

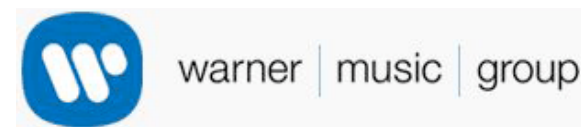
Scala

Verschmelzen von OOP und FP

Jan Kriesten

Java Forum Stuttgart - 2. Juli 2009 [D1]

- ▶ Internetagentur mit Standort Hamburg
- ▶ Web business seit 1995
- ▶ Consulting, Training, Softwareentwicklung
- ▶ Systemadministration
- ▶ Kunden



- ▶ Warum Scala?
- ▶ Spracheigenschaften
- ▶ Scala in der Praxis
- ▶ Ausblick und Resümee
- ▶ Q & A, Referenzen

Warum Scala?

- ▶ OOP - Gestern und heute
- ▶ Dynamische Sprachen
- ▶ Auftritt: Scala

- ▶ Die letzten 10 Jahre:
 - ❖ Java als „Main Stream OOP“
 - ❖ Smalltalk als „Pure OOP“

- ▶ Aktuell:
 - ❖ Java als OOP
 - ❖ Dynamische Sprachen (Groovy, Ruby, Python, ...)

- ▶ Pragmatischer Ansatz
- ▶ Einfache Syntax: leicht zu erlernen + programmieren
- ▶ Übernahme aus der Funktionalen Programmierung, z.B.:
 - ❖ Closures
 - ❖ Listen-Verarbeitung
- ▶ Scriptfähig
- ▶ Keine statische Typisierung
 - ➡ falscher Ansatz!

► Idee:

- ❖ Allgemeine Sprache
- ❖ Konzepte sollen sowohl im Kleinen wie im Großen greifen
- ❖ Symbiose und Generalisierung von funktionalen und objektorientierten Konzepten

► Scala:

- ❖ OOP + FP vollständig kompatibel mit Java (.NET soll folgen)
- ❖ Open Source seit Januar 2004

- ▶ Scala entwickelt vom Team um Martin Odersky an der École Polytechnique Fédérale de Lausanne (EPFL)
- ▶ Scala bietet
 - ❖ ein einheitliches Objekt-Modell
 - ❖ Pattern-matching
 - ❖ Higher-order Functions
 - ❖ neue Wege zu abstrahieren und Programme zu schreiben

Spracheigenschaften

- ▶ Kompatibel
- ▶ Funktional
- ▶ Kompakt
- ▶ Klar und zielgerichtet
- ▶ Erweiterbar
- ▶ Performant

- ▶ Nahtlose Java-Integration
 - ❖ Methoden-Aufruf
 - ❖ Field-Zugriff
 - ❖ Class-Vererbung
 - ❖ Interface-Implementierung
- ▶ JVM Bytecode
- ▶ Code ähnelt Java mit einigen Änderungen

```
object Test1 {  
    def main( args: Array[String] ) {  
        val b=new StringBuilder()  
        for( i<-0 until args.length ) {  
            if( i>0 )  
                b.append( " " )  
            b.append( args(i).toUpperCase )  
        }  
        println( b.toString )  
    }  
}
```

- ▶ Programm im funktionalen Stil:
 - ❖ Abstraktion:
Array als Instanz einer Sequenz
 - ❖ Higher-order Functions
statt Schleifen

```
object Test2 {  
    def main( args: Array[String] ) {  
        println( args  
                  .map( _.toUpperCase )  
                  .mkString( " " ) )  
    }  
}
```

► Scala denkt mit:

- ❖ Folgerung von Typen ("type inference")
- ❖ kompakte Class-Syntax
- ❖ erweiterbare APIs
- ❖ Closures als Kontroll-Abstraktionen

```
var capital = Map( "US" -> "Washington",  
                  "France" -> "paris",  
                  "Japan" -> "tokyo" )  
  
capital += ( "Russia" -> "Moscow" )  
  
for( (country,city) <- capital )  
    capital += (country -> city.capitalize)  
  
assert( capital("Japan") == "Tokyo" )
```

LOC Scala : Java: $\leq 50\%$

- ▶ Code nutzt Abstraktion der Bibliothek - keine spezielle Syntax
- ▶ Vorteil: Bibliotheken sind erweiterbar und geben Kontrolle
- ▶ Fehler werden durch die statische Typisierung früh erkannt

```
import scala.collection.mutable._

val capital = new HashMap[String,String]
               with SynchronizedMap[String,String] {
               override def default(key: String) = "?"
               }

capital += ( "US" -> "Washington",
            "France" -> "Paris",
            "Japan" -> "Tokyo" )

assert( capital("Russia") == "?" )
```


- ▶ Scala ist "schlank" in seinen Sprachkonstrukten
- ▶ Typen lassen sich über Bibliotheken hinzufügen, als wären sie native Sprachelemente

- ▶ Scala generiert viele zusätzliche Konstrukte:
 - ❖ Weiterleitung von Methodenaufrufen
 - ❖ Hilfsobjekte
 - ❖ Innere anonyme Klassen
- ▶ Startzeiten sind z.T. länger aufgrund vieler class-Dateien
- ▶ JIT optimiert diese häufig weg, Laufzeit vergleichbar mit Java
- ▶ Scala ist z.T. schneller in der Ausführung, da die generierten class-Dateien gut zu optimieren sind

Scala in der Praxis

- ▶ Definitionen
- ▶ Functions / Closures
- ▶ Currying
- ▶ Neue Kontrollstrukturen
- ▶ Traits
- ▶ Case Classes / Pattern-matching
- ▶ Actors
- ▶ Tool Support

- ▶ class - entspricht Java class; Besonderheit: "case class"
- ▶ object - Singleton class (kein static wie in Java)
- ▶ trait - entspricht Java Interface
 - ❖ kann Implementierungen enthalten

- ▶ var

```
var mutable = "String reference"
mutable = "String reference changed"
```

- ▶ val

```
val immutable = "fixed String reference"
```

- ▶ lazy val

```
lazy val lval = {
  println( "init" )
  "lazy String reference"
}
```

- ▶ def

```
def function( arg: String ): String = {
  "Hello " + arg
}
```

► First-class functions

- ❖ Funktionen sind Objekte, können übergeben und Variablen zugewiesen werden

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

► Higher-order functions: Funktionen als parameter

```
val list = List(-15, -8, -5, 0, 5, 8, 15)  
list.filter( _ < 0 )
```

► Closures binden "freie Variablen" zur Laufzeit

```
var more = 1  
val addMore = (x: Int) => x + more
```

► Funktionen mit mehreren Argument-Listen

```
def curriedSum(x: Int)(y: Int) = x + y  
curriedSum(1)(2)
```

► Illustration des Prozesses

```
def first(x: Int) = (y: Int) => x + y  
val second = first(1)  
second(2)
```

► Referenz auf die 2. Funktion über Platzhalter (partially applied functions)

```
val onePlus = curriedSum(1) _  
onePlus(2)
```

- ▶ First-class functions + Currying bieten neue Möglichkeiten, Kontrollstrukturen zu etablieren
- ▶ Beispiel: break / continue

```
object BreakLoop {  
  object Break extends RuntimeException  
  object Continue extends Exception  
  def break { throw Break }  
  def continue { throw Continue }  
  def whileTrue( condition: => Boolean )(block: => Unit) {  
    try {  
      while( condition ) try { block } catch { case Continue => }  
    } catch { case Break => }  
  }  
}
```

```
import BreakLoop._  
var i = 0  
whileTrue( i<10 ) {  
  i += 1  
  if( i<2 ) continue  
  if( i>=8 ) break  
  println( i )  
}
```


► Traits können Implementierungen enthalten

```
abstract class IntQueue {  
  def get: Int  
  def put( x: Int )  
}  
  
trait Doubling extends IntQueue {  
  abstract override def put( x: Int ) = super.put( 2 * x )  
}
```

► Traits können Klassen hinzugefügt werden

```
class StandardQueue extends IntQueue {  
  private val buf = new scala.collection.mutable.ArrayBuffer[Int]  
  def get = buf.remove( 0 )  
  def put( x: Int ) = buf += x  
}  
  
val queue = new StandardQueue with Doubling
```


► Case Classes

```
case class Number( value: Int )  
val n = Number(1)
```

► Case Classes geben Einblick in die Objektkonstruktion

► Pattern-matching

```
def isBig( n: Number ) = {  
  n match {  
    case Number(i) if i<5 => false  
    case _ => true  
  }  
}
```

- ▶ Actors sind "bessere" Concurrency-Konstrukte
- ▶ Message-passing concurrency

```
class Ping(count: Int, pong: Actor) extends Actor {  
  def act = { var pingsLeft = count  
    loop { react {  
      case "Start" | "Pong" => println( "Pong" )  
      if( pingsLeft>0 ) { pong ! "Ping"; pingsLeft -= 1 }  
      else { pong ! "Stop"; exit }  
    } }  
  }  
}  
  
class Pong extends Actor {  
  def act = loop { react {  
    case "Ping" => println( "Ping" ); sender ! "Pong"  
    case "Stop" => exit } }  
}
```

- ▶ Plugins sind derzeit für alle gängigen IDEs in Entwicklung
 - ❖ Eclipse-Plugin
<http://www.scala-lang.org/node/94>
 - ❖ Netbeans
<http://wiki.netbeans.org/Scala>
 - ❖ IntelliJ IDEA
<http://plugins.intellij.net/plugin/?id=1347>
- ▶ Maven
 - ❖ <http://scala-tools.org/mvnsites/maven-scala-plugin/>

Ausblick, Referenzen, Q & A

- ▶ Scala 2.8
 - ❖ Named/default parameters
 - ❖ Nested annotations (wichtig für z.B. JPA)
 - ❖ Continuations
 - ❖ Interactive Interpreter (REPL)
 - ❖ Verbesserung der Standardbibliothek

► Scala in the Web:

❖ Scala-Home:

<http://www.scala-lang.org/>

❖ Scala for Java Refugees:

<http://www.codecommit.com/blog/scala/scala-for-java-refugees-part-1>

► Books:

❖ „Programming in Scala“, Artima, ISBN 0981531601

❖ „Programming Scala“, Pragmatic Programmers, ISBN 193435631X

❖ „Beginning Scala“, APress, ISBN 1430219890

❖ „Praxisbuch Scala“, Hanser, ISBN 3446418636 (9/09)



- ▶ Mailinglisten

<http://www.scala-lang.org/node/199>

- ▶ IRC: [#scala@irc.freenode.net](http://irc.freenode.net)

sehr aktiv, Hilfe in allen Lagen

- ▶ Blog-Aggregation

<http://planetscala.com/>

Vielen Dank!