Scala (Programmiersprache)

Ein Teaser und allgemeinere Gedanken

Sebastian Eidecker

16. März 2016

Wer als Werkzeug nur einen Hammer hat, sieht in jedem Problem einen Nagel.

Paul Watzlawick

Worüber reden wir? IT im Wandel

IT im Wandel

Herausforderungen

IT im Wandel

Manifeste

Herausforderungen





IT im Wandel Herausforderungen

Manifeste
Scala

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Ein wenig Code

IT im Wandel

Herausforderungen

Manifeste

Scala

Management Summary

Ein wenig Code

Spannendes

Herausforderungen

IT im Wandel

Software Engineering

Software	Engineering

· Stabilität und Resilienz

- · Stabilität und Resilienz
- Wertbeitrag

- · Stabilität und Resilienz
- Wertbeitrag
- Businesstreiber

- · Stabilität und Resilienz
- Wertbeitrag
- Businesstreiber

Matthias Magnor – CEO Surface und Contract Logistics

IT im Wandel

Manifeste

 Antwortbereit, Widerstandsfähig, Elastisch, Nachrichtenorientiert (2013)

• Gut gefertigt, Stets Mehrwert, Gemeinschaft aus Experten, Produktive Partnerschaften (2009)

 Individuen und Interaktionen, Funktionierende Software, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderung (2001)

- Antwortbereit, Widerstandsfähig, Elastisch, Nachrichtenorientiert (2013)
- Gut gefertigt, Stets Mehrwert, Gemeinschaft aus Experten, Produktive Partnerschaften (2009)
- Individuen und Interaktionen, Funktionierende Software, Zusammenarbeit mit dem Kunden, Reagieren auf Veränderung (2001)

Wo stehen wir?

Wo stehen wir im Wettbewerb?

Scala

Management Summary

Scalable Language

Scalable Language

This means that Scala grows with you. You can play with it by typing one-line expressions and observing the results. But you can also rely on it for large mission critical systems [...]

— www.scala-lang.org



Objektorientiert

- Objektorientiert
- Funktional

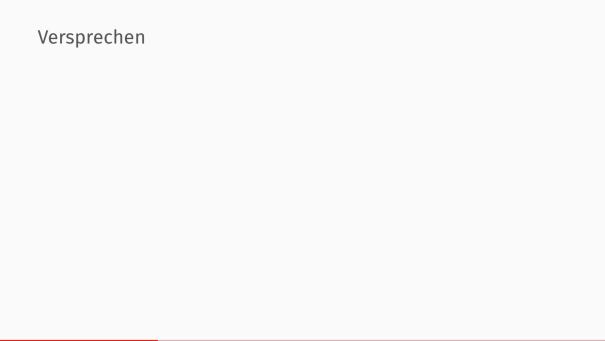
- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- Immutable by default

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- · Immutable by default
- Gewohnte Syntax ("Java ohne Semikolon")

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- · Immutable by default
- Gewohnte Syntax ("Java ohne Semikolon")
- Ausdrucksstark (APIs/DSLs schreiben)

- Objektorientiert
- Funktional
- Statisch typisiert mit Type Inference
- · Immutable by default
- Gewohnte Syntax ("Java ohne Semikolon")
- Ausdrucksstark (APIs/DSLs schreiben)
- Jung (2004, Hype 2011)



Produktivitätssteierung

- Produktivitätssteierung
- · Höhere Codequalität

- Produktivitätssteierung
- · Höhere Codequalität
- Mehr Spaß

- Produktivitätssteierung
- · Höhere Codequalität
- Mehr Spaß durch
- Weniger Code

- Produktivitätssteierung
- Höhere Codequalität
- Mehr Spaß durch
- Weniger Code
- · Höheres Abstraktionsniveau

- Produktivitätssteierung
- · Höhere Codequalität
- Mehr Spaß
 durch
- Weniger Code
- · Höheres Abstraktionsniveau
- Skalierbarkeit

· Java-Bytecode, läuft auf JVM

· Java-Bytecode, läuft auf JVM

· Java-Bibliotheken nutzbar

- James Distracted Linux
 - Java-Bytecode, läuft auf JVM
 - Java-Bibliotheken nutzbar

Bekannte IDFs

- · Java-Bytecode, läuft auf JVM
- · Iava-Bibliotheken nutzbar
- Bekannte IDFs
- · Ähnlicher Paketierungs- und Buildprozess (sbt)

Scala

Ein wenig Code



Eine Java-Klasse

5

7

8

10

11

12 13

```
public class Person {
  private final String firstName;
  private final String lastName:
  public Person(String firstName, String lastName) {
      this.firstName = firstName:
      this.lastName = lastName:
  public String getFirstName() {
      return firstName:
  public String getLastName() {
      return lastName:
```

Eine Java-Klasse

14

15

16

17

18

19

20

21

22

23

24

25 26

```
aOverride
public boolean equals(Object o) {
    if (this == 0) return true:
    if (o == null || getClass() != o.getClass()) return false;
    Person person = (Person) o:
    if (firstName != null ?
        !firstName.equals(person.firstName) :
        person.firstName != null) return false:
    if (lastName != null ?
        !lastName.equals(person.lastName) :
        person.lastName != null) return false:
    return true:
```

Eine Java-Klasse

```
noverride
public int hashCode() {
    int result = firstName != null ? firstName.hashCode() : 0;
    result =
        31 * result + (lastName != null ? lastName.hashCode() :
        0);
    return result;
}
```

Businesslogik?



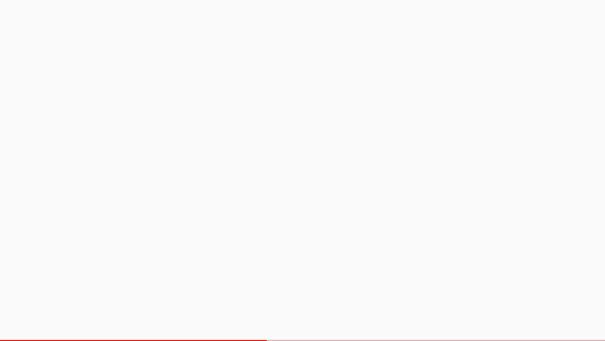
Dasselbe in Scala

Dasselbe in Scala

case class Person(firstName:String, lastName:String)

Es wird funktional – Quicksort

Es wird funktional – Quicksort



val und var

val und var

```
1 val j = 3
```

- **var** k = 3
- **j = 4**
- **k = 4**

```
val und var
```

```
val j = 3
var k = 3
j = 4
```

5 **k = 4**

Compile-Fehler für j, da immutable

Parameter sind vals

def addOne(i: Int): Int = { i += 1; i }

```
Parameter sind vals
```

Parameter sind vals

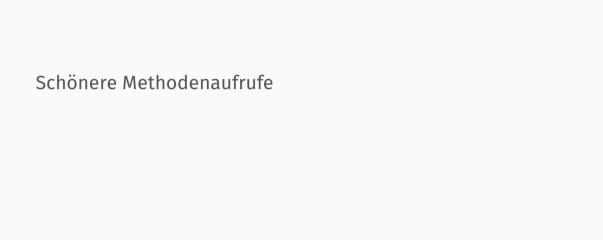
def addOne(i: Int): Int = { i += 1; i }

Compile-Fehler, da i immutable

Funktionen – Benannte Parameter

Funktionen – Benannte Parameter

class



Schönere Methodenaufrufe

```
"Test".startsWith("T")
List(1,2,3).isEmpty
```

Schönere Methodenaufrufe

```
1 "Test".startsWith("T")
2 List(1,2,3).isEmpty
1 "Test" startsWith "T"
```

2 List(1,2,3) isEmpty

Definition eigener Operatoren

Definition eigener Operatoren

```
case class Time(hour:Int, minute:Int) {
    def minus(time: Time) = {new Time(this.hour - time.hour, this.
        minute - time.minute)}
```

Definition eigener Operatoren

Time(10.20).minus(Time(1.10))

```
case class Time(hour:Int, minute:Int) {
  def minus(time: Time) = {new Time(this.hour - time.hour, this.
      minute - time.minute)}
}
```

Definition eigener Operatoren

Time(10,20).minus(Time(1,10))
Time(10.20) minus Time(1.10)

```
case class Time(hour:Int, minute:Int) {
  def minus(time: Time) = {new Time(this.hour - time.hour, this.
      minute - time.minute)}
}
```

Definition eigener Operatoren

Time(10,20).minus(Time(1,10))
Time(10.20) minus Time(1.10)

Definition eigener Operatoren

Time(10,20) minus Time(1,10) Time(10,20) - Time(1,10)

```
case class Time(hour:Int, minute:Int) {
  def minus(time: Time) = {new Time(this.hour - time.hour, this.
        minute - time.minute)}
  def -(time: Time) = minus(time)
}
Time(10.20).minus(Time(1.10))
```

Klassen und Objekte

Klassen und Objekte

class



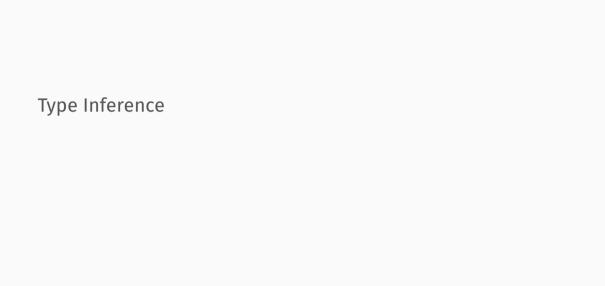
Listen

class

Pattern Matching

Pattern Matching

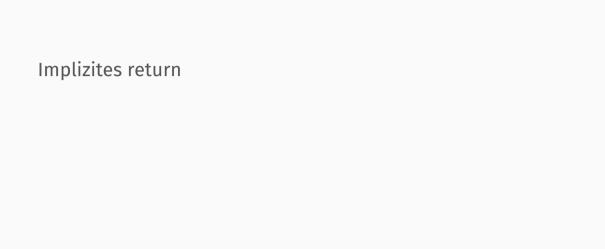
class



```
Type Inference
```

```
1 def f() = 3 * 2
```

def f(): Int = 3 * 2



Implizites return

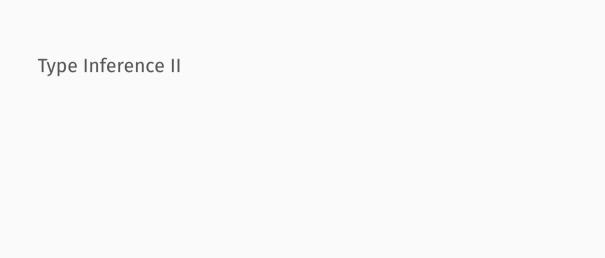
```
1 def f() = {
2    if (something)
3    "A"
4    else
```

"B"

Implizites return

```
1 def f() = {
2    if (something)
3    "A"
4    else
5    "B"
```

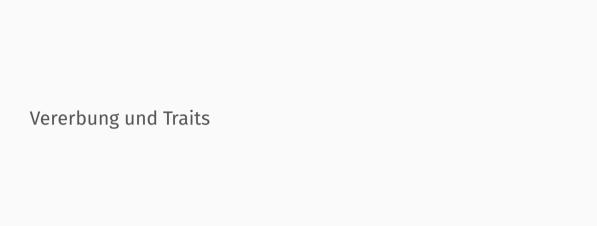
Letzte Anweisung wird zurückgegeben, impliziter Typ String.



Type Inference II

```
def f() = {
    if (something)
    "A"
    else
    1
}
```

Erste gemeinsame Oberklasse, zur Not Any



Vererbung und Traits

class

Funktionen funktional – Lambdas schön

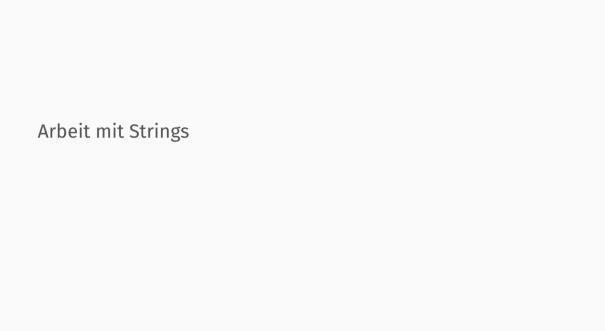
Funktionen funktional – Lambdas schön

class

Flatmap that shit!

ı class

Flatmap that shit!



class

Arbeit mit Strings



Tupel class



Implicits class

??? - Mein heimlicher Star

```
??? - Mein heimlicher Star

case class Time(hour:Int, minute:Int) {
  def minus(time: Time) = ???
```

def -(time: Time) = minus(time)

```
??? - Mein heimlicher Star

case class Time(hour:Int, minute:Int) {
  def minus(time: Time) = ???
  def -(time: Time) = minus(time)
```

Kompilierbar, aber nicht gefährlich.

Scala

Spannendes

Scalable real-time transaction processing

- Scalable real-time transaction processing
 - Nebenläufige Berechnungen

- · Scalable real-time transaction processing
- · Nebenläufige Berechnungen
- Kommunikation ausschließlich über Nachrichten

Aktoren

- Scalable real-time transaction processing
- · Nebenläufige Berechnungen
- · Kommunikation ausschließlich über Nachrichten
- · Elastizität und Resilienz

Aktoren

- Scalable real-time transaction processing
- · Nebenläufige Berechnungen
- · Kommunikation ausschließlich über Nachrichten
- · Elastizität und Resilienz
- · Einfaches Mapping auf Prozessdefinitionen

Fachliche Abstraktion

- Fachliche Abstraktion

- Verständlicher

- Fachliche Abstraktion
- Verständlicher
- Entwicklung schwierig

- Fachliche Abstraktion
- Verständlicher
- Entwicklung schwierig
- · Einschränkungen vorhanden

DSL - Beispiel

- 120 IF ABS('burn) <= 'fuel THEN 150
 130 PRINT "You don't have that much fuel"</pre>
- 4 130 PRINT "You don't have that much fuel" 5 140 GOTO 100
- 5 150 LET ('v := 'v + 'burn * 10 / ('fuel + 'mass))



DSL sinnvoll – ScalaTest

DSL sinnvoll – ScalaTest

Fachlich verständliche Tests

DSL sinnvoll – ScalaTest

- Fachlich verständliche Tests
- Testdatengenerierung

Scalatest - Beispiel

```
"Creating a Time" should {
    "throw an IllegalArgumentException for hours less than 0 or
       greater equal 24" in {
      forAll("hours") { (hours: Int) =>
        whenever(hours < 0 || hours >= 24) {
          an[IllegalArgumentException] should be thrownBy Time(
             hours)
```

Vorteile

 Für moderne Architekturen

- Für moderne Architekturen
- · Verständlich funktional

- Für moderne Architekturen
- · Verständlich funktional
- Java-Ökosystem

- Für moderne Architekturen
- · Verständlich funktional
- Java-Ökosystem
- Macht Spaß

- Für moderne Architekturen
- · Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Vorteile

- Für moderne Architekturen
- · Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Vorteile

- Für moderne Architekturen
- · Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

Nachteile

Komplex

Vorteile

- Für moderne Architekturen
- · Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

- Komplex
- · Zukunftssicher?

Vorteile

- Für moderne Architekturen
- · Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

- Komplex
- · Zukunftssicher?
- Anzahl
 Entwicklungssklaven

Vorteile

- Für moderne Architekturen
- · Verständlich funktional
- Java-Ökosystem
- Macht Spaß
- Statisch typisiert

- Komplex
- · Zukunftssicher?
- Anzahl
 Entwicklungssklaven
- Binärkompatibilität nicht in alle Ewigkeit

We've found that Scala has enabled us to deliver things faster with less code. It's

reinvigorated the team.

— Graham Tackley, Guardian



Mehr für Nerds

Sprecht mich an

Mehr für Nerds

- Sprecht mich an

· Hands on-Termin bei Interesse

Mehr für Nerds

- · Sprecht mich an
- · Hands on-Termin bei Interesse
- Heiko Seeberger: "Durchstarten mit Scala. Tutorial für Einsteiger (2. Aufl.)"

