Hätte, hätte Fahrradkette

Falk Sippach, Mike Sperber

Created: 2025-07-07 Mon 17:00

Falk Sippach

falk.sippach@embarc.de

- Softwarearchitekt bei embarc
- Trainer für iSAQB Foundation und Advanced
- (Co-)Organisator von JUG Darmstadt, CyberLand, SpeaKonf, Tech2Talk
- JavaLand, Java Forum Nord, XP-Days, ...

@ctive group embarc

Mike Sperber

- Geschäftsführer Active Group GmbH
- Universität Tübingen 1995-2003, Bereich Programmiersprachen und Übersetzerbau

`@sperbsen@discuss.systems`

@active group

embarc

Active Group GmbH

- Tübingen
- Projektentwicklung, Beratung, Training
- Scala, Clojure, F#, Haskell, OCaml, Erlang, Elixir, Swift
- iSABQ Foundation, FUNAR, FLEX, DSL, FM

Blog https://funktionale-programmierung.de

Konferenz https://bobkonf.de/

@ctive group embarc

Warum sind wir hier? (F)

Java ist 30

@active group

embarc

Warum sind wir hier? (F)

Java ist 30

- Wie ist Java entstanden?

@active group

embarc

Warum sind wir hier? (F)

Java ist 30

- Wie ist Java entstanden?
- Was ist nicht gut gelöst in Java?

Warum sind wir hier? (F)

Java ist 30

- Wie ist Java entstanden?
- Was ist nicht gut gelöst in Java?
- Wie entwickelt es sich gerade weiter?

Warum sind wir hier? (F)

Java ist 30

- Wie ist Java entstanden?
- Was ist nicht gut gelöst in Java?
- Wie entwickelt es sich gerade weiter?
- Was fehlt in Java, was könnte noch kommen?

Meilensteine von Java (F)

- 1995: 1.0
- 2004: 5.0 (Generics)
- 2014: 8 (Lambdas, Stream API, ...)
- 2017: 9 (JPMS, ...)
- 2018: 10 (halbjährliches Releases)
- 2021: 17 (Pattern Matching, ...)
- 2023: 21 (Virtual Threads, ...)

Entwicklungslinien (M)

OOP Simula 67 -> Smalltalk 80 -> C++ (1985)

FP LISP (1960) -> ML (1973) / Scheme (1975) / HOPE (1980) Haskell/OCaml/F# (ab 1985)

PROC Algol 60 -> Pascal (1970) -> C (1972)

Worüber wir heute nicht reden (F)

- Generics (Java 1.5)
- Lambda (Java 8)
- Streams (Java 8)
- Modulsystem (Java 9)
- Typinferenz (Java 11)
- Virtual Threads (Java 21)

Daten

Java 8: Optional (F)

```
<A> Optional<Integer>
    elementIndex(List<A> list, A element) {

    for (int i = 0; i < list.size(); ++i) {
        if (list.get(i).equals(element))
            return Optional.of(i);
    }
    return Optional.empty();
}
```

Optional

Haskell

```haskell
data Maybe a = Nothing | Just a
```

Standard ML (SML/NJ 1993)

```
datatype 'a option = NONE | SOME of 'a
```

OCaml

```
type 'a t = 'a option =
| None
| Some of 'a
```

Records (F)

```
record Point(int x, int y) {}

Point p = new Point(1, 2);
System.out.println(p.x());
```

ALGOL W (1966)

Programming Languages

D. E. KNUTH, Editor

A Contribution to the Development of ALGOL

NIKLAUS WIRTH
*Stanford University**
Stanford, California

AND

C. A. R. HOARE
Elliott Automation Computers Ltd.,
Borehamwood, England

ALGOL W

```
record Node (reference (Node) left, right)

record Person (string name; integer age; logical male;
reference (Person) father, mother, youngest offspring,
   elder sibling)
```

Alles über Daten und Typen, 1985

On Understanding Types, Data Abstraction, and Polymorphism

Luca Cardelli
AT&T Bell Laboratories, Murray Hill, NJ 07974
(current address: DEC SRC, 130 Lytton Ave, Palo Alto CA 94301)

Peter Wegner
Dept. of Computer Science, Brown University
Providence, RI 02912

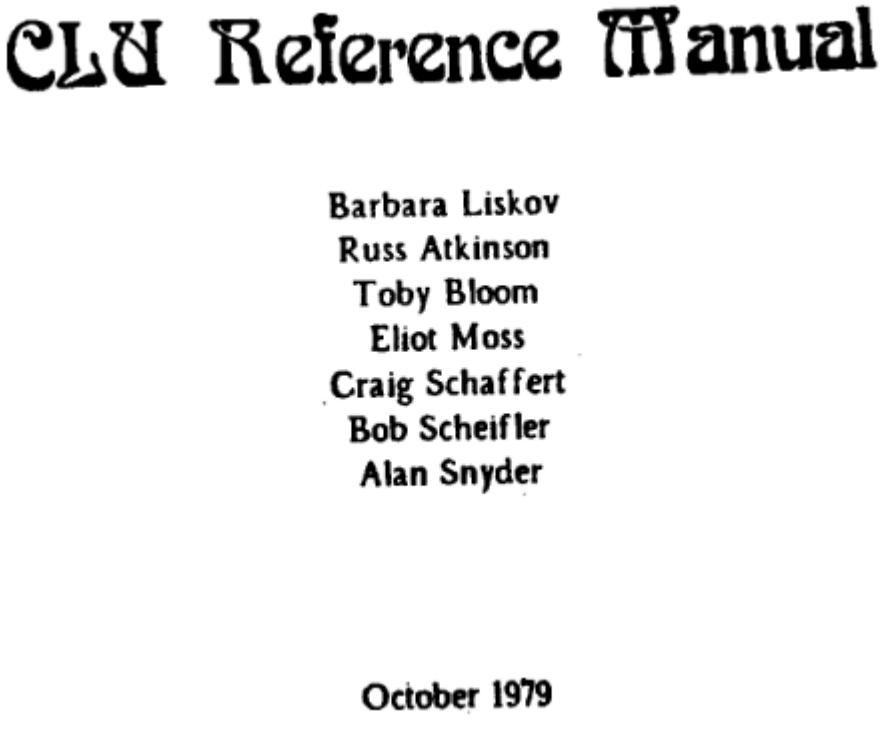
Sealed Classes (F)

```
sealed interface Animal {
    record Armadillo(Liveness liveness, Weight weight)
      implements Animal {}
    record Parrot(String sentence, Weight weight)
      implements Animal {}
}
```

Haskell

```haskell
data Animal =
    MkArmadillo { dilloLiveness :: Liveness,
                  dilloWeight :: Weight }
  | MkParrot { parrotSentence:: String,
               parrotWeight :: Weight }
  deriving Show
```

CLU



CLU Reference Manual

Barbara Liskov
Russ Atkinson
Toby Bloom
Eliot Moss
Craig Schaffert
Bob Scheifler
Alan Snyder

October 1979

https://pmg.csail.mit.edu/ftp.lcs.mit.edu/pub/pclu/CLU/3.Docum
LCS-TR-225.pdf

CLU (1979)

7.12 Oneof Types

A oneof type is a *tagged discriminated union*. A oneof is an immutable labeled object, to be thought of as "one of" a set of alternatives. The label is called the *tag*, and the object is called the *value*. A oneof type specification has the form

oneof [field_spec , ...]

where (as for records)

field_spec ::= name , ... : type_spec

Tags must be unique within a specification, but the ordering and grouping of tags is unimportant.

As an example of a oneof type, the representation type for an immutable linked list of integers, int_list, might be written

```
oneof[empty: null,
      pair:  struct[car: int, cdr: int_list]]
```

As another example, the contents of a "number container" might be specified by

```
oneof[empty:       null,
      integer:     int,
      real_num:    real,
      complex_num: complex]
```

HOPE (1980)

HOPE: AN EXPERIMENTAL APPLICATIVE LANGUAGE

R.M. Burstall
D.B. MacQueen[1]
D.T. Sannella

Department of Computer Science
University of Edinburgh
Edinburgh, Scotland

HOPE: Algebraische Datentypen

```
typevar alpha
```

which when used in a type expression denotes any type (including second- and higher-order types). A general definition of **tree** as a parametric type is now possible:

```
data tree(alpha) == empty ++ tip(alpha)
                  ++ node(tree(alpha)#tree(alpha))
```

Summen und Produkte

funktionale-programmierung.de

Ein Blog der Active Group

Datenmodellierung mit Summen und Produkten

25.11.2024 von Michael Sperber und Stefan Wehr

Für diesen Artikel gibt es auch eine englischsprachige Übersetzung.

Datenmodellierung ist oft ein unterschätzter Aspekt der Softwarearchitektur, spielt jedoch eine entscheidende Rolle, um nicht nur funktionale, sondern auch Nutzbarkeits- und Wartungsziele zu

https://funktionale-programmierung.de/2024/11/25/sums-products.html

@active group embarc

Java 12: Switch Expressions (F)

```
String developerRating( int numberOfChildren ) {
  return switch (numberOfChildren) {
    case 0 -> "open source contributor";
    case 1, 2 -> "junior";
    case 3 -> "senior";
    default -> {
      if (numberOfChildren < 0)
        throw new IndexOutOfBoundsException( numberOf… );
      yield "manager";
    }
  };
}
```

Java 15: Type Patterns (F)

```
private static boolean isNullOrEmpty(Object o) {
    return o == null ||
           o instanceof String s && s.isBlank() ||
           o instanceof Collection c && c.isEmpty();
}
```

Java 17: Pattern-Matching in switch (F)

```
String evaluateTypeWithSwitch( Object o ) {
  return switch(o) {
    case String s -> "String: " + s;
    case Collection c -> "Collection: " + c;
    default -> "Something else: " + o;
  };
}
```

```
boolean isNullOrEmptyWithSwitch( Object o ) {
  return switch(o) {
    case null -> true;
    case String s when s.isBlank() -> true;
    case String s -> false;
    case Collection c when c.isEmpty() -> true;
    default -> false;
  };
}
```

CLU

An example of a **tagcase** statement is

```
pair = struct[car: int, cdr: int_list]
x: oneof[pair: pair, empty: null]
   ...
while true do
   tagcase x
      tag empty:  return(false)
      tag pair (p: pair):  if p.car = i
                              then return(true)
                              else x := down(p.cdr)
                              end
      end
   end
```

Record-Patterns (F)

```
Animal runOver() {
    return switch (this) {
        case Armadillo(Liveness liveness, Weight weight) ->
            new Armadillo(Liveness.DEAD, weight);
        case Parrot(String sentence, Weight weight) ->
            new Parrot("", weight);
    };
}
```

LISP

COMPUTER AIDED MANIPULATION OF SYMBOLS

T H E S I S

submitted in fulfilment of

the requirements for the degree of

DOCTOR OF PHILOSOPHY

in the

QUEEN'S UNIVERSITY OF BELFAST

FREDERICK VALENTINE McBRIDE M.Sc.

MAY 1970

https://personal.cis.strath.ac.uk/conor.mcbride/FVMcB-PhD.pdf

McBride, 1970

d1: d[y;x] → 0 when np[y]

d2: d[x;x] → 1

d3: d[+[u;v];x] → +[d[u;x]; d[v;x]]

d4: d[*[u;v];x] → +[*[u;d[v;x]];*[v;d[u;x]]]

d5: d[/[u;v];x] → /[-[*[v;d[u;x]]; *[u;d[v;x]]]; ↑[v;2]]

d6: d[↑[u;v];x] → *[*[v;↑[u;-[v;1]]]; d[u;x]]when np[v]

@ctive group embarc

HOPE, 1980

```
module ordered_trees
  pubtype otree
  pubconst empty, insert, flatten

  data otree == empty ++ tip(num)
                      ++ node(otree#num#otree)

  dec insert : num#otree -> otree
  dec flatten : otree -> list num

  --- insert(n,empty)  <= tip(n)
  --- insert(n,tip(m))
                <= n<m then node(tip(n),m,empty)
                       else node(empty,m,tip(n))
  --- insert(n,node(t1,m,t2))
                <= n<m then node(insert(n,t1),m,t2)
                       else node(t1,m,insert(n,t2))

  --- flatten(empty)  <= nil
  --- flatten(tip(n)) <= [n]
```

```
--- flatten(node(t1,n,t2)) <= flatten(t1) <> (n::flatten(t2))
```

Java 21: Unnamed Patterns

```
Animal runOver() {
    return switch (this) {
        case Armadillo(Liveness _, Weight weight) ->
            new Armadillo(Liveness.DEAD, weight);
        case Parrot(String _, Weight weight) ->
            new Parrot("", weight);
    };
}
```

Java 23: Primitive Types in Patterns (F)

```
value = 10;
switch (value) {
    case byte _ ->
        System.out.println(value + " instanceof byte");
    case int _ ->
        System.out.println(value + " instanceof int");
}
```

Java ?: Value Objects (F)

Project Valhalla - "Codes like a class, works like an int."

```
value record PLZ(int zahl) {}
```

ALGOL W

```
record Node (reference (Node) left, right)

record Person (string name; integer age; logical male;
reference (Person) father, mother, youngest offspring,
   elder sibling)
```

Warum hat das so lange gedauert?

- OOA/OOD gescheitert
- "program to interfaces not implementations"
- Value Objects vs. Polymorphie

Tupel

```haskell
div_mod :: Integer -> Integer -> (Integer, Integer)
div_mod a b = (a `div` b, a `mod` b)
```

Either

```haskell
data Either a b = Left a | Right b

data ParseDigitError
  = NotADigit Char
  deriving Show

parseDigit :: Char -> Either ParseDigitError Int
parseDigit c =
  case c of
    '0' -> Right 0
    '1' -> Right 1
    ...
    '9' -> Right 9
    _ -> Left (NotADigit c)
```

Functional Update

```haskell
data Armadillo = MkArmadillo { dilloLiveness :: Liveness,
                               dilloWeight :: Weight }

runOverArmadillo dillo = dillo { dilloLiveness = Dead }
```

Nullable Types

Kotlin:

```
val b: String? = null

val l1 = b?.length ?: 0

val l2 = b!!.length
```

@ctive group embarc

Nullable Types - gcc 2.95

4.9 Conditionals with Omitted Operands

The middle operand in a conditional expression may be omitted. Then if the first operand is nonzero, its value is the value of the conditional expression.

Therefore, the expression

```
x ? : y
```

has the value of x if that is nonzero; otherwise, the value of y.

This example is perfectly equivalent to

```
x ? x : y
```

Schlussworte

Woher

Generics	ML	FP	1977
Lambda	LISP	FP	1959
Streams/map	LISP	FP	1960
Typinferenz	ML	FP	1977
Records	ALGOL W/ML	PROC/FP	1966
Sealed Interfaces	HOPE	FP	1980
Virtual Threads	Scheme	FP	~1980
Value Objects	F#/.NET	FP	2001

Was kommt noch (F)

- Tupel, Either
- *richtige* algebraische Datentypen
- functional update
- Value Objects
- `List<int>`
- nullable types
- First-Class-Continuations
- Tail Calls

Yesterday's Features for Today's Programmer

- FP FTW
- Clojure FTW
- Scala FTW

Vielen Dank

Fragen

@active group

embarc