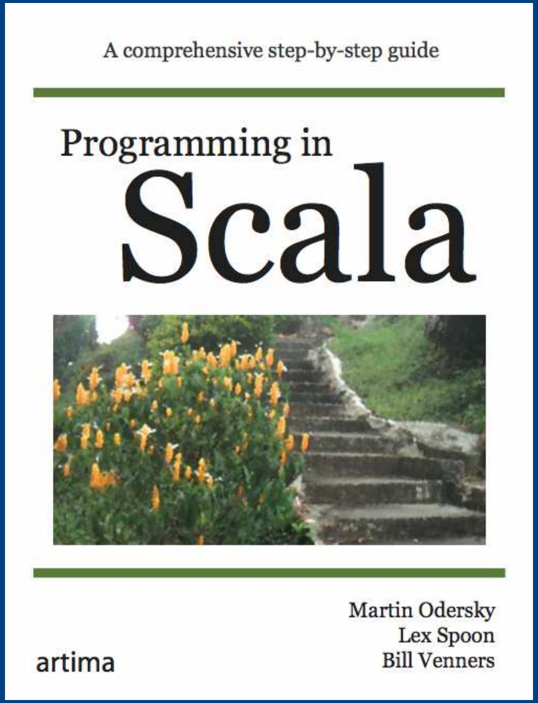
A Scala Firehose

Bill Venners
President
Artima, Inc.
www.artima.com



A comprehensive step-by-step guide

Programming in Scala

Martin Odersky
Lex Spoon
Bill Venners

artima

Enough about me:

- Run the Artima Developer website
- Existing investment in Java knowledge and code
- Wanted a more productive language for JVM
- Didn’t want to give up static typing
- Scala fit my needs
- Scala book, ScalaTest

A bit about Scala:

- Designed by Martin Odersky

Scala is:

- A Scalable language
- Object-oriented
- Functional
- Statically typed

A Scalable language

- From scripts to systems
- Grow new types
- Grow new control constructs

Design libraries that enable
clear, concise client code
that feels like native language support.

Scalable language means:

1. From scripts to systems

Ruby: puts "Hello, world!"

Java:

```
class Hello {

    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Scala

```
println("Hello, world!")
```

Scalable language means:

2. Extensible by growing new types

Java's if statement:

```
if (a > b)
    System.out.println(a);
else
    System.out.println(b);
```

Java’s ternary operator:

```
int m = (a > b) ? a : b;
System.out.println(m);
```

Scala’s if expression:

```
val m = if (a > b) a else b
println(m)
```

java.math.BigInteger:

```
if (x == BigInteger.ZERO)
  BigInteger.ONE
else
  x.multiply(factorial(x.subtract(BigInteger.ONE)))
```

scala.BigInt:

```
if (x == 0) 1 else x * factorial(x - 1)
```

Scalable language means:

3. Extensible by growing new control constructs

JUnit 3:

```
String s = "hi”;
try {
    s.charAt(-1);
    fail();
}
catch (IndexOutOfBoundsException  e) {
    // Expected, so continue
}
```

JUnit 4:

```
@Test(expected=IndexOutOfBoundsException.class)
public void testPassingANegativeToCharAt() {
    s.charAt(-1);
}
```

ScalaTest:

```
intercept(classOf[IndexOutOfBoundsException]) {
  s.charAt(-1)
}
```

How Scala scales:

1. Scala is object-oriented

“Pure” OO language:

Every value is an object;
Every operation is a method call.

1 + 2

(1).+(2)

Domain-specific languages

```
if (x == 0) 1 else x * factorial(x - 1)
```

```
x - 1
```

```
x.-(1)
```

But what about…

$1 - x$

Implicit conversions

```
implicit def int2BigInt(i: Int): BigInt = new BigInt(i)
```

```
1 - x
```

```
int2BigInt (1).-(x)
```

Why i: Int, not int i?

1. Type inference:

```
val i = 0
not
i = 0
```

2. Large type expressions:

```
val m: HashMap[String, (String, List[Char])] = …
not
final HashMap<String, Pair<String, List<Char>>> m = …
```

Java Interfaces

```
interface Ex {
    int abstractMeth(String x);
}
```

(no concrete methods)

(no fields)

Scala Traits

```
trait T {
  def abstractMeth(s: String): Int
  def concreteMeth(s: String) = s + field
  var field = "!"
}
```

Java Interface Implementation

```
Class C extends Super implements Ex
```

Scala Mixin Composition

```
class C extends Super with T
```

Classes and Objects

Java:

```
class Sample {
    private final int x;
    public final int p;
    Sample(int x, int p) {
        this.x = x;
        this.p = p;
    }
    int instMeth(int y) {
        return x + y;
    }
    static int staticMeth(int x, int y) {
        return x * y;
    }
}
```

Scala:

```
class Sample(x: Int, val p: Int) {
  def instMeth(y: Int) = x + y
}

object Sample {
  def mult(x: Int, y: Int) = x * y
}
```

Invoked:

`Sample.mult(1, 2)`

How Scala scales:

2. Scala is functional

A Java method

```
int incr(int x) {
    return x + 1;
}
```

A Scala function

```scala
def incr(x: Int) = x + 1
```

A mathematical function

$$incr(x) = x + 1$$

Anonymizing a function

```
def incr(x: Int) => x + 1
```

A function literal

```
(x: Int) => x + 1
```

Functions are first-class values

```
scala> val f = (x: Int) => x + 1
f: (Int) => Int = <function>

scala> f(1)
res0: Int = 2
```

Functions are first-class values

```
f (1)

f.apply(1)
```

Functional “attitude”

- Prefer immutable objects
- Prefer “initialize-only” variables
- Prefer methods with no side-effects

Immutable tradeoffs

- Simpler
- Can pass them around
- Inherently thread safe
- Safest hashtable keys
- Large graphs expensive to replicate
  - So maybe offer a mutable alternative

Java String

```
String s = "immutable";
String t = s.replace("im", "also im");
System.out.println(s + ", " + t);
```

```
immutable, also immutable
```

Ruby String

```
irb(main):007:0> s = 'immutable'
=> "immutable"
irb(main):008:0> s['im'] = 'quite '
=> ""
irb(main):009:0> puts s
quite mutable
```

Scala String is Java String

```
val s = "immutable"
val t = s.replace("im", "also im")
println(s + ", " + t)
```

```
immutable, also immutable
```

Java List

```
import java.util.List;

List<String> mutable = new ArrayList<String>();
mutable.add("Hi");
List<String> unmodifiable =
    Collections.unmodifiableList(mutable);
```

Scala List

```
val myList = List("Hi" , "there")
```

Scala ListBuffer and Array

```
import scala.collection.mutable.ListBuffer
val muta = new ListBuffer[String]
muta += "Hi"

val arr = Array("Hi")
```

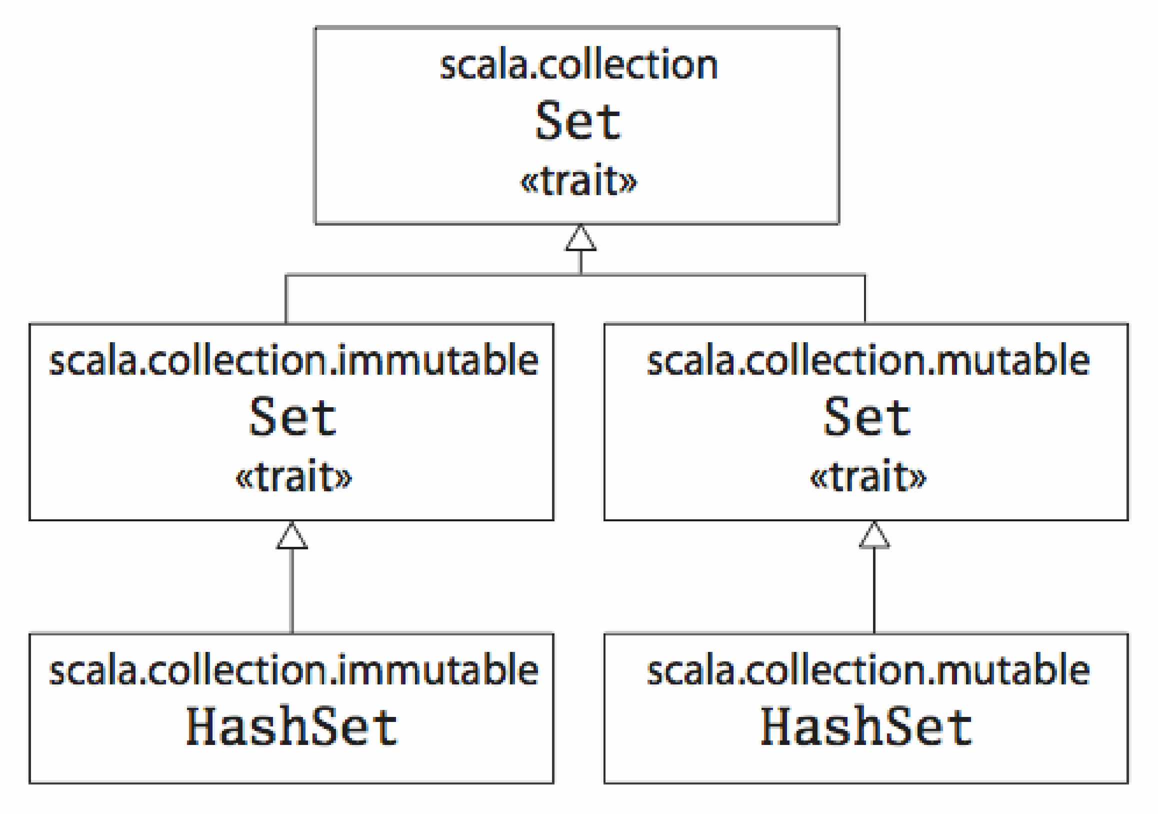
Scala Set

```
val mySet = Set("Hi", "there")
```

Scala mutable.Set

```
import scala.collection.mutable.Set
val muta = Set("Hi")
```

Scala's Set Hierarchy



Java variables

```
String s = "Hi";

final String t = "there";
```

Scala variables

```scala
val s = "Hi";

var t = "there";
```

Java idiom

```
String s = "default";
if (args.length > 0) {
    s = args[0];
}
```

Scala with Java accent

```
var s = "default"
if (args.length > 0) {
    s = args(0)
}
```

Scala idiom

```
val s =
  if (args.length > 0) args(0)
  else "default"
```

gcdLoop

```
def gcdLoop(x: Long, y: Long): Long = {
  var a = x
  var b = y
  while (a != 0) {
    val temp = a
    a = b % a
    b = temp
  }
  b
}
```

gcd

```
def gcd(x: Long, y: Long): Long =
  if (y == 0) x else gcd(y, x % y)
```

How Scala scales:

3. Scala is statically typed

Type annotations in Java

```
Map<Integer, String> myMap =
    new HashMap<Integer, String>();
```

Type annotations in Scala

```
val hm = new HashMap[Int, String]()
```

or

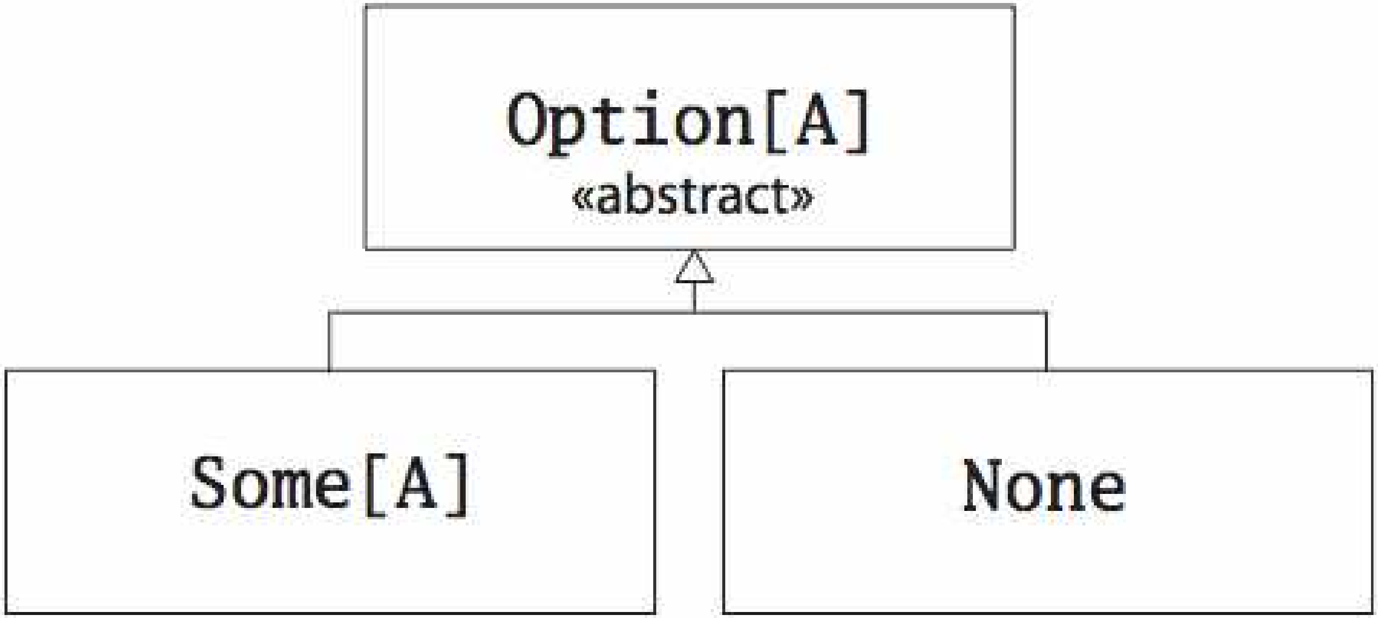
```
val m: Map[Int, String] = new HashMap()
```

javax.servlet.ServletRequest

public java.lang.String
getParameter(java.lang.String name)

Returns the value of a request parameter as a String, or null if the parameter does not exist

Some(param) or None



scala.List

```
def find (p: (A) => Boolean): Option[A]
```

Find and return the first element of the list satisfying a predicate, if any.

Using Option

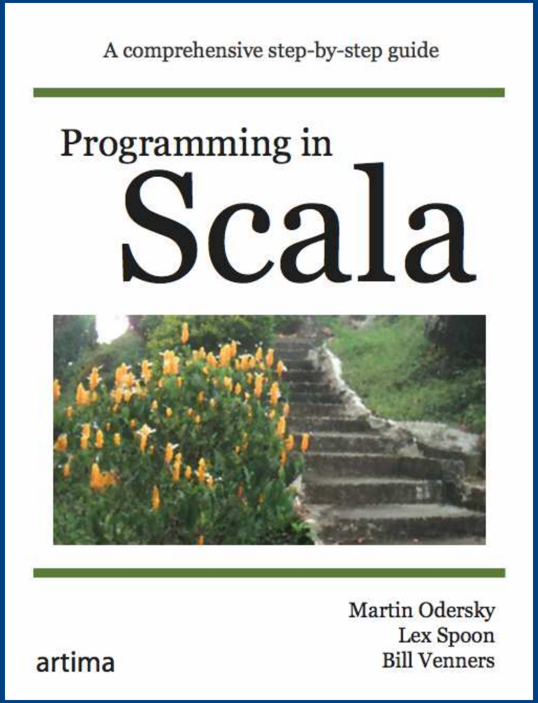
```
val opt =
  args.find(
    arg => { arg.substring(0, 2) == "-g" }
  )

opt match {
   case Some(dashG) =>
     println("Found: " + dashG)
   case None =>
     println("No -g found")
}
```

Getting Started with Scala

- Download Scala www.scala-lang.org
- Get the eBook www.artima.com
- Write scripts
- Do a side project
- "Mix in" Scala with Java

Q & A

Bill Venners
President
Artima, Inc.
www.artima.com


A comprehensive step-by-step guide

Programming in Scala

Martin Odersky
Lex Spoon
Bill Venners

artima