SCALA 3
GEFÜHL UND HÄRTE

Stefan López Romero - MaibornWolff GmbH


TELEFUNKEN
HERGESTELLT BEI DER TELDEC »TELEFUNKEN-DECCA« SCHALLPLATTEN GMBH., HAMBURG 19, GERMANY
TELE FUN KEN
LC 0366
GEMA
6.24 833-01-1
℗ 1981
6.24 833
Seite 1 STEREO
GEFÜHL UND HÄRTE
SCALA 3
1. Metropolitana (2:08)
2. Schizo-Kids (4:05)
3. Ghetto Intern (2:25)
4. Travestie (3:45)
5. ZX (4:30)
(1.–5.: Schumann - Thurley - Lengauer)
Aufgenommen im IC Studio, Februar 1981
Gemischt von SCALA 3, Profenius,
Klaus Schulze
Produzent: Klaus Schulze
ALLE URHEBER- UND LEISTUNGSSCHUTZRECHTE VORBEHALTEN. KEIN VERLEIH!
33
KEINE UNERLAUBTE VERVIELFÄLTIGUNG, VERMIETUNG, AUFFÜHRUNG, SENDUNG!

ÜBER MICH

- IT Architekt MaibornWolff Gmbh
- Muttersprache: Java
- Hobby: FP mit Scala, Haskell, Kotlin...

ZIEL

- Die wichtigsten Scala 3 Features zeigen
- Lust auf Scala machen
- **Keine** vollständige Liste an Feature
- **Keine** Scala Einführung

AGENDA

- Intro - Was ist Scala 3
- Neue Features
- Outro - Persönlicher Eindruck

INTRO

SCALA 3 - CODENAME DOTTY

- Start 2013: Projekt Dotty
- Dezember 2019: Feature Complete
- Februar 2021: Erster Release Candidate

WAS BRINGT SCALA 3

- Neue theoretische Basis: DOT-Kalkül
- Compiler rewrite
- TASTY: Kompatibles Zwischenformat
- Viele neue Features

ZIELE DER NEUEN FEATURES

- Mächtige Konstrukte bändigen
- Unstimmigkeiten und Fallstricke entfernen
- Konsistenz und Ausdruckskraft verbessern
- Bestehende Lücken füllen

NEUE FEATURES

TOP LEVEL DEFINITIONS

- Es sind keine Wrapper Objekte mehr notwendig
- Package Objects werden dadurch obsolet

```
val greeting = "Dear"

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

def name(p:Person) : String = s"$greeting ${p.firstName} ${p.last
```

CREATOR APPLICATIONS

in Scala 3 kann man Klassen ohne das Keyword `new` instanzieren, auch wenn sie keine *apply*-Methode haben.

```
val sb = StringBuilder("abc")
```

@MAIN FUNCTIONS

Folgendes

```
def main(args: Array[String]): Unit =
    println(s"Hello World")
```

ist nicht mehr notwendig

```
@main def hello(): Unit =
    println(s"Hello World")
```

funktioniert auch mit Argumenten

```
@main def sayHello(name: String, age: Int): Unit =
    println(s"Hello $name, you are $age years old")
```

INDENTATION-BASED / BRACE-LESS SYNTAX

Einrückungen und neue Keywords ersetzen Klammern

```
object Algorithm:

    def calc(a: Int, b: Int) : Int =
        var res = a * b
        var c = a
        while c <= b do
            res * b
            c = c + 1
        res

    @main def testCalc(a: Int, b: Int) =
        val result = calc(a, b)

        if result > 1000 then
            println("result is greater than 1000")
```

OPAQUE TYPE ALIASES

Opaque Typ-Aliase bieten eine Typ-Abstraktion ohne jeglichen Overhead

```
object OpaqueType:

  opaque type Nat = Int

  object Nat:
    def apply(d: Int): Nat =
      if(d >= 0) then d
        else throw Exception("Nat must be positiv")


  @main def testOpaque() =
    import OpaqueType._
    val n = Nat(1)
    val i: Int = n //error: found OpaqueType.Nat, required Int
```

ENUMS

Neues Sprach-Konstrukt zur Definition von Enums.

```
enum State: /* extends java.lang.Enum[State] */
    case Solid, Liquid, Gas, Plasma
```

Enum-Methoden

```
scala> State.Solid.ordinal
val res2: Int = 0

scala> State.values
val res0: Array[State] = Array(Solid, Liquid, Gas, Plasma)

scala> State.valueOf("Gas")
val res1: State = Gas
```

ENUMS

Parametrisierte Enums

```
enum Emoticons(icon: String):
    case Smile extends Emoticons(":)")
    case Angry extends Emoticons(":(")
    case Kiss extends Emoticons(":*")
```

Enums als ADTs

```
enum Option[+T]:
  case Some(x: T) : extends Option[T]
  case None       : extends Option[Nothing]
```

EXTENSION METHODS

Definition

```
object StringExtensions:
  extension (str: String)
    def toCamelCase: String =
      str.toLowerCase.split("\\s").foldLeft("")((acc, elem) =>
        s"$acc${elem.substring(0,1).toUpperCase}${elem.substring(
```

Nutzung

```
@main def toCamelCase(str: String) : Unit =
    import StringExtensions.*
    println(str.toCamelCase)
```

UNION TYPES

Werden durch die Notation `A | B` definiert

```
def size(number: String | Int): Int =
    number match
        case s: String => s.size
        case n: Int => n

val numberSize  = size(1234)   //1234
val stringSize  = size("1234") //4
```

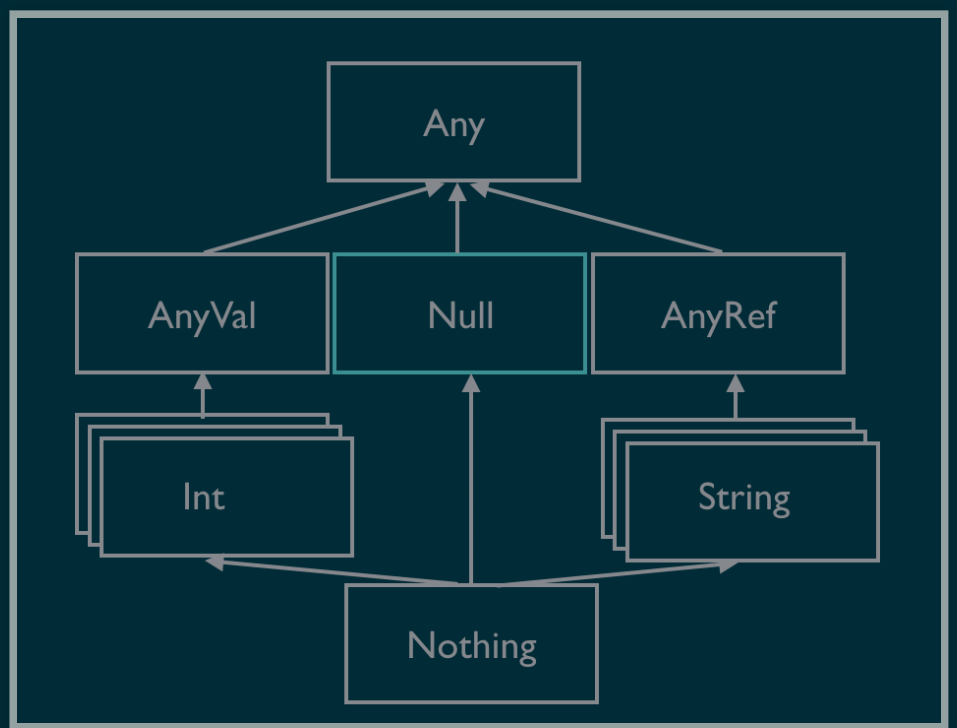
INTERSECTION TYPES

Werden durch die Notation `A&B` definiert

```
trait UpperCaseable:
  def uppercase(s: String) : String =  s.toUpperCase

trait Reversable:
  def reverse(s: String) : String = s.reverse

def convert(in: String, c: UpperCaseable & Reversable) : String =
    c.uppercase(c.reverse(in))

class Converter
  extends UpperCaseable with Reversable

@main def testIntersection() =
    println(convert("racecar", Converter())) //RACECAR
```

EXPLICIT NULLS

- opt-in Feature
- Modifiziert das Typ-System
- Referenz-Typen werden non-nullable



EXPLICIT NULLS

```
def reverse(string: String|Null ): String =
        string match
            case s: String => s.reverse
            case null => ""

def upperReverse(string: String): String =
    val res: String|Null = string.toUpperCase
    if res != null then res.reverse else "" //flow typing

@main def test(): Unit =
    val r1 = reverse(null)
    val r2 = reverse("racecar")
    val r3 = upperReverse(null) // Found: Null Required: String
```

IMPLICITS TURN INTO GIVENS


GREAT POWER COMES GREAT RESPONSIBI

GIVEN INSTANCES

```
trait Monoid[T]:
    def combine(a: T, b: T) : T
    def unit : T
```

```
given sumMonoid: Monoid[Int] with
    def combine(a: Int, b: Int) : Int =
        a + b
    def unit : Int = 0

given strMonoid: Monoid[String] with
    def combine(a: String, b: String) : String =
        a + b
    def unit : String = ""
```

Das Keyword `given` soll die Intension besser greifbar machen

USING CLAUSES

```
object MonoidOps
    def reduce[T](x: List[T])(using m: Monoid[T]) : T =
        x.foldLeft(m.unit)(m.combine)
```

Compiler sucht im Scope nach einer `given`-Instance mit dem passenden Typ

GIVEN IMPORT

```
@main def monoidTest(): Unit =
import Monoids.{given, *}
println(reduce(List(1,2,3,4))) //10
println(reduce(List("Hello", " World"))) //Hello World
```

Verhindert, dass durch einen Wildcard import versehentlich `given`-Intances in den Scope gelangen

UND NOCH VIELES MEHR

- Context functions
- Match types
- Type lambdas
- Dependent Function types
- Kind Polymorphism
- ...

Siehe: Scala3 Documentation

OUTRO

ZUSAMMENFASSUNG

- Scala 3 erleichtert den Einstieg in die Sprache
- Viele Verbesserungen und neue Features für die tägliche Arbeit
- Die meisten neuen Konstrukte existieren erstmal parallel zu den alten
- Mit TASTY als Zwischenformat ist eine sanfte Migration möglich
- Scala 3 compiler kann Scala 2 Code migrieren

DANKESCHÖN :) UND...