



---

Programming in  
**Scala**



---

artima

Martin Odersky  
Lex Spoon  
Bill Venners

Overschrijven zonder bronvermelding is plagiaat.

Overschrijven met bronvermelding is wetenschap.  
Programming Scala



## Inhoud

- **Historie**
- **Functioneel Programmeren**
- **Scala's eigenschappen: traits, patternmatching, implicits closures voorbeelden en vergelijkingen in Java.**
- **Conclusie**



## Scala Historie

- **Gestart in 2001 door Martin Odersky**
- **2004 gepubliceerd**
- **2006 nieuwe versie, huidige versie 2.0.7**



## Functioneel programmeren In Scala.

- **Wikipedia: functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data**
- **Programmeren zonder bijwerking, geen status verandering op scherm, file, database etc.**
- **Methods zijn first class values, closures.**



## Scala behalve Functionele Programmeertaal.

- **Zuiver Object Georiendeerd**
- **Compatible met Java.**
- **Compiler genereert bytecode.**
- **Interpreter functionaliteit**
- **Test functionaliteit**



## Voorbeeld (List)

- **Input: lijst met integers**  
**Output: lijst met alleen de even integers**

- **Scala**

```
val myList = List(1,2,3,4,8,4,2)
def isEven(x : Int):Boolean =
{ x % 2 == 0}
myList.filter(isEven)
myList.filter(x => x % 2 ==0}
myList.filter(_ % 2 ==0 }
```



## Vervolg Voorbeeld (List)

- `myList.map(_ => _*_)`
- `myList.sort((s1, s2) => s1 < s2)`



## Block Resultaat

- Elk code blok in Scala retourneert resultaat
- Normaal gesproken het laatste stmt.
- Val URL = try { new URL(path) }  
catch { case e : MalformedURLException  
=> new URL(<http://www.njug.org>); }



## Definities

- **val: immutable) val x : Int = 5**
- **var: mutable) var y : String = “jeroen”**
- **def: methods) def z (x : int) : Double =  
{ Math.sqrt(x.doubleValue) }  
methods zijn zgn “first class values”.**



## Method

- Een Scala method is vergelijkbaar met een Java method.
- Retourneert: of geen waarde (Unit), of 1 waarde, of een tuple.
  - def f = {"return"} geeft return
  - def f {"return"} geeft Unit (void)
  - def f = {"jeroen", 10}
  - val (a, b) = f // a en b zijn bekend.
- Methods en Methods



## Methods en Closures

- vgl Anonymous classes in java.
- Een method accepteert een andere method volgens de syntax:  
def apply(f: (T, ..) => S)  
class A (val meer: Int) {  
 def verhoog(i: Int) : Int = {c + meer;}}  
 def (f:(Int) => Int, x : Int) : Int = { f(x) \* x }  
 val a1 = new A(" ", 5);
- p(a1.c, 5)



## Visibility

- **Access modifiers werken ongeveer gelijk aan Java: public, protected, private**
- **Methods/class definities zijn default public**
- **Java's default modifier heeft geen scala equivalent (?)**



## Traits

- Traits zijn scala's interfaces
- **Mogelijkheid tot concrete methods!!**
- Toegankelijkheid van this, en eigen methods
- Thin interfaces alleen abstract methods
- Thick interfaces extensie op thin interfaces, met concrete methods



## Traits voorbeeld

- **Comparators ==, <, >, <=, >= ,!=**
- **We maken hier een interface van.**
- **..**
- **En classes die bovenstaande intf implementeren...**
- **We gaan een Scala trait maken:**



## Traits vervolg voorbeeld

- **trait CompareTrait [T] {  
 def compare(that : T) : Int  
 def < (that : T) : Boolean = (this  
 compare that) < 0  
 def <> (that: T) : Boolean = (this  
 compare that) != 0  
 ..etc..**



## Traits toegepast

### Leningen. (Rente en looptijd)

- ```
class Lening (r :double, l : int) extends (Object with)
CompareTrait[Lening] {
  compare(that : Lening) : Int {
    l / r compare that.l / that.r
  }
}
val l1 : Lening = new Lening(5.2, 10)
val l2 : Lening = new Lening(2.6, 6)
l2 < l1 en alle andere comparators gratis
```

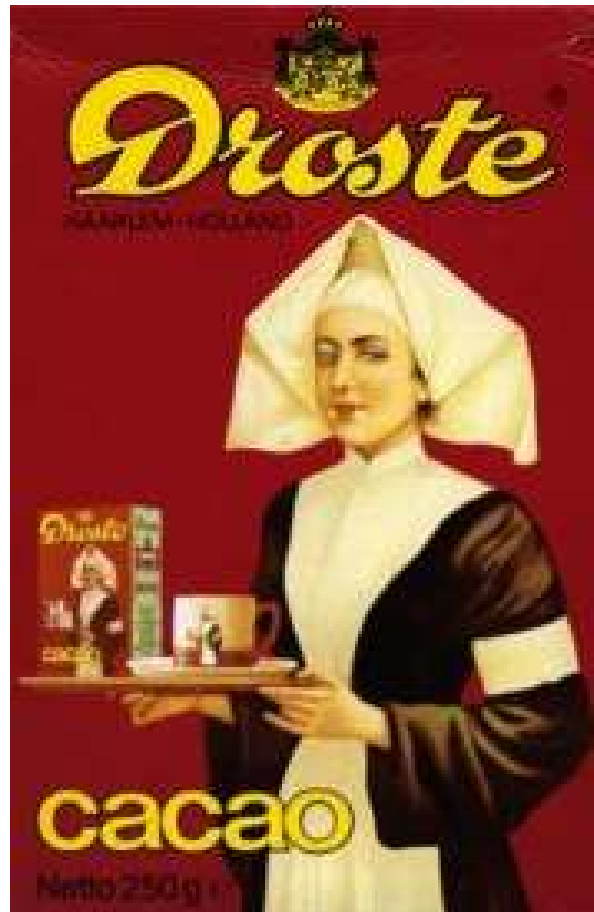


## Case Classes & PatternMatching

- **Boom of recursieve datastructuren in zijn belangrijke usecases in FP**
- **Onderscheid maken in soort data (Java inheritance)**
- **Onderscheid maken in combinaties van soorten data (Java method overriding, Scala case classes)**



## Recursief patroon: Droste effect



Programming Scala



## Case Classes

- **Voorbeeld: rekenkundige expressie**
- **Expressie kan een variabele, een nummer of een expressie zijn.**
- **2 mogelijkheden: expressie met 1 argument (sin expr, wortel expr)**
- **Expressie met 2 argumenten  $\text{expr1} + \text{expr2}$**



## Case Classes

- **abstract class Expr**  
**case class Var (name: String) extends Expr**  
**case class Num (num: Double) extends Expr**  
**case class Op1 (operator: String**  
**, exp:Expr) extends Expr**  
**case class Op2 (operator: String, left : Expr**  
**, right : Expr) extends Expr**



## Case Classes

**case maakt:**

- **new overbodig.**
- **alle argumenten fields (val).**  
**val op = Op2("+", Var("x"), Num(2))**  
**op.operator**
- **print, hashCode en equals gratis (ook in boom structuur)**
- **Pattern Matching mogelijk.**



## Pattern Matching

- Expressies versimpelen (oplossen)
- ```
def versimpel(expr : Expr) : Expr =  
  expr match {  
    case Op1("-", Op1("-", e)) => e  
    case Op2("+", Number(0), e) => e  
    case Op2("*", Number(1), e) => e  
    case _ => e  
  }
```



## Pattern Matching

**Java's switch? Niet helemaal!**

- **Niet veroordeeld tot primitives.**
- **Geen break.**
- **match is een expressie en geeft een resultaat**
- **Geen match geeft een MatchError!**



## cc & PM vervolg

- **case List(0,\_) => “match”**  
een lijst beginnend met een 0
- (“een”, “lijstje”, true) match {  
  **case (var1, var2, var3) => var1, var2 en var3 zijn hier**  
  bruikbaar (tupel)
- **Patterns zijn lineair:**  
**case Op2(“+”, x, x) => Op2(“\*”, Num(2), x)**  
**case Op2(“+”, x, y) if x == y => Op2(“\*”, Num(2), x)**



## Implicits Hemels

- Perl x operator: “- ” x 5: “- - - - - “
- Scala Helaas! “- “ x 5: **value x is not a member of java.lang.String**
- ```
class StringHelper(orig: String) {  
  def x (aantal: Int) = {  
    def maal (aantal: Int): String = {  
      if(aantal > 1) orig + maal(aantal - 1)  
      else orig  
    }  
    maal(aantal)}}
```



## Implicits Hemels (2)

- `val c = new StringHelper("- ");`  
`c x 5: "- - - - -";` (We zijn er bijna!)
- `implicit def string2StringHelper(orig: String) = new HunkemollerSchap(orig);`
- `"- " x 5: "- - - - -"`

(bron [scalada.blogspot.com](http://scalada.blogspot.com))



## Implicits Duivels

- **Implicits altijd lokaal gebruiken, extra toevoegingen zijn lastig te achterhalen!**
- **Let op je return statements:  
StringHelper.x retourneerde een String,  
maar wat als het een (gevulde)  
StringHelper retourneerde?  
“- “ x 5 == “- . . . . - “; ??**



## Scala caveat auditor

- **Taal is in ontwikkeling, wacht altijd met downloaden van de nieuwste alpha versie, vaak een of 2 revisies**
- **IDE Support is bedroevend**
- **Maven builds draaien soms niet**
- **Wilde mailing list**
- **Documentatie in ontwikkeling**
- **Vreemde implementatie (Tuple1, Tuple2, etc).**
- **Onduidelijk wanneer je types moet opgeven.**



## Meer over Scala

- **Liftweb**
- [www.artima.com](http://www.artima.com)
- **Code.google.com**
- [www.google.com](http://www.google.com)
- [www.parleys.com](http://www.parleys.com)
- **Jeroen@dijkmeijer.com**



## Conclusie

- **Scala bundelt de voordelen van een functionele programmeertaal en zuiver object georiënteerde programmeertaal in één taal met behoud van alle bestaande java libraries.**
- **Niemand kan meer om Scala heen!**



# Vragen / Opmerkingen

Programming Scala