

Vorlesung

Programmieren I und II

Unit 5

Rekursive Programmierung und rekursive Datenstrukturen,
Lambda-Ausdrücke und Streams

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
1

**Prof. Dr. rer. nat.
Nane Kratzke**

*Praktische Informatik und
betriebliche Informationssysteme*

- Raum: 17-0.10
- Tel.: 0451 300 5549
- Email: kratzke@fh-luebeck.de

@NaneKratzke
Updates der Handouts auch über Twitter #prog_inf und #prog_itd

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
2

Units

1. Semester

Unit 1 Einführung und Grundbegriffe	Unit 2 Grundelemente Imperativer Programme	Unit 3 Selbstdefinierbare Basistypen und Collections	Unit 4 Einfache I/O Programmierung
Unit 5 Rekursive Programmierung, rekursive Datenstrukturen, Lambdas	Unit 6 Objektorientierte Programmierung und UML	Unit 7 Konzepte objektorientierter Programmiersprachen, Klassen vs. Objekte, Pakete und Erbschaften	Unit 8 Testen (objektorientierter) Programme

2. Semester

Unit 9 Generische Datentypen	Unit 10 Objektorientierter Entwurf und objektorientierte Designprinzipien	Unit 11 Graphical User Interfaces	Unit 12 Multithread Programmierung
---------------------------------	--	--------------------------------------	---------------------------------------

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
3

Abgedeckte Ziele dieser UNIT

Können existierender Programmierparadigmen und Laufzeitmodelle	Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)	Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenztypen (insbesondere Liste, Stack, Mapping)	Verstehen des Unterschieds zwischen Wert- und Referenzsemantik
Können und Anwenden des Prinzips der rekursiven Programmierung und rekursiver Datenstrukturen, sowie Lambda Funktionen	Können des Algorithmusbegriffs, Implementieren einfacher Algorithmen	Können objektorientierter Konzepte Datenkapselung, Polymorphie und Vererbung	Sicheres Anwenden programmiersprachlicher Konzepte der Objektorientierung (Klassen und Objekte, Schnittstellen und Generics, Streams, GUI und MVC)
Können von UML Klassendiagrammen, sicheres Übersetzen von UML Klassendiagrammen in Java (und von Java in UML).	Können der Grenzen des Testens von Software und erste Erfahrungen im Testen (objektorientierter) Software	Sammeln erster Erfahrungen in der Anwendung objektorientierter Entwurfsprinzipien	Sammeln von Erfahrungen mit weiteren Programmiermodellen und -paradigmen, insbesondere Multithread Programmierung sowie funktionale Programmierung

Am Beispiel der Sprache JAVA

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
4

Themen dieser Unit

Rekursive Routinen

- Rekursiv definierte Methoden
- Beispiele für rekursive Methoden
- Formulierung rekursiver Methoden

Rekursive Algorithmen und Datenstrukturen

- Algorithmus
- Rekursive Datenstrukturen (Binärbäume)
- BubbleSort (imperativ) vs BinSort (rekursiv)

Lambdas

- Anonyme Funktionen
- Streams
- Filter
- Map
- Reduce

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
5

Zum Nachlesen ...

Kapitel 6

Methoden und Unterprogramme

Abschnitt 6.2

Rekursiv definierte Methoden

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme
6

Rekursive Routinen

- Unter rekursiven Methoden versteht man Methoden, die sich selber aufrufen.
- Rekursionen können dazu genutzt werden, um Kontrollanweisungen – wie z.B. Schleifen – zu vermeiden und durch Aufrufstrukturen abzubilden.
- Diverse Funktionen und Strukturen der Mathematik sind rekursiv definiert.
- Einige komplexe Probleme (z.B. Türme von Hanoi) lassen sich erstaunlich einfach rekursiv formulieren und lösen.
- Die Entwicklung rekursiver Methoden fördert ferner das **abstrakte Denk- und Analysevermögen** und ist so wertvoll im Rahmen der Informatikausbildung.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Beispiel: Fakultät – eine rekursiv definierte mathematische Funktion

- Die Fakultät von 0 ist 1.
- Die Fakultät einer positiven Zahl n ist n multipliziert mit der Fakultät von n-1.

$$n! = \begin{cases} 1 & \text{falls } n = 0 \\ n \cdot (n-1)! & \text{falls } n \neq 0 \end{cases}$$

- Die Fakultät einer Zahl n ist also das Produkt aller ganzen Zahlen von 1 bis zu dieser Zahl n.

$$n! = n * (n-1) * (n-2) * \dots * 1$$

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Exkurs: Rekursion Beispiel – Fakultät in JAVA (I)

```
public int fac(int n) {
    if (n == 0) return 1;
    return fac(n-1) * n;
}
```



- $fac(3) =$
- $fac(2) * 3 =$
- $fac(1) * 2 * 3 =$
- $fac(0) * 1 * 2 * 3 =$
- $1 * 1 * 2 * 3 =$
- 6

$$fac(1) = 1 * fac(0)$$

$$fac(2) = 2 * fac(1 * fac(0))$$

$$fac(3) = 3 * fac(2 * fac(1 * fac(0)))$$

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Exkurs: Rekursion Beispiel – Fakultät in JAVA (II)

```
public int fac(int n) {
    if (n == 0) return 1;
    return fac(n-1) * n;
}
```

- Die Anzahl der Multiplikationen wird also in der dynamischen Aufrufstruktur der Methode fac abgebildet und nicht explizit codiert
- Iterativ kann das wie folgt durch eine for-schleife ausgedrückt werden.

```
public int facit(int n) {
    int ret = 1;
    for (int i = n; i > 0; i--)
        ret *= i;
    return ret;
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Rekursionsabbruch

- Bei rekursiv definierten Methoden ist es wesentlich, sicherzustellen, dass die Rekursion endet.
- Die Bedingung die hierfür zuständig ist, nennt man **Rekursionsabbruchbedingung**
- Sie sollte beim Programmieren als erster Problembestandteil formuliert werden und stellt zumeist den einfachsten Fall
 - trivialen
 - aber häufig übersehenen Sonderfall
- dar.



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Entwicklung einfacher rekursiver Methoden

- Schritt 1:**
Formulierung des Methodenkopfes (Signatur)
- Schritt 2:**
Definition des Rekursionsabbruchs
- Schritt 3:**
Formulierung des Falls n und Rückführung auf den Fall n-1

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Beispiel: Rekursive Summe (I)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Rekursive Implementierung der Addition der Zahlen 0 bis n.

$$sum(n) = \sum_{i \in \{0, \dots, n\}} i = 0 + 1 + 2 + \dots + n$$

$$sum(n) = \begin{cases} 0 & \text{falls } n = 0 \\ n + sum(n-1) & \text{falls } n > 0 \end{cases}$$

Formulierung des Methodenkopfes (Signatur)

Definition des Rekursionsabbruchs

Formulierung des Falls n und Rückführung auf den Fall n-1

```
public static int sum(int n) {
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Beispiel: Rekursive Summe (II)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Rekursive Implementierung der Addition der Zahlen 0 bis n.

$$sum(n) = \sum_{i \in \{0, \dots, n\}} i = 0 + 1 + 2 + \dots + n$$

$$sum(n) = \begin{cases} 0 & \text{falls } n = 0 \\ n + sum(n-1) & \text{falls } n > 0 \end{cases}$$

Formulierung des Methodenkopfes (Signatur)

Definition des Rekursionsabbruchs

Formulierung des Falls n und Rückführung auf den Fall n-1

```
public static int sum(int n) {
    if (n == 0) {
        return 0;
    }
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Beispiel: Rekursive Summe (III)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Rekursive Implementierung der Addition der Zahlen 0 bis n.

$$sum(n) = \sum_{i \in \{0, \dots, n\}} i = 0 + 1 + 2 + \dots + n$$

$$sum(n) = \begin{cases} 0 & \text{falls } n = 0 \\ n + sum(n-1) & \text{falls } n > 0 \end{cases}$$

Formulierung des Methodenkopfes (Signatur)

Definition des Rekursionsabbruchs

Formulierung des Falls n und Rückführung auf den Fall n-1

```
public int sum(int n) {
    if (n == 0) {
        return 0;
    } else {
        return n + sum(n-1);
    }
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Beispiel: Rekursive Summe (IV)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Rekursive Implementierung der Addition der Zahlen 0 bis n.

$$sum(n) = \sum_{i \in \{0, \dots, n\}} i = 0 + 1 + 2 + \dots + n$$

$$sum(n) = \begin{cases} 0 & \text{falls } n = 0 \\ n + sum(n-1) & \text{falls } n > 0 \end{cases}$$

```
// Ich schreibe Rekursionen gerne auch so ...
// Vergleiche mit mathematischer Definition oben!
// else ist häufig unnötig.
public int sum(int n) {
    if (n == 0) return 0; // Rekursionsabbruch
    return n + sum(n-1); // Rekursion
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Beispiel: Rekursive Summe (V)

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Rekursive Implementierung der Addition der Zahlen 0 bis n.

$$sum(n) = \sum_{i \in \{0, \dots, n\}} i = 0 + 1 + 2 + \dots + n$$

$$sum(n) = \begin{cases} 0 & \text{falls } n = 0 \\ n + sum(n-1) & \text{falls } n > 0 \end{cases}$$

```
// Und man kann das naturlich auch elegant
// mit einem einzigen Ausdruck formulieren.
public int sum(int n) {
    return n == 0 ? 0 : n + sum(n-1);
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Entwicklung rekursiver Methoden bei sequenzbasierten Problemstellungen

FACH HOCHSCHULE LÜBECK
University of Applied Sciences



Bei Problemen, die sich nicht durch einen Parameter ausdrücken lassen, sondern Sequenzen von Werten verarbeiten sollen, bietet es sich häufig an, im Schritt 3 den Kopf der Sequenz vom Rest der Sequenz zu trennen.

Schritt 1:
Formulierung des Methodenkopfes (Signatur)

Schritt 2:
Triviale Fälle beantworten (Rekursionsabbruch)

Schritt 3:
Zerlegung des Problems (n auf n-1 zurückführen)

3.1 Kopf des Problems (head)	3.2 Rest des Problems (rest)	3.3 Rekursionsausdruck
------------------------------	------------------------------	------------------------

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Beispiel: Anzahl eines Zeichens in einer Zeichenkette bestimmen (rekursiv)

FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Methodenkopf (Signatur)
 Triviale Fälle
 Zerlegung des Problems
 Kopf Rest Rekursionsausdruck

```
public static int count(String s, char c) {  
  
}
```

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 19

Beispiel: Anzahl eines Zeichens in einer Zeichenkette bestimmen (rekursiv)

FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Methodenkopf (Signatur)
 Triviale Fälle
 Zerlegung des Problems
 Kopf Rest Rekursionsausdruck

```
public static int count(String s, char c) {  
    if (s.isEmpty()) return 0;  
  
}
```

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 20

Beispiel: Anzahl eines Zeichens in einer Zeichenkette bestimmen (rekursiv)

FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Methodenkopf (Signatur)
 Triviale Fälle
 Zerlegung des Problems
 Kopf Rest Rekursionsausdruck

```
public static int count(String s, char c) {  
    if (s.isEmpty()) return 0;  
  
    char head = s.charAt(0);  
  
}
```

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 21

Beispiel: Anzahl eines Zeichens in einer Zeichenkette bestimmen (rekursiv)

FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Methodenkopf (Signatur)
 Triviale Fälle
 Zerlegung des Problems
 Kopf Rest Rekursionsausdruck

```
public static int count(String s, char c) {  
    if (s.isEmpty()) return 0;  
  
    char head = s.charAt(0);  
    String rest = s.substring(1);  
  
}
```

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 22

Beispiel: Anzahl eines Zeichens in einer Zeichenkette bestimmen (rekursiv)

FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Methodenkopf (Signatur)
 Triviale Fälle
 Zerlegung des Problems
 Kopf Rest Rekursionsausdruck

```
public static int count(String s, char c) {  
    if (s.isEmpty()) return 0;  
  
    char head = s.charAt(0);  
    String rest = s.substring(1);  
  
    return (c == head ? 1 : 0) + count(rest, c);  
  
}
```

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 23

Komplexe Probleme mit einfachen Rekursionen lösen

FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Wie bekommen Sie alle Scheiben von dem linken Stab auf den rechten Stab, wenn nur kleinere Scheiben auf größeren Scheiben auf den Stäben liegen dürfen und pro Zug nur ein Stein bewegt werden darf?

Das sogenannte „Türme von Hanoi“-Problem

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 24

Geschichte der Türme von Hanoi



Nach dem franz. Mathematiker **Eduard Lucas** (1842 - 1891) geht das Problem angeblich auf indische Mönche im großen Tempel zu Benares (im vermeintlichen Mittelpunkt der Welt) zurück, die einen Turm aus 64 goldenen Scheiben versetzen müssen. Wenn Ihre Arbeit vollendet ist, ist das Ende der Welt gekommen.



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Die Strategie der Mönche

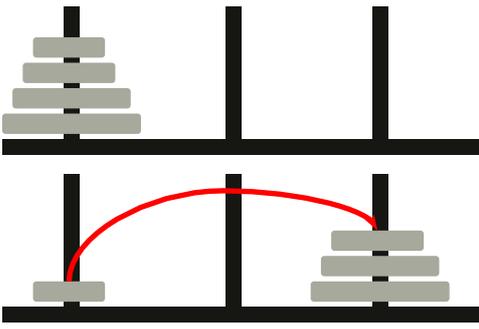


- Der älteste Mönch erhält die Aufgabe, den Turm aus 64 Scheiben zu versetzen.
- Da er die komplexe Aufgabe nicht bewältigen kann, gibt er dem zweitältesten Mönch die Aufgabe, die oberen 63 Scheiben auf einen Hilfsplatz zu versetzen.
- Er selbst (der Älteste) würde dann die große letzte Scheibe zum Ziel bringen.
- Dann könnte der Zweitälteste wieder die 63 Scheiben vom Hilfsplatz zum Ziel bringen.
- Der zweitälteste Mönch fühlt sich der Aufgabe ebenfalls nicht gewachsen.
- So gibt er dem drittältesten Mönch den Auftrag, die oberen 62 Scheiben zu transportieren, und zwar auf den endgültigen Platz.
- Er selbst (der Zweitälteste) würde dann die zweitletzte Scheibe an den Hilfsplatz bringen.
- Schließlich würde er wieder den Drittaltesten beauftragen, die 62 Scheiben vom Zielfeld zum Hilfsplatz zu schaffen.
- Dies setzt sich bis zum 64. Mönch (dem Jüngsten) fort, der die obenauf liegende kleinste Scheibe alleine verschieben kann.



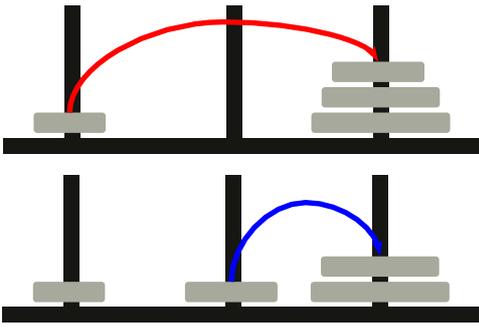
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Die Problemlösung aus Sicht des ersten Mönchs

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Die Problemlösung aus Sicht des zweiten Mönchs (usw. usw.)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Die Strategie der Mönche als Algorithmus (I)



- Der Algorithmus besteht im Wesentlichen aus einer Funktion *bewege*
 - *Parameter i* ist die Anzahl der zu verschiebenden Scheiben bezeichnet,
 - *Parameter a* ist der Stab von dem verschoben werden soll,
 - *mit b* der Stab, der als Zwischenziel dient und
 - *mit c* der Stab, auf den die Scheiben verschoben werden sollen.

Stab A
↓
Quelle

Stab B
↓
Zwischenziel

Stab C
↓
Ziel

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Die Strategie der Mönche als Algorithmus (II)



- Die Funktion *bewege* löst ein Teilproblem dadurch, dass es dieses in drei einfachere Probleme aufteilt.
- Die drei Teilprobleme werden sequentiell ausgeführt.
 - Zunächst wird der um eine Scheibe kleinere Turm von a auf das Zwischenziel b verschoben. Die Stäbe b und c tauschen dabei ihre Rollen.
 - Anschließend wird die einzig verbliebene Scheibe von a nach c verschoben.
 - Zum Abschluss wird der zuvor auf b verschobene Turm auf seinen Bestimmungsort c verschoben, wobei hier a und b die Rollen tauschen



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Die Strategie der Mönche als JAVA-Methode



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

```
public void bewege(int i, Stack a, Stack b, Stack c)
{
    if (i == 0) return; // Rekursionsabbruch
    bewege(i - 1, a, c, b); // nächster Mönch
    c.push(a.pop()); // Setze deinen Stein
    bewege(i - 1, b, a, c); // nächster Mönch
}
```



Das Problem der Türme von Hanoi ist also mit einem rekursiven Vier-Zeiler zu lösen!

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 31

Die Türme von Hanoi



FACH HOCHSCHULE LÜBECK
University of Applied Sciences






 Sie finden eine Menge Hinweise zu den Türmen von Hanoi, z.B. unter Wikipedia:

http://de.wikipedia.org/wiki/Türme_von_Hanoi

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 32

Miniübung:






FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Gegeben sei folgende Liste:

```
List v = new LinkedList();
for (int i = 0; i < 10; i++) v.add(i);
```

Entwickeln Sie nun eine Methode rekprint, um eine Liste oben angegebener Art in folgender Form als String zurückzugeben:

0-1-2-3-4-5-6-7-8-9

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 33

Miniübung:






FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Gegeben sei folgendes Array:

```
int[] as = { 1, 2, 3, 4, 5, 6 };
```

Entwickeln Sie nun eine Methode rekprinta, um ein Array oben angegebener Art in folgender Form als String zurückzugeben:

1-2-3-4-5-6

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 34

Zusammenfassung



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

- **Rekursive Methoden**
 - Rufen sich selber auf
 - Komplexe Probleme einfach ausdrücken (Türme von Hanoi)
 - Ersetzen Kontrollanweisungen durch Aufrufstrukturen
- **Beispiele für Rekursionen**
 - Fakultät (einfach)
 - Summe von 0 bis n (einfach)
 - Türme von Hanoi (komplex)
- **Formulierung einer rekursiven Methode**
 - Definiere die Signatur (Methodenkopf)
 - Implementiere die Abbruchbedingung(en)
 - Implementiere den Fall n unter Rückgriff auf den Fall n-1 bzw. n+1




Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 35

Themen dieser Unit



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Rekursive Routinen	Rekursive Algorithmen und Datenstrukturen	Lambdas
<ul style="list-style-type: none"> • Rekursiv definierter Methoden • Beispiele für rekursive Methoden • Formulierung rekursiver Methoden 	<ul style="list-style-type: none"> • Algorithmus • Rekursive Datenstrukturen (Binärbäume) • BubbleSort vs BinSort 	<ul style="list-style-type: none"> • Anonyme Funktionen • Streams • Filter • Map • Reduce

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 36

Zum Nachlesen ...



WIKIPEDIA
Die freie Enzyklopädie

Algorithmus
<http://de.wikipedia.org/wiki/Algorithmus>

Binärbaum
<http://de.wikipedia.org/wiki/Binärbaum>

BubbleSort
<http://de.wikipedia.org/wiki/Bubblesort>

BinSort
<http://de.wikipedia.org/wiki/Binarytreesort>

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Algorithmen

- Ein Algorithmus ist eine aus endlich vielen Schritten bestehende eindeutige Handlungsvorschrift zur Lösung eines Problems
- Eigenschaften eines Algorithmus
 - Das Verfahren muss in einem endlichen Text **eindeutig beschreibbar** sein (Finitheit)
 - Jeder Schritt des Verfahrens muss tatsächlich **ausführbar** sein (Ausführbarkeit)
 - Das Verfahren darf zu jedem Zeitpunkt nur **endlich viel Speicherplatz** benötigen (Dynamische Finitheit)
 - Das Verfahren darf nur **endlich viele Schritte** benötigen (terminierend)
- **Oftmals wird gefordert, dass ein Algorithmus deterministisch ist**
 - Der Algorithmus muss bei denselben Voraussetzungen das gleiche Ergebnis liefern (Determiniertheit des Ergebnisses)
 - Die nächste anzuwendende Regel im Verfahren ist zu jedem Zeitpunkt eindeutig definiert (Determinismus des Verfahrens)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Ein Beispiel für einen natürlichsprachigen Algorithmus zum Sortieren

Sie haben eine Liste von Zahlen und sollen diese in eine aufsteigende Reihenfolge bringen.

Ein **Verfahrungsanweisung** hierfür könnte bspw. so aussehen:

1. Gehe davon aus, dass die Liste nicht sortiert ist.
2. Ist die Liste sortiert? Wenn ja Springe zu Schritt 6.
3. Behaupte ab sofort die Liste sei sortiert (denn das sollen die folgenden Schritte bezwecken).
4. Durchlaufe die Liste in aufsteigender Richtung Element für Element. Pro Schritt tue das folgende
 1. Betrachte immer zwei benachbarte Elemente
 2. Stehen diese in falscher Ordnung vertausche die Elemente
 3. Behaupte die Liste ist nicht sortiert (ein Fehler wurde ja gefunden)
5. Springe zu Schritt 2
6. Beende den Algorithmus (denn die Liste ist nun sortiert).

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

BubbleSort Prinzip am Beispiel

6 5 3 1 8 7 2 4



WIKIPEDIA
Die freie Enzyklopädie

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Der Algorithmus in JAVA (bubbleSort)

```

public static void bubbleSort(int[] xs) {
    boolean unsorted=true; Schritt 1: Behaupte Liste sei unsortiert
    while (unsorted) { Schritt 2: Ist die Liste sortiert?
        unsorted = false; Schritt 3: Behaupte Liste wird sortiert sein
        for (int i=0; i < xs.length-1; i++) { Schritt 4: Listendurchlauf
            if (xs[i] > xs[i+1]) { Schritt 4.1: Betrachte ben. Elemente
                int dummy = xs[i]; Schritt 4.2: Wenn in falscher Ordnung vertausche
                xs[i] = xs[i+1];
                xs[i+1] = dummy;
                unsorted = true; Schritt 4.3: Liste war doch nicht sortiert
            }
        }
        Schritt 5: Springe zu Schritt 2
    }
    Schritt 6: Ende des Algorithmus
}
    
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Miniübung:





Geben Sie den **Listenzustand** nach Ausführung von **bubbleSort** Durchlaufen an:

Ausgangsliste: 5 4 1 7 6 3 2 8

Nach 1. Durchlauf:

Nach 2. Durchlauf:

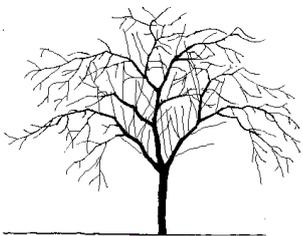
Nach 3. Durchlauf:

Nach 4. Durchlauf:

Nach 5. Durchlauf:

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Von Listen und Bäumen

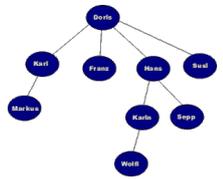


FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

Die Datenstruktur Baum

- Unter einem Baum versteht man in der Informatik eine Datenstruktur, die ausgehend von einem Wurzelknoten, einen oder mehrere Kindknoten haben kann, die wiederum weitere Kindknoten haben können.
- Bäume gehören zu häufig genutzten Datenstrukturen in der Informatik
- Bekannte Anwendungen von Bäumen sind z.B.
 - Dateisysteme (Verzeichnishierarchien)
 - Webseiten (Der HTML Code wird als Baum eingelesen und durch Webbrowser dargestellt)

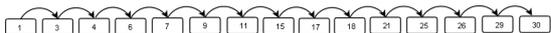


FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

Algorithmen zur rekursiven Datenstruktur Baum

Von der Liste zum Baum:



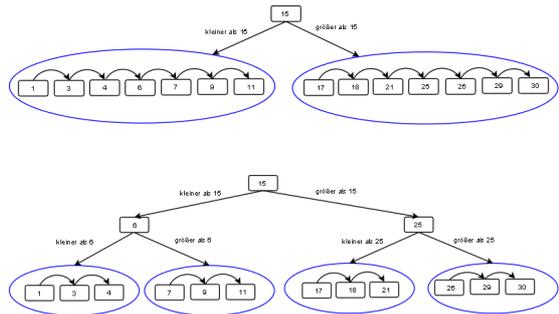
In einer sortierten Liste erscheint das Suchen eines bestimmten Elements sehr einfach:

- Sucht man beispielsweise das Element 9 in obiger Liste, wird die Liste durchlaufen und man erhält das Element nach 6 Vergleichen.
- Sucht man das Element 10, sind 7 Vergleiche notwendig.
- Ist das gesuchte Element jedoch am Ende der Liste sind sehr viele Vergleichsoperationen notwendig, für das Element 30 benötigt man 15 Vergleiche.
- Bei großen Listen (z.B. ein Wörterbuch) hat eine solche Suche ein sehr schlechtes Zeitverhalten.
- Die Laufzeit der Suche ist bei dieser Suche linear von n abhängig.
- Die Suche wird effizienter, wenn man die Liste in der Mitte teilt.

FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

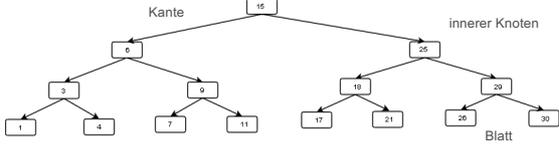
Von der Liste zum Baum



FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

Sortierter Binärbaum



Ein solcher Baum heißt **Binärbaum**.

- Bei den Knoten unterscheidet man zwischen **inneren Knoten** (mit Nachfolger) und **Blättern** (kein Nachfolger).
- Die Referenzen zwischen den Knoten nennt man **Kanten**.
- Die **Tiefe** eines Knotens ist die Anzahl der Kanten + 1, die beim Durchlauf von der Wurzel bis zum Knoten Schritten werden.
- Der oberste Knoten heißt **Wurzel** und hat die Tiefe 1.
- Alle Knoten mit der gleichen Tiefe beschreiben eine **Ebene** des Baumes.
- Die **Höhe** des Baumes ist festgelegt durch die größtmögliche Tiefe.

FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

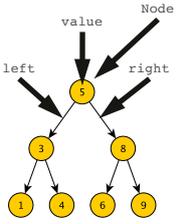
JAVA Referenztyp Knoten eines Baums

```

class Node {
    public Node left;
    public Node right;
    public int value;

    public Node(int v, Node l, Node r) {
        value = v;
        left = l;
        right = r;
    }

    public String toString() {
        return value + " ";
    }
}
    
```



FACH HOCHSCHULE LÜBECK
 University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

Ausdrücken eines Baums in JAVA

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Dieser Baum lässt sich in folgender Form in JAVA ausdrücken:

```
Node tree = new Node(5,
    new Node(3,
        new Node(1, null, null),
        new Node(4, null, null)
    ),
    new Node(8,
        new Node(6, null, null),
        new Node(9, null, null)
    )
);
```

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 49

Baumdurchlauf

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Möchte man alle Elemente eines Baumes ausgeben, muss man sich Strategien überlegen, in welcher Reihenfolge der Baum durchlaufen wird. Hierzu existieren Algorithmen, die beim Durchlaufen eines Baumes jeden Knoten genau einmal besuchen und auswerten. Folgende drei (rekursive) Algorithmen existieren hierzu:

- **inorder-Durchlauf (Merke: L K R)**
 - Beauftrage den linken Teilbaum des Knotens k mit inorder-Durchlauf
 - Besuche den Knoten k selbst
 - Beauftrage den rechten Teilbaum des Knotens k mit inorder-Durchlauf
- **preorder-Durchlauf (Merke: K L R)**
 - Besuche den Knoten k selbst
 - Beauftrage den linken Teilbaum des Knotens k mit preorder-Durchlauf
 - Beauftrage den rechten Teilbaum des Knotens k mit preorder-Durchlauf
- **postorder-Durchlauf (Merke: L R K)**
 - Beauftrage den linken Teilbaum des Knotens k mit postorder-Durchlauf
 - Beauftrage den rechten Teilbaum des Knotens k mit postorder-Durchlauf
 - Besuche den Knoten k selbst

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 50

Baumdurchlauf

Beispiel: inorder Ausgabe eines Binärbaums

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Inorder: Linker Ast – Knoten – Rechter Ast

```
public static String inorder(Node n) {
    if (n == null) return "";
    return inorder(n.left) +
        n +
        inorder(n.right);
}
```

1 3 4 5 6 8 9

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 51

Einfügen in sortierte Binäräume

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

- Um einen sortierten Binärbaum erweitern oder schrittweise erzeugen zu können, benötigt man eine insert Methode
- Auch diese Methode kann rekursiv definiert werden.

- Zunächst muss geprüft werden, ob der Baum leer ist. Für diesen Fall ist das einzufügende Element die Wurzel des Baumes.
- Ist der Baum nicht leer, wird ausgehend von der Wurzel zunächst geprüft, ob das einzufügende Element mit dem Knoten übereinstimmt. Ist dies der Fall, wird das Element nicht eingefügt.
- Ist das einzufügende Element kleiner als der aktuelle Knoten und hat einen linken Teilbaum, wird die insert Methode für diesen linken Teilbaum aufgerufen.
- Ist es größer und existiert ein rechter Teilbaum, wird sie für den rechten Teilbaum aufgerufen.
- Hat der aktuelle Knoten keinen Nachfolger, kann das Element als Abbruchbedingung der Rekursion eingefügt werden.

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 52

Insert Operation auf einem sortierten Binärbaum

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

```
void insert(int v, Node tree) {
    if (tree == null) return;
    if (tree.value == v) return;

    if (v < tree.value) {
        if (tree.left == null) {
            tree.left = new Node(v, null, null);
        } else {
            insert(v, tree.left);
        }
    }

    if (v > tree.value) {
        if (tree.right == null) {
            tree.right = new Node(v, null, null);
        } else {
            insert(v, tree.right);
        }
    }
}
```

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 53

Mit Binäräumen sortieren

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

- Durchlaufe eine unsortierte Liste Element für Element von vorne nach hinten
- Füge jedes Element mittels der insert Operation in einen Binärbaum ein

- Man erhält einen sortierten Binärbaum
- Durchlaufe diesen Binärbaum in inorder Durchlauf
- Man erhält eine sortierte Liste

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme 54

**Mit Binärbäumen sortieren
Veranschaulichung**



University of Applied Sciences

5 →

4 →

1 →

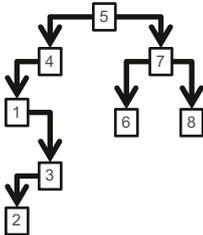
7 →

6 →

3 →

2 →

8 →



Inorder Durchlauf ergibt:
1-2-3-4-5-6-7-8

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 55

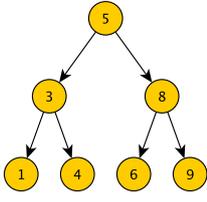
Miniübung:






University of Applied Sciences

Geben Sie die **postorder** Ausgabe unten stehenden Baumes an:



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 56

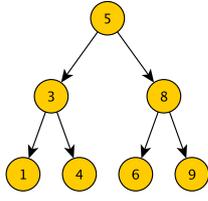
Miniübung:






University of Applied Sciences

Drücken Sie unten stehenden Baum in JAVA aus. Nutzen Sie dabei den Node Referenztyp wie er in der VL definiert wurde.



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 57

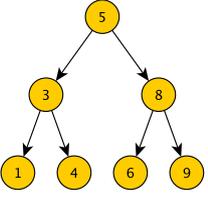
Miniübung:






University of Applied Sciences

Gegeben sei ein Baum. Bestimmen Sie die Anzahl der Knoten dieses Baums (und beliebig anderer) mit einer Funktion `nodes()`.



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 58

Miniübung:

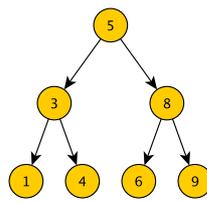
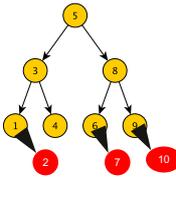





University of Applied Sciences

Gegeben ist folgender Baum tree. Es werden die folgenden `insert` Operationen auf `tree` ausgeführt.

```
insert(7, tree);
insert(2, tree);
insert(10, tree);
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 59

Miniübung:



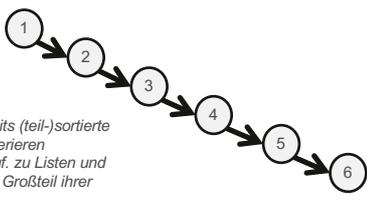



University of Applied Sciences

Gegeben ist folgende Liste:

1-2-3-4-5-6

Überführen Sie diese in einen Binärbaum indem Sie diese sequentiell von vorne nach hinten durchlaufen und mit der `insert` Operation in einen Binärbaum speichern.



Hinweis:
Existieren bereits (teil-)sortierte Liste so degenerieren Binärbäume ggf. zu Listen und verlieren einen Großteil ihrer Effizienz

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 60

Zusammenfassung

- Algorithmus**
 - Eigenschaften
 - Determinismus
- BubbleSort**
 - Verfahrensanweisung
 - JAVA Implementierung
- Rekursive Datenstruktur (Binär)Bäume**
 - Knotendefinition als Referenztyp
 - In-/pre-/post-order Durchläufe
 - Insert in sortierten Binärbaum
- BinSort**
 - Aufbau eines sortierten Baums mittels insert
 - Inorder Durchlauf








Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 41

Themen dieser Unit

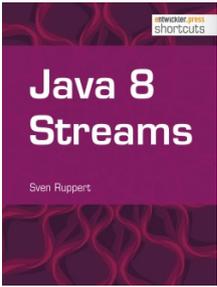
Rekursive Routinen	Rekursive Algorithmen und Datenstrukturen	Lambdas
<ul style="list-style-type: none"> Rekursiv definierter Methoden Beispiele für rekursive Methoden Formulierung rekursiver Methoden 	<ul style="list-style-type: none"> Algorithmus Rekursive Datenstrukturen (Binärbäume) BubbleSort vs BinSort 	<ul style="list-style-type: none"> Anonyme Funktionen Funktionsstypen Streams Filter Map Reduce





Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 42

Wo können Sie es nachlesen?



Kapitel 2

Core Methods

- 2.1 forEach
- 2.2 map
- 2.3 filter
- 2.5 reduce
- 2.6 limit/skip
- 2.7 distinct
- 2.9 allMatch/anyMatch/noneMatch/count



<http://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html>

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 43

Streams

- Pipeline für Datenströme (bitte nicht mit I/O Streams aus Unit 4 verwechseln!)
- Streams sind für den Einsatz von Lambdas konzipiert worden
- Streams ermöglichen keinen wahlfreien Zugriff, nur auf das erste Element
- Streams sind **lazy**, sie liefern Elemente erst, wenn eine Operation auf einem Element angewendet werden soll
- Streams können **unendlich** sein. Mittels Generatorfunktionen können Streams die permanente Lieferung von Daten generieren.
- Streams lassen sich gut parallelisieren (dazu mehr im 2. Semester)





Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 44

Erzeugen von Streams

Üblicherweise erzeugt man Streams aus Collections mittels der `stream()` Methode.

```
List<String> list = Arrays.asList("Ich", "bin", "ein", "Beispiel");
Stream<String> s1 = list.stream();
```

Es geht aber auch mit der `Stream.of(T...)` Methode

```
Stream<String> s2 = Stream.of("Noch", "ein", "Beispiel");
```

Oder so ...

```
Stream<String> s3 = Stream.of("Noch,ein,Beispiel".split(","));
```



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 45

Unendliche Streams (I)

Klingt komisch (*Jeder Rechner ist letztlich endlich, wie soll da etwas Unendliches hineinpassen?*). Geht aber.

Hier mal ein Beispiel für einen Stream, der unendlich viele ganzzahlige Zufallswerte zwischen 0 und 1000 erzeugt.

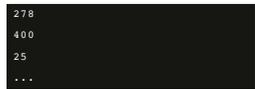
```
Stream<Integer> rand = Stream.generate(() -> (int)(Math.random() * 1000));
```

Der Trick ist, dass man in der Programmierung natürlich nicht unendliche Streams komplett ausliest :-)

Wir wollen hier nur einmal die ersten 100 davon ausgeben (es werden also 100 angefordert) und mit jeder Anforderung wird dann (lazy) eine Zufallszahl erzeugt.

```
rand.limit(100).forEach(r -> {
    System.out.println(r);
});
```





Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 46

Unendliche Streams (II)

Mittels `iterate(T, UnaryOperator<T>)` kann man auch Streams mittels eines Generatorlambdas generieren.

Hier mal ein Beispiel für einen Stream, der unendlich viele ganzzahlige Werte beginnend ab 1 in Dreierschritten erzeugt.

```
Stream<Integer> incs = Stream.iterate(1, x -> x + 3);
```

```
incs.limit(100).forEach(r -> {
    System.out.println(r);
});
```

```
1
4
7
10
13
16
...

```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Wie machen wir aus Streams wieder Collections?

Streams sind gut um Daten zu verarbeiten. Aber irgendwann brauchen wir die Daten wieder in einem „direkteren“ Zugriff (zumindest in Java).

So können wir bspw. eine Liste mit 10 Zufallszahlen erzeugen.

```
Stream<Integer> rand = Stream.generate(() -> (int)(Math.random() * 1000));
List<Integer> rs = rand.limit(10)
    .collect(Collectors.toList());
System.out.println(rs);
```

```
[978, 323, 331, 583, 484, 421, 916, 296, 476, 525]
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Wie machen wir aus Streams einen String?

Insbesondere für Konsolenausgaben ist es hilfreich, einen Stream in einen String konvertieren zu können.

So lässt sich bspw. eine Liste mit 10 kommaseparierten Zufallszahlen erzeugen und ausgeben.

```
Stream<Integer> rand = Stream.generate(() -> (int)(Math.random() * 1000));
String out = rand.limit(10)
    .map(i -> "" + i) // Integer -> String
    .collect(Collectors.joining(", "));
System.out.println(out);
```

```
978, 323, 331, 583, 484, 421, 916, 296, 476, 525
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Wie machen wir aus Streams ein Mapping?

Mappings sind Key-Value Paare. Insbesondere wenn Sie in Streams nach Gruppen von Elementen suchen, sind Mappings ggf. ein geeignetes „Zwischenformat“ für ihre Verarbeitung.

Beispielsweise wollen wir in einem Stream von ganzzahligen Zufallswerten [0..1000] bestimmen, welche Zufallszahlen im Bereich von [0..333[(Gruppe „small“), welche im Bereich von [333..666[(Gruppe „medium“) und welche im Bereich [666..1000] (Gruppe „big“) liegen.

```
Stream<Integer> rand = Stream.generate(() -> (int)(Math.random() * 1000));
Map<String, List<Integer>> map = rand.limit(10)
    .collect(Collectors.groupingBy(r ->
        if (r >= 0 && r < 333) return "small";
        if (r >= 333 && r < 666) return "medium";
        return "big";
    ));
System.out.println(map);
```

```
{small=[73, 178, 234], big=[947, 843, 774, 976], medium=[625, 359, 605]}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Miniübung:





Bestimmen Sie aus einem Stream von 1.000.000 ganzzahligen Zufallszahlen zwischen 0 und 100 wie viele Zufallszahlen in den Bereichen [0..19], [20..39], [40..59], [60..79], [80..100] anteilig (bezogen auf alle gezogenen Zufallszahlen) liegen.

Sie sollen etwa folgenden Konsolenoutput erzeugen und nur Lambdas und Streams nutzen:

```
[80..100]: 19.9497%
[60..79]: 20.0396%
[20..39]: 19.9402%
[40..59]: 20.0571%
[0..19]: 20.0134%
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Miniübung:





Lösung:

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Lambda Funktionen (I)



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Uns ist in den letzten Beispielen eine \rightarrow Notation über den Weg gelaufen.

```
String out = rand.limit(10)
    .map(i -> "" + i) // Integer -> String
    .collect(Collectors.joining(", "));
```

Diese Notation definiert eine anonyme Funktion (oder auch Ad hoc Funktion, Lambda Ausdruck).

Wir hätten für $i \rightarrow "" + i$ auch folgendes schreiben können

```
String int2String(int i) {
    return "" + i;
}
```

und (wenn Java konsequent wäre) folgendes schreiben können.

```
String out = rand.limit(10)
    .map(int2String) // Integer -> String
    .collect(Collectors.joining(", "));
```

Hinweis: Aber dann hätten wir uns einen Namen ausdenken müssen, der nur an einer Stelle genutzt wird und außerdem Typparameter rumschleppen müssen.

Hinweis: Leider ist Java nicht konsequent und die Notation geht aus Gründen der Abwärtskompatibilität nicht. Andere Sprachen können so etwas.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 73

Lambda Funktionen (II)



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Um Lambda-Ausdrücke zu formulieren, gehen wir wie folgt vor.

```
(Typ1 n1, Typ2 n2, Typ3 n3, ...) -> { anweisungen; }
```

Beispiel: Lambda-Ausdruck zum Multiplizieren.

```
(int x, int y) -> { return x * y; }
```

Auf Typen kann aber dank Typinferenz verzichtet werden.

```
(x, y) -> { return x * y; }
```

Besteht die rechte Seite nur aus einer Anweisung kann auf die Klammer verzichtet werden.

Ferner kann auf `return` verzichtet werden, wenn der Anweisungsblock nur dazu dient einen Ausdruck auszuwerten.

```
(x, y) -> x * y
```

Hinweis: Allgemeine und verbesserte Form. Üblicherweise werden vereinfachte Formen genutzt (siehe unten).

Hinweis: Typinferenz bedeutet, dass der Datentyp aus der Verwendung des Lambda-Ausdrucks abgeleitet werden kann.

Hinweis: Aufgrund dieser kompakten Notation, ist dies die präferierte Form wie Lambda-Ausdrücke genutzt werden.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 74

Lambda Funktionen (III)



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Hat ein Lambda-Ausdruck nur einen Parameter, so kann auch noch die Klammer um die Parameter auf der linken Seite weggelassen werden.

```
x -> x * x
```

Hat ein Lambda-Ausdruck keinen Parameter, so kann dies wie folgt notiert werden.

```
() -> System.out.println("Hello World")
```

```
() -> 10
```

Hinweis: Einfacher geht es nicht über wirklich nicht mehr :-)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 75

Funktionen an Funktionen übergeben



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Was soll das alles?

Mit Java 8 können sie nun auch (über den Umweg neuer Funktionstypen) Funktionen (also ausführbare Logik!) als Parameter für Methoden definieren.

Oder anders gesagt:
Man kann einer Methode eine „Methode“ übergeben.
Klingt komisch, klingt innovativ, ist es aber nicht.



Nur Java konnte das jahrelang nicht!

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 76

Funktionen an Funktionen übergeben



FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Die Stream-Klasse definiert eine Reihe von Methoden, die Lambda-Funktionen (also Code!) als Parameter erwarten.

Es gibt dabei unterschiedliche Arten von Funktionen, hier eine Auswahl, die mit Java 8 definiert worden sind.



Name	Beschreibung
Predicate<T>	Prüft einen Parameter des Typs T auf eine Eigenschaftserfüllung (liefert Boolean)
BiPredicate<R, S>	Prüft zwei Parameter des Typs R und S darauf, ob sie in einer Relation zueinander stehen oder nicht (liefert Boolean)
Function<T, R>	Funktionen die Parameter des Typs T auf Ergebnisse des Typs R abbildet. Bspw. Länge einer Zeichenkette T == String, R == Integer
UnaryOperator<T>	Ein Operator der Parameter des Typs T in Ergebnisse desselben Typs umrechnet. Bspw. Kann die Fakultät als unärer Operator angesehen werden. 3! = 6 (int -> int)
BinaryOperator<T>	Ein Operator der zwei Parameter des Typs T in Ergebnisse desselben Typs umrechnet. Bspw. + 4 + 3 = 7 (int, int) -> int
BiFunction<T, U, R>	Funktionen die zwei Parameter des Typs T und U auf Ergebnisse des Typs R abbilden. Bspw. Funktion zum Suchen der Häufigkeit eines Zeichens in einer Zeichenkette (String, Character) -> Integer

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 77

Miniübung:






FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Definieren sie eine Lambda-Funktionen und weisen sie diese einem geeigneten Funktionstyp zu.

Prüfen ob eine ganzzahlige Zahl gerade ist.

```
even.test(5) == false
```

Bestimmen der Länge einer Zeichenkette.

```
length.apply("Hello") == 5
```

Quadrieren einer Fließkommazahl.

```
sqr.apply(5.0) == 25.0
```

Bestimmung des Divisionsrests (Modulo) zweier ganzzahliger Zahlen.

```
modulo.apply(5, 2) == 1
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 78

Miniübung:

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Bestimmung der Anzahl eines Zeichens in einer Zeichenkette.

```
count.apply('o', "Hello World") == 2
```

```
count.apply('o', "Hello World") == 2
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Stream::forEach()

Streams bieten nun eine Reihe von Methoden an, die solche Funktionstypen als Parameter nutzen.

Möchte man bspw. einen Stream einfach Element für Element durchgehen und für jedes Element Anweisungen ausführen, so geht dies mittels `forEach()`

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
Consumer<Integer> print = i -> System.out.println(" " + i);
stream.forEach(print);
```

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
stream.forEach(i -> System.out.println(" " + i));
```

```
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 0
```

Oder auch kürzer:

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
stream.forEach(i -> System.out.println(" " + i));
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Stream::map()

Möchte man in einem Stream auf jedem Element eine Funktion ausführen, so geht dies mittels `map()`.

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
Function<Integer, String> toString = i -> "" + i + "";
Stream<String> is = stream.map(toString);
System.out.println(is.collect(Collectors.toList()));
```

```
['1', '2', '3', '4', '5', '6', '7', '8', '9', '0']
```

Hinweis:
Die Langformen werden hier nur angegeben, um den Funktionstyp explizit zu machen.

Oder auch in dieser Form:

```
Stream<String> is = stream.map(i -> "" + i + "");
System.out.println(is.collect(Collectors.toList()));
```

Hinweis:
Die kürzeren Formen sind gebräuchlicher und auch flexibler einsetzbar.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Stream::sorted()

Möchte man einen Stream sortieren, so geht dies mittels `sorted()`.

```
Stream<String> strings =
    Stream.of("Dies", "ist", "ein", "Beispiel");
Comparator<String> byLength = (s1, s2) -> s1.length() - s2.length();
Stream<String> sorted = strings.sorted(byLength);
sorted.forEach(s -> System.out.println(s));
```

```
ist
ein
Dies
Beispiel
```

Hinweis:
Ein Comparator vergleicht zwei Werte w1 und w2.
Liefert der Comparator für w1, w2 etwas kleiner als Null so steht w1 vor w2 in der Ordnung (ist kleiner).
Liefert der Comparator für w1, w2 etwas größer als Null so steht w1 hinter w2 in der Ordnung (ist größer).
Liefert der Comparator für w1, w2 Null so steht w1 und w2 in derselben Ordnung (ist gleich).
Mittels `sorted` können Sie also eine beliebige Ordnung definieren (vgl. Informatik I) und anhand dieser sortieren.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Stream::filter()

Möchte man in einem Stream Elemente herausfiltern, die einer Bedingung genügen (z.B. nur ungerade Werte), geht dies mittels `filter()`.

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
Predicate<Integer> odd = i -> i % 2 != 0;
Stream<Integer> is = stream.filter(odd);
System.out.println(is.collect(Collectors.toList()));
```

```
[1, 3, 5, 7, 9]
```

Hinweis:
Die Langformen werden hier nur angegeben, um den Funktionstyp explizit zu machen.

Oder auch in dieser Form:

```
Stream<Integer> is = stream.map(i -> i % 2 != 0);
System.out.println(is.collect(Collectors.toList()));
```

Hinweis:
Die kürzeren Formen sind gebräuchlicher und auch flexibler einsetzbar.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

FACH
HOCHSCHULE
LÜBECK
University of Applied Sciences

Stream::limit()

Möchte man in einem Stream nur die ersten *n* Elemente verarbeiten, so geht dies mittels `limit()`.

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
Stream<Integer> is = stream.limit(4);
System.out.println(is.collect(Collectors.toList()));
```

```
[1, 2, 3, 4]
```

Hinweis:
Ist *n* größer als Elemente in dem Stream *m* vorhanden sind, wird ein Stream der Länge *m* < *n* zurückgeliefert.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Stream::skip()

Möchte man in einem Stream die ersten n Elemente **nicht** verarbeiten, so geht dies mittels `skip()`.

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
Stream<Integer> is = stream.skip(4);
System.out.println(is.collect(Collectors.toList()));
```

```
[5, 6, 7, 8, 9, 0]
```

Hinweis:
Ist n größer als Elemente in dem Stream m vorhanden sind, wird ein leerer Stream zurückgeliefert.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 85

Stream::distinct()

Möchte man in einem Stream nur unterschiedliche Elemente verarbeiten, so geht dies mittels `distinct()`.

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 5, 4, 3, 2, 1);
Stream<Integer> is = stream.distinct();
System.out.println(is.collect(Collectors.toList()));
```

```
[1, 2, 3, 4, 5]
```

Hinweis:
Hierbei wird die Wertgleichheit herangezogen, nicht die Referenzgleichheit. D.h. die Elemente innerhalb des Streams werden mittels `equals()` und nicht mittels `==` verglichen!

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 86

Miniübung:

Bestimmen Sie wie viele gleiche Zufallszahlen in folgendem Stream im Bereich der Zufallszahlen 100 bis 200 (jeweils inklusive) gezogen wurden.

```
Stream<Integer> rands = Stream.generate(() -> (int)(Math.random() * 100));
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 87

Miniübung:

Gegeben sei folgende for-Schleife.

```
for(int i = 10; i <= 100; i += 3) {
    System.out.println(i);
}
```

Übersetzen sie diese in einen Lambdaausdruck mit demselben Verhalten.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 88

Stream::reduce()

Möchte man alle Werte in einem Stream auf einen Wert „zusammenrechnen“ so geht dies mittels `reduce()`.

`reduce(id, op)` hat dabei zwei Parameter.

- Der initiale Wert `id` mit dem die Reduktion begonnen wird (das neutrale Element `id` der Reduktionsoperation `op`, d.h. $x \text{ op } id == x$)
- Die Reduktionsoperation `op`, diese muss assoziativ sein (d.h. $(x \text{ op } y) \text{ op } z == x \text{ op } (y \text{ op } z)$)

`op` sei die Addition, d.h. `+`
`id` für `+` ist `0`

0 +	1	2	3	4	5
1 +	2	3	4	5	
3 +	3	4	5		
6 +	4	5			
10 +	5				
15					

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 89

Stream::reduce()

Da die Reduktionsoperation assoziativ ist, kann das ganze auch in einer beliebigen anderen Reihenfolge (und auch **parallel**) erfolgen.

Hinweis:
Initiale Operation mit neutralem Element aus Gründen der Anschaulichkeit weggelassen!

Hinweis:
Das geht natürlich auch in einer beliebigen anderen Reduktionsabfolge, vorausgesetzt die Reduktionsoperation ist assoziativ!

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 90

Stream::reduce()

Möchte man alle Werte in einem Stream auf einen Wert „zusammenrechnen“ so geht dies mittels `reduce()`.

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 0);
BinaryOperator<Integer> plus = (x, y) -> x + y;
int sum = stream.reduce(0, plus);
System.out.println(sum);
```

45

Oder auch in dieser Form:

```
int sum = stream.reduce(0, (x, y) -> x + y);
System.out.println(sum);
```

Hinweis:
Die Langformen werden hier nur angegeben, um den Funktionstyp explizit zu machen.

Hinweis:
Die kürzeren Formen sind gebräuchlicher und auch flexibler einsetzbar.

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 91

Miniübung:

Gegeben ist folgender Stream von Zeichenketten.

```
Stream<String> strings = Stream.of("Hello", "functional", "crazy", "World");
```

Verknüpfen sie diesen Stream (und beliebige andere) zu einer Zeichenkette in der jedes Element durch ein Leerzeichen getrennt ist. Nutzen sie ausschließlich die `reduce()` Methode.

"Hello functional crazy World"

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 92

Miniübung:

Gegeben ist folgender Stream von Integerwerten.

```
Stream<Integer> rands = Stream.generate(() -> (int)(Math.random() * 1000));
```

Bestimmen sie aus den ersten 100 Einträgen des Streams das Maximum. **Nutzen sie zur Reduktion nur die `reduce()` Funktion.**

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 93

Miniübung:

Gegeben ist folgender Stream von Integerwerten.

```
Stream<Integer> rands = Stream.generate(() -> Math.random(1000));
```

Geben sie die ersten 10 Einträge in folgender Form
[877, 567, 678, 400, 300, 177, 999, 675, 444, 666]
als Zeichenkette aus. **Nutzen sie zur Reduktion nur die `reduce()` Funktion.**

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 94

Stream::count()

Möchte man in einem Stream die Anzahl vorhandener Elemente bestimmen, so geht dies mittels `count()`.

```
Stream<Integer> stream = Stream.of(1, 2, 3, 4, 5, 5, 4, 3, 2, 1);
long quantity = stream.count();
System.out.println(quantity);
```

10

Hinweis:
Achtung der Rückgabetype von `count()` ist `long`. D.h. es wird von der Möglichkeit seeeeeehr langer Streams ausgegangen :-)

Die Länge eines unendlich langen Streams, dauert unendlich lange zu berechnen (irgendwie logisch). Unten stehende Zeile wird also nie terminieren ...

```
long nr = Stream.iterate(1, x -> x + 1).count(); // terminiert nie!
```

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 95

Stream::all/any/noneMatch()

Möchte man in einem Stream alle Elemente gegen eine Bedingung prüfen, so geht dies mit den Methoden

1. `boolean allMatch(Predicate<T>)` alle Elemente genügen einer Bedingung
2. `boolean anyMatch(Predicate<T>)` mindestens ein Element genügt einer Bedingung
3. `boolean noneMatch(Predicate<T>)` kein Element genügt einer Bedingung

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 96

Stream::all/any/noneMatch()

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Gegeben sei folgende Liste und folgende Prädikate:
`List<Integer> is = Arrays.asList(1, 2, 3, 4, 5);`
`Predicate<Integer> even = i -> i % 2 == 0;`
`Predicate<Integer> isZero = i -> i == 0;`

Geben Sie bitte an, welche Ausdrücke zu false und welche zu true ausgewertet werden.

<code>is.stream().allMatch(even)</code>	// false
<code>is.stream().anyMatch(even)</code>	// true
<code>is.stream().noneMatch(even)</code>	// false
<code>is.stream().allMatch(isZero)</code>	// false
<code>is.stream().anyMatch(isZero)</code>	// false
<code>is.stream().noneMatch(isZero)</code>	// true

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Miniübung:

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Gegeben sei folgende (klassische) Methode zur Erzeugung von Primzahlen bis n .

```
public static List<Integer> erathostenes(int n) {
    boolean[] deleted = new boolean[n + 1];
    List<Integer> primes = new LinkedList<>();

    for (int i = 2; i < Math.sqrt(n); i++) {
        if (!deleted[i]) primes.add(i);
        for (int j = i * i; j <= n; j += i) deleted[j] = true;
    }

    for (int i = (int)Math.sqrt(n) + 1; i <= n; i++)
        if (!deleted[i]) primes.add(i);

    return primes;
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Miniübung:

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Für kryptografische Anwendung benötigen sie eine einfache Methode Primzahlen zu erzeugen. Sie stoßen bei Wikipedia

<https://de.wikipedia.org/wiki/Primzahlgenerator>

auf die folgenden Funktionen, die auf Euler zurück gehen sollen (Euler gilt im Allgemeinen ja als verlässliche Quelle), die beide jeweils nur Primzahlen erzeugen sollen.

$$p_1(n) = n^2 + n + 17 \quad \text{Euler 1}$$

$$p_2(n) = n^2 - n + 41 \quad \text{Euler 2}$$

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Miniübung:

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Sie können die Methode `List<Integer> erathostenes(int)` als korrekt annehmen. Prüfen Sie Euler 1 und Euler 2 für die ersten 20 generierten Primzahlen auf Korrektheit. Geben Sie im Falle fehlerhafter Primzahlen falsch generierte Primzahlen für Euler 1 und Euler 2 aus.

Was passiert, wenn Sie Euler 2 für die ersten 100 generierten Primzahlen prüfen?

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Und nun HTML mit Lambdas erzeugen zu sein wird

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Um zu prüfen, ob verstanden Du hast ...

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Miniübung:

FACH HOCHSCHULE LÜBECK
University of Applied Sciences

Gegeben sind Strings folgenden Formats:

```
String students = "Mustermann, Max" + '\n' +
    "Musterfrau, Maren" + '\n' +
    "Hilfos, Holger" + '\n' +
    "Loniki, Tessa";
```

Wandeln Sie Strings in HTML Zeichenketten, so dass sie in einem HTML Browser als Tabelle dargestellt würden.

```
<table>
<tr><td>Max</td><td>Mustermann</td></tr>
<tr><td>Maren</td><td>Musterfrau</td></tr>
<tr><td>Holger</td><td>Hilfos</td></tr>
<tr><td>Tessa</td><td>Loniki</td></tr>
</table>
```

Nutzen sie ausschließlich Lambdas, um dies zu realisieren.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

Miniübung: 

```
String students = "Mustermann, Max" + '\n' +  
"Musterfrau, Maren" + '\n' +  
"Hilflos, Holger" + '\n' +  
"Loniki, Tessa";
```

Lösung:

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 183

Zusammenfassung  

- **Streams**
 - Erzeugen mittels `.stream()` oder `Stream.of()`
 - Mittels `.collect()` in Collections wandeln
- **Lambda Funktionen**
 - Anonyme Funktionen
 - `(a, b, c) -> expression(a, b, c)`
- **Funktionsarten**
 - Predicate
 - Function, BiFunction
 - UnaryOperator, BinaryOperator
- **Streammethoden**
 - `forEach()`
 - `map()` und `reduce()`
 - `filter()`, `distinct()` und `sorted()`
 - `limit()`, `skip()`
 - `allMatch()`, `anyMatch()`, `noneMatch()`
 - `count()`




Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme 184