Opening Your Mind: **Scala – The Next Java!?**

Insights into the awesome Scala programming language

Version: 1.2.0 (2010-07-08)

.consulting .solutions .partnership

Table of Contents

We will encourage you to develop
the three great virtues of a programmer:
laziness, impatience, and hubris.
— *Larry Wall*

A language that doesn‘t affect the way you think about programming, is not worth knowing.

— *Alan Perlis*

Part I: Motivation

- Motivation: Some Quotes
- Sneak Preview: Hello World
- 4x2 Primary Aspects of Scala
- Question: Why Scala at all?
- Question: I'm forced into Java anyway, so why Scala?

Motivation: Some Quotes

- "If I were to pick a language to use today other than Java, it would be Scala."
  — ***James Gosling***, creator of **Java**

- "I can honestly say if someone had shown me the 'Programming Scala' book […] back in 2003 I'd probably have never created Groovy."
  — ***James Strachan***, creator of **Groovy**

Sneak Preview: Hello World

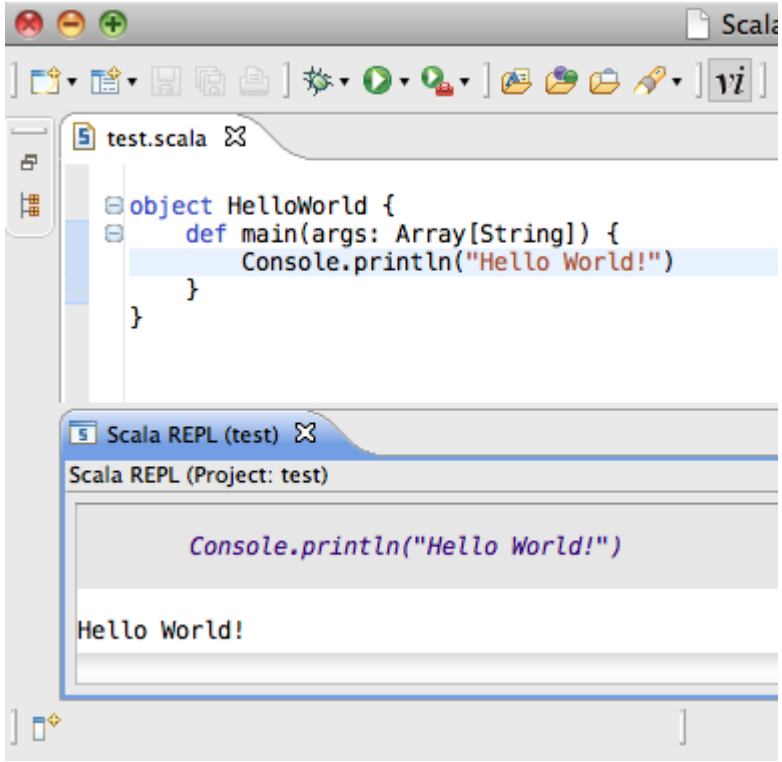
- **Scala CLI:**

```
$ scala
This is a Scala shell.
Type in expressions to have them evaluated.
Type :help for more information.

scala> object HelloWorld {
     |     def main(args: Array[String]) {
     |         Console.println("Hello World!")
     |     }
     | }
defined module HelloWorld

scala> HelloWorld.main(Array())
Hello World!
unnamed0: Unit = ()
scala> :q
```

- **Scala IDE:**



'Hello World!' :17 errors, 31 warnings

4x2 Primary Aspects of Scala

- **Scalability & Type Safety:**
  Scala („scalable language“) is an extensible, general purpose programming language designed to express common programming patterns in a very concise, elegant, and fully type-safe and type-inferred way.

- **Expressiveness & Productivity:**
  Scala is very expressive, its source code sizes are typically reduced by a factor of 2-3 when compared to *Java*, and hence *Scala* dramatically boosts a programmer’s productivity.

- **OO/FP Concepts & Integration**
  Scala smoothly integrates language features of *Object-Orientation* (OO) and *Functional Programming* (FP) to allow programmers to leverage from concepts of both successful paradigms at the same time.

- **Interoperability & Ecosystem:**
  Scala is compiled into *Java* byte-code, executed on the *JVM*, able to fully bi-directionally interoperate with regular *Java* code and hence fully leverages from the extremely large software ecosystem of reusable *Java* libraries and frameworks.

> The key to successful leadership today is influence, not authority.
> — *Kenneth Blanchard*

Question: Why Scala at all?

- **What is the problem?**
  Java as an ecosystem is powerful, but Java as language is rather weak.
- **Why is the problem a problem?**
  Java causes lots of boilerplate code, requires lots of run-time tricks for advanced features.
- **What is the solution?**
  Leverage from a scalable and more flexible language like Scala.
- **Why is the solution a solution?**
  Scala is very expressive, causes mostly no boilerplate code, is stronger typed than Java and mostly all of its features are implemented under compile-time.

Java is the answer,
but only if you phrase
the question very carefully.

Question: I'm forced into Java anyway, so why Scala?

- **It opens your mind as a programmer**
  (think about: Scala's seamless OOP & FP integration)
- **It opens your mind as an architect**
  (think about: Scala Component Oriented Programming features like Traits)
- **It already can be used in projects for lots of border topics**
  like application testing, build-time parser/generators, etc.
  (think about: better tooling)
- **You do not loose anything from your existing Java ecosystem**
  …and instead you just gain a lot more.
  (think about: Scala is Java bytecode and fully interoperates with plain Java)

Goodbye Java,
Welcome Scala!

The limits of your language
are the limits of your world.
— *L. Wittgenstein*

Part II: Java vs. Scala Tour

- Java vs. Scala Tour 1/5: define Person class
- Java vs. Scala Tour 2/5: “select and sort Persons“
- Java vs. Scala Tour 3/5: create list of Person objects
- Java vs. Scala Tour 4/5: convert Persons into XML DOM
- Java vs. Scala Tour 5/5: find oldest Person in XML DOM

Java vs. Scala Tour (1/5)

```java
//  Java: define the Person class

public class Person {
    private final String  firstName;
    private final String  lastName;
    private final Integer age;
    public Person(
        String  firstName,
        String  lastName,
        Integer age       ) {
        this.firstName = firstName;
        this.lastName  = lastName;
        this.age       = age;
    }
    public String getFirstName() {
        return firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public Integer getAge() {
        return age;
    }
    public Boolean isValid() {
        return age > 18;
    }
}
```

```scala
//  Scala: define the Person class

class Person (
    val firstName: String,
    val lastName:  String,
    val age:       Int    ) {
    def isValid = age > 18
}
```

Programming is similar to the game of Golf. The point is not getting the ball in the hole, but how many strokes it takes.

— Harlan D. Mills

Java vs. Scala Tour (2/5)

```java
//  Java: select valid Persons,
//  sort them by increasing age,
//  return their first-name

public static List<String>
validByAge(List<Person> in) {
   List<Person> valid =
       new ArrayList<Person>();
   for (Person p: in)
       if (p.isValid())
           valid.add(p);
   Collections.sort(valid,
       new Comparator<Person>() {
           public int compare(
               Person a, Person b) {
               return a.getAge() –
                      b.getAge();
           }
       }
   );
   List<String> ret =
       new ArrayList<String>();
       for (Person p: valid)
           ret.add(p.getFirstName());
   return ret;
}
```

```scala
//  Scala: select valid Persons,
//  sort them by increasing age,
//  return their first-name

def validByAge(in: List[Person]) =
    in filter   { _.isValid }
       sortWith { _.age < _.age }
       map      { _.firstName }
```

About Scala syntax complexity:
"Yes, but are you sure it's the language and not just you getting older?"
— *unknown*

Java vs. Scala Tour (3/5)

```java
//  Java: create list of Persons
//  and select sorted valid ones

List<Person> x = new ArrayList<Person>();
x.add(new Person("John",  "Doe",    32));
x.add(new Person("Frank", "Doe",    17));
x.add(new Person("Sue",   "Sample", 19));


List<String> people = validByAge(x)
```

```scala
//  Scala: create list of Persons
//  and select sorted valid ones

val x = List(
    new Person("John",  "Doe",    32),
    new Person("Frank", "Doe",    17),
    new Person("Sue",   "Sample", 19)
)

val people = validByAge(x)
```

A programming language is low level when its programs require attention to the irrelevant.

— Alan Perlis

Java vs. Scala Tour (4/5)

```java
//  Java: convert Persons into XML DOM

DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder loader =
    factory.newDocumentBuilder();
Document xml = loader.newDocument();
Element root = xml.createElement("persons");
xml.appendChild(root);
for (Person p: x) {
    Element pe = xml.createElement("person");
    Element fn = xml.createElement("first");
    fn.appendChild(
        xml.createTextNode(p.getFirstName()));
    pe.appendChild(fn);
    Element ln = xml.createElement("last");
    ln.appendChild(
        xml.createTextNode(p.getLastName()));
    pe.appendChild(ln);
    Element age = xml.createElement("age");
    age.appendChild(
        xml.createTextNode(p.getAge()));
    pe.appendChild(age);
    root.appendChild(pe);
}
```

```scala
//  Scala: convert Persons into XML DOM

val xml =
  <persons> {
    x map {
      <person>
        <first>{ _.firstName }</first>
        <last> { _.lastName  }</last>
        <age>  { _.age       }</age>
      </person>
    }
  } </persons>
```

XML is like violence –
if it doesn't solve your problems,
you are not using enough of it.
— http://nokogiri.org

Java vs. Scala Tour (5/5)

```java
//  Java: find oldest Person's
//  first-name directly in XML

XPath xp =
    XPathFactory.newInstance().newXPath();
NodeList nl = (NodeList)xpath.evaluate(
    "person", xml, XPathConstants.NODESET);
List<Person> p = new List<Person>();
for (int i = 0; i < nl.getLength(); i++) {
    Node n = nl.item(i);
    String fn = (String)xpath.evaluate(
        "first/text()", n,
        XPathConstants.STRING);
    int age = (Int)xpath.evaluate(
        "age/text()", n,
        XPathConstants.INT);
    p.add(new Person(fn, null, age));
}
Collections.sort(p,
    new Comparator<Person>() {
        public int compare(Person a,
                           Person b) {
            return a.getAge() - b.getAge();
} } );
String oldest = p[0].getFirstName();
```

```scala
//  Scala: find oldest Person's
//  first-name directly in XML

val oldest =
    (xml \ "person")
    map { n =>
        ( (n \ "first").text,
          (n \ "age"  ).text.toInt )
    }
    sortWith { _._2 < _._2 }
    head map { _._1        }
```

A good programming language is a conceptual universe for thinking about programming.

— A. Perlis

It's really hard to design products by focus groups. A lot of times, people don't know what they want until you show it to them.
— *Steve Jobs*

Part III: More About Scala

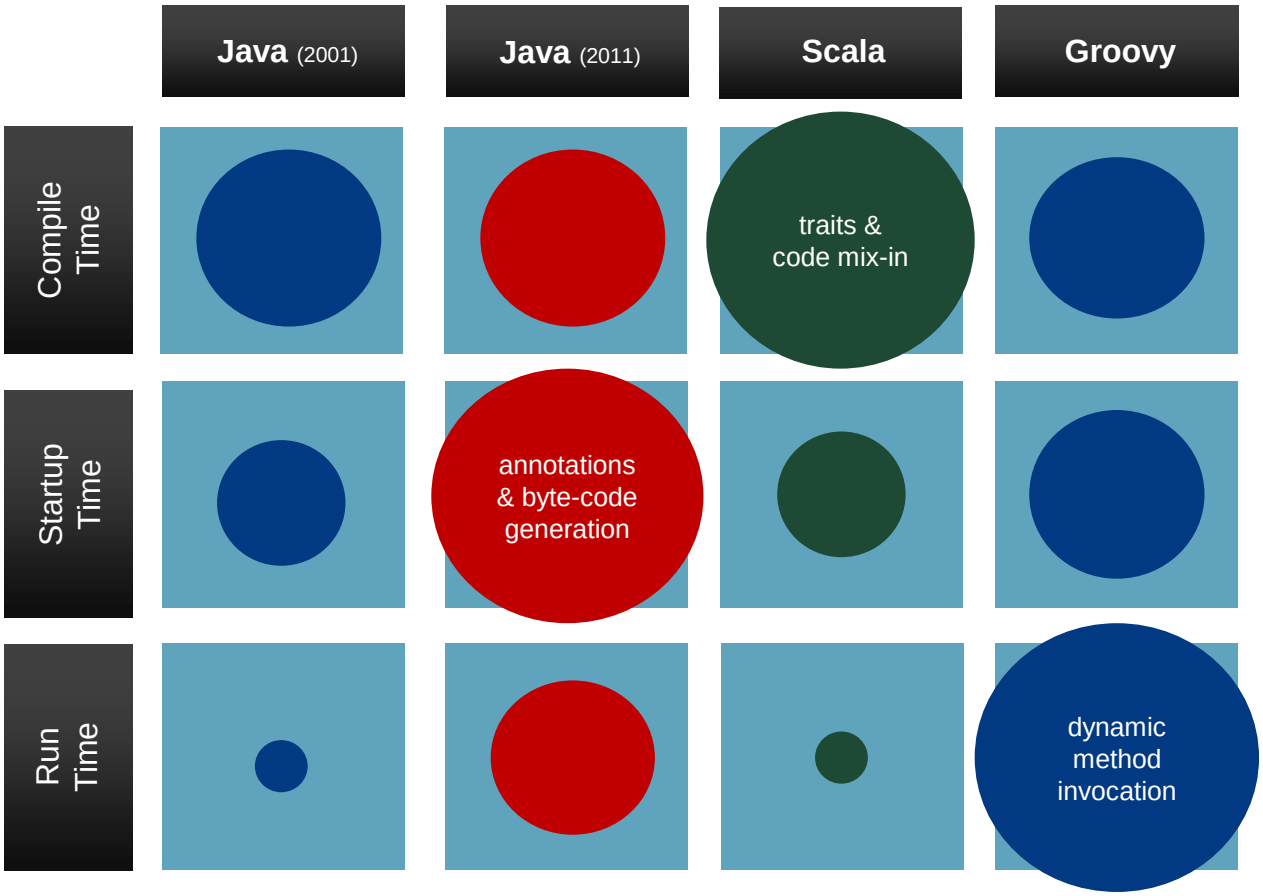
- Short History of Scala
- Efforts & Focus Domains
- Scala: Best of Breed
- Scala: Unique Features
- But… Business vs. Technical Programming

Short History of Scala

- 1995:
  Java created by *James Gosling* at Sun Microsystems
- 2000/2004:
  Generic Java (GJ)'s Java compiler (for Java 1.2) and Generics (for Java 5) contributed by *Martin Odersky*, *Philip Wadler*, *Gilad Bracha* and *David Stoutamire*.
- 2004:
  Scala created by *Martin Odersky* at LAMP/IC/EPFL

If I have seen farther than others,
it is because I was standing
on the shoulder of giants.
— *Isaac Newton*

Efforts & Focus Domains: Java vs. Scala vs. Groovy



…because Scala‘s philosophy is that everything which can be done at compile-time, also should be done there

…because Java language is too weak and hence one today has to trick with lots of on-the-fly code generation – effectively leading to a „Java on steroids“

…because that‘s simply Groovy‘s nature and intentional idea behind MOP, etc

Back to the roots…

Scala: Best of Breed

- From **Smalltalk/Ruby**:
  Unified Object Model
- From **C/C++/Java/C#:**
  Syntax Style
- From **Algol/Simula/Beta:**
  Universal Construct Nesting
- From **Eiffel**:
  Uniform Access Principle
- From **SML/OCaml/F#**:
  Functional Programming Aspects
- From **ML/Haskell**:
  Higher-Order Functions
- From **Haskell**:
  Implicit Parameters
- From **Erlang**:
  Actor Multithreading Model
- From **Smalltalk/Iswim**:
  Infix Operator Functions,
  Function Literals as Parameters
- From **Smalltalk/LISP**:
  Flexible Syntax for DSLs
- From **LISP/C++:**
  Extensibility, Scalability
- From **Ruby/Smalltalk/**
  Python/OCaml/F#:
  OOP+FP Integration

> There are only two kinds of programming languages:
> those people always bitch about and those nobody uses.
>
> — *B. Stroustrup*

Scala: Unique Contributions

- **Seamless OOP+FP Integration**:
  - Everything is an object
  - Everything is an expression
- **Sophisticated Type System:**
  - Powerful Type Inferencing
  - Abstract Types
  - Sealed Classes
  - Type Covariance
  - Type Contravariance
  - View Bounds
  - Structural Types
  - Existential Types
- **Powerful Pattern Matching:**
  - Representation-independent
  - Based on Extractor methods
- **Component Oriented Programming (COP) with Traits:**
  - Define interfaces with Traits
  - Mixin behaviour with Traits
  - Assemble at class definition time
  - Assemble at object instanciation time

Always remember you're unique,
just like everyone else...
— *unknown*

But… Business vs. Technical Programming

- **Scala for Business Aspects:**
  - Scala allows very concise Business Programming
  - But most of its advanced features can and should(!) be intentionally ignored during Business Programming.

Feature	Java	Scala
Fluent API	+	+ +
Combinators	+	+ +
Control Structures	✖	+ +
Internal DSL	✖	+
Type-Safe Calls	+	+ +

- **Scala for Technical Aspects:**
  - Many of Scala's features especially target the technical implementation of elegant and powerful APIs in libraries and frameworks.
  - Nevertheless, this requires mastering a certain degree of language complexity first.

> Simple things should be simple, complex things should be possible.
> — *Bo Leuf & Ward Cunningham*

If you don't think carefully,
you might believe that programming is just
typing statements in a programming language.
— *W. Cunningham*

Part IV: Unique Scala Features

- Unique Scala Features 1/3
- Unique Scala Features 2/3
- Unique Scala Features 3/3

Unique Scala Features 1/3

- **Non-Interpolated String Literals**
```
"""(?s)([^\]]+|\][^\]])*""".r
```

- **Call-by-Name Parameters**
```
def loop(e: => Boolean, s: => Unit) =
    while (e) { s }
```

- **Default Parameters**
```
def f(x: Int = 0, y: Int = 0): …
```

- **Named Parameters**
```
def f(x: Int, y: Int): …
f(y = 7, x = 42)
```

- **Implicit Parameters**
```
def f(x: Int)(implicit x: Foo): …
implicit val foo = new Foo;
f(42)  ▶  f(42)(foo)
```

- **Implicit Functions**
```
implicit def i2f(i: Int): Foo = Foo(i)
def f(foo: Foo) = …
42.onFoo() ▶ i2f(42).onFoo()
f(42)      ▶ f(i2f(42))
```

- **Method Pretending**
```
x.foo(42, "bar")  ▶
x.applyDynamic("foo", Seq("42", "bar"))
```

- **Mutable/Immutable Variables**
```
var x = 7
val y = 42
```

- **Closures**
```
val prefix = "Mr./Mrs. "
val contacts = persons map {
    (p) => prefix + p.lastName
}
```

- **Parameter Placeholders**
```
_.toUpperCase + _
▶ (a1, a2) => a1.toUpperCase + a2
```

- **Partially Applied Functions (Currying)**
```
def modN(n: Int)(x: Int) =
  (x % n) == 0
modN(2) ▶  (x: Int) => modN(2, x)
```

> Make everything as simple as possible, but not simpler.
> — *Albert Einstein*

Unique Scala Features 2/3

- **Case Classes**

```
case class Person(
  name: String,
  age: Int
)
val person = Person("foo", 42)
println(person.name)
```

- **Pattern Matching**

```
node match {
   case Circle(p, r) if (r < 1) =>
       Ellipse(p, r, r)
   case Circle(Pos(x, y), r) =>
       Rect(Pos(x-r, y-r), 2*r, 2*r) …
```

- **Tuples**

```
def xml2pos(p: String): (Int, Int) =
  ( (p \ "x").text.toInt,
    (p \ "y").text.toInt  )
val (x, y) = xml2pos(
  "<pos><x>7</x><y>42</y></pos>")
(42, "foo") ▶ Tuple2[Int, String]
```

- **Lazy Evaluation**

```
lazy val a = b + 1
lazy val b = 1
```

- **Structural Types**

```
type Observer = { def notify() }
observers foreach { _.notify() }
```

- **Covariant & Contravariant Types**

```
trait Function1[-P, +R] {
  def apply(p: P): R }
```

"A function *A* is a sub-type of another function *B* if the *parameter* type of *A* is a *super-type* of the parameter type of *B* while the *return* type of *A* is a *sub-type* of the return type of *B*."

- **Traits**

```
trait Similarity {
   def isSimilar(x: Any): Boolean
   def isNotSimilar(x: Any): Boolean =
       !isSimilar(x)
}
```

- **Sequence Comprehensions**

```
for (i <- 0 until n;
     j <- 0 to i if i % 2 == 0)
    yield Pair(i, j)
```

> Before software can be reusable it first has to be usable.
>
> — *Ralph Johnson*

Unique Scala Features 3/3

- **Function Literals**
```
val inc = (i: Int) => i + 1
▶ def inc (i: Int) = i +1
```

- **Custom Controls**
```
def using [A, B <: {def close(): Unit}]
  (res: B)(f: B => A): A =
    try { f(res) }
    finally { res.close() }
▶ using (resource) { action-on-resource }
```

- **Custom Operators**
```
def ~~(re: String): Option[Match] =
    new Regex(re).findFirstMatchIn(this)
implicit def elvisOperator[A](a: => A) =
    new { def ?:[O >: A](o: O) =
          if (o == null) a else o }
```

- **Finer-Grained Access Control**
```
private[this] val foo = 42
```

- **Import Wildcards & Renames**
```
import scala.xml._
import java.util.{Date => UDate}
import java.sql. {Date => SDate}
```

- **Null-Safety**
```
def lookup(key: String): Option[Int] = {
  val value = db(key)
  if (value != null) Some(value)
  else               None
}
lookup("foo").getOrElse(42)
```

- **Uniform Object Model**
```
42.toString
```

- **Uniform Access Principle**
```
val foo: String = "foo"
override def foo(): String = "foo"
```

- **Operators vs Methods**
```
a foreach b ▶ a.foreach(b)
a + b ▶ a.+(b)
```

Motto for a research laboratory:
"What we work on today,
others will first think of tomorrow."
— *Alan Perlis*

Programming languages should be designed
not by piling feature on top of feature,
but by removing the weaknesses and restrictions
that make additional features appear necessary.

— *unknown*

Part V: Summary

- Scala is…
- Further Cool Stuff
- More About Scala

Scala is…

- **consistent** — everything is an object, everything can be nested
- **flexible** — every operator is a method call (and vice versa)
- **scalable** — libraries can implement new language constructs
- **type-safe** — fully statically typed at compile-time
- **light-weight** — usable like a dynamically-typed scripting language
- **expressive** — lots of semantics expressable with very less syntax
- **concise** — mostly free of „syntactic sugar“ in syntax
- **simple(r)** — no special syntax for arrays, no specialized "switch", etc
- **object-oriented** — unified object system, no primitive types, no statics, etc
- **functional** — higher-order functions, pattern matching, etc
- **interoperable** — call Java from Scala and with care also vice versa
- **scriptable** — supports on-the-fly compilation and Unix she-bang
- **sophisticated** — very elegant and sophisticated collection library design
- **extensive** — leverages from existing Java ecosystem of libraries

> The most likely way for the world to be destroyed,
> most experts agree, is by accident. That's where we come in.
> We're computer professionals. We cause accidents.
> — *Nathaniel Borenstein*

Further Cool Stuff

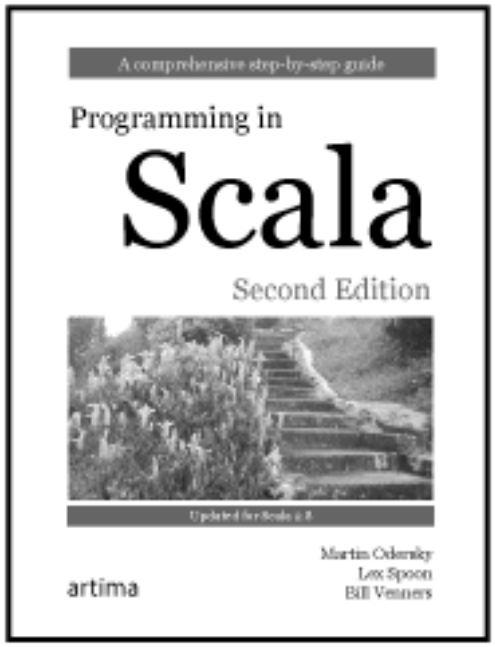
- Standard Libraries & Frameworks:
  - **Parallel Collections**
    `scala.collection.parallel.*`
  - **Multithreading Actors**
    `scala.actors.*`
  - **Parser Combinators**
    `scala.util.parsing.combinator.*`
  - **Scala Swing**
    `scala.swing.*`
- Third-Party Utility Libraries:
  - **ScalaJ-Collection** & Java
    https://github.com/scalaj/scalaj-collection
  - **Scalaz** & FP
    http://code.google.com/p/scalaz/
  - **Grizzled Scala**
    http://software.clapper.org/grizzled-scala/
  - **Parboiled** & PEG
    https://github.com/sirthias/parboiled
  - **Kiama** & Language Processing
    http://code.google.com/p/kiama/
  - **ScalaLA** & Linear Algebra
    https://github.com/scalala/Scalala
  - **ScalaSTI** & String Templating
    http://software.clapper.org/scalasti/
  - **Scalate** & String Tempating
    http://scalate.fusesource.org/
- Third-Party Network/Testing Frameworks:
  - **Akka** & Actors
    http://akka.io/
  - **Lift** & AJAX/Comet
    http://liftweb.net/
  - **Play!** & REST
    http://www.playframework.org/
  - **Specs** & BDD/TDD
    http://specs2.org/
  - **ScalaTest** & BDD/TDD
    http://www.scalatest.org/
- Third-Party Network/Database Libraries:
  - **Dispatch** & HTTP Client
    http://dispatch.databinder.net/
  - **Unfiltered** & HTTP Server
    http://unfiltered.databinder.net/
  - **ScalaQuery** & JDBC/SQL
    http://scalaquery.org/
  - **SqueryL** & JDBC/SQL
    http://squeryl.org/
  - **Querulous** & JDBC/SQL
    https://github.com/nkallen/querulous
  - **QueryDSL** Scala & JDBC/SQL
    http://www.querydsl.com/

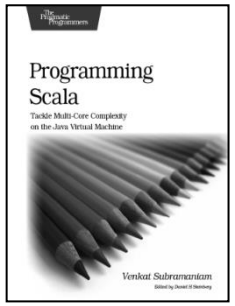
Think twice,
code once.

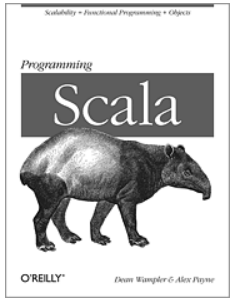
More About Scala

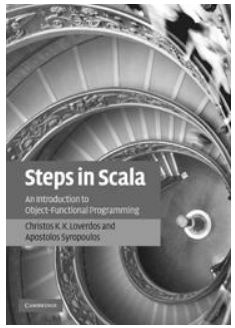
- **Scala Language:**
  http://www.scala-lang.org/
- **Scala IDE** (Plugin for Eclipse IDE)
  http://www.scala-ide.org/
- **Scala Community** (at Stackoverflow)
  http://stackoverflow.com/questions/tagged/scala
- Various Books on Scala…

> It's not the mountain we conquer, but ourselves.
> — *Sir Edmund Hillary*









Your Next Steps?!

1. **At least test-drive Scala**
   (install Eclipse with Scala plugin)
2. **Feel the joy of Scala**
   (read foreign Scala code and play with your own code)
3. **Add Scala to your toolbox**
   (know when it can and should be reasonably used)

- **Risk-Less Immediate Use Cases in Software Development Projects:**
  - Unit Testing
    (Specs or ScalaTest,
    with/without ScalaCheck)
  - DSL Parsing
    (Scala Parser Combinators)
  - Build Tooling
    (conversions, generation, etc)

> If you think Scala is just another language
> and it will be gone soon again,
> I predict: you'll be plain wrong!
>
> — *Ralf S. Engelschall*

Thank you for your attention

Ralf S. Engelschall

msg Applied Technology Research,
Principal IT Consultant, Department Manager

Phone: +49 89 96101-1913
ralf.engelschall@msg-systems.com

www.msg-systems.com

.consulting .solutions .partnership