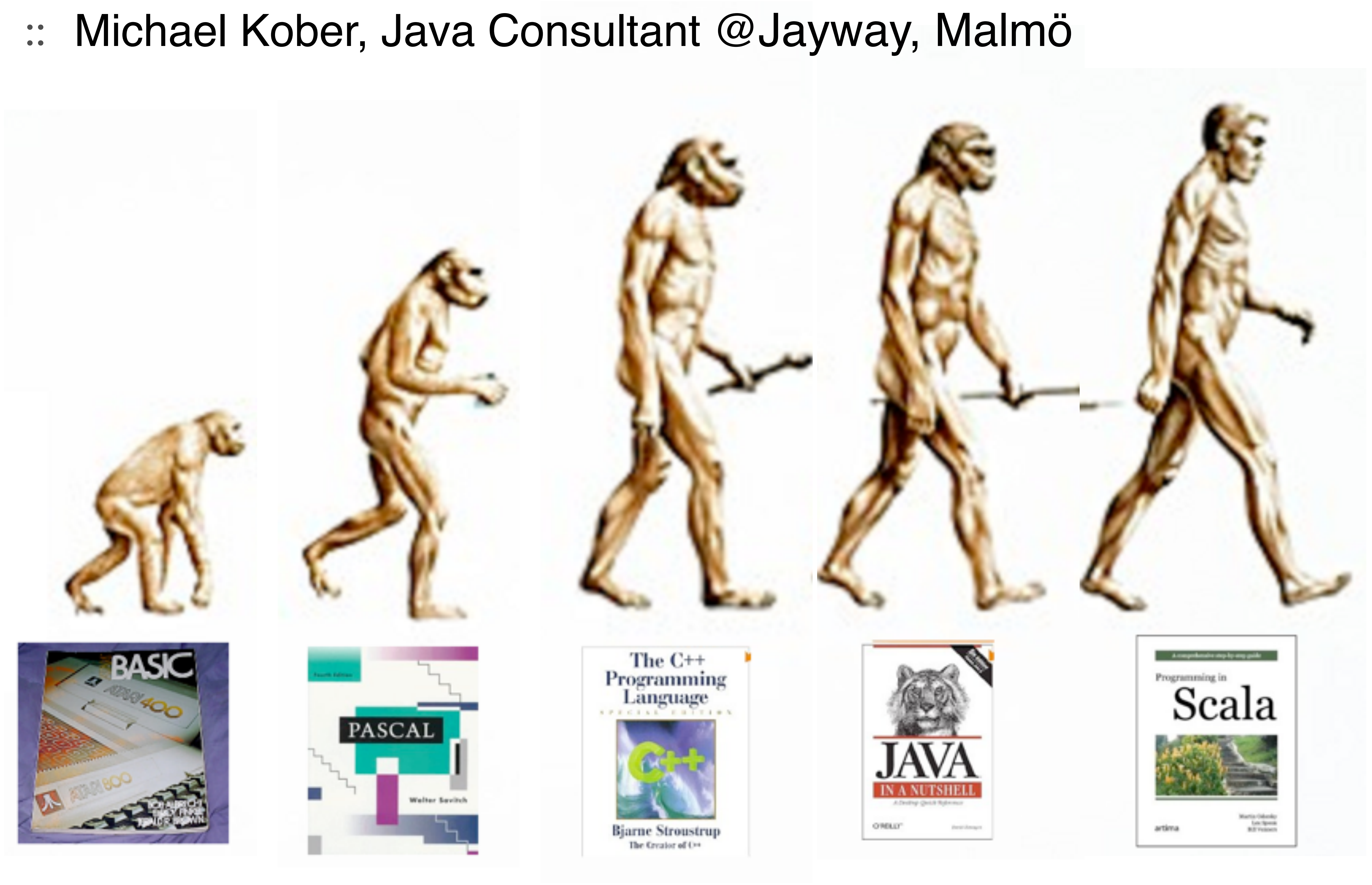
Scala

:: Introduction to a scalable language

JAYWAY

About me

:: Michael Kober, Java Consultant @Jayway, Malmö



Outline

:: History

:: A scalable language

:: Why Scala

:: First steps

:: Functional programming

:: Pattern matching

:: Mixins

:: Implicit conversions

:: Structural types

:: Actors

:: XML

:: Testing

:: Building

:: Tools and frameworks

:: Is Scala too complex?

:: Scala in your company

:: Resources

:: Conclusion

JAYWAY

History

- :: 1999 Martin Odersky joined EPFL (Ecole polytechnique de Lausanne)
- :: ~2001 programming methods lab EPFL starts developing Scala
  - :: Initial goals: a better Java, merge of functional and object oriented programming
- :: 2004 first public release on JVM
- :: 2006 major rewrite
- :: 2010 Scala Days
- :: Current version: 2.8.0

JAYWAY

A scalable language

- Scala is
  - a functional language
  - an object oriented language
  - a scripting language
  - concise
  - high-level
  - statically typed

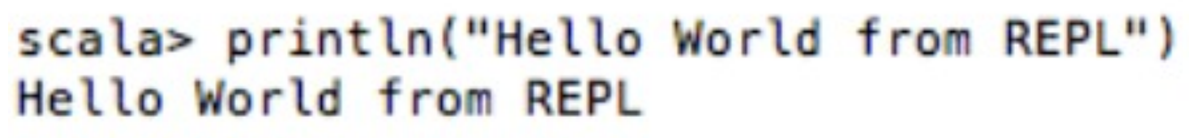
Why Scala?

- Runs on JVM
- Java compatibility
- first class functions
- type inference keeps code clean
- less “boilerplate”
- reusable behaviour through mixins
- simpler concurrency through Actors
- powerful pattern matching
- performance same as Java
- there’s also a .NET version
- don’t need to abandon Java

First steps - HelloWorld

```scala
package com.jayway.sample

object HelloWorld {

  def main(args: Array[String]) {
    println("Hello World")
  }
}
```

Or even easier from the REPL (Read Eval Print Loop)



```
scala> println("Hello World from REPL")
Hello World from REPL
```

DEMO

JAYWAY

First steps - Scala compared to Java

- :: pure object oriented
- :: first class functions
- :: operator overloading
- :: closures
- :: mixin composition with traits
- :: pattern matching
- :: abstract types
- :: there's much more...

- :: no static members
- :: no primitive types
- :: no break, continue
- :: no raw types
- :: no checked exceptions
- :: no enums

JAYWAY

First steps - Definitions

Scala

```
// variable definitions
 val greeting = "Hello World"
 var count = 0

 // Scala method definition
 def fun(x: Int) : Int = {
   x * x
 }

 // parameterless method
 def today : String = {
   val sdf = new SimpleDateFormat("yyyy.MM.dd")
   sdf.format(new Date())
 }
```

Java

```
// variable definitions
final String greeting = "Hello World";
int count = 0;

// Java method definition
int fun(int x) {
    return x * x;
    }

// no parameterless methods
```

JAYWAY

First steps - Expressions

Scala

```
// method calls
object.method(arg)
// operator notation
object method arg

// choice expressions
if (condition) expr1 else expr2

epxr match {
  case pattern1 => expr1
  ...
  case patternN => exprN
}
```

Java

```
// method calls
object.method(arg)



// choice expressions
condition ? expr1 : expr2
if (condition) return expr1;
  else return expr2;

switch expression {
  case pattern1 => return expr1
  ...
  case patternN => return exprN
} // statement only
```

JAYWAY

First steps - Objects and Classes

DEMO

Scala

```scala
class Complex(val re: Double, val im: Double) {

  def +(b: Complex) : Complex = {
      new Complex(re + b.re, im + b.im)
  }

  def -(b: Complex) : Complex = {
      new Complex(re - b.re, im - b.im)
  }

  override def toString : String = {
    (re, im) match {
      case (0, 0) => "0"
      case (0, im) => im + "i"
      case (re, 0) => re + ""
      case (re, im) => re + " + " + im + "i"
    }
  }
}

object Complex {
    def add(a: Complex, b: Complex) = a + b
}
```

Java

```java
public class Complex {
    private final double re;
    private final double im;

    public Complex(double re, double im) {
        this.re = re;
        this.im = im;
    }

    public double getRe() {
        return re;
    }

    public double getIm() {
        return im;
    }

    public Complex plus(Complex b) {
        return new Complex(re + b.getRe(), im + b.getIm());
    }

    public Complex minus(Complex b) {
        return new Complex(re - b.getRe(), im - b.getIm());
    }

    // static version of plus
    public static Complex add(Complex a, Complex b) {
        return a.plus(b);
    }
}
```

JAYWAY

First steps - collections

:: scala.collection.mutable vs. scala.collection.immutable

:: Scala encourages use of immutable collections

Scala

```
// List
val list1 = List(1, 2, 3, 4, 5)
// same as 'literal' syntax
val list2 = 1 :: 2 :: 3 :: 4 :: 5 :: Nil

println(list1.head) // == 1
println(list1.tail) // == List(2,3,4,5)

// Map
val capitals = Map(
        "Denmark" -> "København",
        "Sweden" -> "Stockholm",
        "Germany" -> "Berlin")
```

Java

```
//List
List<Integer> list1 =
    Arrays.asList(1, 2, 3, 4, 5);

// Map
Map<String, String> capitals =
     new HashMap<String, String>();
capitals.put("Denmark", "København");
capitals.put("Sweden", "Stockholm");
capitals.put("Germany", "Berlin");
```

JAYWAY

First steps - for comprehensions

DEMO

Scala

```
val filesHere =
  (new java.io.File(".")).listFiles
for(file <- filesHere) {
  println(file)
}

// using a range
for(i <- 0 to 5) {
  println("iteration " + i)
}

// filtering
for(file <- filesHere
    if file.isDirectory) {
  println("DIR " + file)
}

val dirs = for(file <- filesHere
    if file.isDirectory) yield file
```

Java

```
File[] filesHere =
    (new java.io.File(".")).listFiles();
for (File file : filesHere) {
    System.out.println(file);
}

for (int i = 0; i <= 5; i++) {
    System.out.println("iteration " + i);
}

// filtering
for (File file : filesHere) {
    if (file.isDirectory()) {
        System.out.println("DIR " + file);
    }
}
```

JAYWAY

First steps - Tuples

DEMO

:: immutable

:: can contain different types

```
def getHostAndPort() : Tuple2[String, Int] = {
  ("localhost", 8080)
}

val (host, port) = getHostAndPort
println("use host " + host + “ and port ” + port)

val pair = getHostAndPort
println(pair._1)

val (server, _) = getHostAndPort
```

JAYWAY

Functional programming - what?

- functions in mathematics: no side effects
- programming paradigm
  - computation is evaluation of mathematical functions
  - avoid state and mutable data
  - emphasize application of functions instead of change of state
  - first class and higher order functions
- Scala does not require functions to be pure, nor variables to be immutable, but encourages it

Functional programming - simple example

:: function definition in math

$$\text{max(a,b)} = \begin{cases} a & \text{if } (a >= b) \\ b & \text{if } (b > a) \end{cases}$$

:: function definition in Scala

```
def max(a: Int, b:Int) : Int = if (a >= b) a else b

// max3
def max3(a: Int, b:Int, c: Int) : Int = if (a >= b) max(a, c) else max(b, c)

// max3 slightly more functional style, composition of functions
def max3(a: Int, b:Int, c: Int) : Int = max(a, max(b, c))
```

JAYWAY

Function literals

DEMO

Scala

```scala
val increase = (x : Int) => x + 1

var func = increase

var func = (x: Int) => {
  println("this is nice")
  x + 1
}

def functionFactory(y: Int) = {
  (x : Int) => x + y
}

// placeholder syntax
var filter = (_: Int) > 0
```

Java 8 ?

```java
#int(int) increase = #(int x)(x + 1);

#int(int) func = increase;

#int(int) func = #(int x) {
    System.out.println("this is nice");
    return x + 1;
}
```

JAYWAY

Higher order functions

DEMO

```
// foreach
List(1, 2, 3, 4, 5).foreach(println(_))

// same as
List(1, 2, 3, 4, 5) foreach println

// for comprehension example using foreach
val filesHere = (new java.io.File(".")).listFiles
filesHere.foreach(println(_))

// operator notation
filesHere foreach println

// or even shorter
(new java.io.File(".")).listFiles foreach println
```

Higher order functions - continued

DEMO

```
// map
List(1,2,3,4,5) map { _ * 2 }   // '_ * 2' is shorthand for (i:Int) => i * 2

val twoTimes = (i: Int) => i * 2   // function literal

val newList = List(1,2,3,4,5) map twoTimes

// filter
List.range(1, 20).filter(_ % 2 == 0)

// get directory names: filter & map
val dirNames = filesHere.filter(_.isDirectory).map(_.getName)


// reduceLeft
List(1,2,3,4,5) reduceLeft { _ * _ }
```

There's much more: find, foldLeft, forall, reduceRight, count, flatMap, ...

JAYWAY

Closures

:: first-class function with free variables that are bound in the lexical environment

```
// example
var factor = 3
val multiplier = (i: Int) => i * factor   // factor defined in enclosing scope
val l1 = List(1,2,3,4,5) map multiplier

factor = 10
val l2 = List(1,2,3,4,5) map multiplier
println(l1)
println(l2)
```

Currying

- named after mathematician Haskell Curry
- transform function with multiple parameters into a chain of functions
- makes it easy to specialize functions

```scala
// curried function
def multiplier(factor: Int)(i: Int) = factor * i

val byFour = multiplier(4)(_)
val byTen = multiplier(10)(_)

println(multiplier(5)(3))
println(byTen(5))
```

DEMO

JAYWAY

Currying

```scala
def times(i: Int)(block: => Unit) {
  (0 until i).foreach(_ => block)
}

val twice = times(2)(_)

times(5) {
  println("Scala is cool")
}

twice {
  println(“Hello”)
}
```

DEMO

Pattern matching

:: switch/case statement on steroids

:: not limited on matching against values of ordinal types

```
def describe(x: Any) = x match {
    case 5 => "five"
    case true => "true"
    case "ping" => "pong"
    case x: String => "a string"
    case Nil => "empty list"
    case (a: Int, b:Int) => "a tuple of ints"
    case _ => "something else"
  }
```

JAYWAY

Case classes

- equality check scala ==

```
case class Point(x: Double, y: Double)
```

- ‘case’ keyword suggests an association with case expressions in pattern matching
- causes compiler to add some features:
  - adds factory method with name of class
  - all constructor args implicitly get a val prefix
  - equals, hashCode, toString for free

JAYWAY

Pattern matching with case classes

```scala
case class Point(x: Double, y: Double)
abstract class Shape()
case class Circle(center: Point, radius: Double) extends Shape()
case class Rectangle(lowerLeft: Point, height: Double, width: Double) extends Shape()
case class Square(lowerLeft: Point, height: Double) extends Shape()

def stretch(s: Shape) : Shape = s match {
  case Circle(c,r) => Circle(c, r * 2)
  case Square(l,h) => Square(l, h * 2)
  case Rectangle(l, h, w) => Rectangle(l, h * 2, w * 2)
  case _ => throw new IllegalArgumentException("don't know how to stretch")
}

def main(args: Array[String]) {
  val shapes: List[Shape] = List(
            Circle(Point(0.0,1.0), 2.0),
            Circle(Point(0.0,0.0), 1.0),
            Square(Point(2.0,2.0), 3.0),
            Rectangle(Point(0.0,0.0), 2.0, 3.0))
  val stretchedShapes = shapes.map(stretch(_))
  println(stretchedShapes)
}
```

DEMO

JAYWAY

Traits as mixins

- like Java interfaces with optional implementations
- makes behaviour reusable
- mixin to classes or when creating instances

```scala
abstract class Widget
class Label extends Widget

trait Clickable {
    def click() : Unit = println("I'm clicked")
}

class Button extends Widget with Clickable
class Calendar extends Widget with Clickable

val clickableLabel = new Label() with Clickable
```

Stackable traits

- Order of mixins is significant
- traits further to the right take effect first
- differs from multiple inheritance
  - call to super is determined by a linearization of the classes and traits that are mixed in

```
trait Doubling extends IntQueue {
  abstract override def put(x: Int) { super.put(2 * x) }
}
trait Incrementing extends IntQueue {
  abstract override def put(x: Int) { super.put(x + 1) }
}
val queue1 = new BasicIntQueue with Incrementing with Doubling
val queue2 = new BasicIntQueue with Doubling with Incrementing
```

DEMO

JAYWAY

Implicit conversions - Pimp your library

```
val name: java.lang.String = "scala"
// Predef.stringWrapper
name.capitalize.reverse

class MyRichString(str: String) {
  def acronym = str.toCharArray.foldLeft("Acronym: ") { (t, c) =>
    t + (if (c.isUpperCase) c.toString else "")
  }
}

implicit def str2MyRichString(str: String) = new MyRichString(str)

def main(args: Array[String]) {
  println("HyperText Transfer Protocol".acronym)
}
```

DEMO

JAYWAY

Structural types

:: type safe way of ‘Duck typing’

Ruby

```
class Duck
  def quack
    'Quack!'
  end
end

class Frog
  def quack
    'Quaaack Quaaack!'
  end
end

def doQuack(duck)
  duck.quack     # Lets hope it works!
end
```

Scala

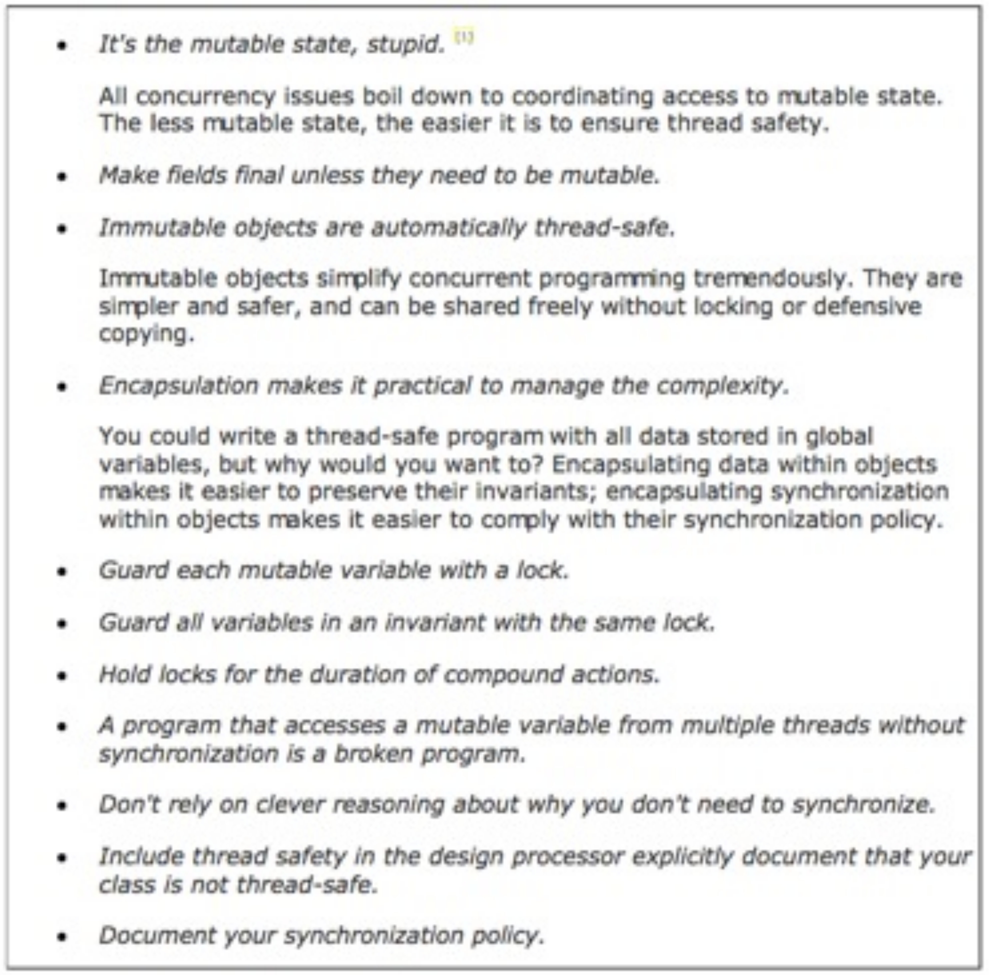
```
class Duck {
    def quack = "Quack!"
  }

  class Frog {
    def quack = "Quaaack Quaaack!"
  }

  def doQuack(duck: { def quack : String }) = {
    duck.quack // compiler checks that it will work
  }
```

DEMO

JAYWAY

Actors - why?

- From Java Concurrency in Practice:
  - "It's the mutable state, stupid."
  - All concurrency issues boil down to coordinating access to mutable state.The less mutable state, the easier it is to ensure thread safety.

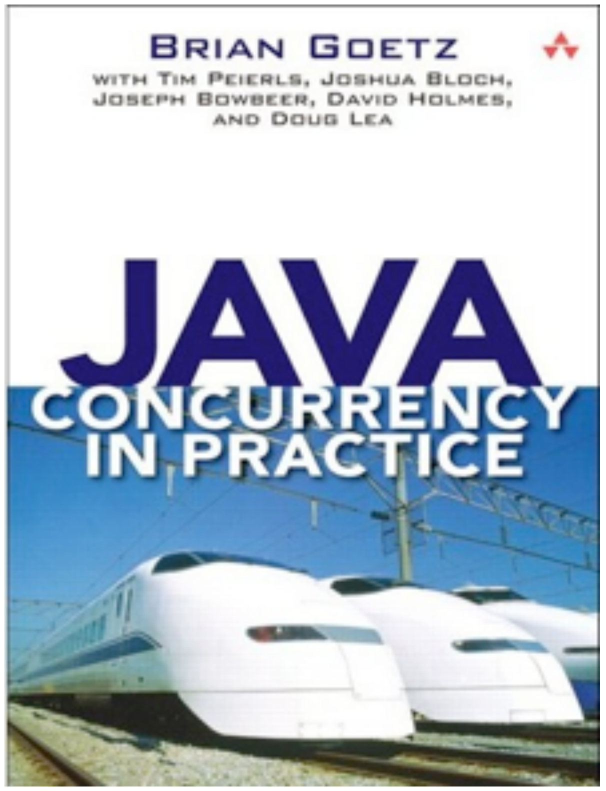


- *It's the mutable state, stupid.* [1]

  All concurrency issues boil down to coordinating access to mutable state. The less mutable state, the easier it is to ensure thread safety.
- *Make fields final unless they need to be mutable.*
- *Immutable objects are automatically thread-safe.*

  Immutable objects simplify concurrent programming tremendously. They are simpler and safer, and can be shared freely without locking or defensive copying.
- *Encapsulation makes it practical to manage the complexity.*

  You could write a thread-safe program with all data stored in global variables, but why would you want to? Encapsulating data within objects makes it easier to preserve their invariants; encapsulating synchronization within objects makes it easier to comply with their synchronization policy.
- *Guard each mutable variable with a lock.*
- *Guard all variables in an invariant with the same lock.*
- *Hold locks for the duration of compound actions.*
- *A program that accesses a mutable variable from multiple threads without synchronization is a broken program.*
- *Don't rely on clever reasoning about why you don't need to synchronize.*
- *Include thread safety in the design processor explicitly document that your class is not thread-safe.*
- *Document your synchronization policy.*


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA
CONCURRENCY
IN PRACTICE

Actors

- No shared state
- lightweight processes
- asynchronous message passing concurrency
- buffer messages in a mailbox
- receive messages with pattern matching

Actors - example

```
class Simple extends Actor {
  def act = {
    loop {
      react {
        case "ping" => println("pong")
        case _ => println("something else")
      }
    }
  }
}

val simple = new Simple()
simple.start

simple ! “ping”
simple ! “hello”
```

DEMO

JAYWAY

Testing

- JUnit
- Specs
  - Behaviour Driven Design framework
  - DSL for specifications
- ScalaCheck
  - based on property specifications and automatic test data generation
- Scalatest
  - flexible, support BBD style specifications and ‘ordinary’ JUnit style
  - choose your favourite: Spec, WordSpec, FlatSpec, FeatureSpec

JAYWAY

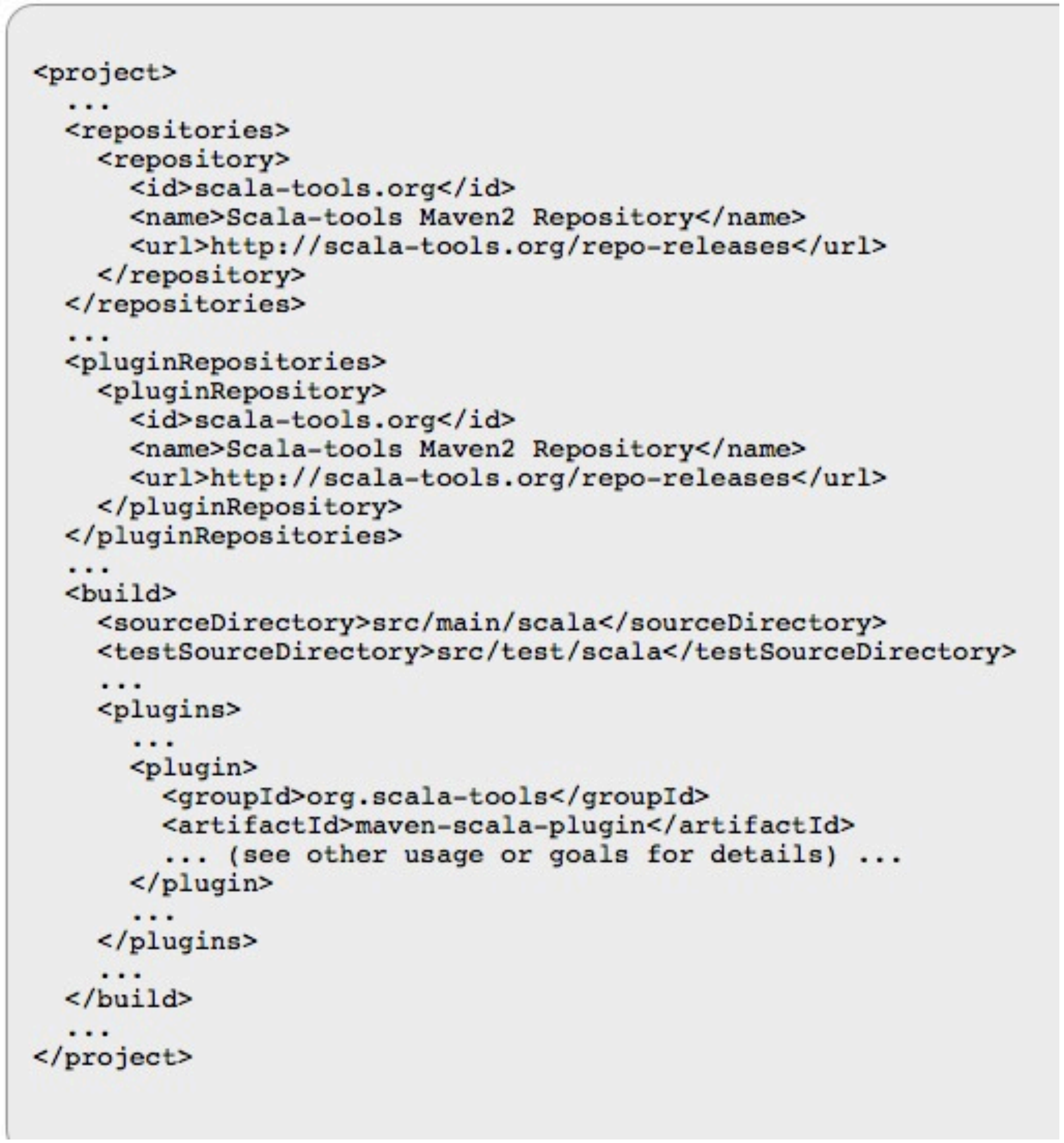
Testing with Scalatest

```scala
class StackSpec extends Spec with ShouldMatchers {
  describe("A Stack") {
    describe("(when empty)") {
      val stack = new Stack[Int]

      it("should be empty") {
        stack should be ('empty)
      }

      it("should complain when popped") {
        evaluating { stack.pop() } should produce [NoSuchElementException]
      }
    }

    describe("(when filled)") {
      it("should pop values in last-in-first-out order") {
        stack.push(1)
        stack.push(2)
        assert(stack.pop() === 2)
        assert(stack.pop() === 1)
      }
    }
  }
}
```

DEMO

Building

- Maven 2 & 3
  - maven scala plugin
- Gradle
  - Groovy DSL
- Buildr
  - Ruby DSL
- SBT
  - Scala DSL



```
<project>
  ...
  <repositories>
    <repository>
      <id>scala-tools.org</id>
      <name>Scala-tools Maven2 Repository</name>
      <url>http://scala-tools.org/repo-releases</url>
    </repository>
  </repositories>
  ...
  <pluginRepositories>
    <pluginRepository>
      <id>scala-tools.org</id>
      <name>Scala-tools Maven2 Repository</name>
      <url>http://scala-tools.org/repo-releases</url>
    </pluginRepository>
  </pluginRepositories>
  ...
  <build>
    <sourceDirectory>src/main/scala</sourceDirectory>
    <testSourceDirectory>src/test/scala</testSourceDirectory>
    ...
    <plugins>
      ...
      <plugin>
        <groupId>org.scala-tools</groupId>
        <artifactId>maven-scala-plugin</artifactId>
        ... (see other usage or goals for details) ...
      </plugin>
      ...
    </plugins>
    ...
  </build>
  ...
</project>
```

Tools & Frameworks

eclipse HELIOS

IntelliJ IDEA

NetBeans

Lift WEBFRAMEWORK

akka

configgy

scalax

ostrich

squeryl

scalaz

gizzard

scala-query

scalate

specs

scalatest

kestrel

scalatra

scala-json

sbt

play

... and of course everything ever written in Java!

JAYWAY

Is Scala too complex?

Which one is simpler ?

http://lamp.epfl.ch/~odersky/blogs/isscalacomplex.html

JAYWAY

Scala in the wild

foursquare

LinkedIn

GridGain
COMPUTE · DATA · CLOUD

guardian.co.uk

twitter

SONY PICTURES imageworks

Thatcham

JAYWAY

Introducing Scala in your company

- History: Going from C++ to Java
  - rewrite existing applications
  - new libraries, new tools
  - new deployment mechanism
- Going from Java to Scala
  - replace components
  - reuse libraries, reuse tools
  - same deployment mechanism
  - Don’t need to abandon Java !

JAYWAY

Resources

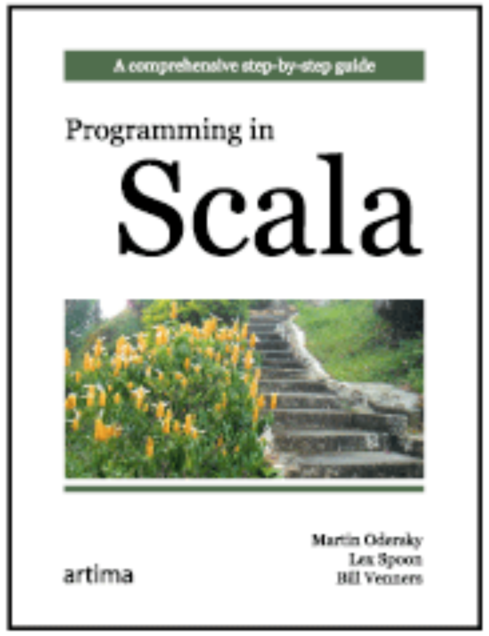
Research

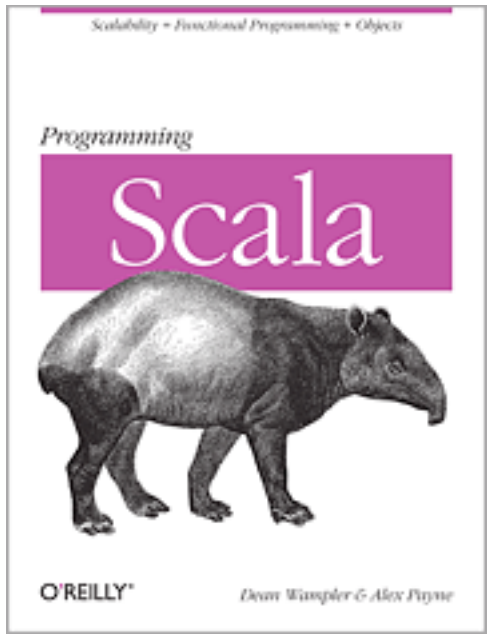
Scala opens new frontiers in programming language research. Find out about the theory and the practice behind the Scala language.

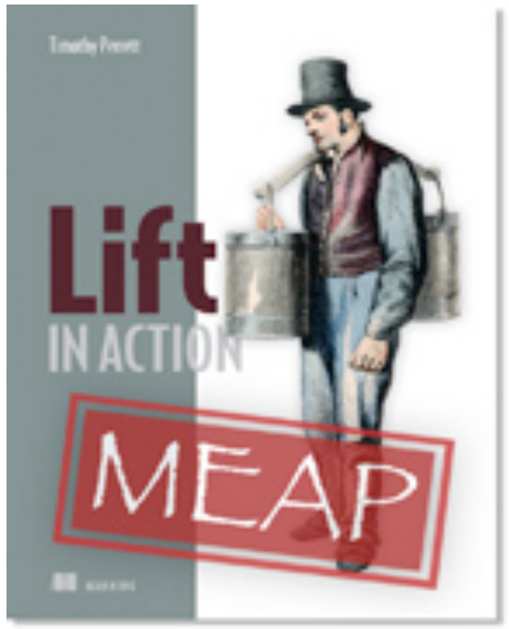
Introducing Scala

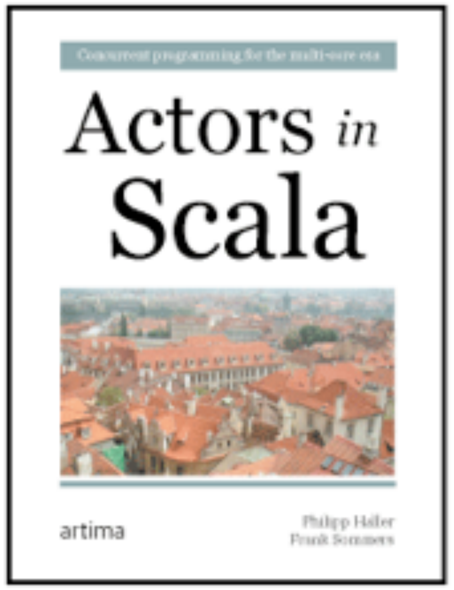
Scala is a general purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It smoothly integrates features of object-oriented and functional languages, enabling Java and other programmers to be more productive. Code sizes are typically reduced by a factor of two to three when compared to an equivalent Java application. Read more

- Scala: http://www.scala-lang.org/
- Style Guide: http://davetron5000.github.com/scala-style/ScalaStyleGuide.pdf
- Lots of books




Scalability • Functional Programming • Objects
Programming
Scala
O'REILLY
Dean Wampler & Alex Payne





Conclusion

“I'm very impressed with it! I can honestly say if someone had shown me the Programming in Scala book by by Martin Odersky, Lex Spoon & Bill Venners back in 2003 I'd probably have never created Groovy.”

James Strachan, Groovy

James Gosling

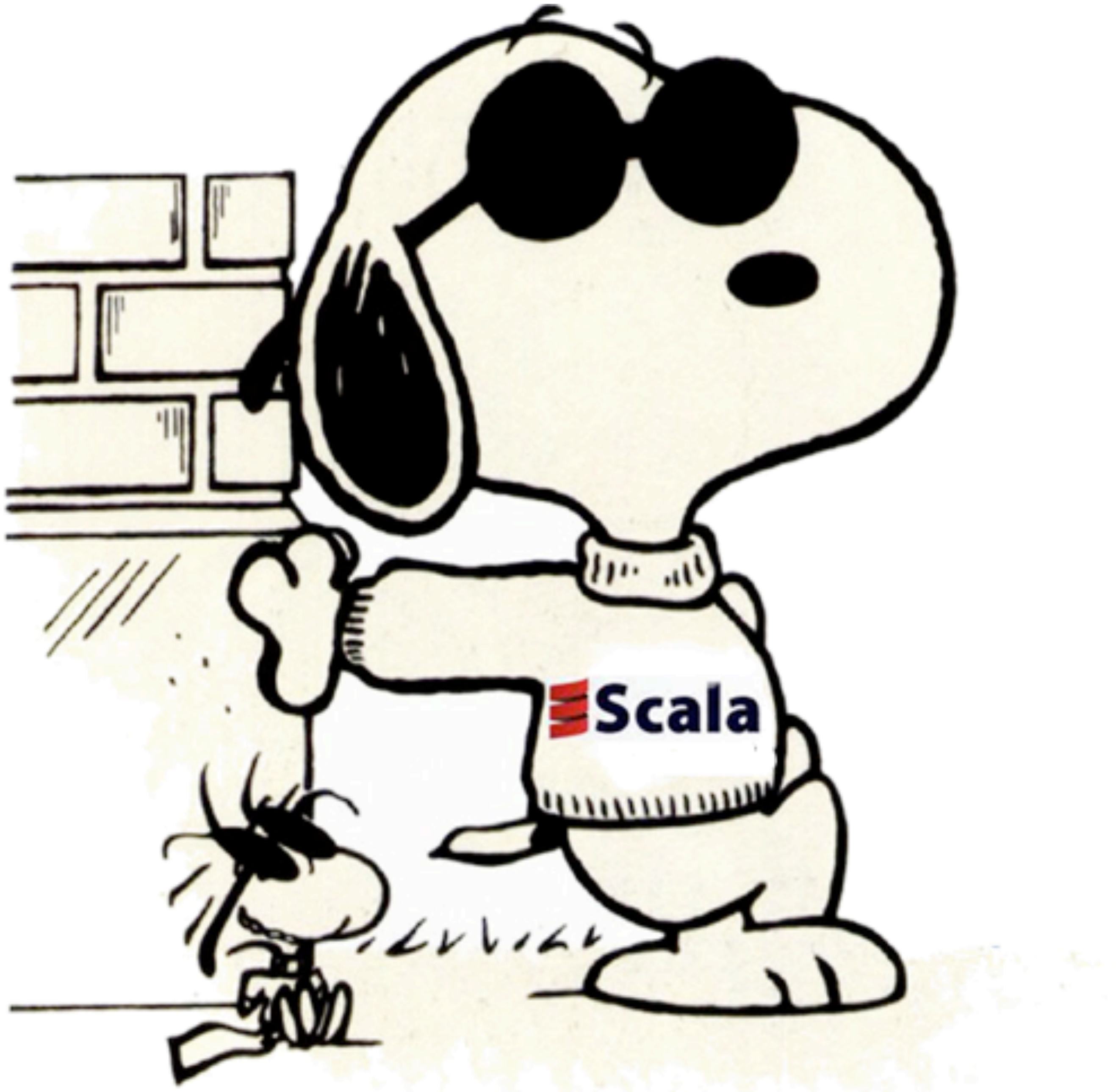
James Gosling on other JVM languages:
“I'm a big fan. I can't say I use them a lot. You know, the Java design was not that I thought it was the perfect language. The whole design concept there was about being comfortable with existing C and C++ developers and seeing bits and pieces they felt cool with.”

“Scala, it must be stated, is the current heir apparent to the Java throne. No other language on the JVM seems as capable of being a ‘replacement for Java’ as Scala, and the momentum behind Scala is now unquestionable.“

Charles Nutter, JRuby

JAYWAY

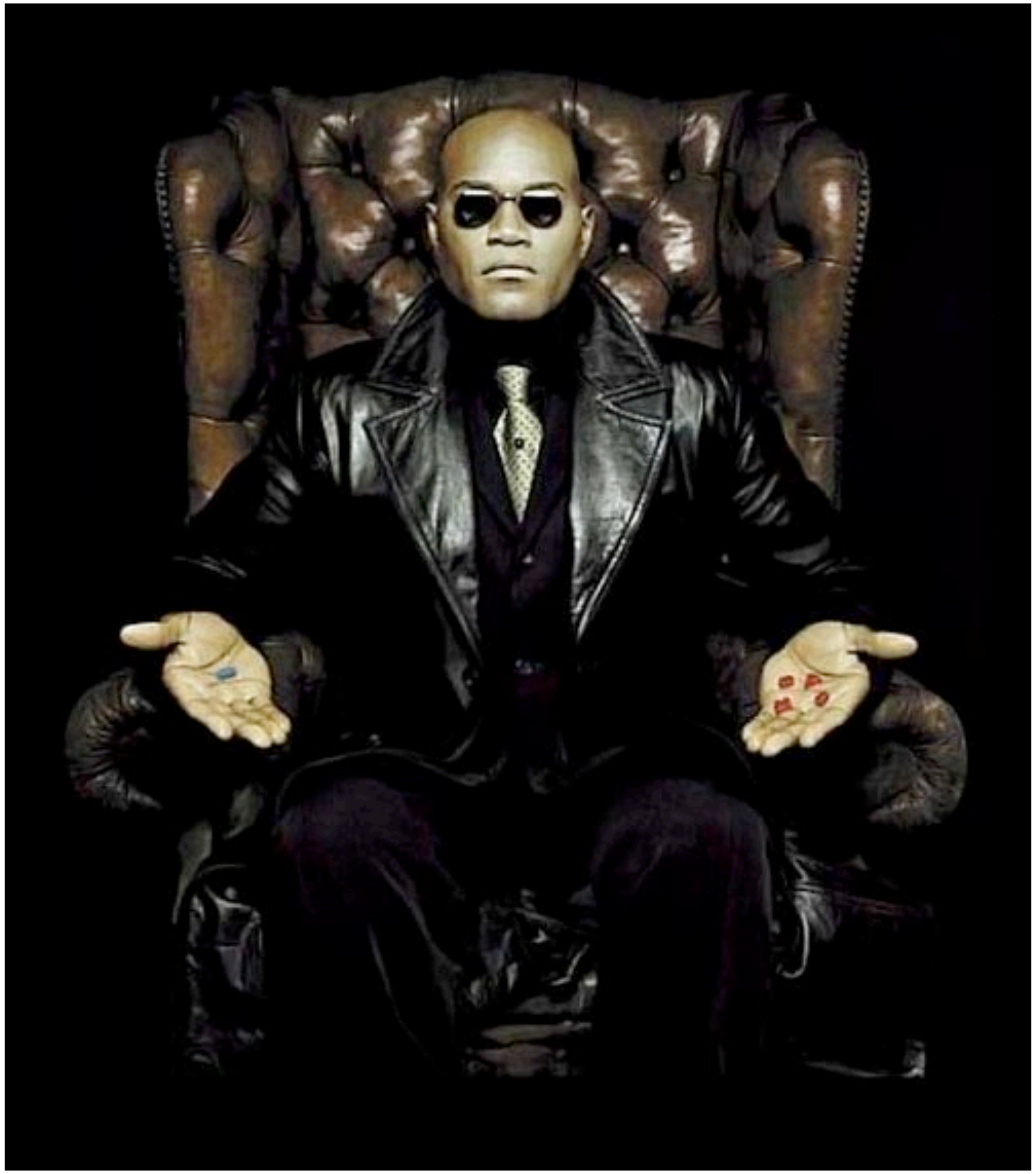
Scala is really, really cool!

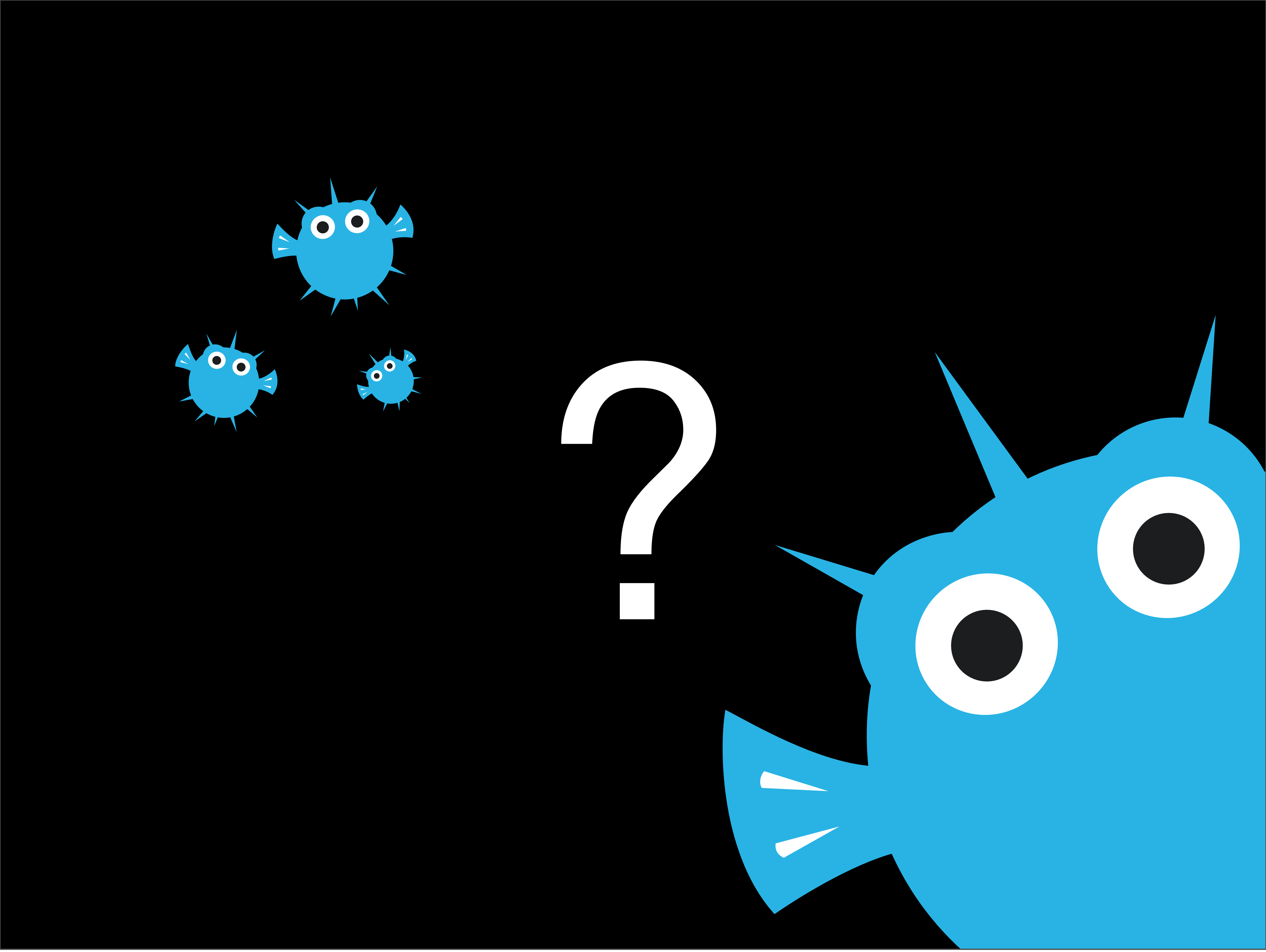


It's easy to start coding Scala, but it's hard to stop!

JAYWAY

How about you - red pill or blue pill?





Building - sbt

- cross compiling
- configuration, customization, and extension in Scala
- continuous compilation and testing
- Multiple project/subproject support
- Dependency management support
- Scala REPL with project classes and dependencies on the classpath
- supports testing with ScalaCheck, specs, scalatest, jUnit

DEMO

XML

:: ‘natural’ part of Scala, XML literals

:: partly library, some build in syntax support

:: extract sub-elements with ‘\’ and ‘\\’ (XPath uses ‘/’ and ‘//’)

```
case class Person(name: String, age: Int) {
  def toXml = <person>
      <name>{name}</name>
      <age>{age}</age>
      </person>
}
val persons = List(Person("Bob", 22), Person("Tom", 31), Person("Sally", 29))
val xml: Node = <employees>{for(person <- persons) yield person.toXml}</employees>
val names = (xml \\ "name").map(elem => elem.text)
// save
XML.saveFull("/Users/michaelkober/employees.xml", xml, "UTF-8", true, null)
// load
val xml2 = XML.loadFile("/Users/michaelkober/employees.xml")
```

JAYWAY

Recursion

```
// recursion
def factorial(i: Long) : Long = i match {
  case 1L => 1L
  case _ => i * factorial(i-1)
}

 // tail call recursion
def factorialTail(i: Long) : Long = {
  def fact(i: Long, accumulator: Long) : Long = i match {
    case 1L => accumulator
    case _ => fact(i-1, i * accumulator)
  }
  fact(i,1L)
}
```

JAYWAY

Recursion

- scalac can do tail recursion optimizations
- tail call recursion
  - can easily be optimized by conversion into loop
  - method that call itself must be private or final, defined in an object, or nested
  - @tailrec will trigger error if annotated method can’t be optimized