Konzepte & Nutzen von λ-Ausdrücken in Java 8

Christoph Benzmüller
FU Berlin

11. Januar 2012

Vortrag an der Beuth Hochschule für Technik Berlin

Inhalt der Vorlesung

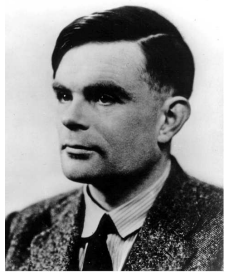
- Historische Vorbemerkungen
- λ-Ausdrücke in Java 8 und Scala – Motivation
- Eine (sehr kurze) Einführung in den λ-Kalkül
  - Motivation und Syntax
  - Reduktions- und Konversionsregeln
  - Currying (Schönfinkeln)
  - Reduktionsstrategien
  - ...
- Zusammenfassung

$\longrightarrow$ Definitionen und Beispiele an der Tafel

$\longrightarrow$ Demonstrationen in Scala

Historische Vorbemerkungen

Bilder: Wikipedia



Allgemeine Theorien von Berechnung (30er Jahre)

Bilder: Wikipedia

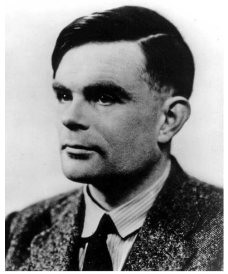
Turing (1912-54)

Gödel (1906-1978)

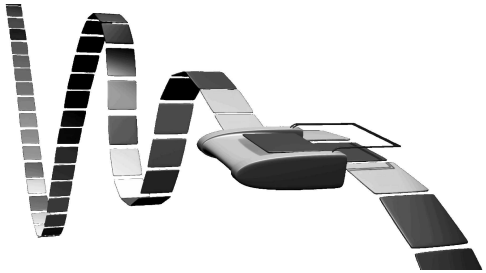
Church (1903-95)

Allgemeine Theorien von Berechnung (30er Jahre)

Bilder: Wikipedia



Turing (1912-54)



Turing Maschine

- x := x + 1
- Prozeduren
- Seiteneffekte

Gödel (1906-1978)

$$\mu f(x_1,\ldots,x_k) = \begin{cases} \min M(f,x_1,\ldots,x_k) & \text{falls } M(f,x_1,\ldots,x_k) \neq \emptyset \\ \text{undefiniert} & \text{sonst.} \end{cases}$$

μ-rekursive Funktionen

- while . . . do . . .

Church (1903-95)

$$\begin{aligned} & (\lambda x.x\,x)\,(\lambda z.z) \\ \longrightarrow_\beta \quad & (\lambda z.z)\,(\lambda z.z) \\ \longrightarrow_\beta \quad & (\lambda z.z) \end{aligned}$$

λ-Kalkül

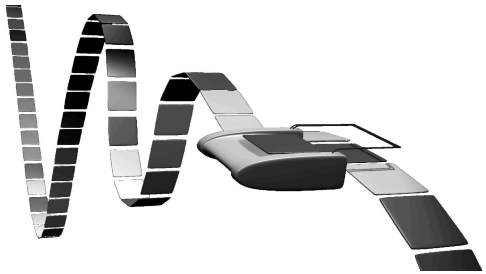
- Funktionen als
  - Objekte
  - Argumente & Resultate
- keine Seiteneffekte

Allgemeine Theorien von Berechnung (30er Jahre)

Bilder: Wikipedia

(Algol, Fortran, Pascal, C) — Imperative Programmierung

Turing (1912-54)



Turing Maschine

- x := x + 1
- Prozeduren
- Seiteneffekte

Gödel (1906-1978)

$$\mu f(x_1,\ldots,x_k) = \begin{cases} \min M(f,x_1,\ldots,x_k) & \text{falls } M(f,x_1,\ldots,x_k) \neq \emptyset \\ \text{undefiniert} & \text{sonst.} \end{cases}$$

μ-rekursive Funktionen

- while . . . do . . .

Church (1903-95)

$$\begin{aligned} & (\lambda x.x\,x)\,(\lambda z.z) \\ \longrightarrow_\beta \quad & (\lambda z.z)\,(\lambda z.z) \\ \longrightarrow_\beta \quad & (\lambda z.z) \end{aligned}$$

λ-Kalkül

- Funktionen als
  - Objekte
  - Argumente & Resultate
- keine Seiteneffekte

(LISP, ML, OCAML, HASKELL) — Funktionale Programmierung

λ-Ausdrücke in Java 8 und Scala – Motivation

Java — Motivation & Konzepte

- Objektorientierte, Imperative Programmierung
- Einfachheit, Lesbarkeit, Vertrautheit (C, C++ angelehnte Syntax)
- Robustheit und Sicherheit (Typsicherheit)
- Architekturneutralität, Portabilität, Interpretierbarkeit
- Leistungsfähigkeit
- Parallelisierbarkeit & Dynamisierung

Java — Motivation & Konzepte

- Objektorientierte, Imperative Programmierung
- Einfachheit, Lesbarkeit, Vertrautheit (C, C++ angelehnte Syntax)
- Robustheit und Sicherheit (Typsicherheit)
- Architekturneutralität, Portabilität, Interpretierbarkeit
- Leistungsfähigkeit
- Parallelisierbarkeit & Dynamisierung

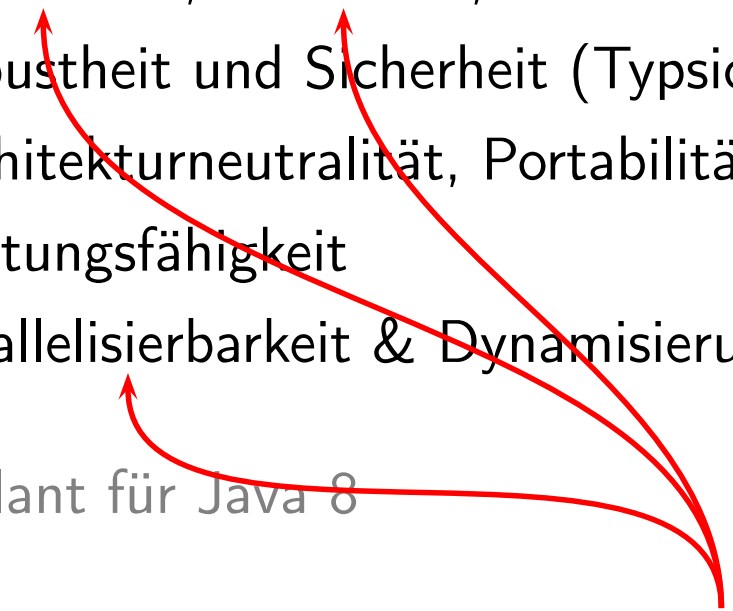
geplant für Java 8

\+ λ-Ausdrücke (Funktionale Programmierung) — Projekt 'Lambda'

\+ Modulsystem — Projekt 'Jigsaw'

Java — Motivation & Konzepte

- Objektorientierte, Imperative Programmierung
- Einfachheit, Lesbarkeit, Vertrautheit (C, C++ angelehnte Syntax)
- Robustheit und Sicherheit (Typsicherheit)
- Architekturneutralität, Portabilität, Interpretierbarkeit
- Leistungsfähigkeit
- Parallelisierbarkeit & Dynamisierung



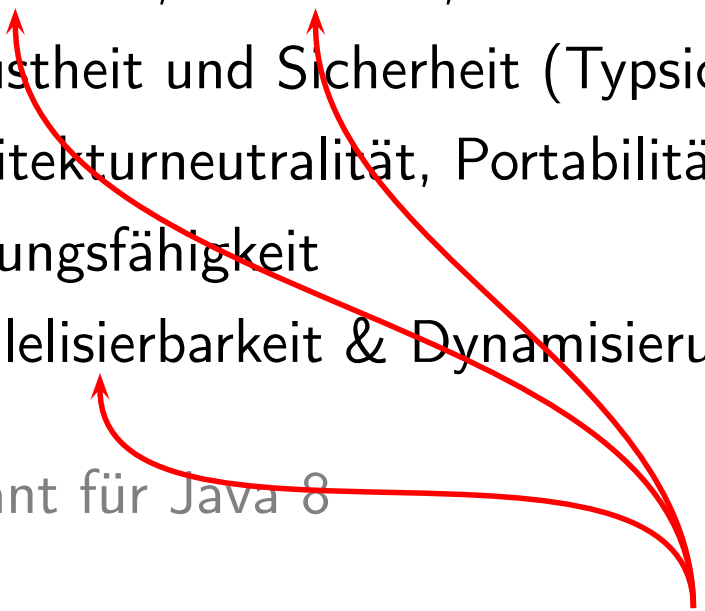
geplant für Java 8

\+ λ-Ausdrücke (Funktionale Programmierung) — Projekt 'Lambda'

\+ Modulsystem — Projekt 'Jigsaw'

Java — Motivation & Konzepte

- Objektorientierte, Imperative Programmierung
- Einfachheit, Lesbarkeit, Vertrautheit (C, C++ angelehnte Syntax)
- Robustheit und Sicherheit (Typsicherheit)
- Architekturneutralität, Portabilität, Interpretierbarkeit
- Leistungsfähigkeit
- Parallelisierbarkeit & Dynamisierung



geplant für Java 8

\+ λ-Ausdrücke (Funktionale Programmierung) — Projekt 'Lambda'

\+ Modulsystem — Projekt 'Jigsaw'

Insbesondere zur Unterstützung paralleler Algorithmen in Multicore & Multiprozessor Umgebungen

Auszug der Scala Website: `http://www.scala-lang.org/node/25`

Introducing Scala

Scala is a general purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It smoothly integrates features of object-oriented and functional languages, enabling Java and other programmers to be more productive. Code sizes are typically reduced by a factor of two to three when compared to an equivalent Java application. Many existing companies who depend on Java for business critical applications are turning to Scala to boost their development productivity, applications scalability and overall reliability.

For example, at Twitter, ...

Auszug der Scala Website: `http://www.scala-lang.org/node/25`

Introducing Scala

Scala is a general purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. It smoothly integrates features of object-oriented and functional languages, enabling Java and other programmers to be more productive. Code sizes are typically reduced by a factor of two to three when compared to an equivalent Java application. Many existing companies who depend on Java for business critical applications are turning to Scala to boost their development productivity, applications scalability and overall reliability.

For example, at Twitter, ...

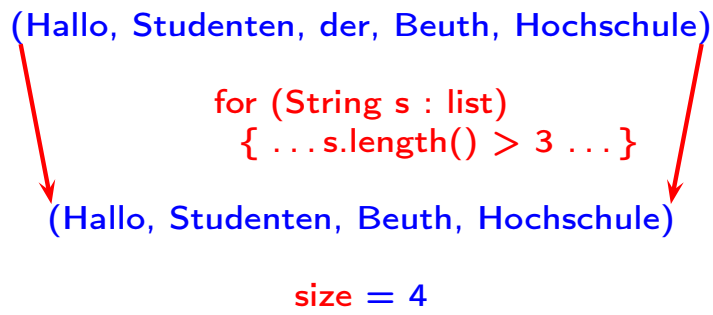
Insbesondere: Versuch einer nahtlosen Integration von Java!

Scala – das bessere Java? Ich meine ja!

Java 8 und Scala – Beispiele

```
import java.util.*;

class TestJava {

  static int simpleExample(ArrayList<String> list) {
    ArrayList<String> resultlist = new ArrayList<String>();
    for (String s : list) {
      if (s.length() > 3) {resultlist.add(s);}
    };
    return resultlist.size();
  }

  public static void main(String[] args) {
    ArrayList<String> examplelist =
    new ArrayList<String>(Arrays.asList(
      "Hallo","Studenten","der","Beuth","Hochschule"));
    System.out.println( simpleExample(examplelist) );
  }
}
```

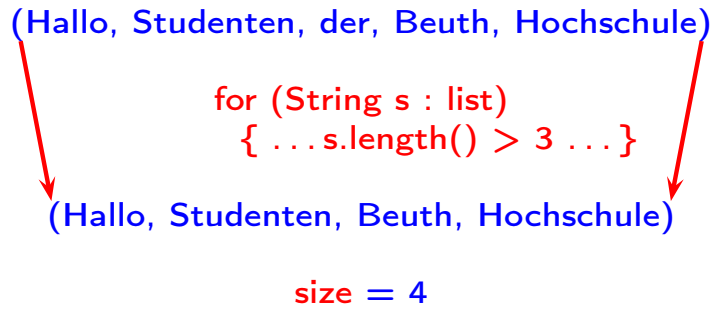
Java



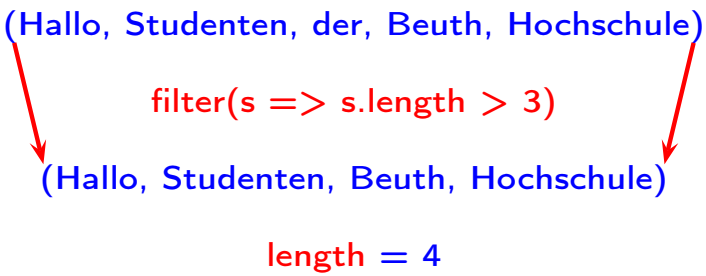
Java 8 und Scala – Beispiele

```
import java.util.*;

class TestJava {

  static int simpleExample(ArrayList<String> list) {
    ArrayList<String> resultlist = new ArrayList<String>();
    for (String s : list) {
      if (s.length() > 3) {resultlist.add(s);}
    };
    return resultlist.size();
  }

  public static void main(String[] args) {
    ArrayList<String> examplelist =
    new ArrayList<String>(Arrays.asList(
      "Hallo","Studenten","der","Beuth","Hochschule"));
    System.out.println( simpleExample(examplelist) );
  }
}
```

Java



Scala (mit λ-Ausdrücken)




```
object TestScala {

   def simpleExample(list: List[String]): Int =
       list.filter(s => s.length > 3).length

   def main(args: Array[String]) {
       val examplelist =
         List("Hallo","Studenten","der","Beuth","Hochschule")
       println( simpleExample(examplelist) )
   }
}
```

Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...

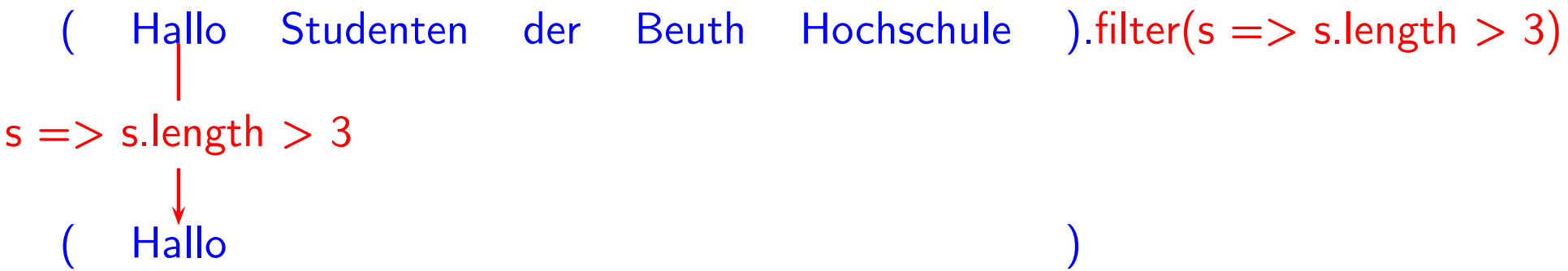
(Hallo Studenten der Beuth Hochschule).filter(s => s.length > 3)

Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...

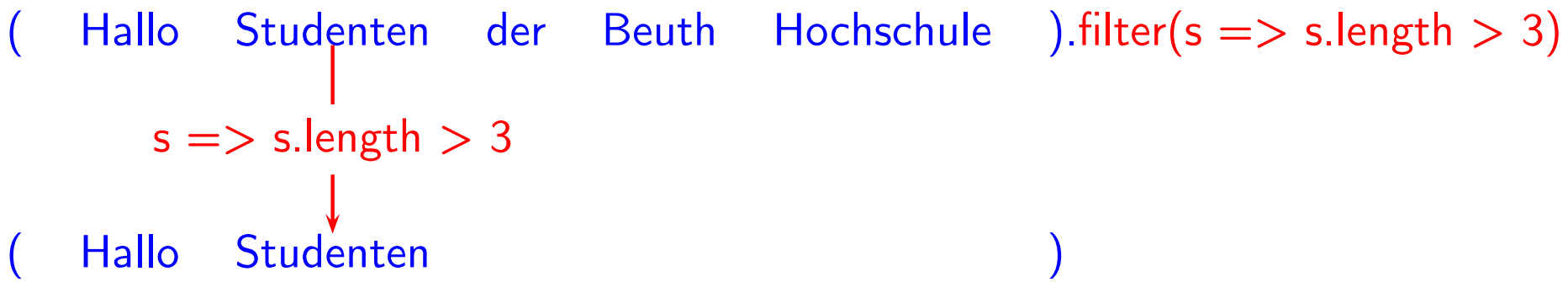
(Hallo Studenten der Beuth Hochschule).filter(s => s.length > 3)

( )

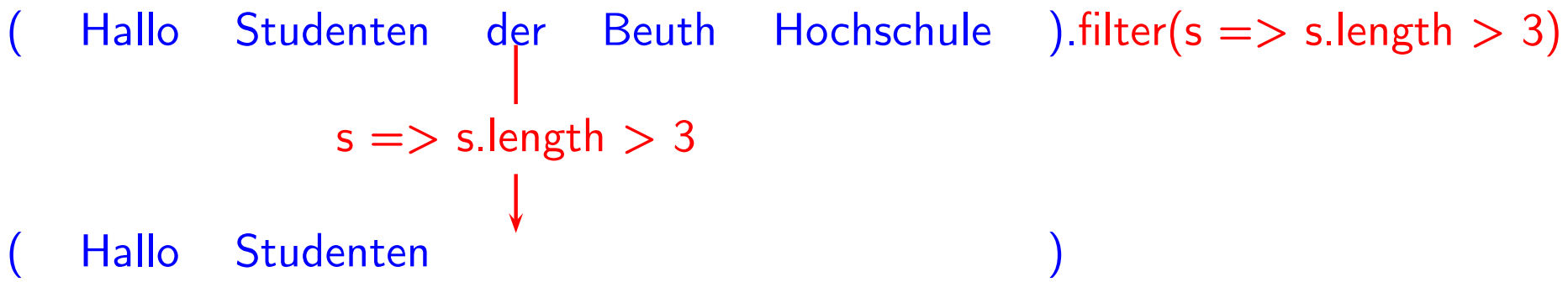
Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...



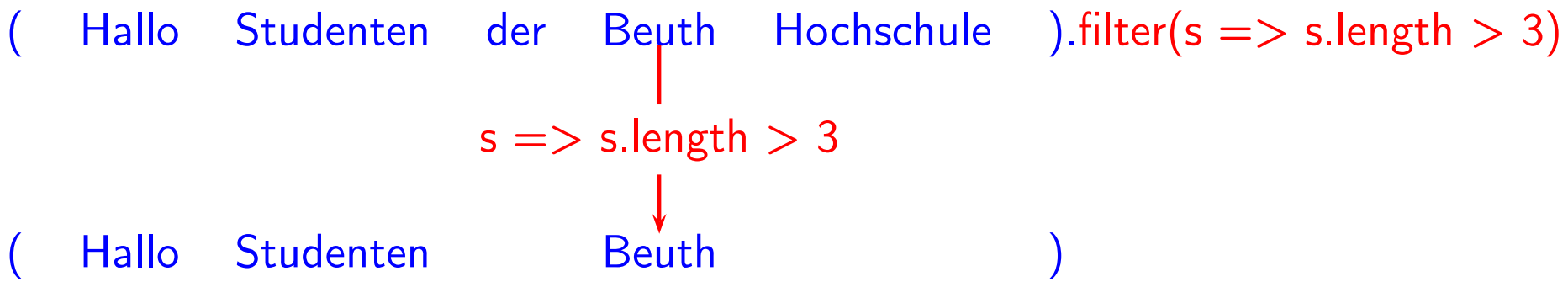
Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...



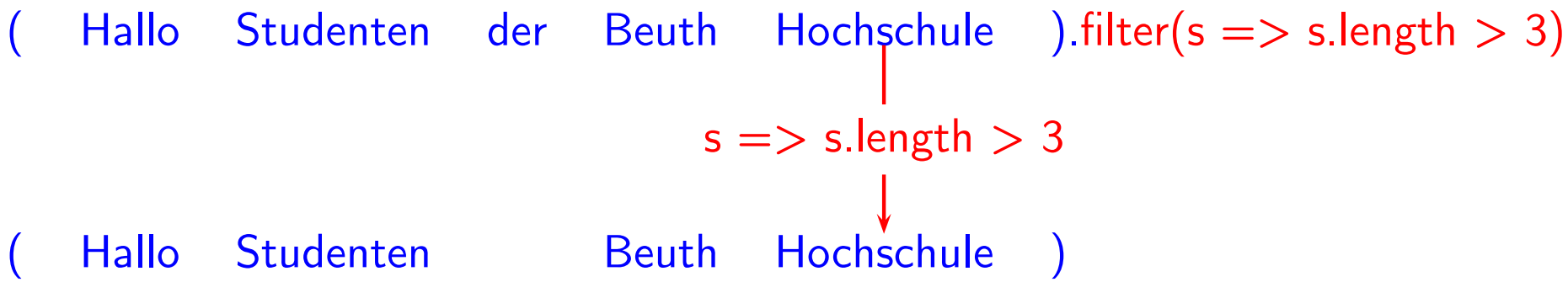
Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...



Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...



Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...



Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...

```
( Hallo Studenten der Beuth Hochschule ).filter(s => s.length > 3)
```

```
( Hallo Studenten Beuth Hochschule ).map(s => s.length)
```

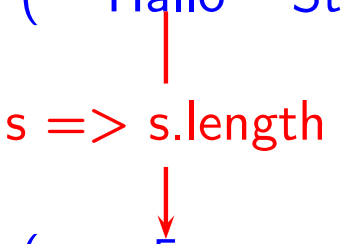
Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...

```
( Hallo Studenten der Beuth Hochschule ).filter(s => s.length > 3)

( Hallo Studenten      Beuth Hochschule ).map(s => s.length)

(                                       )
```

Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...

(Hallo Studenten der Beuth Hochschule).filter(s => s.length > 3)

(Hallo Studenten Beuth Hochschule).map(s => s.length)

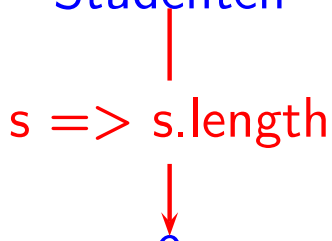


( )

Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...

```
(  Hallo  Studenten  der  Beuth  Hochschule  ).filter(s => s.length > 3)
```

```
(  Hallo  Studenten        Beuth  Hochschule  ).map(s => s.length)
```

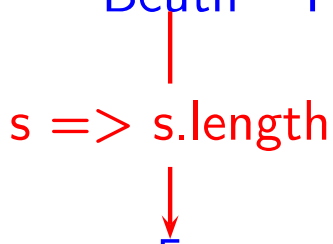



```
(    5         9                                )
```

Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...

(Hallo Studenten der Beuth Hochschule).filter(s => s.length > 3)

(Hallo Studenten Beuth Hochschule).map(s => s.length)

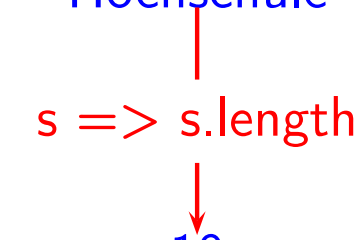


(5 9 5)

Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...

(Hallo Studenten der Beuth Hochschule).filter(s => s.length > 3)

(Hallo Studenten Beuth Hochschule).map(s => s.length)



(5 9 5 10)

Methoden mit λ-Ausdrücken als Argumenten: filter, map, fold, ...

(Hallo Studenten der Beuth Hochschule).filter(s => s.length > 3)

(Hallo Studenten Beuth Hochschule).map(s => s.length)

(5 9 5 10)

Typische Methoden für Listen (Collections):

map, filter, fold, forall, exists, foreach, etc.

In der funktionalen Programmierung können Sie solche (und noch komplexere) Methoden natürlich auch selbst einführen!

Was will Java 8 erreichen?

Was will Java 8 erreichen?

```
def simpleExample(list: List[String]): Int =
    list.filter(s => s.length > x)
        .length
```

Was will Java 8 erreichen?

```
static int simpleExample(list ArrayList<String>) =
    list.filter(s => s.length() > x)
        .size()
```

Was will Java 8 erreichen?

```
static int moreChallenging(list ArrayList<String>, ...) =
    list.filter(s => something-expensive)
        .map(s => something-expensive)
        .fold(...)(s => something-expensive)
        ....
```

z.b. komplexe, parallele Suche

Was will Java 8 erreichen?

```
static int moreChallenging(list ArrayList<String>, ...) =
    list.parallel()
        .filter(s => something-expensive)
        .map(s => something-expensive)
        .foldleft(...)(s => something-expensive)
        ....
```

z.b. komplexe, parallele Suche

Was will Java 8 erreichen?

```
static int moreChallenging(list ArrayList<String>, ...) =
    list.parallel()
        .filter(s => something-expensive)
        .map(s => something-expensive)
        .foldleft(...)(s => something-expensive)
        ....
```

z.b. komplexe, parallele Suche

Automatische Behandlung von Aspekten wie:

- Zerlegung des Problems
- Starten und Verwalten von Threads
- Synchronisation von Resultaten

Eine (kurze) Einführung in den λ-Kalkül

Eine (kurze) Einführung in den λ-Kalkül

– folgende Themen werden an der Tafel behandelt –

Motivation und Syntax

- λ-Abstraktionen modellieren anonyme Funktionen und Prädikate
- Notation in Scala und (voraussichtlich) Java 8
- Wichtige Definitionen:
  - λ-Ausdrücke: Variablen, λ-Abstraktionen, λ-Applikationen
  - freie und gebundene Variablen

Reduktions- und Konversionsregeln

- β-Reduktion und Beispiele
- α-Konversion und Beispiele

Eine (kurze) Einführung in den λ-Kalkül

– so könnte es weitergehen –

Currying (Schönfinkeln)

- λ-Kalkül unterstützt einstellige Funktionen
- Mehrstellige Funktionen können als sukzessive Anwendungen einstelliger Funktionen

Reduktionsstrategien

- Welche Reihenfolge bei der Anwendung von Reduktions- bzw. Konversionsregeln?

Church-Rosser Theorem

- Reihenfolge von Reduktions- bzw. Konversionsregeln spielt kein Rolle
- Es gibt höchstens eine Normalform für λ-Terme

Eine (kurze) Einführung in den λ-Kalkül

– so könnte es weitergehen –

Expressivität des λ-Kalküls

- Beispiel Church's Numerals – Ganze Zahlen kodiert als λ-Terme

Der getypte λ-Kalkül

- ... Wiederholung von Themen; diesmal mit Typen ...
- einfache Typen
- Polymorphie und Typinferenz

Bedeutung des (getypten) λ-Kalküls für die Vorlesung

- λ-Kalkül und funktionales Programmieren
- λ-Ausdrücke und Closures in Java 8 (oder Scala)

Zusammenfassung

λ-Ausdrücke in Java 8

- bereichern die Sprache um Aspekte funktionaler Programmierung
- verbessern die Lesbarkeit und Einfachheit von Java Programmen
- stehen im Zusammenhang zur Einführung von leistungsfähigen, abstrakten Methoden zur Unterstützung paralleler Algorithmen

Warum aber nicht gleich Scala?

- Scala ohnehin bereits ein besseres Java?
- Scala wird offensichtlich bereits von der Industrie aufgegriffen
- Scala stellt einen einfachen interaktiven Shell-Interpreter bereit
  $\longrightarrow$ besondere Eignung zur Einbindung in Vorlesungen