A Brief Intro to **Scala**

Angel J. Rey

About Me

- Angel Rey
- Software engineer. Big Data developer
- Utad master
- Working at GFT, and before at ZED and Bongiorno/Lumata as Big data Devolper / Architect

- Before **Scala** default languages were **Java** and **C++**

Dynamic vs. Static

Dynamic (Java Script)

- Concise
- Scriptable
- **R**ead-**E**val-**P**rint **L**oop (irb)
- Higher Order Functions
- Extend existing classes
- Duck Typing
- method_missing

Static (Java)

- Better IDE Support
- Fewer Tests
- Documentation
- Open Source Libs
- Performance
- JVM Tools (VisualVM)
- True Multi-threading

Scala

- ✓ Concise
- ✓ Scriptable
- ✓ **R**ead-**E**val-**P**rint **L**oop
- ✓ Higher Order Functions
- ✓ Extend existing classes
- ✓ Duck Typing
- ✓ method_missing

- ✓ Better IDE Support
- ✓ Fewer Tests
- ✓ Documentation
- ✓ Open Source Libs
- ✓ Performance
- ✓ JVM Tools (VisualVM)
- ✓ True Multi-threading

Scalable

language

Scala is a **modern multi**-**paradigm** programming language designed to express **common** programming **patterns** in a **concise**, **elegant**, and **type**-**safe** way.

Scala

- Statically Typed
- Runs on JVM, full inter-op with Java
- Object Oriented
- Functional
- Dynamic Features

Scala is Practical

- Can be used as drop-in replacement for Java
  - Mixed Scala/Java projects
- Use existing Java libraries
- Use existing Java tools (Ant, Maven, JUnit, etc...)
- Decent IDE Support (NetBeans, IntelliJ, Eclipse)

Scala Installation

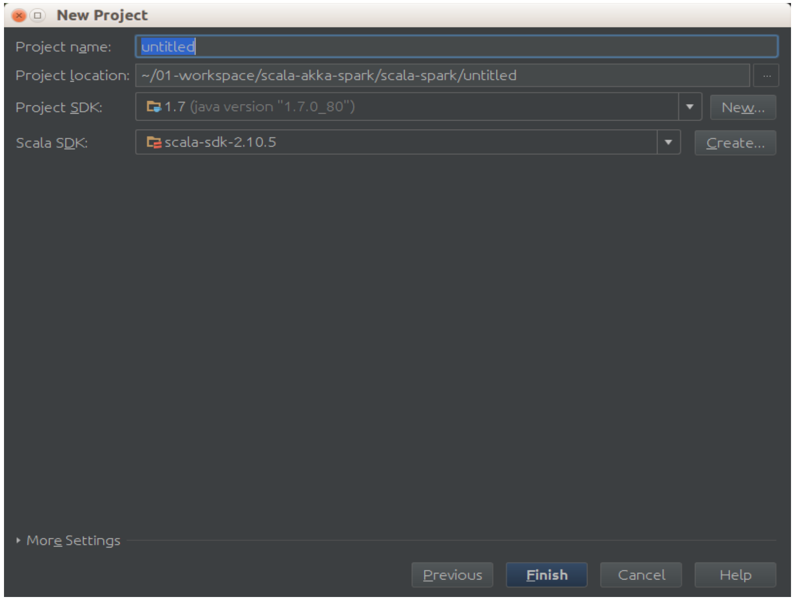
- For using with spark 1.3.1 → Scala 2.10.5
- Download and unpack: `http://www.scala-lang.org/download/2.10.5.html`
- Start the Scala interpreter (aka the "REPL") by launching `scala` from where it was unarchived.
- Start the Scala compiler by launching `scalac` from where it was unarchived.
- For quick access, add scala and scalac to your path. For example:

Environment	**Variable**	**Value (example)**
`Linux / Unix`	`$SCALA_HOME`	`/usr/local/share/scala`
	`$PATH`	`$PATH:$SCALA_HOME/bin`
`Windows`	`%SCALA_HOME%`	`c:\Progra~1\Scala`
	`%PATH%`	`%PATH%;%SCALA_HOME%\bin`

- Consider install SBT for building your projects (sbt is like a "maven for scala" with steroids): `http://www.scala-sbt.org/`
- Installing instructions at: `http://www.scala-sbt.org/0.13/tutorial/Manual-Installation.html`

IntelliJ Installation

- IntelliJ is a modern IDE which supports Scala development using a plugin
- Download IntellijI in the desired flavor (Ultimate Edition recommended) from: `https://www.jetbrains.com/idea/download/`
- Unpack & Install using the provided script
- The Installation wizard will ask you if you want to install Scala plugin choose yes to install
- You can install later or check if it's installed: CTRL + SHIFT + A / type `plugins` / in the search box type `Scala`.
- Link Scala plugin with the installed Scala version: when creating a New Project select `Scala` or `SBT` and choose scala-sdk-2.10.5 as Scala-sdk (or scala version 2.10.4 in SBT)



New Project

Project name: untitled

Project location: ~/01-workspace/scala-akka-spark/scala-spark/untitled

Project SDK: 1.7 (java version "1.7.0_80")

Scala SDK: scala-sdk-2.10.5

Scala is Concise

Type Inference

```
val sum = 1 + 2 + 3

val nums = List(1, 2, 3)

val map = Map("abc" -> List(1,2,3))
```

Explicit Types

```
val sum: Int = 1 + 2 + 3

val nums: List[Int] = List(1, 2, 3)

val map: Map[String, List[Int]] = ...
```

Higher Level

```java
// Java - Check if string has uppercase character
boolean hasUpperCase = false;
for(int i = 0; i < name.length(); i++) {
    if(Character.isUpperCase(name.charAt(i))) {
        hasUpperCase = true;
        break;
    }
}
```

Higher Level

```
// Scala
val hasUpperCase = name.exists(_.isUpper)
```

Less Boilerplate

```
// Java
public class Person {
  private String name;
  private int age;
  public Person(String name, Int age) {  // constructor
    this.name = name;
    this.age = age;
  }
  public String getName() {              // name getter
    return name;
  }
  public int getAge() {                  // age getter
    return age;
  }
  public void setName(String name) {     // name setter
    this.name = name;
  }
  public void setAge(int age) {          // age setter
    this.age = age;
  }
}
```

Less Boilerplate

```
// Scala
class Person(var name: String, var age: Int)
```

Less Boilerplate

```
// Scala
class Person(var name: String, private var _age: Int) {
  def age = _age             // Getter for age
  def age_=(newAge:Int) {  // Setter for age
    println("Changing age to: "+newAge)
    _age = newAge
  }
}
```

Variables and **Val**ues

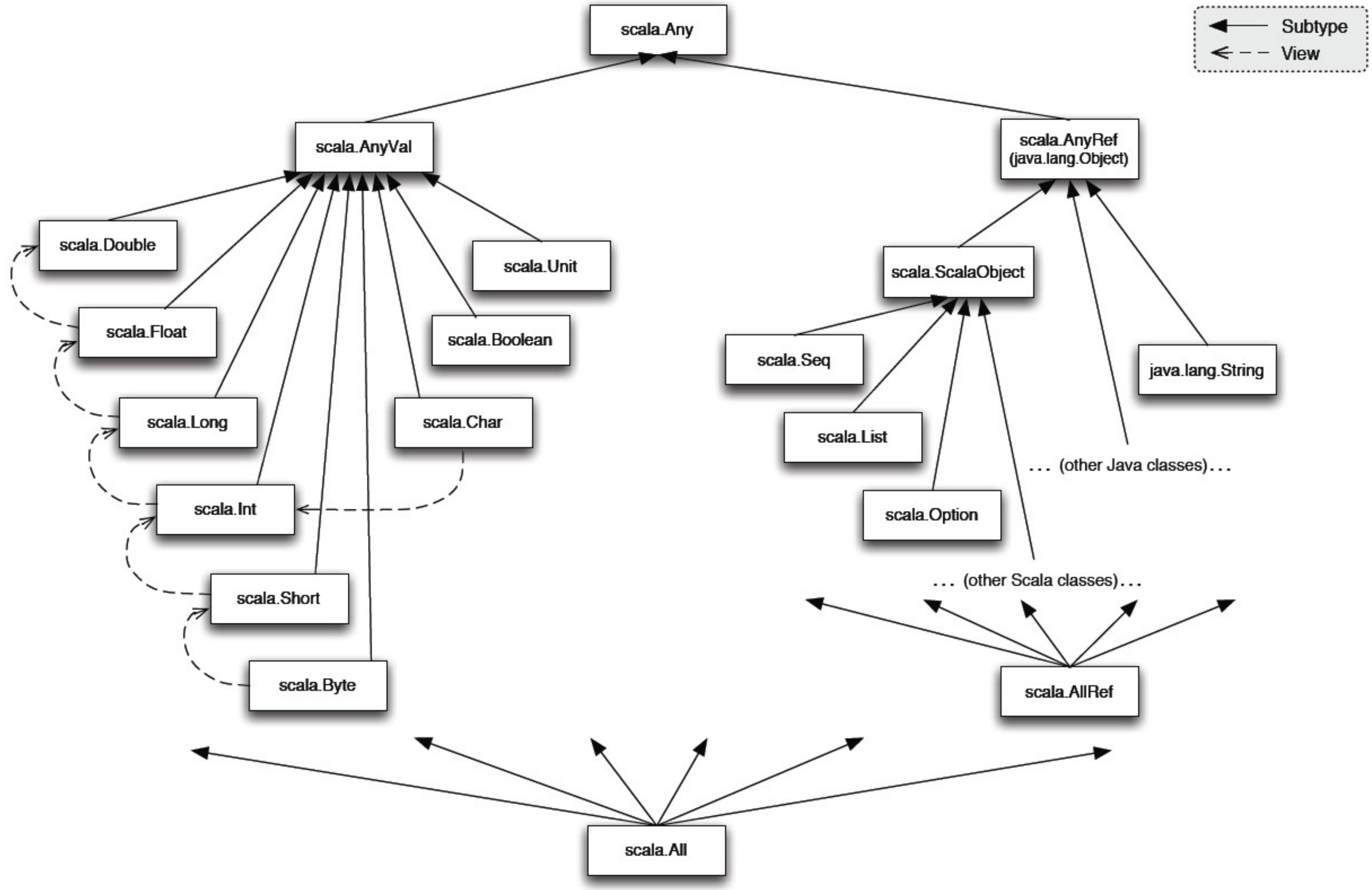
```
// variable
var foo = "foo"
foo = "bar"  // okay

// value
val bar = "bar"
bar = "foo"  // nope
```

Scala is **O**bject **O**riented

Pure O.O.

```
// Every value is an object
1.toString

// Every operation is a method call
1 + 2 + 3   □   (1).+(2).+(3)

// Can omit . and ( )
"abc" charAt 1   □   "abc".charAt(1)
```

Scala class hierarchy



Classes

```
// Classes (and abstract classes) like Java
abstract class Language(val name:String) {
  override def toString = name
}

// Example implementations
class Scala extends Language("Scala")

// Anonymous class
val scala = new Language("Scala") { /* empty
*/ }
```

Traits

```
// Like interfaces in Java
trait Language {

  val name:String

  // But allow implementation
  override def toString = name
}
```

Traits

```
trait JVM {
  override def toString = super.toString+" runs on
JVM" }
trait Static {
  override def toString = super.toString+" is
Static" }

// Traits are stackable
class Scala extends Language with JVM with Static {
  val name = "Scala"
}

println(new Scala) ⊓ "Scala runs on JVM is Static"
```

Singleton Objects

```
// Replaces static methods from Java
// Can extend/implement classes &
traits

object Hello {
  def world = println("Hello World"}
}

Hello.world  ▭  Hello World
```

Traits & Object example

```
trait Similarity {
  def isSimilar(x: Any): Boolean
  def isNotSimilar(x: Any): Boolean = !isSimilar(x)
}
class Point(xc: Int, yc: Int) extends Similarity {
  var x: Int = xc
  var y: Int = yc
  def isSimilar(obj: Any) =
    obj.isInstanceOf[Point] &&
      obj.asInstanceOf[Point].x == x
}
object TraitsTest extends App {
  val p1 = new Point(2, 3)
  val p2 = new Point(2, 4)
  val p3 = new Point(3, 3)
  println(p1.isNotSimilar(p2))
  println(p1.isNotSimilar(p3))
  println(p1.isNotSimilar(2))
}
```

Scala is Functional

First Class Functions

```
// Lightweight anonymous functions
(x:Int) => x + 1


// Calling the anonymous function
val plusOne = (x:Int) => x + 1
plusOne(5)    ⇒  6
```

Closures

```
// plusFoo can reference any
values/variables in scope
var foo = 1
val plusFoo = (x:Int) => x + foo

plusFoo(5)      ▭  6

// Changing foo changes the return
value of plusFoo
foo = 5
plusFoo(5)      ▭  10
```

Higher Order Functions

```
val plusOne = (x:Int) => x + 1
val nums = List(1,2,3)

// map takes a function: Int => T
nums.map(plusOne)           ▭   List(2,3,4)

// Inline Anonymous
nums.map(x => x + 1)        ▭   List(2,3,4)

// Short form
nums.map(_ + 1)             ▭   List(2,3,4)
```

Higher Order Functions

```
val nums = List(1,2,3,4)

// A few more examples for List class
nums.exists(_ == 2)               ▭   true
nums.find(_ == 2)                 ▭   Some(2)
nums.indexWhere(_ == 2)           ▭   1
nums.reduceLeft(_ + _)            ▭   10
nums.foldLeft(100)(_ + _)         ▭   110

// Many more in collections library
```

Higher Order Functions

```
// functions as parameters
def call(f: Int => Int) = f(1)


call(plusOne)      ▭   2
call(x => x + 1)   ▭   2
call(_ + 1)        ▭   2
```

Higher Order Functions

```
// functions as parameters
def each(xs: List[Int], fun: Int => Unit) {
  if(!xs.isEmpty) {
    fun(xs.head)
    each(xs.tail, fun)
  }
}

each(List(1,2,3), println)
```

```
    ▭   1
    ▭   2
    ▭   3
```

Curriying

```
/**
 * Created by chicochica10 on 4/06/15.
 */
object Curry extends App {

    def filter(xs: List[Int], p: Int => Boolean): List[Int] =
      if (xs.isEmpty) xs
      else if (p(xs.head)) xs.head :: filter(xs.tail, p)
      else filter(xs.tail, p)

    def modN(n: Int)(x: Int) = ((x % n) == 0)

    val nums = List(1, 2, 3, 4, 5, 6, 7, 8)

    println(filter(nums, modN(2)))
    println(filter(nums, modN(3)))
}
```

Pattern matching

```
// More complex example with generics & pattern
matching

@tailrec
def each[T](xs: List[T], fun: T => Unit): Unit = xs
match {
   case Nil =>
   case head :: tail => fun(head); each(tail, fun)
}

each(List(1,2), println)
      ▭   1
      ▭   2

each(List("foo", "bar"), println)
      ▭   foo
      ▭   bar
```

Pattern Matching

```
def what(any:Any) = any match {
  case i:Int => "It's an Int"
  case s:String => "It's a String"
  case _ => "I don't know what it is"
}

what(123)       →   "It's an Int"
what("hello")   →   "It's a String"
what(false)     →   "I don't know what it is"
```

Patter Matching - Case Class

```scala
object CaseClasses extends App{
  abstract class Term
  case class Var(name: String) extends Term
  case class Fun(arg: String, body: Term) extends Term
  case class App(f: Term, v: Term) extends Term
  //no new is needer (they have a companion object)
  val t =Fun ("x", Fun ("y", App (Var ("x"), Var ("y"))))
  val x = Var ("x")
  x.name
  // equals and toString
  val x1 = Var ("x")
  val x2 = Var ("x")
  val y1 = Var ("y")
  println(s"  $x1  ==  $x2  =>  ${x1 == x2}")
  println(s"  $x1  ==  $y1  =>  ${x1 == y1}")
  def printTerm(term: Term) {
    term match {
      case Var(n) => print(n)
      case Fun(x, b) =>
        print("^" + x + ".")
        printTerm(b)
      case App(f, v) =>
        print("(")
        printTerm(f)
        print(" ")
        printTerm(v)
        print(")")
    }
  }
  printTerm (t)
}
```

Pattern Matching

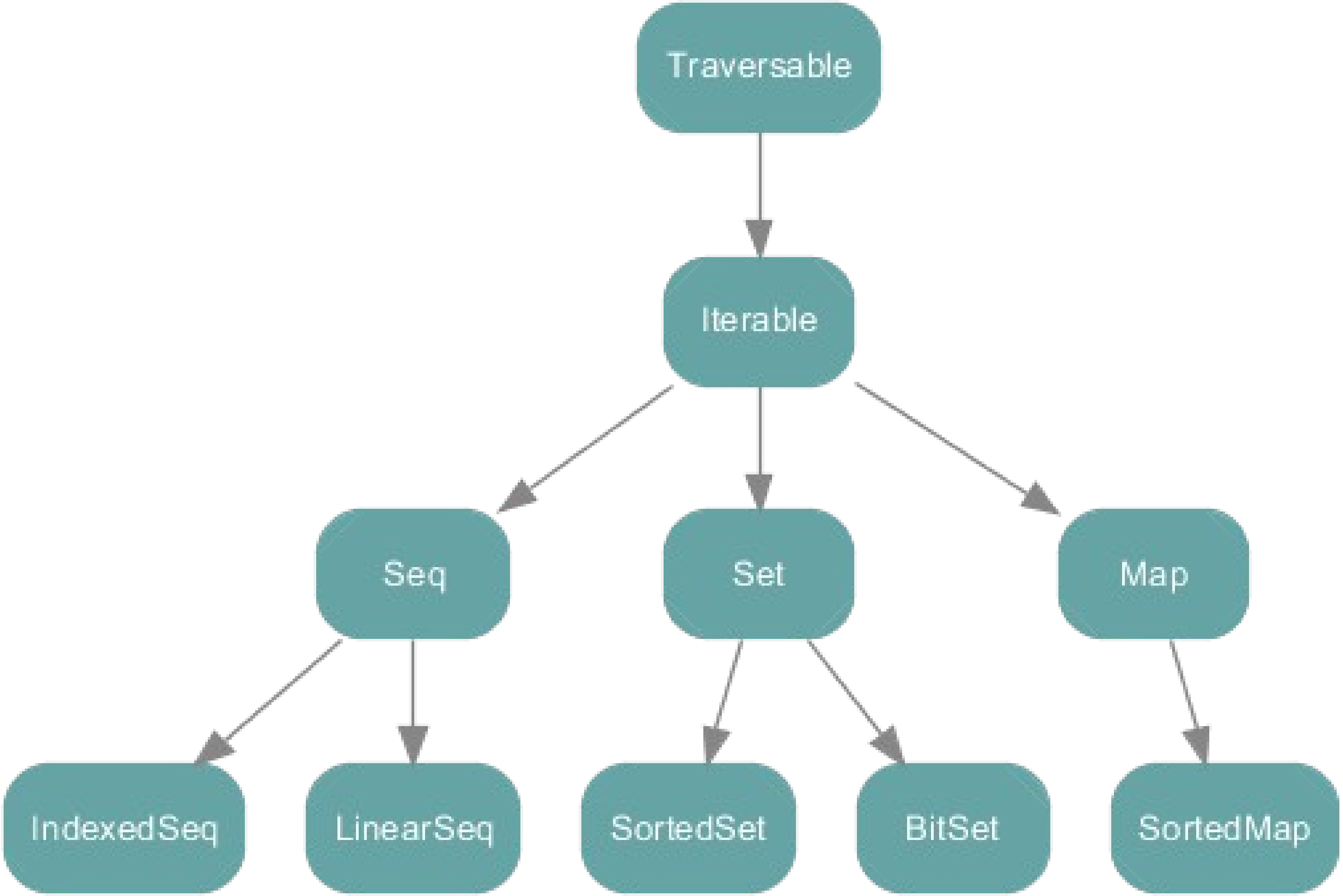
```
val nums = List(1,2,3)

// Pattern matching to create 3 vals
val List(a,b,c) = nums
```

a	▭	1
b	▭	2
c	▭	3

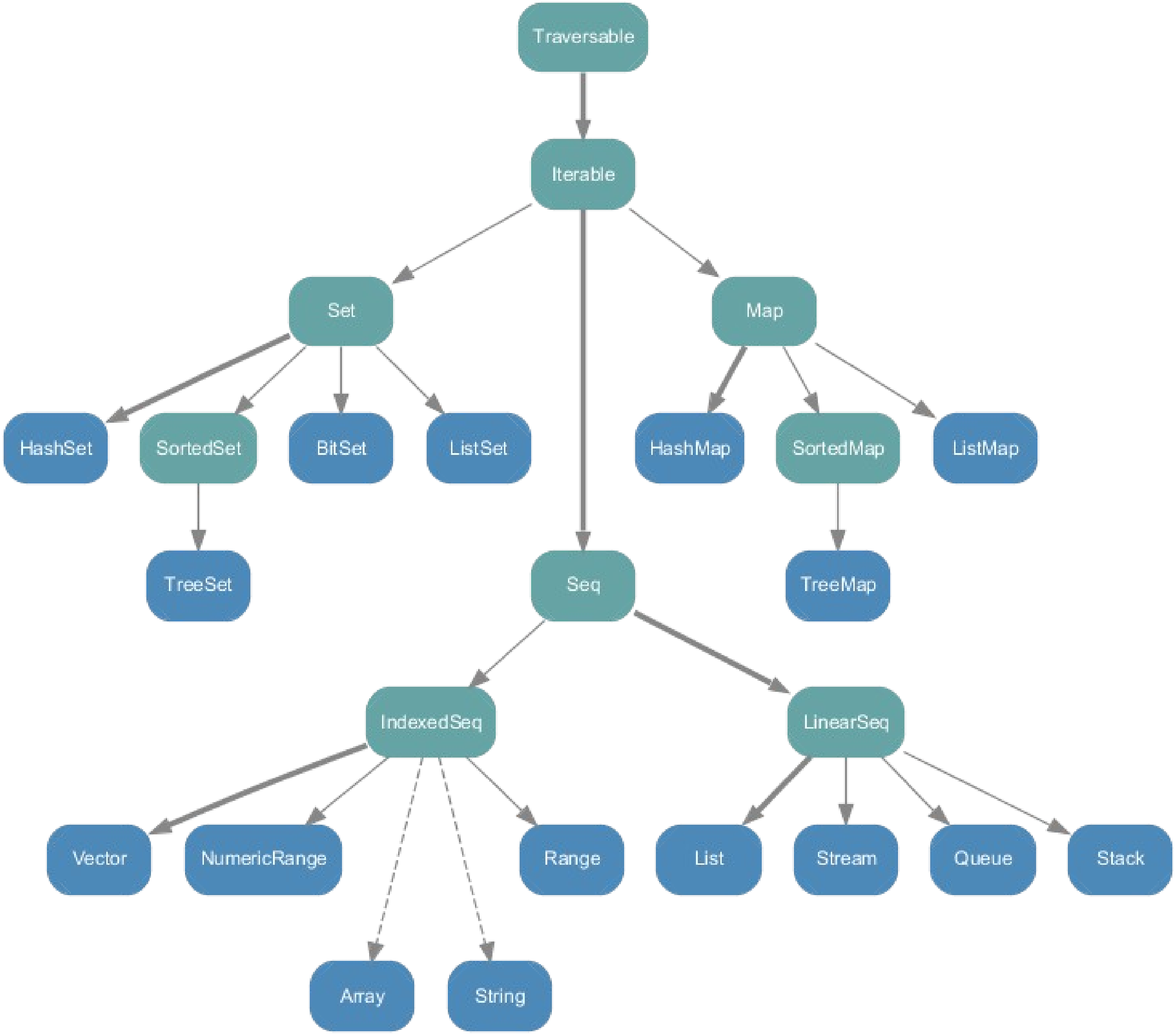
Immutable Types

```
// Immutable types by default
var nums = Set(1,2,3)
nums += 4   ▭   nums = nums.+(4)

// Mutable types available
import scala.collection.mutable._

val nums = Set(1,2,3)
nums += 4   ▭   nums.+=(4)
```

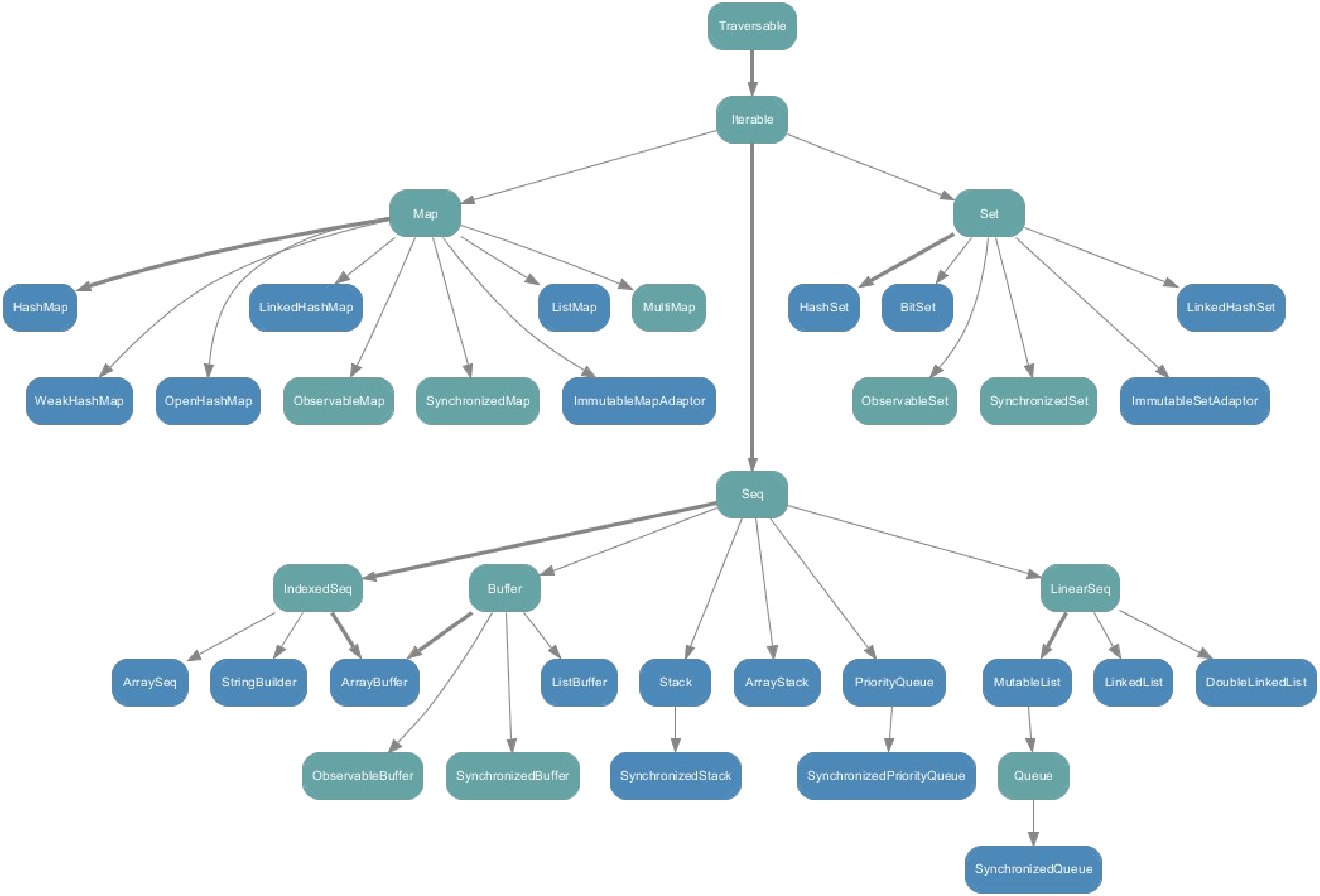
scala.collection



scala.collection.immutable



scala.collection.mutable



Or Use Existing Java Collections

- java.util
- Apache Commons Collections
- fastutil
- Trove
- Google Collections

- scala.collection.JavaConversion available to convert to and from java.util Interfaces

Scala is Dynamic

(Okay not really, but it has lots of features typically only found in Dynamic languages)

Scriptable

```
// HelloWorld.scala
println("Hello World")
```

```
bash$ scala HelloWorld.scala
Hello World
```

```
bash$ scala -e 'println("Hello World")'
Hello World
```

Read-Eval-Print Loop

```
bash$ scala
Welcome to Scala version 2.8.1.final (Java HotSpot(TM)
64-Bit Server VM, Java 1.6.0_22).
Type in expressions to have them evaluated.
Type :help for more information.

scala> class Foo { def bar = "baz" }
defined class Foo

scala> val f = new Foo
f: Foo = Foo@51707653

scala> f.bar
res2: java.lang.String = baz
```

Structural Typing

```
// Type safe Duck Typing
def doTalk(any:{def talk:String}) {
  println(any.talk)
}

class Duck { def talk = "Quack" }
class Dog  { def talk = "Bark"  }

doTalk(new Duck)   □   "Quack"
doTalk(new Dog)    □   "Bark"
```

Implicit Conversions

```
// Extend existing classes in a type safe way

// Goal: Add isBlank method to String class
class RichString(s:String) {
  def isBlank = null == s || "" == s.trim
}

implicit def toRichString(s:String) = new RichString(s)


// Our isBlank method is now available on Strings
" ".isBlank      ▭   true
"foo".isBlank    ▭   false
```

Implicit Conversions

```
// Does not type check
"abc".isBlank

// Search in-scope implicits defs that take a
// String & return a type with an isBlank
method
implicit def toRichString(s:String):RichString


// Resulting code that type checks
new RichString("abc").isBlank
```

method_missing (Scala 2.9 Feature)

```
// Dynamic is a marker trait used by the compiler
class Foo extends Dynamic {
  def typed[T] = error("not implemented")

  def applyDynamic(name:String)(args:Any*) = {
    println("called: "+name+"("+args.mkString(",")+")")
  }

}

val f = new Foo
f.helloWorld       ▭   called: helloWorld()
f.hello("world")   ▭   called: hello(world)
f.bar(1,2,3)       ▭   called: bar(1,2,3)
```

Scala has tons of other cool stuff

Default Parameter Values

```
def hello(foo:Int = 0, bar:Int = 0) {
  println("foo: "+foo+"  bar: "+bar)
}

hello()          □   foo: 0  bar: 0
hello(1)         □   foo: 1  bar: 0
hello(1,2)       □   foo: 1  bar: 2
```

Named Parameters

```
def hello(foo:Int = 0, bar:Int = 0) {
  println("foo: "+foo+" bar: "+bar)
}

hello(bar=6)          ▭   foo: 0  bar: 6
hello(foo=7)          ▭   foo: 7  bar: 0
hello(foo=8,bar=9)    ▭   foo: 8  bar: 9
```

Everything Returns a Value

```
val a = if(true) "yes" else "no"

val b = try{
  "foo"
} catch {
  case _ => "error"
}

val c = {
  println("hello")
  "foo"
}
```

Lazy Vals

```
// initialized on first access
lazy val foo = {
  println("init")
  "bar"
}

foo  ▭  init
foo  ▭
foo  ▭
```

Nested Functions

```
// Can nest multiple levels of functions
def outer() {
    var msg = "foo"
    def one() {
        def two() {
            def three() {
                println(msg)
            }
            three()
        }
        two()
    }
    one()
}
```

By-Name Parameters

```
// msg parameter automatically wrapped in closure
def log(doLog:Boolean, msg: => String) {
  if(doLog) {
    msg  // evaluates msg
    msg  // evaluates msg again!
  }
}

def foo:String = {
  println("in foo"); "Foo"
}

log(true, foo+" Bar")   // foo called twice
    □   in foo
    □   in foo

log(false, foo+" Bar")  // foo never called
```

Another Clousure example

```
object TargetTest1 extends App {
  def whileLoop(cond: => Boolean)(body: =>
Unit): Unit =
    if (cond) {
      body
      whileLoop(cond)(body)
    }
  var i = 10
  whileLoop (i > 0) {
    println(i)
    i -= 1
  }
}
```

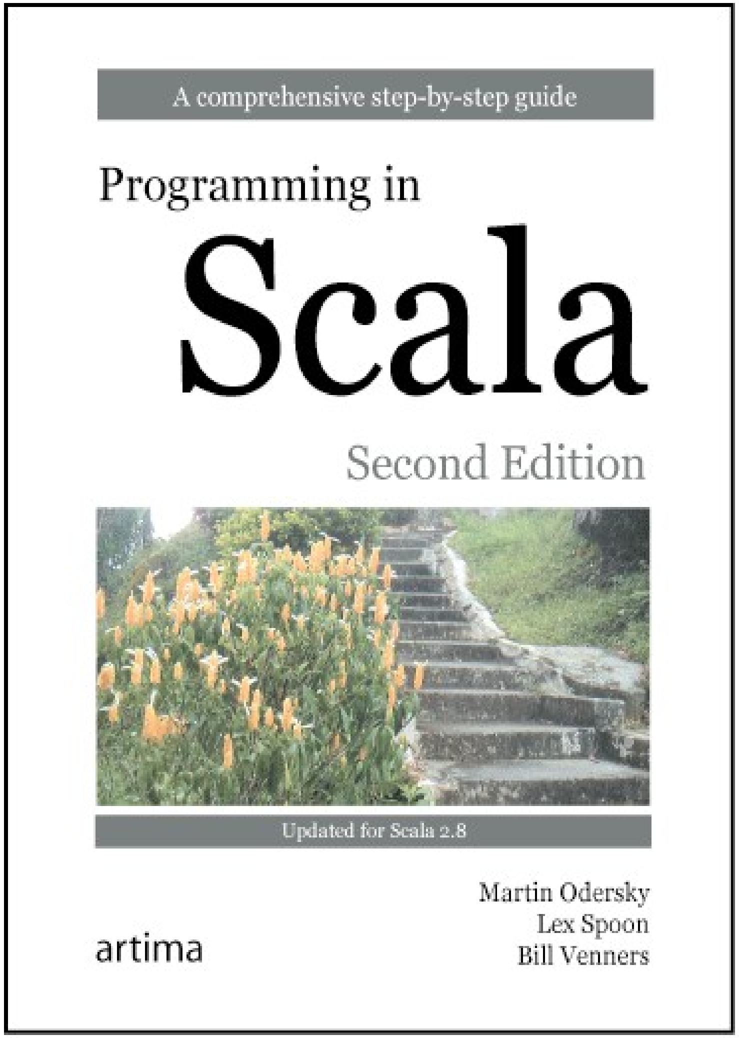
Many More Features

- **Actors**
- **Annotations** □ `@foo def hello = "world"`
- **Case Classes** □ `case class Foo(bar:String)`
- **Currying** □ `def foo(a:Int,b:Boolean)(c:String)`
- **For Comprehensions** □ `for(i <- 1.to(5) if i % 2 == 0) yield i`
- **Generics** □ `class Foo[T](bar:T)`
- **Package Objects**
- **Partially Applied Functions**
- **Tuples** □ `val t = (1,"foo","bar")`
- **Type Specialization**
- **XML Literals** □ `val node = <hello>world</hello>`
- **etc…**

Personal Experiences

- Productive from Day 1
- Drop in replacement for Java giving you more Ruby-like syntax and features
- Can pickup the functional and higher-level programming concepts as you go

Great Books for a Deep Dive into Scala



A comprehensive step-by-step guide

Programming in

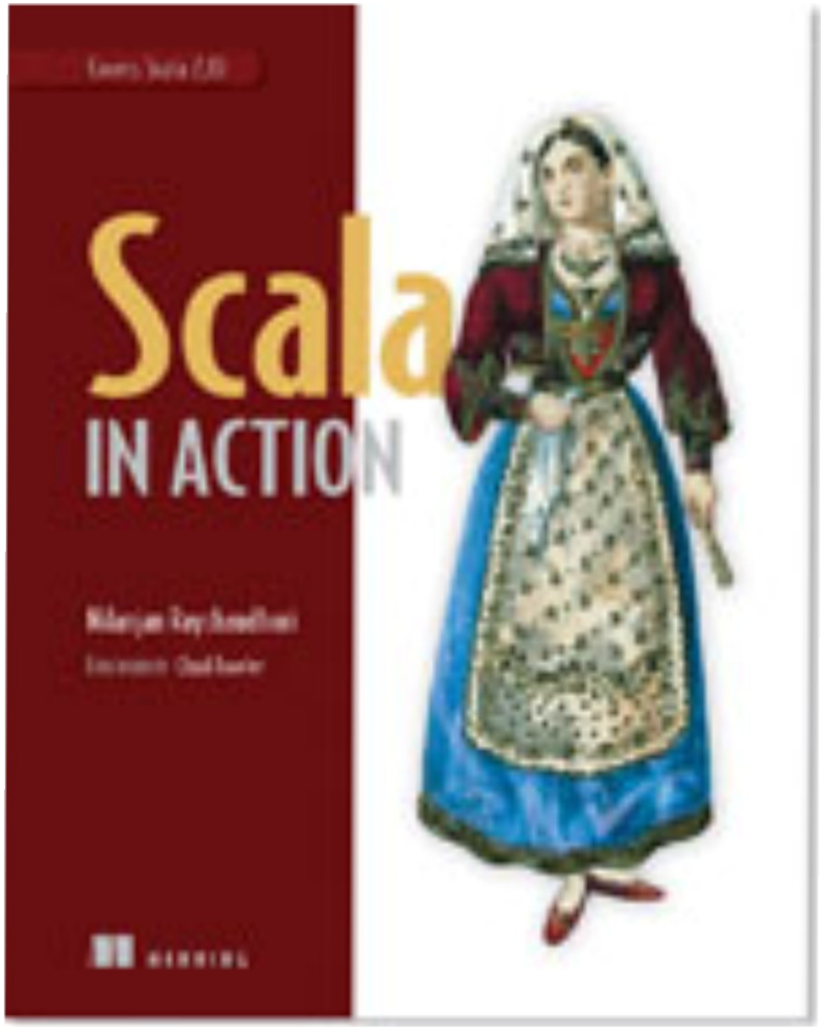
Scala

Second Edition

Updated for Scala 2.8

Martin Odersky
Lex Spoon
Bill Venners

artima



Scala

www.scala-lang.org