

Introducing Scala in Java territory

Peter Maas
2011-Q4



Peter Maas

- Team lead at eBay Classifieds Group working on B2C portfolio for Marktplaats & DBA
 - Clicks / PageViews / Money
- Interested in programming languages in general; every new concept you learn might help you solve problems in a smarter / cleaner way.
- Hasn't worked on a MS Windows machine for quite a while.
- Background in Sound & Music, specialized in pattern recognition. But spend the last decade on web development.



In the past I have successfully introduced:

- Spring/Hibernate in J2EE heavy organizations
- A Ruby development street within a Java shop
- Groovy/Grails
- NoSQL (CouchDB) in a postgres environment
- Camel

Etc...

SCALA IS DIFFERENT

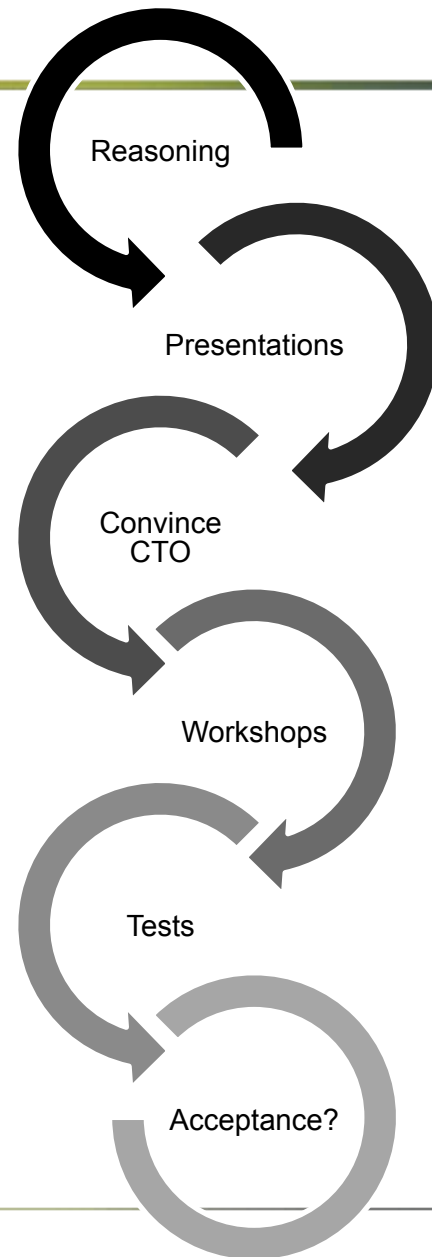
IT SOLVES PROBLEMS MANY PEOPLE DON'T RECOGNIZE

- *Scalable* Language
- Statically typed
- First class citizen on the JVM
- Multi paradigm: OO, FP

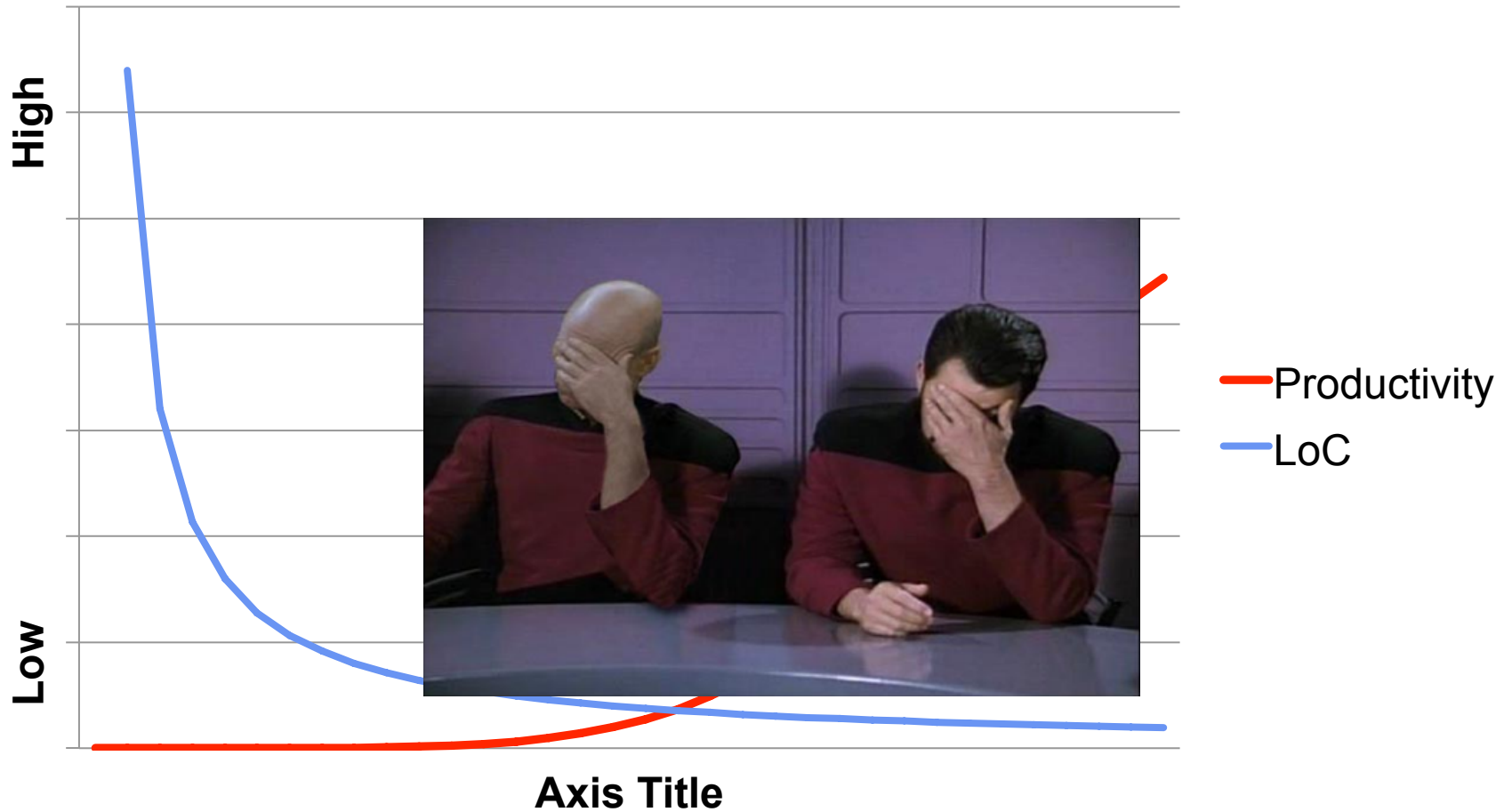
Who uses it



The process:



Reasoning – “productivity”



Reasoning

Most algorithms can be characterized as:

- Searching (some find find-if mismatch)
- Sorting (sort merge remove-duplicates)
- Filtering (remove remove-if mapcan)
- Mapping (map mapcar mapc)
- Combining (reduce mapcan)
- Counting (count count-if)

These functions abstract common control patterns.

Code that uses them is:

- Concise
- Self-documenting
- Easy to understand
- Often reusable
- Usually efficient (Better than a non-tail recursion)



Source: Luv slides, Peter Norvig, 1993

Java - Filtering

```
public interface Predicate<T> { boolean apply(T type); }

public static <T> Iterable<T> filter(Iterable<T> target, Predicate<T> predicate) {
    final Collection<T> result = new ArrayList<T>();
    for (T element: target) {
        if (predicate.apply(element)) {
            result.add(element);
        }
    }
    return result;
}

Predicate<User> isAuthorized = new Predicate<User>() {
    @Override public boolean apply(User user) {
        // binds a boolean method in User to a reference
        return user.isAuthorized();
    }
};

// allUsers is a Collection<User>
Collection<User> authorizedUsers = filter(allUsers, isAuthorized);
```

// Scala

`allUsers.filter(_.authorized)`

// (J)Ruby

`allUsers.select{|u| u.authorized}`

// Clojure

`(filter #(not= (:authorized)) allUsers)`

// Python

`filter(lambda u: u.authorized, allUsers)`

General feeling after a couple of weeks of scala:

A lot of "stuff that's hard or impossible in Java is simple in Scala," Scala is a very easy language. Dealing with collections is super easy in Scala. Isolating business logic making programs much more maintainable is vastly easier in Scala than it is in Java.

David Pollak

Creator of lift

Snippets

```
// Regex to split a date in the format Y/M/D.
```

```
val regex = "(\\d+)/ (\\d+)/ (\\d+)".r
```

```
val regex(year, month, day) = "2010/1/13"
```

```
// Structural types
```

```
def printName(f: { def getName(): String }) { println(f.getName) }
```

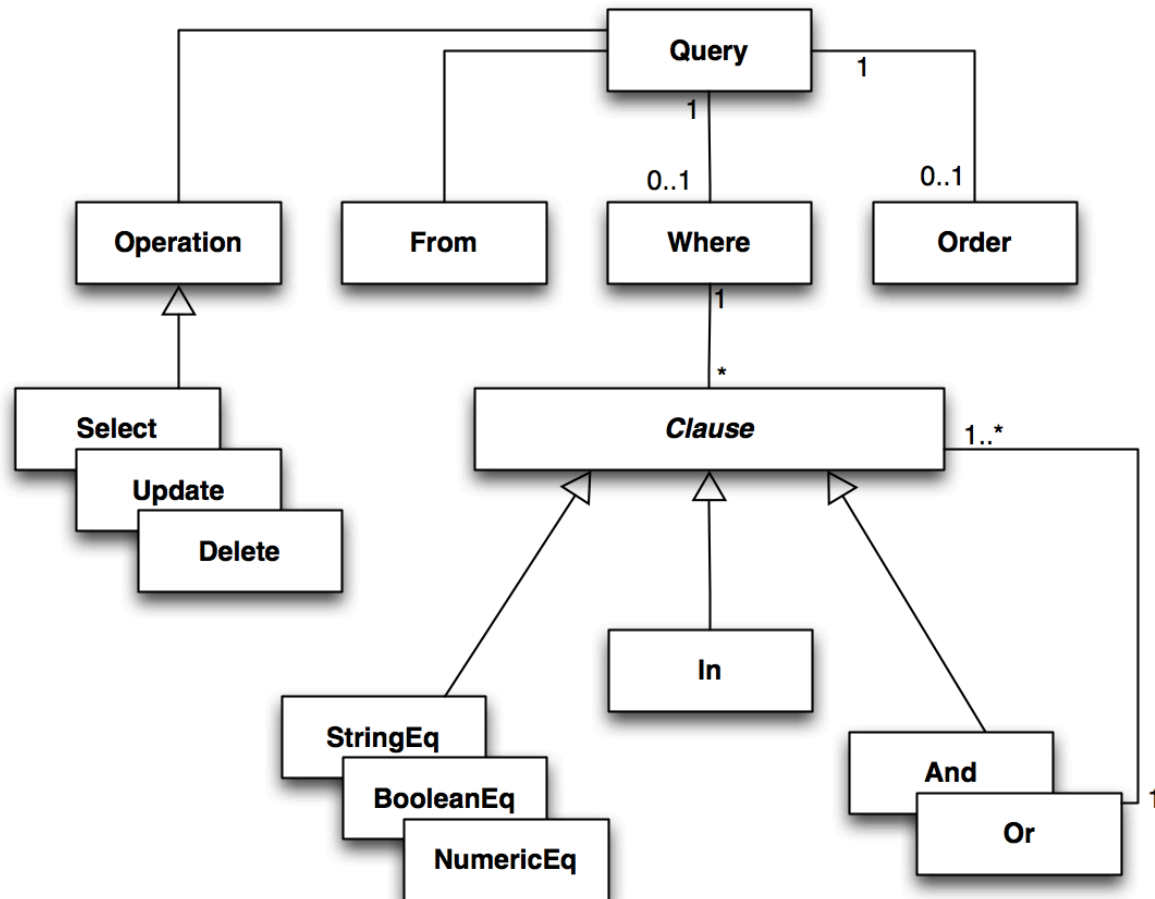
```
printName(new File(...))
```

```
printName(user)
```

```
// Options
```

```
Map(1 -> 2).get(2) match {  
  case Some(d) => println(d)  
  case None => ...  
}
```

```
Map(1 -> 2).getOrElse(2, 4)
```



Parser combinators

```
class SQLParser extends JavaTokenParsers {

  def query:Parser[Query] = operation ~ from ~ opt(where) ~ opt(order) ^^ {
    case operation ~ from ~ where ~ order => Query(operation, from, where, order)
  }

  def operation:Parser[Operation] = {
    ("select" | "update" | "delete") ~ repsep(ident, ",") ^^ {
      case "select" ~ f => Select(f:_)
      case _ => throw new IllegalArgumentException("Operation not implemented")
    }
  }

  def from:Parser[From] = "from" ~> ident ^^ (From(_))

  def where:Parser[Where] = "where" ~> rep(clause) ^^ (Where(_:_*))

  def clause:Parser[Clause] = (predicate|parens) * (
    "and" ^^ { (a:Clause, b:Clause) => And(a,b) } |
    "or" ^^ { (a:Clause, b:Clause) => Or(a,b) }
  )

  def parens:Parser[Clause] = "(" ~> clause <~ ")"

  ...
}
```

Workshops

- Introduction to basic Scala features
- TDD using Scala/ScalaTest
- Writing simple webapps in Scala with the Play! framework
- Messaging with Actors







TDD Exercise

Build a program to calculate the best scores for a given Yahtzee dice roll. Dive the code from tests. Mutable state is not allowed.

The first person in the line starts writing a test.
The next person makes it green.

Apart from calculating the score for a roll it would be very nice to have a function for calculating the best score for a given dice roll.

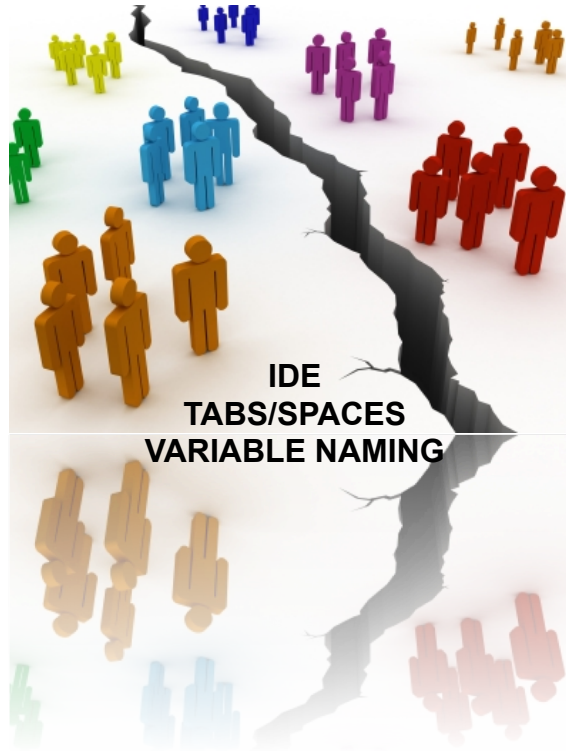
Yahtzee Name _____

UPPER SECTION	HOW TO SCORE	GAME #1	GAME #2	GAME #3	GAME #4	GAME #5	GAME #6
Aces  = 1	Count and Add Only Aces						
Twos  = 2	Count and Add Only Twos						
Threes  = 3	Count and Add Only Threes						
Fours  = 4	Count and Add Only Fours						
Fives  = 5	Count and Add Only Fives						
Sixes  = 6	Count and Add Only Sixes						
TOTAL SCORE	→						
BONUS <small>If total score is 63 or over</small>	SCORE 35						
TOTAL <small>Of Upper Section</small>	→						
LOWER SECTION							
3 of a kind	Add Total Of All Dice						
4 of a kind	Add Total Of All Dice						
Full House	SCORE 25						
Sm. Straight <small>Sequence of 4</small>	SCORE 30						
Lg. Straight <small>Sequence of 5</small>	SCORE 40						
YAHTZEE <small>5 of a kind</small>	SCORE 50						
Chance	Score Total Of All 5 Dice						
YAHTZEE BONUS	✓ FOR EACH BONUS						
	SCORE 100 PER ✓						
TOTAL <small>Of Lower Section</small>	→						
TOTAL <small>Of Upper Section</small>	→						
GRAND TOTAL	→						

Something interesting happened

```
def street(dices:Seq[Int], size:Int, points:Int) = {  
  val sortedDices = dices.sorted  
  val containsSlice = sortedDices.zip(sortedDices.tail)  
    .map(t => t._1 - t._2)  
    .containsSlice(List.fill(size)(-1))  
  
  if (containsSlice) points else 0  
}  
  
def smallStreet(dices:Seq[Int]) = street(dices, 3, 30)  
def bigStreet(dices:Seq[Int]) = street(dices, 4, 40)
```

Observation



So... focus on solving problems

```
object ECGRouter extends App {
  distributeAccounts()

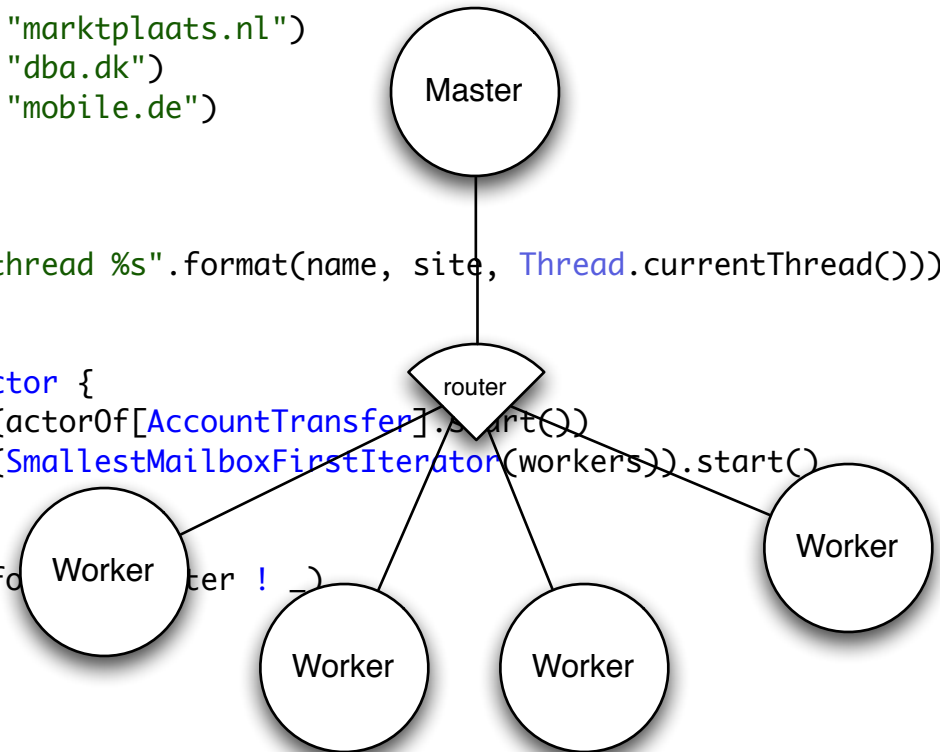
  case class Person(name: String, country: String)

  class AccountTransfer extends Actor {
    def receive = {
      case Person(n: String, "nl") => log(n, "marktplaats.nl")
      case Person(n: String, "dk") => log(n, "dba.dk")
      case Person(n: String, "de") => log(n, "mobile.de")
    }

    def log(name: String, site: String) =
      println("sending %s to %s from thread %s".format(name, site, Thread.currentThread()))
  }

  class Master(nrOfWorkers: Int) extends Actor {
    val workers = Vector.fill(nrOfWorkers)(actorOf[AccountTransfer].start())
    val router = Routing.loadBalancerActor(SmallestMailboxFirstIterator(workers)).start()

    def receive = {
      case persons: Seq[Person] => persons.foreach { person ! router }
    }
  }
}
```



Composing

```
abstract class Repository[T] {  
  def withConn(f: (Connection => Option[T])): Option[T]  
}  
  
trait Retrying[T] extends Repository[T] {  
  abstract override def withConn(f: (Connection => Option[T])): Option[T] = retry(f)  
  
  private def retry(f: (Connection => Option[T]), times: Int = 3): Option[T] = {  
    try {  
      println("number of tries left: %d".format(times))  
      super.withConn(f)  
    } catch {  
      case _ if times == 0 => None  
      case _ => retry(f, times - 1)  
    }  
  }  
}  
  
val userRepo = new UserRepository with Retrying[User]
```

Summarizing

Start by identifying issues developer *actually* have and fix these.

Avoid crossing the complexity line to early

Don't push; facilitate

Issues we (still) face(d)

Housekeeping

- Yes... it's getting old... but IDE integration is (still) a nightmare. Remember the part about the Dart editor this morning?
- Language version updates tend to be more invasive then Java people expect.
- Compiler speed...
- ScalaDoc (Scalas' version of JavaDoc) is very hard to digest for people not living on planet TypeSystem.
- Integration with Java isn't always 'seamless'. Numbers in `Object[]`, `JavaConversions._`

Mindset

- To get the best out of Scala a functional approach to solve problems is a must.
- You need to be *very* strict about patterns and style to avoid extreme inconsistencies in the codebase. Comparable to JavaScript.

Acceptation?

A question asked in a recent interview on infoq:

Is Scala worth the effort for average developer teams to learn? Do the benefits outweigh the complexity and learning curve?

it depends....

**IF YOU WANT TO BE READY
FOR THE FUTURE YOU NEED
TO KEEP INNOVATING!**

TEAM/Culture Checklist

- 'Fluent' developers?
- Not afraid of doing something twice?
- Commitment to maintain the code?
- Not dependent on specific IDE support?
- Working code is just the beginning?
- Wouldn't be scared away by my previous Parser Combinator and Zipper examples?
- Is capable of teaching new developers their craft?

Fear is the mind killer

Embrace Risk

One more thing:

Q&A

(yes: We are hiring)