

# Scala for Java developers

– A three-hour crash course

Björn Regnell

Dept. of Computer Science, LTH  
Lund University, Sweden

2016 April 20

## Agenda

- 1 Introduction to Scala
  - What is Scala?
  - What can you do with Scala?
  - Similarities and differences between Scala and Java
- 2 Live Scala coding
- 3 Overview of new course at LTH: EDAA45 Scala + Java
- 4 Workshop: Exercises in Scala

# Background

The aim of Scala: A scalable, pragmatic, real-world language

- [en.wikipedia.org/wiki/Scala programming language](http://en.wikipedia.org/wiki/Scala_programming_language)
- **Multi-paradigm:** object-oriented, functional, imperative, concurrent
- **Designed by:** Martin Odersky
- **Developer:** EPFL, Lightbend, OSS
- **First appeared:** January 20, 2004
- **Stable release:** 2.11.8 / March 8, 2016
- **Typing:** static, strong, inferred, structural
- **Platform:** JVM, JavaScript
- **License:** BSD 3-clause
- **File ext:** .scala
- **Official site:** [www.scala-lang.org/](http://www.scala-lang.org/)



# Scala History

Heritage: Algol, Modula-2, Simula, Pizza, Java, Beta, OCaml, Haskell, ...

Time line:

- 2004: 1.0, 1.1, 1.2, 1.3
- 2005: 1.4
- 2006: 2.0, 2.1, 2.2, 2.3; **scalac written in Scala**
- 2007: 2.4, 2.5, 2.6
- 2008: 2.7;
- 2010: 2.8; **Play** gets a scala plug-in, **Akka**
- 2011: 2.9; **Typesafe**; **scala.collection.parallel**, Play in Scala
- 2013: 2.10 value classes, implicit classes, string interpolators, Try, Future, Promise, Dynamic, Akka actors
- 2014: 2.11; optimizations; 10x faster compilation
- 2016: 2.12; Java 8, Scala.js, **Scala Center@EPFL**, **Lightbend**

[Bill Venners, Frank Sommers]

[Marconi Lanna]

# Scala – the simple parts

Lecture by **Martin Odersky**: [www.youtube.com/watch?v=ecekSCX3B4Q](http://www.youtube.com/watch?v=ecekSCX3B4Q)

Scala for every-day dev actions:

- 1 **Compose**: everything is a composable **expression**
- 2 **Match**: decompose data with **pattern**-matching
- 3 **Group**: everything can be grouped and **nested**
- 4 **Recurse**: compose at any depth; better loops **@tailrec**
- 5 **Abstract**: functions are objects
- 6 **Aggregate**: collections aggregate & **transform** data
- 7 **Mutate**: local, private mutability to optimize perf.



SF Scala: Martin Odersky, Scala -- the Simple Parts

# Some similarities between Scala and Java

- Both are object-oriented and imperative
- Both are statically typed (~ 100 times faster than Python)
- Both have C-like block syntax { }
- Both have lambdas (Java 8)
- Both run on the JVM
- Both can execute each other's byte code

# Some differences between Scala and Java

- Scala is a more "pure" OO language:  
instances of **Int**, **Double**, **Char**, etc. are **real objects**
- Scala is a more advanced functional language:  
easy to transform immutable data in functional collections
- Scala unifies OO and functional programming:  
functions are objects with an apply-method
- singleton **object** instead of Java's static
- Some syntax differences:

# Some differences between Scala and Java

- Scala is a more "pure" OO language:  
instances of **Int**, **Double**, **Char**, etc. are **real objects**
- Scala is a more advanced functional language:  
easy to transform immutable data in functional collections
- Scala unifies OO and functional programming:  
functions are objects with an apply-method
- singleton **object** instead of Java's static
- Some syntax differences:
  - semicolons are inferred; newline btw statements is enough
  - no need for **return** as blocks are values
  - Type *after* names and colon: **val** name: String = "Kim"
  - generic types in [T] instead of <T>
  - Five types of members: **def**, **val**, **lazy val**, **var**, **type**  
Methods: **def** isChild: Boolean = age < 18  
Immutable fields: **val** gender = "Female"  
Delayed init: **lazy val** r = List.fill(1000)(math.random)  
Mutable fields: **var** age: Int = 42  
Type alias: **type** Matrix = Map[Int, Map[Int, String]]



# Classes in Java and Scala

```
// this is Java

public class JPerson {
    private String name;
    private int age;

    public JPerson(String n, int a) {
        name = n;
        age = a;
    }

    public JPerson(String n) {
        name = n;
        age = 42;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int a) {
        age = a;
    }
}
```

# Classes in Java and Scala

```
// this is Java
```

```
public class JPerson {  
    private String name;  
    private int age;  
  
    public JPerson(String n, int a) {  
        name = n;  
        age = a;  
    }  
  
    public JPerson(String n) {  
        name = n;  
        age = 42;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int a) {  
        age = a;  
    }  
}
```

```
// same in (non-idiomatic) Scala
```

```
class SPerson(n: String, a: Int) {  
    private var name: String = n  
    private var age: Int = a  
  
    def this (n: String): Unit = {  
        this(n, 42)  
    }  
  
    def getName = name  
  
    def getAge = age  
  
    def setAge(a: Int): Unit = {  
        age = a  
    }  
}
```

# Classes in Java and Scala

```
// this is Java
```

```
public class JPerson {  
    private String name;  
    private int age;  
  
    public JPerson(String n, int a) {  
        name = n;  
        age = a;  
    }  
  
    public JPerson(String n) {  
        name = n;  
        age = 42;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public int getAge() {  
        return age;  
    }  
  
    public void setAge(int a) {  
        age = a;  
    }  
}
```

```
// same in (non-idiomatic) Scala
```

```
class SPerson(n: String, a: Int) {  
    private var name: String = n  
    private var age: Int = a  
  
    def this (n: String): Unit = {  
        this(n, 42)  
    }  
  
    def getName = name  
  
    def getAge = age  
  
    def setAge(a: Int): Unit = {  
        age = a  
    }  
}
```

```
// this is idiomatic Scala
```

```
case class Person(  
    name: String,  
    age: Int = 42  
)
```

# Live coding: code-along

Start the REPL:

```
$ scala
Welcome to Scala 2.11.8 (Java HotSpot(TM) VM, Java 1.8.0_60)
Type in expressions for evaluation. Or try :help.

scala> case class Person(name: String, age: Int = 42)
defined class Person

scala> Person("Björn", 48)
res0: Person = Person(Björn,48)

scala> Person("Kim")
res1: Person = Person(Kim,42)
```

# Functions are first-class values; Try this in REPL:

```
def öka(i: Int) = i + 1

val nums = Vector(1, 2, 3, 4, 42)

nums.map(öka)

nums.map(i => i + 1)

nums.map(_ + 1)

def mappa(xs: Vector[Int], f: Int => Int) = xs.map(f)

mappa(nums, öka)

def upprepa(n: Int)(block: => Unit) = for (i <- 1 to n) block
```

# Overview of new LTH Course EDAA45 (was EDA016)

Open Source project, on-going course dev:

<https://github.com/lunduniversity/introprog>

<i>W</i>	<i>Modul</i>	<i>Övn</i>	<i>Lab</i>
W01	Introduktion	expressions	kojo
W02	Kodstrukturer	programs	–
W03	Funktioner, Objekt	functions	simplewindow
W04	Datastrukturer	data	textfiles
W05	Sekvensalgoritmer	sequences	cardgame
W06	Klasser, Likhet	classes	shapes
W07	Arv, Gränssnitt	traits	turtlerace-team
KS	KONTROLLSKRIVN.	–	–
W08	Mönster, Undantag	matching	chords-team
W09	Matriser, Typparametrar	matrices	maze
W10	Sökning, Sortering	sorting	surveydata-team
W11	Scala och Java	scalajava	scalajava-team
W12	Trådar	threads	life
W13	Design	Uppsamling	Inl.Uppg.
W14	Tentaträning	Extenta	–
T	TENTAMEN	–	–

# Workshop: Exercises in Scala

Test our **DRAFT** exercises in Scala:

[github.com/lunduniversity/introprog/blob/master/compendium/exercises.pdf](https://github.com/lunduniversity/introprog/blob/master/compendium/exercises.pdf)

**Feedback welcome!**

- 1 Övning expressions – skim if too simple
- 2 Övning programs – do hello world then skim
- 3 Övning **functions** – check out Scala's functional aspects
- 4 Övning data – check out Scala's basic OO aspects
- 5 Övning sequences – only half-ready; try scala collections

If time permits:

We will close with live coding of some more advanced aspects