Scala 2.8.0 – Was gibt's Neues?

Prof. Dr. Oliver Braun
o.braun@fh-sm.de

Fakultät Informatik
Fachhochschule Schmalkalden

01. Juli 2010

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Überblick

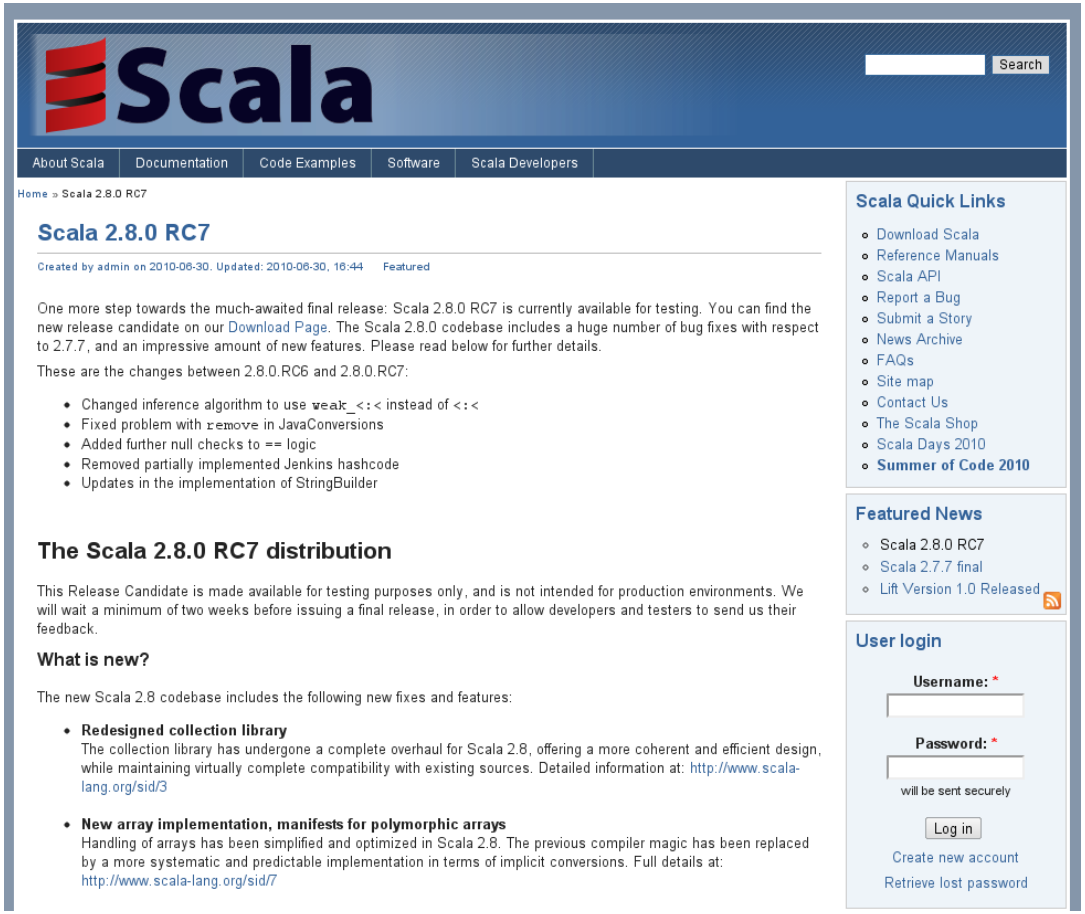
- ursprünglich nur Collections neu schreiben
- 18 Monate Entwicklungszeit
- parallel viele andere Erneuerungen
  - bessere IDE-Unterstützung
  - neue Tools: `scalap`, `scaladoc2`
  - Typ-Inferenz für Typ-Konstruktoren
  - verbesserte Actors
  - Unterstützung für Continuations
  - ...
- Martin Odersky auf den Scala Days 2010 (sinngemäß):

es wäre eigentlich 3.0 angebracht
aber 2.8 ist schon announced
und wird bereits in Büchern etc. referenziert

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Inhalt

- Named Arguments
- Default Arguments
- Scala 2.8 Collections
- Package-Objekte
- Arrays in Scala 2.8
- Typ-Spezialisierung
- Nested Packages
- Scaladoc 2
- Scala IDE for Eclipse



FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Named Arguments

- Parameter in Funktionsdefinitionen haben Namen, z.B.

```
def f(a: Int, b: Boolean) = if (b) a else 0
```

- diese Namen können ab Scala 2.8.0 nicht nur innerhalb der Funktion, sondern auch beim Aufruf der Funktion in der Parameterliste genutzt werden, z.B.

```
f(a = 7, b = false)
f(b = true, a = 12)
f(18, b = true)
```

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Named Arguments (2)

- nicht erlaubt

```
f(b = true, 1) // error: positional after named argument
f(true, a = 2) // error: parameter specified twice: a
```

- (fast) keine Verwechslung mit Zuweisungen möglich

```
var a = 5
f(a = a + 1, b = false) // named argument
```

  eine Zuweisung der Form `a = a + 1` hat den Typ `Unit`

- bei *by-name*-Argumenten mit dem Ergebnistyp `Unit` kann es Probleme geben $\rightsquigarrow$ Compiler-Error

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Default Arguments

- bei der Definition einer Funktion können *Default Arguments* angegeben werden, z.B.

```
def f(a: Int, b: Boolean = true) =
        if (b) a else 0
```

- damit kann das Argument beim Aufruf weg gelassen werden

```
f(7)
f(a = 12)
```

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Default Arguments (2)

- Named und Default Arguments zusammen machen einfache Überladung obsolet

```
class Rational(val numer: Int = 0,
               val denom: Int = 1)
```

- 4 „Konstruktoren“

```
new Rational()          // 0/1
new Rational(2)         // 2/1
new Rational(denom = 2) // 0/2
new Rational(5,3)       // 5/3
```

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Default Arguments (3)

- Achtung: Default Arguments ersetzen nicht die Implementierung überladener Methoden
- Beispiel:

```
trait MyTrait {
  def f(i: Int): Double
  def f(i: Int, d: Double): Double
}
class MyClass extends MyTrait { // error
  def f(i: Int, d: Double = 1) = i * d
}
```

- führt zu

```
error: class MyClass needs to be abstract,
    since method f in trait MyTrait of
        type (i: Int)Double is not defined
```

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

copy-Methode für Case-Klassen

Für Case-Klassen wird copy-Methode dank Named und Default Arguments automatisch generiert.

Beispiel:

```
case class Conference(name: String, year: Int)

val jfs2010 = Conference("Java Forum Stuttgart",
                         2010)

val jfs2011 = jfs2010 copy (year = 2011)
              // ⇝ Conference(Java Forum Stuttgart,2011)
```

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

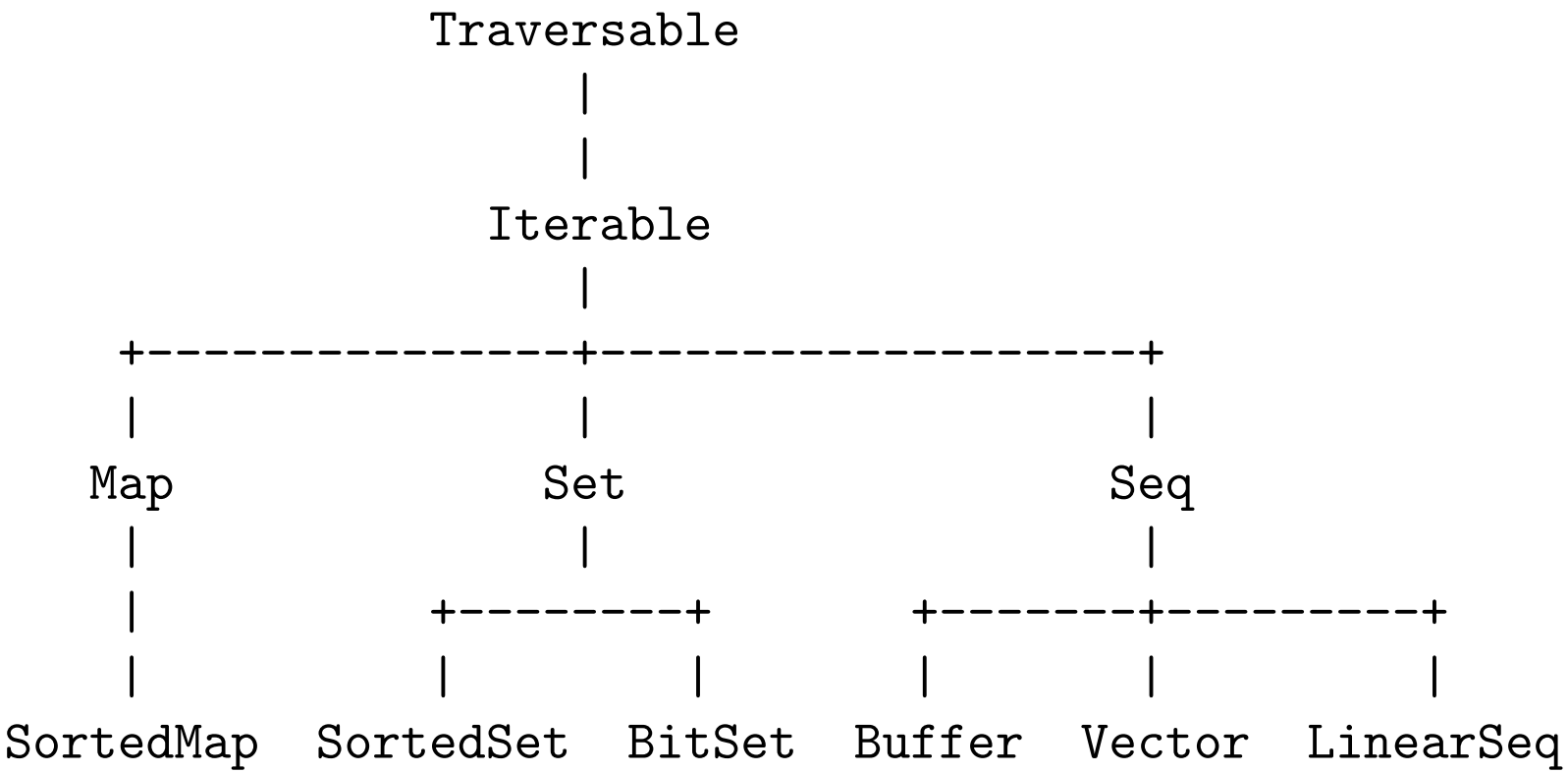
Scala 2.8 Collections

- mit 2.8.0 Redesign der Scala-Collections-Bibliothek
- alle Collection-Klassen sind in `scala.collection`
- die meisten Basis-Klassen[1] existieren in 3 Formen
  1. in `scala.collection.immutable`
  2. in `scala.collection.mutable`
  3. in `scala.collection`
- einzige Ausnahme ist `Buffer`-Trait (immer mutable)
- die Klassen in `scala.collection` haben das selbe Interface wie die entsprechende in `scala.collection.immutable`
  - ...sind aber nicht garantiert unveränderbar
- die Klassen in `scala.collection.mutable` haben zusätzlich noch Methoden die den Zustand verändern können

[1] siehe nächste Folie

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Die wichtigsten Collection-Basisklassen



Erzeugen einer Instanz

- ...durch Angabe des Klassennamens und der Elemente in Klammern, z.B.

```
Traversable(1, 2, 3)
Map("B" -> "Berlin", "S" -> "Stuttgart")
Set("Java", "Forum", "Stuttgart")
```

- genauso für spezifische Implementierungen

```
List(1, 2, 3)
HashMap(1 -> "one", 2 -> "two")
```

- Repräsentation als Zeichenkette mit `toString` auf die gleiche Weise

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Seq

- Sequenzen sind partielle Funktionen von `Int` zu dem Elementtyp, beginnend bei `0`
- Sequenzen definieren eine Methode `apply`[2] zum Indizieren
- Beispiel

```
val list = List("Hallo","Welt")
list(1) // ⇝ Welt
```

[2] Ein Ausdruck der Form `<objectname>(<parameterlist>)` wird in Scala immer expandiert zu `<objectname>.apply(<parameterlist>)`

Set

- `apply` ist identisch zur `contains`-Methode, d.h.

```
val set = Set(1,2,3)
set(2) // ⇝ true, entspricht set.contains(2)
set(4) // ⇝ false, entspricht set.contains(4)
```

- Elemente hinzufügen oder entfernen
  - immutable: `Set(1,2) + 3` und `Set(1,2) - 2`
  - mutable: `set += 3` und `set -= 2`
- Argumentliste mit mehreren Argumenten

```
val set2 = set - (3,2) // ⇝ Set(1)
```

- Mengenoperationen wie Vereinigung (`union`, `|`), Schnitt (`intersect`, `&`), Mengendifferenz (`diff`, `&~`)

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Map

- `apply` gibt Wert zurück oder wirft Exception

```
map(key)  // ⇝ value oder Exception
```

- Lookup-Methode `get` nutzt den `Option`-Datentyp[3]

```
def get(key): Option[Value]
```

- im Wesentlichen gleiche Methoden wie `Set`: `+`, `-`, `+=`, `-=`, ...

[3] `Option[A]` hat die beiden Werte `None` und `Some(x)` für ein `x: A`

Map (2)

- Updates mit verschiedener Syntax möglich[4]

```
map(5) = "five"
map.update(5, "five")
map += (5 -> "five")
```

- für immutable Maps

```
map.updated(5, "five")
map + (5 -> "five")
```

[4] Ein Ausdruck der Form
`<objectname>(<parameterlist>) = <expression>`
wird in Scala immer expandiert zu
`<objectname>.update(<parameterlist>, <expression>)`

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Migration zu neuen Collection-Klassen

- die in Scala 2.8.0 neu eingeführten Package-Objekte erleichtern Migration

```
package object scala {
  type List[+A] =
       scala.collection.immutable.List[A]
  val List =
       scala.collection.immutable.List
  // ...
}
```

- das Scala-Package-Objekt muss in `scala/package.scala` gespeichert werden
- die dort definierten Member sind dann Teil des Packages
- damit gibt es neben `scala.collection.immutable.List` auch `scala.List`

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Arrays in Scala

- Spannungsfeld
  - Interoperabilität mit Java / Effizienz von Java-Arrays

  vs.
  - die Vielzahl von Methoden der Scala-Collections soll auch mit Arrays nutzbar sein
- Scala bis 2.7.x nutzt Boxing / Unboxing / Compiler Magic

  $\rightsquigarrow$ Probleme und zum Teil schlechte Performanz
- auf Scala-Arrays konnten zwar viele Collection-Methoden angewendet werden, aber das Ergebnis war kein Array mehr
- gleiches Problem mit `String` und `RichString`

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Scala 2.8 Arrays

- entsprechen Java Arrays
- implizite Umwandlung in `ArrayOps` für die Collection-Methoden $\Rightarrow$ geben Arrays zurück
- moderen VMs können die impliziten Konversionen eliminieren
  $\rightsquigarrow$ Overhead nahe Null
- zweite implizite Umwandlung in „echte“ `Seq`: `WrappedArray`
- Auswahl von überladenen Methoden und Implicits in 2.8.0 liberalisiert:
  $\rightsquigarrow$ Priorisierung von Implicits
- Analog: `StringOps` und `WrappedString`

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Generische Arrays

- Java erlaubt keine Typparameter im Zusammenhang mit Arrays
- Scala erlaubt aber beispielsweise

```
new Array[T] // für T ist Typparameter
```

⇒ benötigt Runtime-Information über `T`

- Mechanismus heisst `Manifest`
- `Manifest[T]` kann für bekannte Typen vom Compiler generiert werden und wird als impliziter Parameter übergeben
- abgeschwächte Version `ClassManifest` kann erzeugt werden wenn nur Top-Level-Klasse eines Typen bekannt ist (reicht für Arrays)

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Generische Arrays (2)

- Beispiel

```
def listToArray[T](list: List[T])
      (implicit m: ClassManifest[T]) = {
  val xs = new Array[T](list.length)
  for (i <- 0 until list.length) xs(i) = list(i)
  xs
}
```

- kürzer mit Context Bound

```
def listToArray[T: ClassManifest]
                (list: List[T]) = { // ...
```

Generische Arrays (3)

- Funktion die `listToArray` nutzt und selbst Typparameter hat muss auch `Manifest` bieten:

```
def mkArray[T: ClassManifest](x: T*) =
  listToArray(x.toList)
```

- `GenericArray` ist immer Java-Referenz-Array und braucht daher kein `Manifest`

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Typ-Spezialisierung

- Scalas parametrischer Polymorphismus basiert auf Type Erasure
- für primitive Typen heisst das Un-/Boxing
  - `Unit`, `Boolean`, `Byte`, `Short`, `Char`, `Int`, `Long`, `Float`, `Double`

  $\Rightarrow$ ca. 10 mal langsamer

  $\Rightarrow$ Programmierer nehmen keine generischen Collections

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Typ-Spezialisierung (2)

- Ausweg: Type Specialization in Scala 2.8

```
class Vector[@specialized A] {
  def apply(i: Int): A = // ...
  def map[@specialized(Int, Boolean) B]
         (f: A => B) = // ...
}
```

- Compiler erzeugt generische Klasse `Vector` und spezialisierte für jeden primitiven Typ
- `map` wird für `Int` und `Boolean` spezialisiert

FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Nested Packages

- Java hat nur absolute Packages, in Scala können Packages verschachtelt werden
- Pre-2.8:

```
package net.obraun
import java.util.Scanner
```

  schlägt fehl, wenn es ein Sub-Package `net.java` gibt
- 2.8:

```
package net.obraun
```

  sucht **nicht** in `net`
-

```
package net
package obraun
```

  sucht in `net`

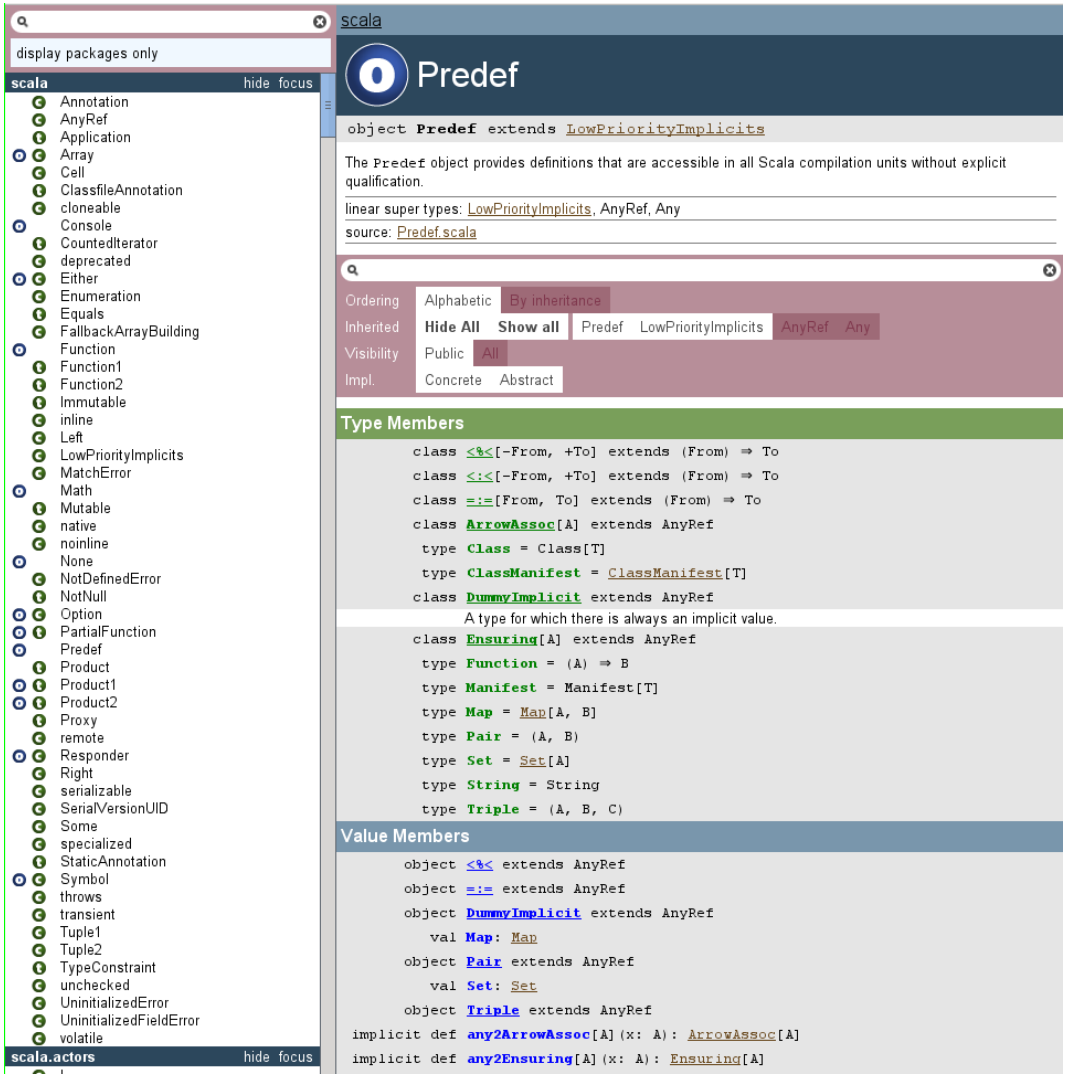
FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Neues Scaladoc

- moderneres Layout
- Tags, wie in Javadoc
  - `@author`
  - `@param`
  - `@return`
  - `...`
- Wiki-Syntax in Sourcecode-Kommentaren
- Makros

  `@define <name> <body>`

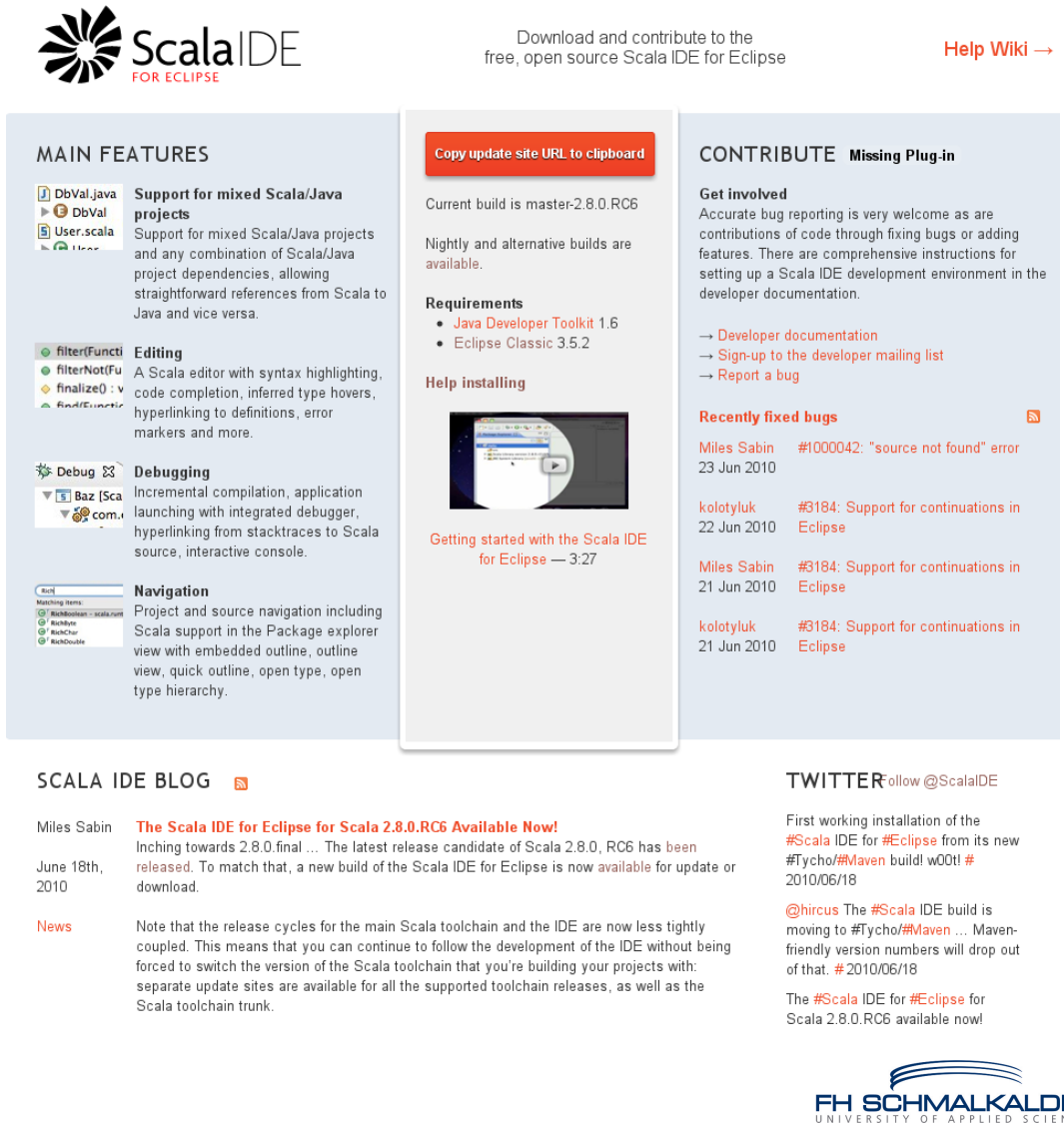
  nutzbar als `$name`



FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Eclipse-Plugin

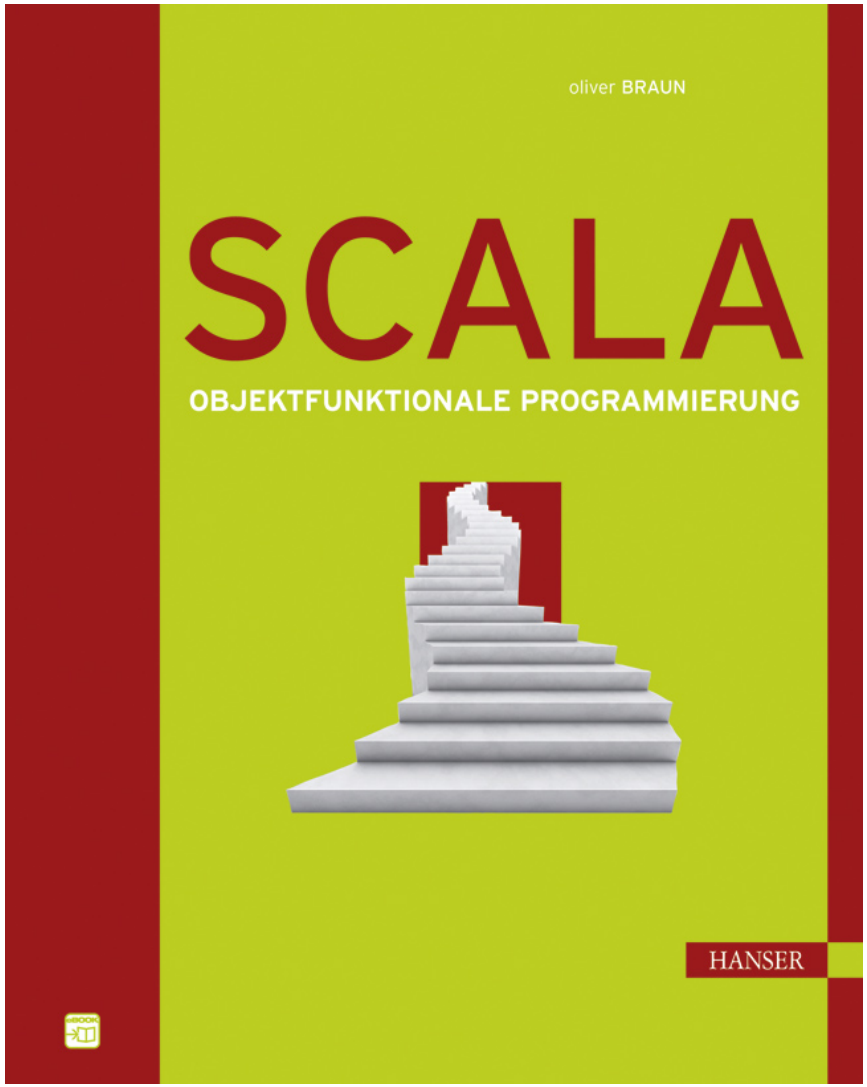
- gemischte Scala/Java-Projekte
- Syntax Highlighting
- Code-Completion
- Hyperlinks zu Definitionen
- Error Markers
- Debugging
- ...



FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES

Vielen Dank für die Aufmerksamkeit – Fragen?

- 02.12.2010
- Die Sprache Scala
- Die Tools
  - Interpreter / Compiler
    ...
  - Simple Build Tool
  - ScalaDoc / ScalaCheck / ScalaTest / ...
- Die Frameworks
  - Lift
  - Akka
  - ...



FH SCHMALKALDEN
UNIVERSITY OF APPLIED SCIENCES