Don't blow the stack!

## IF YOU THINK IT'S TAILRECURSIVE, SAY SO!

```
case class Node(value: Int, edges: List[Node])

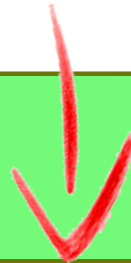
def bfs(node: Node, pred: Node => Boolean): Option[Node] = {
  @scala.annotation.tailrec
  def search(acc: List[Node]): Option[Node] = acc match {
    case Nil => None
    case x :: xs =>
      if (pred(x)) Some(x)
      else search(xs ::: xs.edges)
  }
  search(List(node))
}
```





# String interpolation

```
case class Complex(real: Int, im: Int) {  
  override def toString: String =  
    real + " + " + im + "i"  
}
```



```
case class Complex(real: Int, im: Int) {  
  override def toString: String =  
    s"$real + ${im}i"  
}
```



*interpolator!*



*Mantras*

# Use the REPL

```
Mircos-MacBook-Pro:~ mirco$ scala
Welcome to Scala version 2.10.1 (Java HotSpot(TM) 64-Bit Server VM, Java 1.6.0_45).
Type in expressions to have them evaluated.
Type :help for more information.

scala> class Greet {
      | def hello(m: String) = println("Hello, "+m)
      | }
defined class Greet

scala> new Greet
res0: Greet = Greet@1a61c596

scala> res0.hello("SoftShake hackers!")
Hello, SoftShake hackers!

scala> []
```

# On the Worksheet :-)

sheet.sc ✕

```
object sheet {  
  println("Welcome to the Scala worksheet")      > Welcome to the Scala worksheet  
  
  case class Greet {  
    def hello(m: String) = println("Hello, " + m)  
  }  
  
  val greet = new Greet                          > greet : sheet.Greet = Greet()  
  greet.hello("SoftShake hackers!")              > Hello, SoftShake hackers!  
}
```



Write Expressions,  
*not* Statements

# *Expressions!*

- **SHORTER**
- **SIMPLER**
- **COMPOSABLE!**


```
def home(): HttpPage = {  
  var page: HttpPage = null  
  try page = prepareHttpPage()  
  catch {  
    case _: TimeoutEx => page = TimeoutPage  
    case _: Exception => page = NoPage  
  }  
  return page  
}
```



```
def home(): HttpPage =  
  try prepareHttpPage()  
  catch {  
    case _: TimeoutEx => TimeoutPage  
    case _: Exception => NoPage  
  }
```

# Expressions compose!

```
def home(): HttpPage =  
  try prepareHttpPage()  
  catch {  
    case _: TimeoutEx => TimeoutPage  
    case _: Exception => NoPage  
  }
```



*Try..catch is  
an expression*

*This is a* PartialFunction[Throwable, HttpPage]

```
def timeoutCatch = {  
  case _: TimeoutEx => TimeoutPage  
}  
def noPageCatch = {  
  case _: Exception => NoPage  
}
```

```
def home(): HttpPage =  
  try prepareHttpPage()  
  catch timeoutCatch orElse  
    noPageCatch
```

*compose!*



*Don't use* null

null *is a disease*

- **NULLCHECKS WILL SPREAD**
- **CODE IS BRITTLE**
- **NPE WILL STILL HAPPEN**
- **ASSERTIONS WON'T HELP**

# *Forget* null, *use* Option

- NO MORE “IT MAY BE NULL”
- TYPE-SAFE
- DOCUMENTATION

```
def authenticate(session: HttpSession,  
                 username: Option[String],  
                 password: Option[String]) =  
  for {  
    user <- username  
    pass <- password  
    if canAuthenticate(user,pass)  
    privileges <- privilegesFor(user)  
  } yield inject(session, privileges)
```

# But don't overuse Option

## SOMETIME A NULL-OBJECT CAN BE A MUCH BETTER FIT

```
def home(): HttpPage =  
  try prepareHttpPage()  
  catch {  
    case _: TimeoutEx => TimeoutPage  
    case _: Exception => NoPage  
  }
```

*Null Object!*

*Stay immutable*



# *Immutability*

## **3 REASONS WHY IT MATTERS?**

- **SIMPLIFIES REASONING**
- **SIMPLIFIES REASONING**
- **SIMPLIFIES REASONING**

# Immutability

**ALLOWS CO/CONTRA VARIANCE**

**MUTABILITY  
+ VARIANCE**

**TROUBLES**

JAVA

```
String[] a = {" "};  
Object[] b = a;  
b[0] = 1;  
String value = a[0];
```

`java.lang.ArrayStoreException`

# *Immutability*

- **CORRECT EQUALS & HASHCODE!**
- **IT *ALSO* SIMPLIFIES REASONING ABOUT CONCURRENCY**
- **THREAD-SAFE BY DESIGN**

# *Immutability*

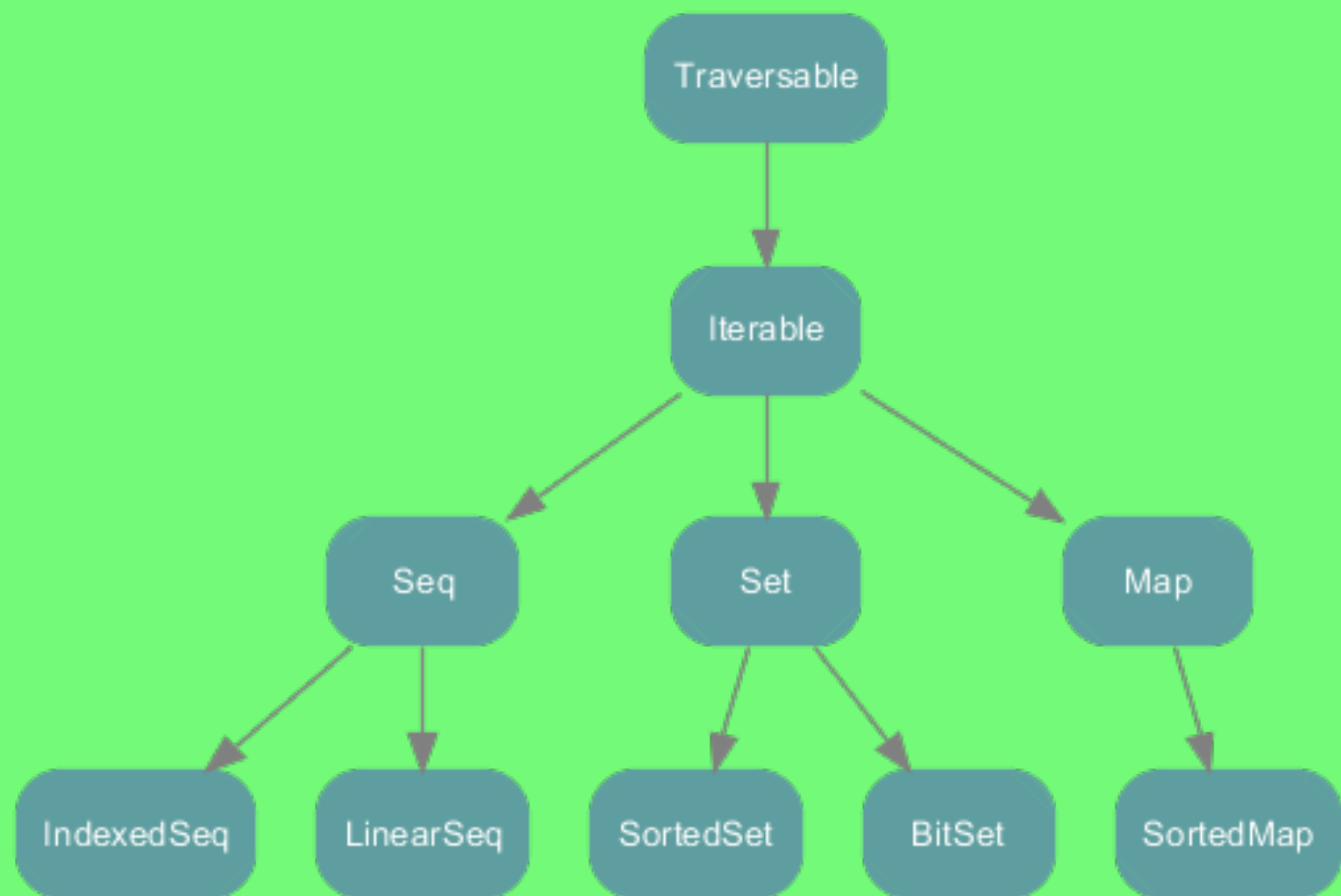
- **MUTABILITY IS STILL OK, BUT  
KEEP IT IN LOCAL SCOPES**
- **API METHODS SHOULD RETURN  
IMMUTABLE OBJECTS**

*Do you want faster Scala compilation?*

**PROGRAM TO AN INTERFACE, NOT  
AN IMPLEMENTATION.**



*Collection-foo*






# Learn the API

```
!=, ##, ++, ++:, +:, /:, /\, :+, ::, :::, :\, <init>, ==, addString, aggregate,
andThen, apply, applyOrElse, asInstanceOf, canEqual, collect, collectFirst,
combinations, companion, compose, contains, containsSlice, copyToArray,
copyToBuffer, corresponds, count, diff, distinct, drop, dropRight, dropWhile,
endsWith, eq, equals, exists, filter, filterNot, find, flatMap, flatten, fold,
foldLeft, foldRight, forall, foreach, genericBuilder, getClass, groupBy, grouped,
hasDefiniteSize, hashCode, head, headOption, indexOf, indexOfSlice, indexWhere,
indices, init, inits, intersect, isDefinedAt, isEmpty, isInstanceOf,
isTraversableAgain, iterator, last, lastIndexOf, lastIndexOfSlice, lastIndexWhere,
lastOption, length, lengthCompare, lift, map, mapConserve, max, maxBy, min, minBy,
mkString, ne, nonEmpty, notify, notifyAll, orElse, padTo, par, partition, patch,
permutations, prefixLength, product, productArity, productElement, productIterator,
productPrefix, reduce, reduceLeft, reduceLeftOption, reduceOption, reduceRight,
reduceRightOption, removeDuplicates, repr, reverse, reverseIterator, reverseMap,
reverse_:::, runWith, sameElements, scan, scanLeft, scanRight, segmentLength, seq,
size, slice, sliding, sortBy, sortWith, sorted, span, splitAt, startsWith,
stringPrefix, sum, synchronized, tail, tails, take, takeRight, takeWhile, to,
toArray, toBuffer, toIndexedSeq, toIterable, toIterator, toList, toMap, toSeq,
toSet, toStream, toString, toTraversable, toVector, transpose, union, unzip,
unzip3, updated, view, wait, withFilter, zip, zipAll, zipWithIndex
```

# Know when to breakOut

```
def adultFriends(p: Person): Array[Person] = {  
  val adults = // <- of type List  
  for { friend <- p.friends // <- of type List  
    if friend.age >= 18 } yield friend  
  adults.toArray  
}
```



## CAN WE AVOID THE 2ND ITERATION?

```
def adultFriends(p: Person): Array[Person] =  
  (for { friend <- p.friends  
    if friend.age >= 18  
  } yield friend)(collection.breakOut)
```

# Common pitfall

```
def getUsers: Seq[User]
```

*mutable or immutable?*

**GOOD QUESTION!**

*remedy*

```
import scala.collection.immutable  
def getUsers: immutable.Seq[User]
```





*Implicit*



**LIMIT THE SCOPE OF IMPLICIT**

# *Implicit Resolution*

- **IMPLICITS IN THE LOCAL SCOPE**
  - **IMPLICITS DEFINED IN CURRENT SCOPE (1)**
  - **EXPLICIT IMPORTS (2)**
  - **WILDCARD IMPORTS (3)**
- **PARTS OF A TYPE**
  - **COMPANION OBJECT OF THE TYPE (AND INHERITED TYPES)**
  - **COMPANION OBJECTS OF TYPE ARGUMENTS OF THE TYPE**
  - **OUTER OBJECTS FOR NESTED TYPES**

# Implicit Scope (Parts)

```
trait Logger {  
  def log(m: String): Unit  
}  
  
object Logger {  
  implicit object StdLogger extends Logger {  
    override def log(m: String): Unit = println(m)  
  }  
  def log(m: String)(implicit l: Logger) = l.log(m)  
}
```

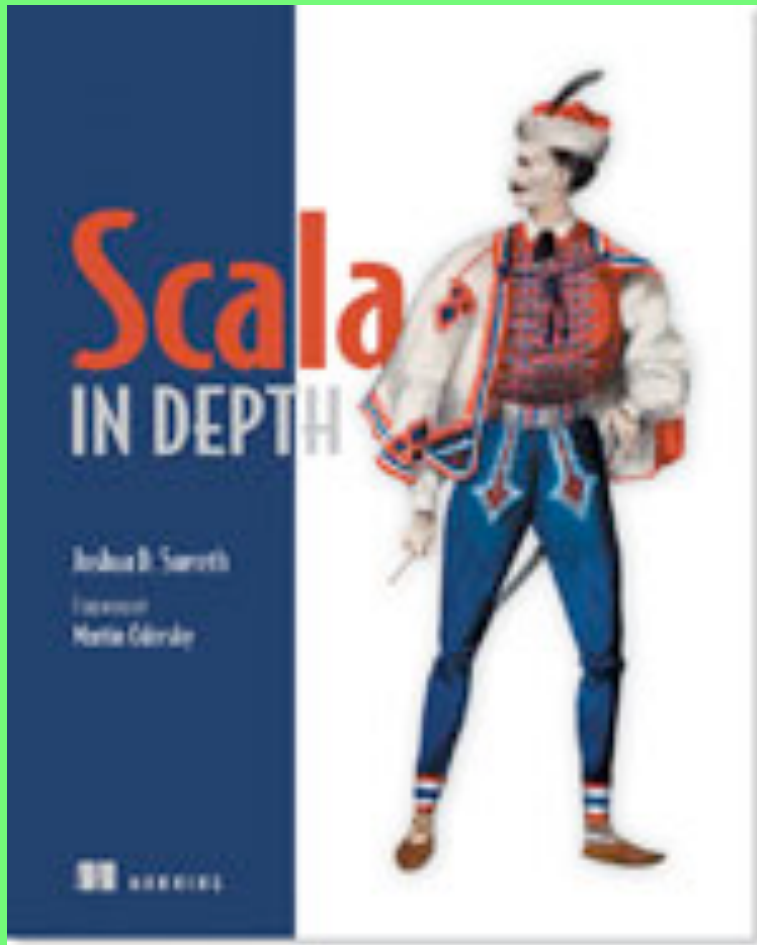
```
Logger.log("Hello")
```





*Looking ahead*

# Level up!



*Scala In Depth*  
*Joshua Suereth*

<http://www.manning.com/suereth/>

# EFFECTIVE SCALA

Thanks to [@heathermiller](#) for the presentation style

**MIRCO DOTTA**  
**TWITTER: @MIRCODOTTA**

