

# 23. Teams - Fixe Netze mit Datenfluss für die Programmierung des Internet

## Kommunikation mit Iteratoren, Senken, Kanälen und Konnektoren

Prof. Dr. Uwe Aßmann  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
Technische Universität Dresden  
Version 18-0.2, 20.04.12

- 1) Teams und Konnektoren  
(Connectors)
- 2) Kanäle
- 3) Entwurfsmuster Channel
  - 1) Entwurfsmuster Iterator  
(Stream)
  - 2) Entwurfsmuster Sink
  - 3) Channel
- 4) I/O und Persistente  
Datenhaltung mit Channels
- 5) Ereigniskanäle



DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

# Betreff: "Softwaretechnologie für Einsteiger" 2. Auflage

- ▶ zur Info: o.g. Titel steht zur Verfügung:
  - 20 Exemplare ausleihbar in der Lehrbuchsammlung
  - 1 Präsenz-Exemplar im DrePunct
  - <https://katalogbeta.slub-dresden.de/id/0011358900/#detail>
- ▶ Jeweils unter ST 230 Z96 S68(2).

# Be Careful, The Exam Will be Coming!

[https://de.wikipedia.org/wiki/A\\_Londonderry\\_Air](https://de.wikipedia.org/wiki/A_Londonderry_Air)

[https://de.wikipedia.org/wiki/Danny\\_Boy](https://de.wikipedia.org/wiki/Danny_Boy)

3

Softwaretechnologie (ST)

Oh, Danny boy, the pipes, the pipes are calling  
From glen to glen, and down the mountain side  
The summer's gone, and all the roses falling  
'Tis you, 'tis you must go and I must bide.

But come ye back when summer's in the meadow  
Or when the valley's hushed and white with snow  
'Tis I'll be there in sunshine or in shadow  
Oh, Danny boy, oh Danny boy, I love you so!



**And when ye come, and all the flow'rs are dying  
If I am dead, as dead I well may be  
Ye'll come and find the place where I am lying  
And kneel and say an Ave there for me.**

And I shall hear, though soft you tread above me  
And all my grave will warmer, sweeter be  
For you will bend and tell me that you love me,  
And I shall sleep in peace until you come to me.

Frederic Weatherly (1910)  
Text aus Wikipedia/McCourt, Danny  
Boy, S. 87 f.

# Warum müssen Softwareingenieure fortgeschrittenes Wissen besitzen?

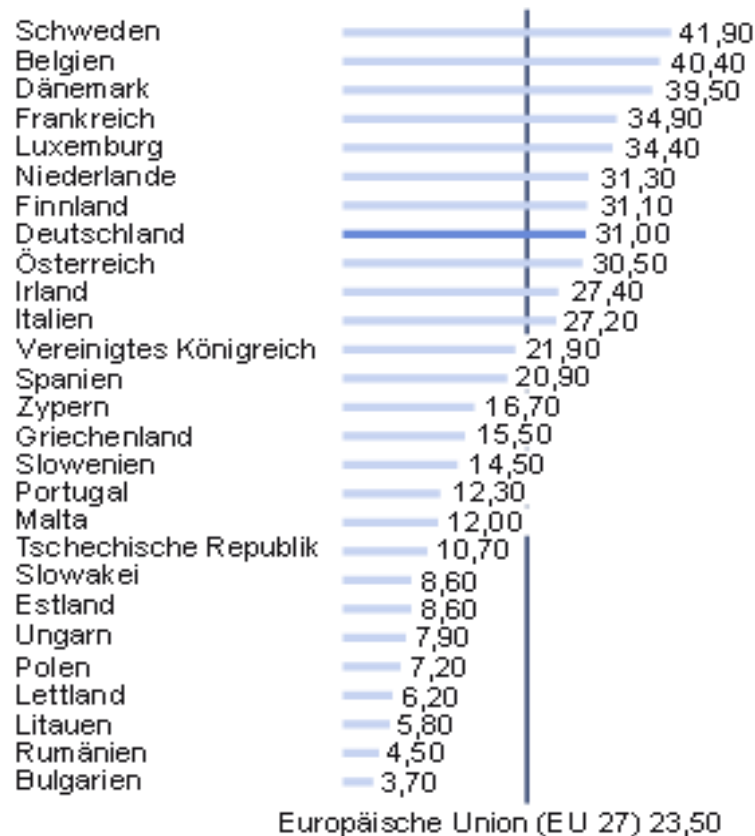
4

Softwaretechnologie (ST)

- ▶ Die Konkurrenz ist hart: Zu den Kosten der Arbeit:

## Arbeitskosten in der Privatwirtschaft 2012

je geleistete Stunde in EUR



© Statistisches Bundesamt, Wiesbaden 2013

<http://www.heise.de/resale/imgs/17/1/0/0/1/3/4/1/ArbeitskostenEULAenderStart2012-9bb2e8b041f1342e.png>

# Überblick Teil II

5

Softwaretechnologie (ST)

Höhere  
Sprachen

Graphen als Sprachkonstrukte

Teams mit Konnektoren (Rollen-Kollaborationen)

Teams als Entwurfsmuster

Java

Teams mit Datenfluss mit Konnektoren

Graphen als Bibliotheken (Java)

Graphen als Collections abgeflacht (Java)

Graphen als Endoassoziationen:  
Netze von Unterobjekten in komplexen Objekten (Java)

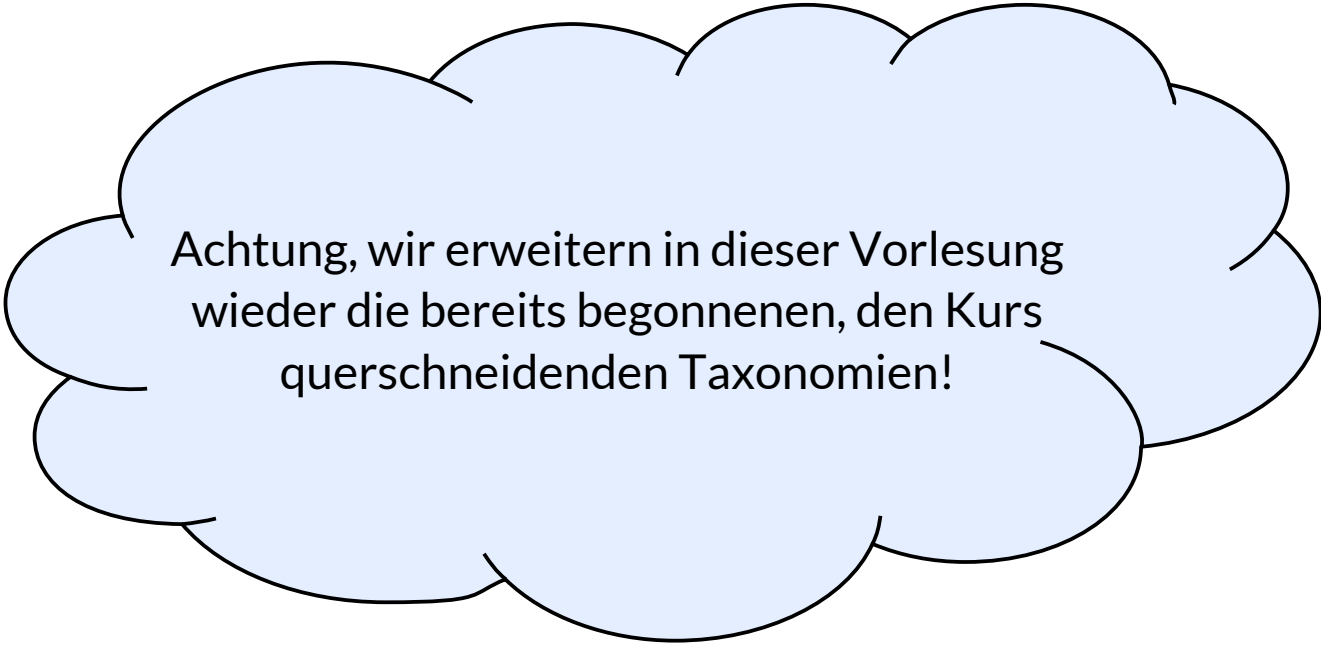
System-  
programmier-  
sprache (C)

Graphen als Datenstrukturen fester Länge abgeflacht

# E.23.1 Lernen mit Begriffshierarchien, die die Vorlesung querschneiden



- ▶ Wie lernt man mit Ihnen?
  - Klassen-Taxonomie
  - Methoden-Taxonomie
  - Realisierungen von Graphen



Achtung, wir erweitern in dieser Vorlesung  
wieder die bereits begonnenen, den Kurs  
querschneidenden Taxonomien!

## 23.1 Teams

- ▶ Objekte kommunizieren in Teams (fixen Netzen)
- ▶ Teams kommunizieren oft auf kontinuierliche Art, mit wechselnden Partnern, aber in einem fixen Netz
- ▶ Auf dem Internet ist das die Regel





Def.: Ein **Team** besteht aus einer festen Anzahl von interagierenden und kommunizierenden Objekten.

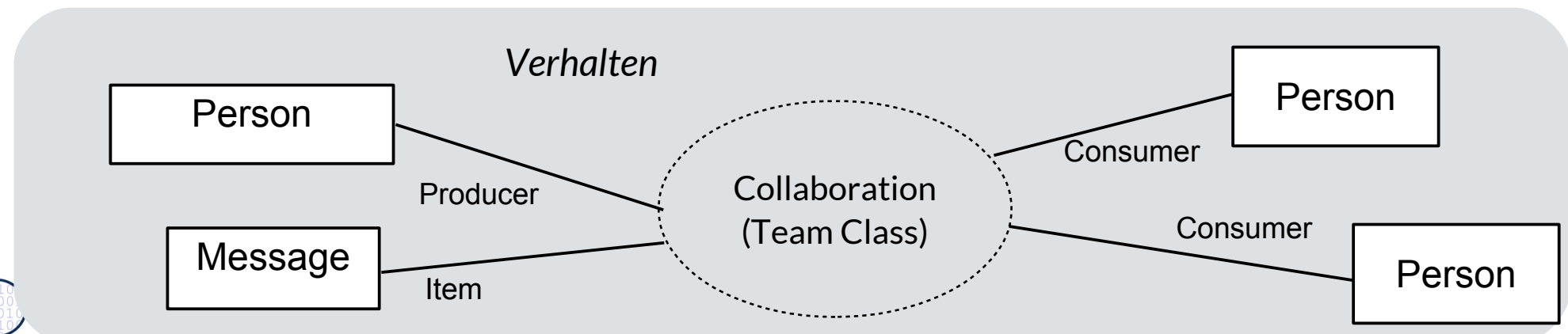
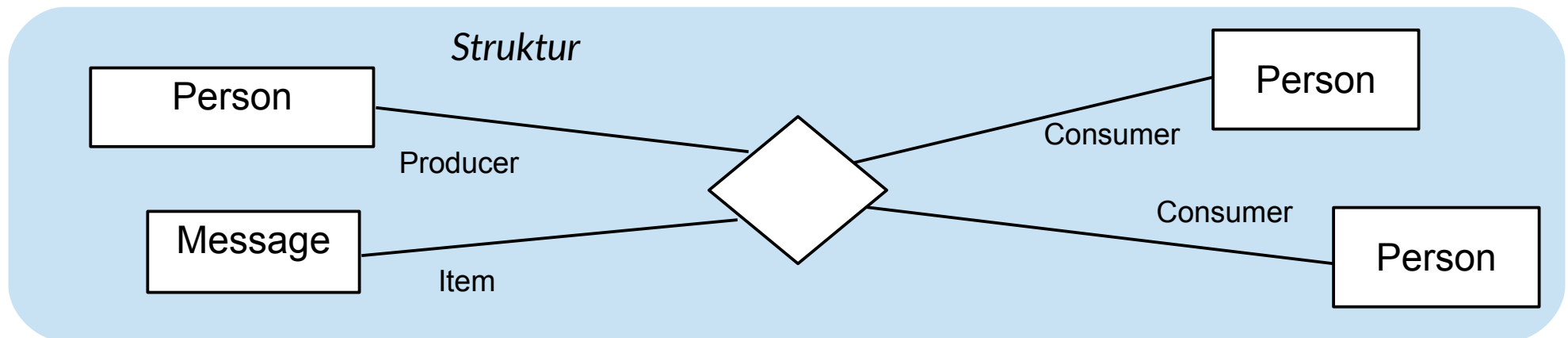
Die Interaktion des Teams wird in UML durch eine **Kollaboration** beschrieben.

# Kollaborationen kapseln das Verhalten von Netzen

10

Softwaretechnologie (ST)

- ▶ Die *Struktur* von Netzen wird in UML durch n-stellige Assoziationen dargestellt
- ▶ Das Verhalten eines Netzes und seine Kommunikation durch Kollaborationen
- ▶ Def.: Eine **Kollaboration (collaboration)** realisiert die Kommunikation eines fixen Netzes mit einem festen anwendungsspezifischen Protokoll
- ▶ Eine **Teamklasse (Konnektor)** kapselt eine Kollaboration in einer Klasse

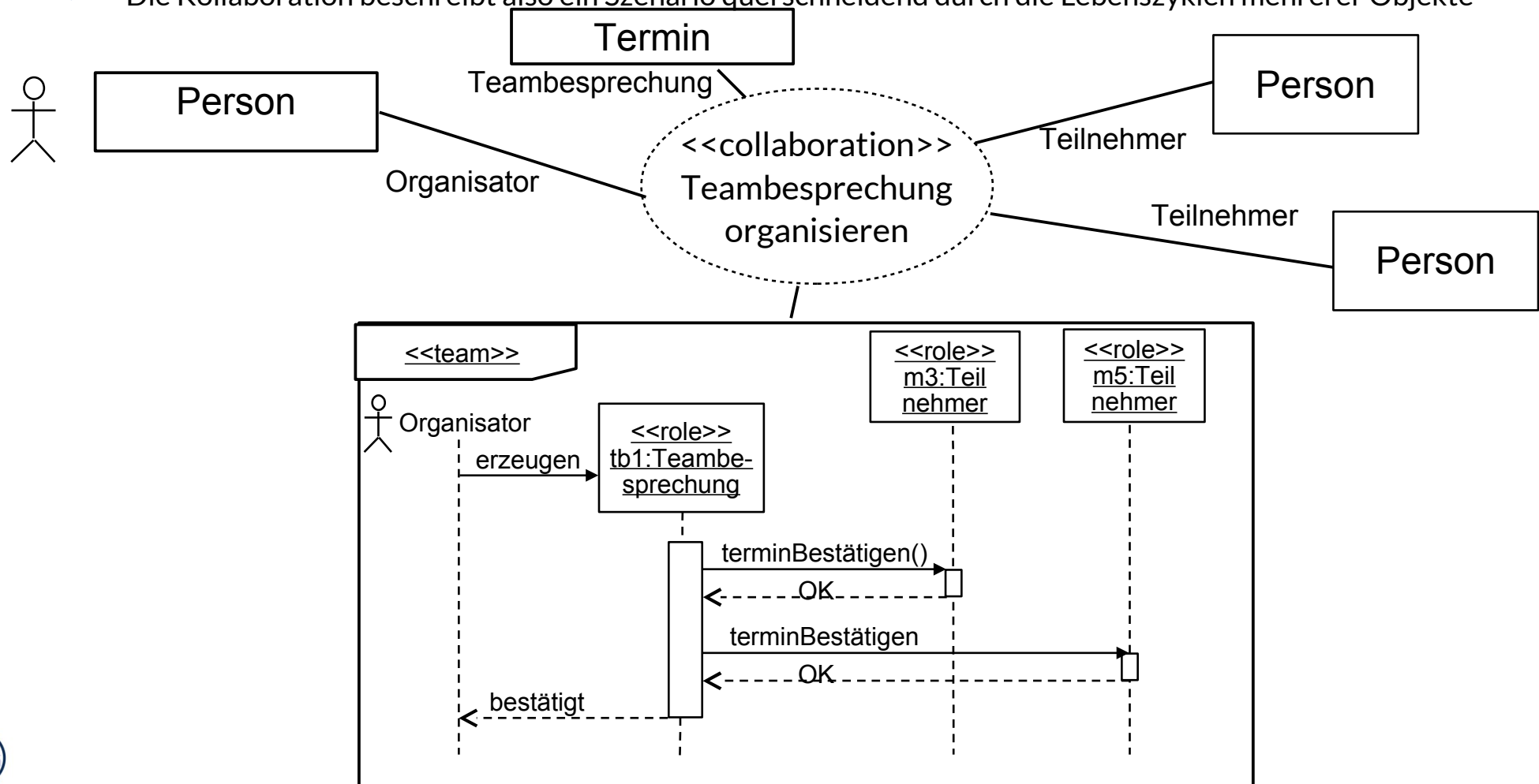


# Kollaborationen kapseln Verhalten durch Interaktionsprotokolle

11

Softwaretechnologie (ST)

- ▶ **Kollaborationen** beschreiben die anwendungsspezifische Interaktion, Nebenläufigkeit und Kommunikation eines Teams von Beteiligten
- ▶ Def.: Ein **Team** realisiert eine Kollaboration durch eine feste Menge von Rollenobjekten, koordiniert durch ein Hauptobjekt. Es wird oft mit einem Sequenzdiagramm als Verhalten unterlegt
  - Die einzelnen Lebenslinien geben das Verhalten einer Rolle der Kollaboration an
- ▶ Die Kollaboration beschreibt also ein Szenario querschneidend durch die Lebenszyklen mehrerer Objekte



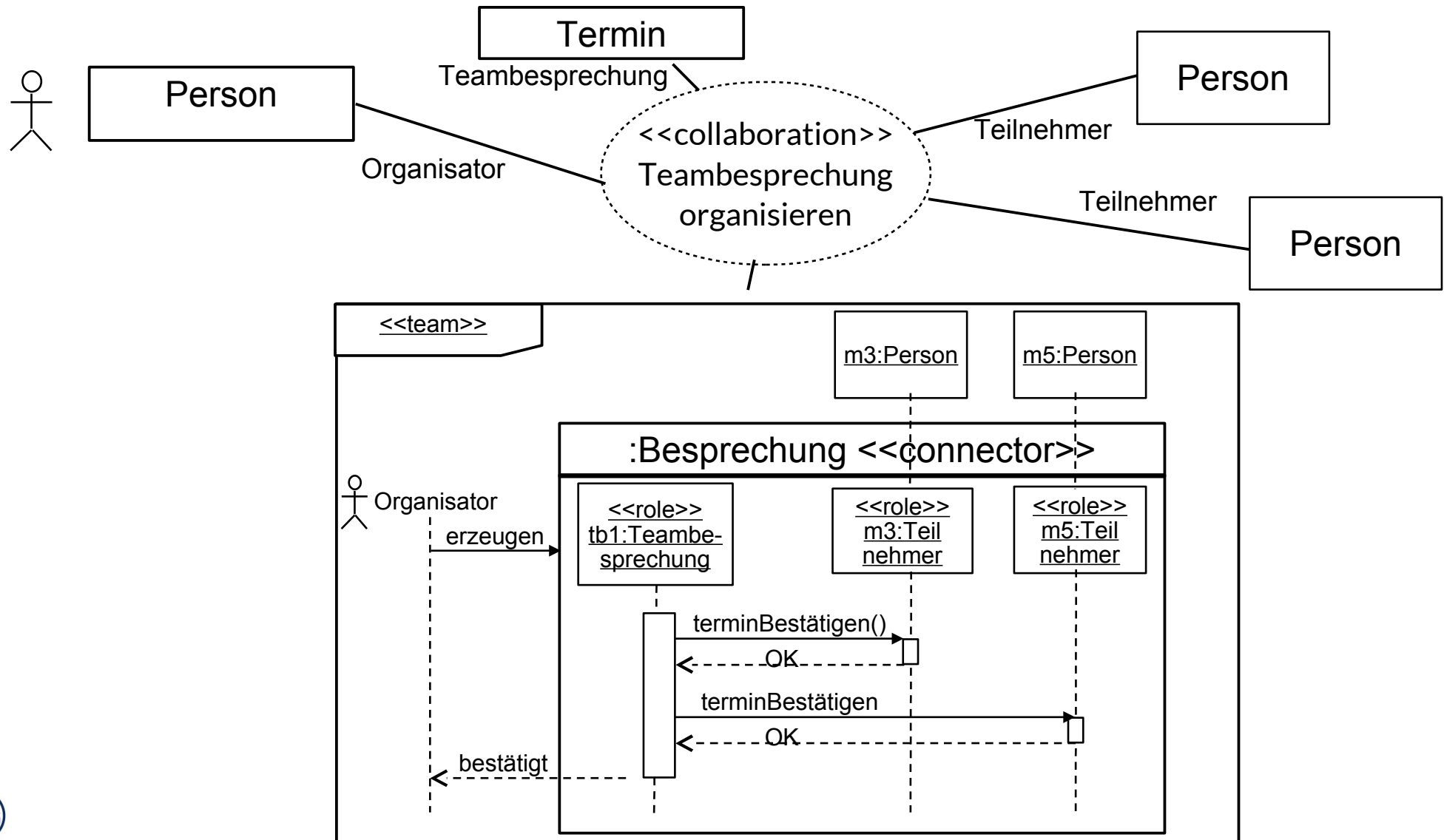
Def.: Kann eine Kollaboration durch eine Klasse gekapselt werden, spricht man von einer **Teamklasse (Konnektor)**.

# Kollaborationen kapseln Verhalten durch Interaktionsprotokolle

13

Softwaretechnologie (ST)

- Def.: Ein **Teamklasse (Konnektor)** realisiert eine Kollaboration durch eine Klasse (ein Hauptobjekt)



Beobachtung:  
Viele Entwurfsmuster beschreiben  
Schemata für **Teams** und ihre **Kollaborationen**.

## 23.2 Kanäle (Channels) für die Programmierung des Internets

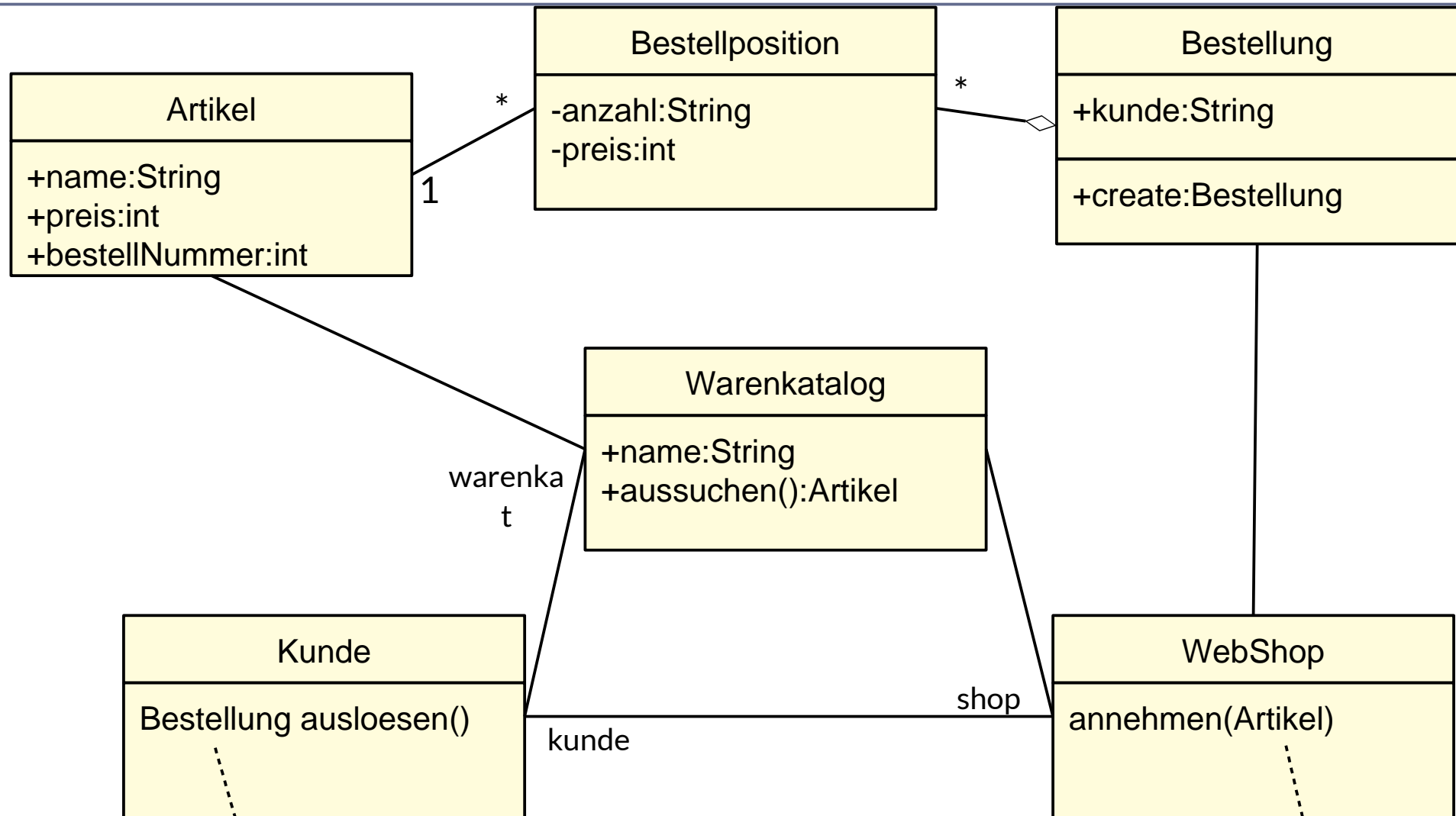
- ▶ Kanäle bilden einfach Konnektoren (Teamklassen)
- ▶ Web-Objekte kommunizieren oft in Teams auf kontinuierliche Art, mit wechselnden Partnern, aber in einem fixen Netz



# Beispiel: Bestellung auf einem Webshop

16

Softwaretechnologie (ST)



```
while (hatBedarf())
{
    artikel = warenkat.aussuchen();
    shop.bestellen(artikel);
}
```

```
while (true)
{
    b = Bestellung.create();
    b.workOn();
    b.ship();
}
```



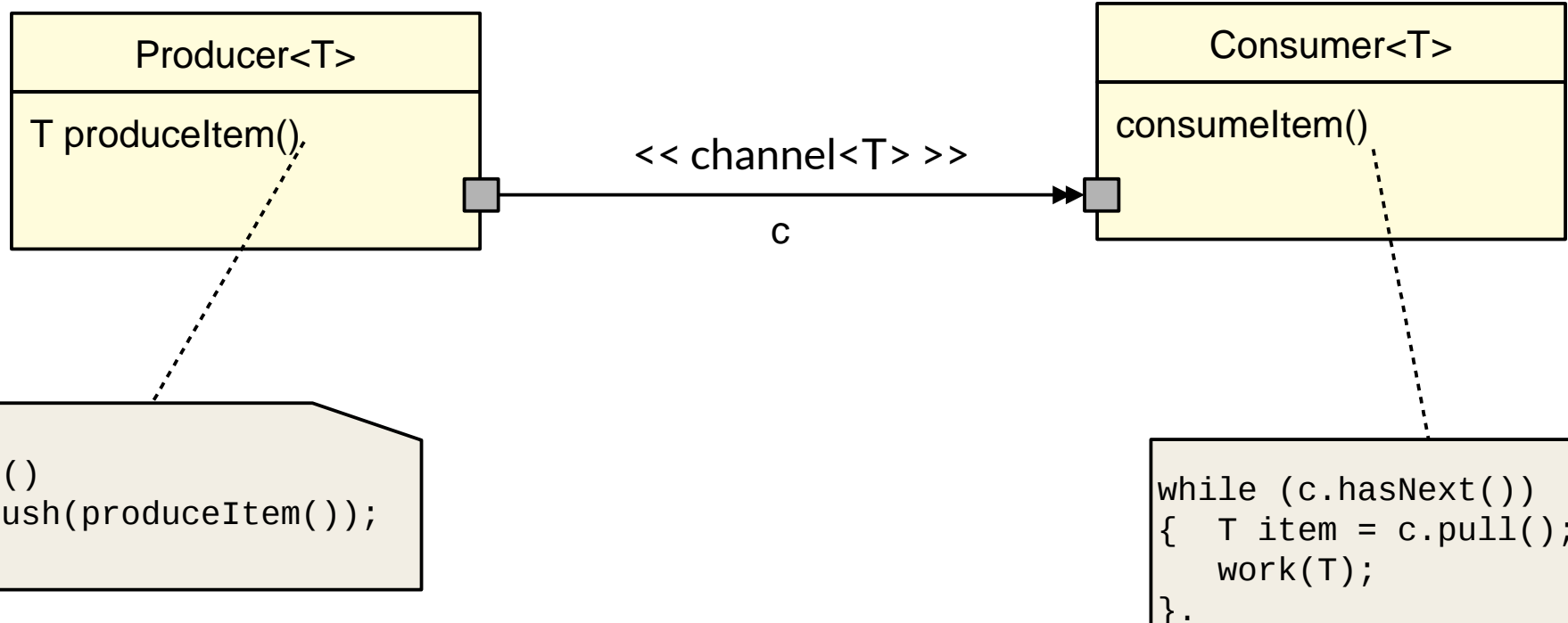
# Kanäle bilden Netze mit Datenfluss

17

Softwaretechnologie (ST)

**Def.:** Ein **Kanal (channel, pipe, stream)** ist ein Konnektor, der zur Kommunikation von Anwendungsklassen mit *Datenfluss* dient. In den Stream werden Daten gegeben und aus dem Stream werden Daten entnommen

- ▶ UML Notation: Andocken eines Kanals an *sockets (ports)*
- ▶ [https://en.wikipedia.org/wiki/Douglas\\_McIlroy](https://en.wikipedia.org/wiki/Douglas_McIlroy)
- ▶ [https://en.wikipedia.org/wiki/Pipeline\\_\(Unix\)](https://en.wikipedia.org/wiki/Pipeline_(Unix))



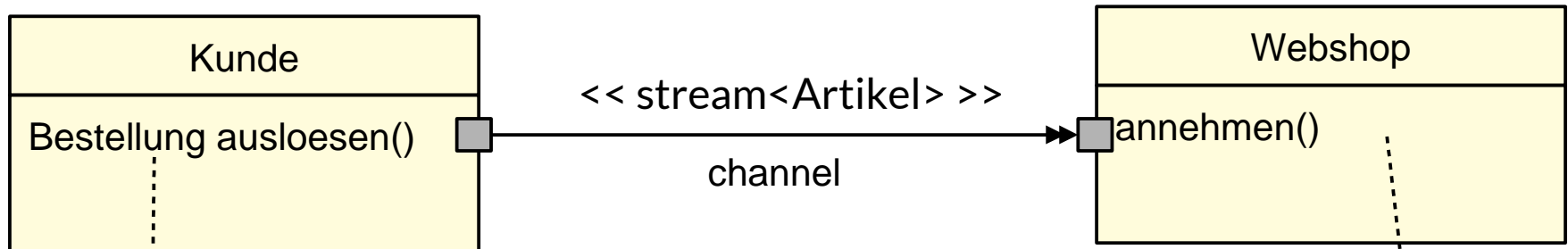
# Vorteil von Kanälen: Partnerwechselnde Kommunikation

Problem  
Partnerwechsel

18

Softwaretechnologie (ST)

- ▶ Webshops dürfen die konkreten Objekte ihrer Kunden nicht kennen
- ▶ Kanäle erlauben, die Partner zu wechseln, ohne Kenntnis des Netzes
  - Ideal für Netze mit dynamisch wechselnden Partnern
  - Ideal für Webprogrammierung



```
while (hatBedarf())
{
    artikel = warenkat.aussuchen();
    channel.bestellen(artikel);
}
```

```
while (true)
{
    b = Bestellung.create();
    artikel = channel.pull();
    b.workOn(artikel);
    b.ship();
}
```

## 23.3 Entwurfsmuster Channel

- ▶ Eines der wesentlichen Entwurfsmuster für Internet- und Webprogrammierung.

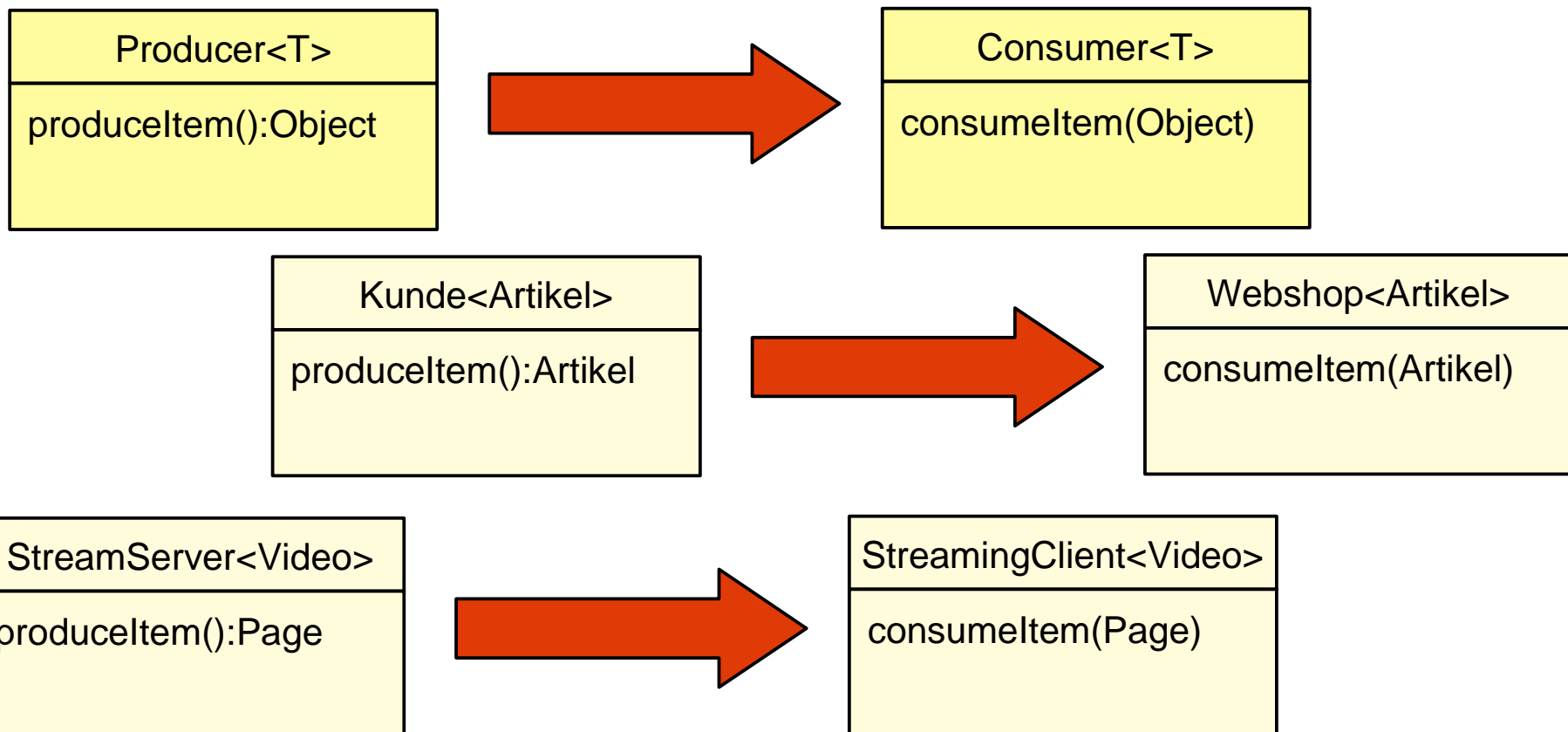


# Wie organisiere ich die “unendlich lange” Kommunikation zweier Akteure?

20

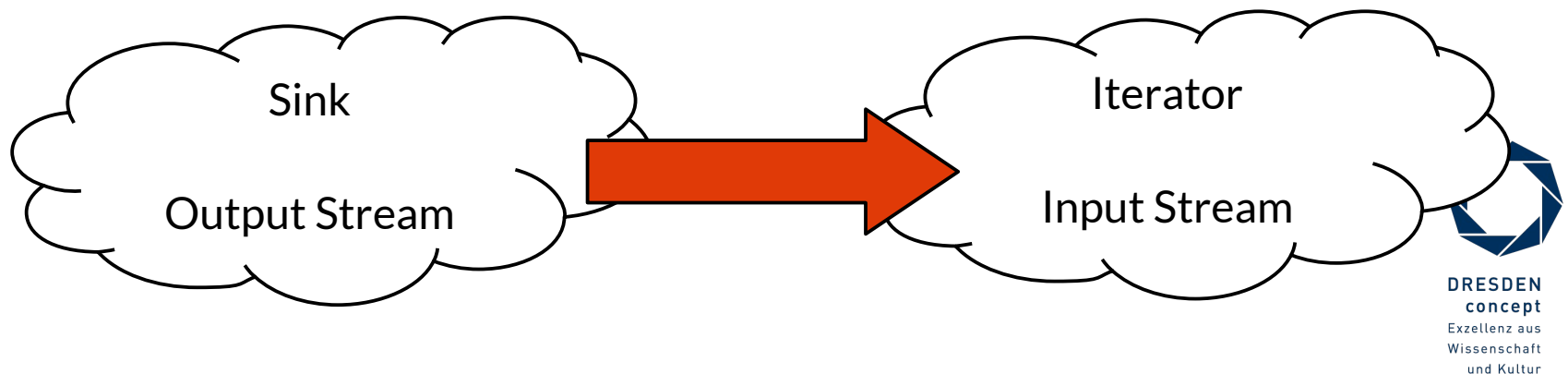
Softwaretechnologie (ST)

- ▶ **Problem:** Über der Zeit laufen in einem Webshop eine Menge von Bestellungen auf
  - Sie sind aber nicht in endlicher Form in Collections zu repräsentieren
- ▶ **Frage:** Wie repräsentiert man potentiell unendliche Collections?
- ▶ **Antwort:** mit Kanälen.



## 23.3.1 Entwurfsmuster Iterator (Eingabestrom, input stream)

Kanäle bestehen aus mit einander verbundenen Enden, mindestens zweien

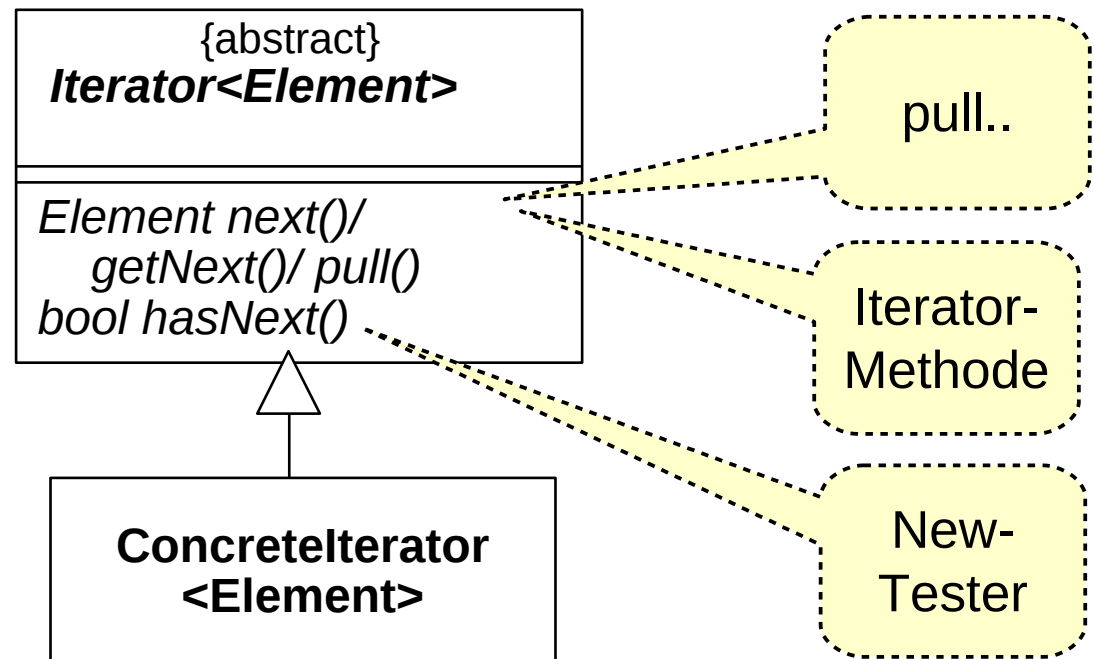


# Entwurfsmuster Iterator (Input Stream) (Implementierungsmuster)

22

Softwaretechnologie (ST)

- ▶ Ein **Eingabestrom (input stream)** ist eine potentiell unendliche Folge von Objekten (zeitliche Anordnung einer pot. unendlichen Folge)
- ▶ Eingabe in eine Klasse oder Komponente

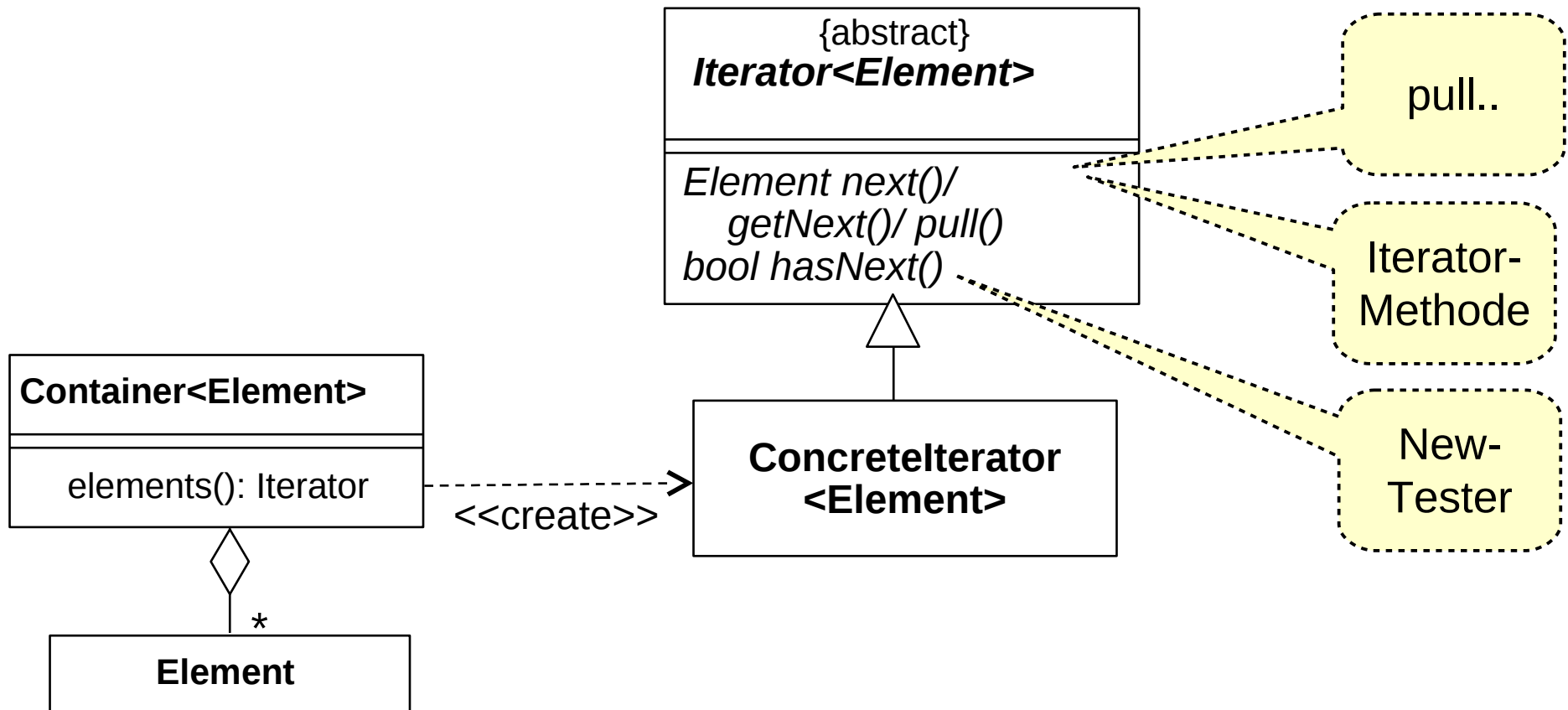


# Entwurfsmuster Iterator (Input Stream) (Implementierungsmuster)

23

Softwaretechnologie (ST)

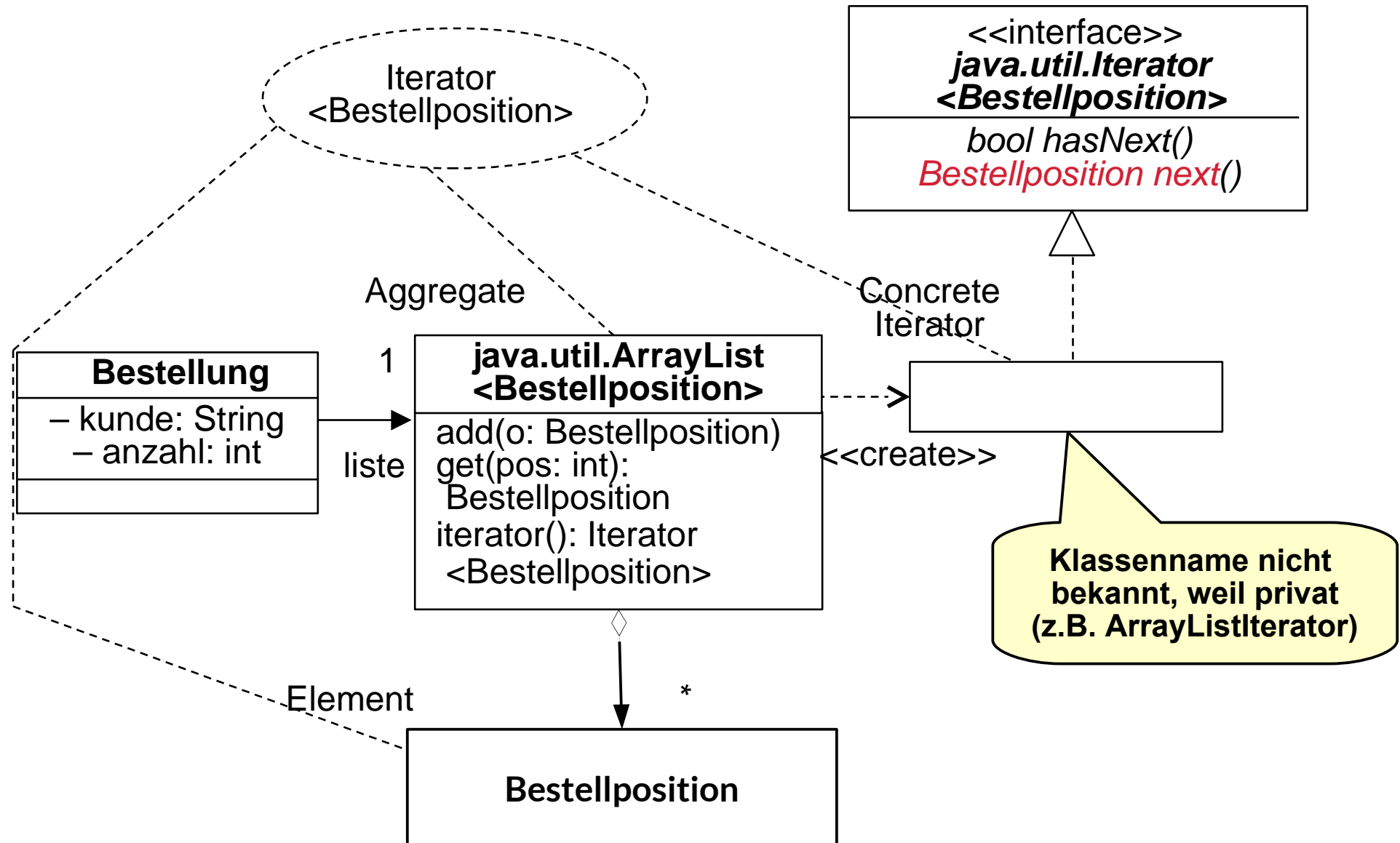
- ▶ Containerklassen haben Iteratoren



# Iterator-Beispiel in der JDK (ArrayList)

24

Softwaretechnologie (ST)





# Implementierungsmuster Iterator

25

Softwaretechnologie (ST)

## ► Verwendungsbeispiel:

```
T thing;  
List<T> list;  
  
..  
Iterator<T> i = list.iterator();  
while (i.hasNext()) {  
    doSomething(i.next());  
}
```

## ► Einsatzzwecke:

- Verbergen der inneren Struktur
- **bedarfsgesteuerte Berechnungen** auf der Struktur
- “unendliche” Datenstrukturen

# Anwendungsbeispiel mit Iteratoren

26

Softwaretechnologie (ST)

```
import java.util.Iterator;
...

class Bestellung {

    private String kunde;
    private List<Bestellposition> liste;

    ...
    public int auftragssumme() {
        Iterator<Bestellposition> i = liste.iterator();
        int s = 0;
        while (i.hasNext())
            s += i.next().positionspreis();
        return s;
    }
    ...
}
```

Online:  
Bestellung2.java

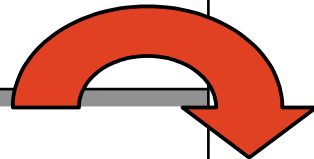
# For-Schleifen auf Iterable-Prädikatschnittstellen

27

Softwaretechnologie (ST)

- ▶ Erbt eine Klasse von `Iterable`, kann sie in einer vereinfachten *for*-Schleife benutzt werden
- ▶ Typisches Implementierungsmuster

```
class BillItem extends Iterable {  
    int price; }  
class Bill {  
    int sum = 0;  
    private List billItems;  
    public void sumUp() {  
        for (BillItem item: billItems) {  
            sum += item.price;  
        }  
    }  
    return sum;  
}
```



```
class BillItem { int price; }  
class Bill {  
    int sum = 0;  
    private List billItems;  
    public void sumUp() {  
        for (Iterator i = billItems.iterator();  
             i.hasNext(); ) {  
            item = i.next();  
            sum += item.price;  
        }  
    }  
    return sum;  
}
```

## 23.3.2 Senken (Sinks, OutputStream)

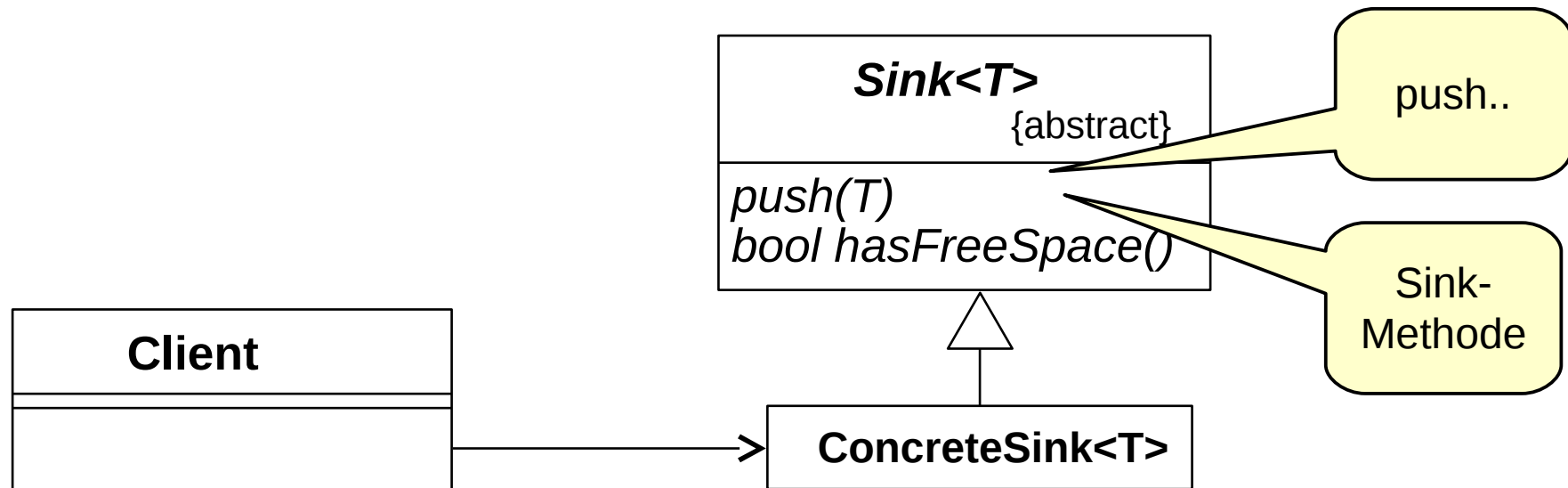


# Entwurfsmuster Senke (Implementierungsmuster)

29

Softwaretechnologie (ST)

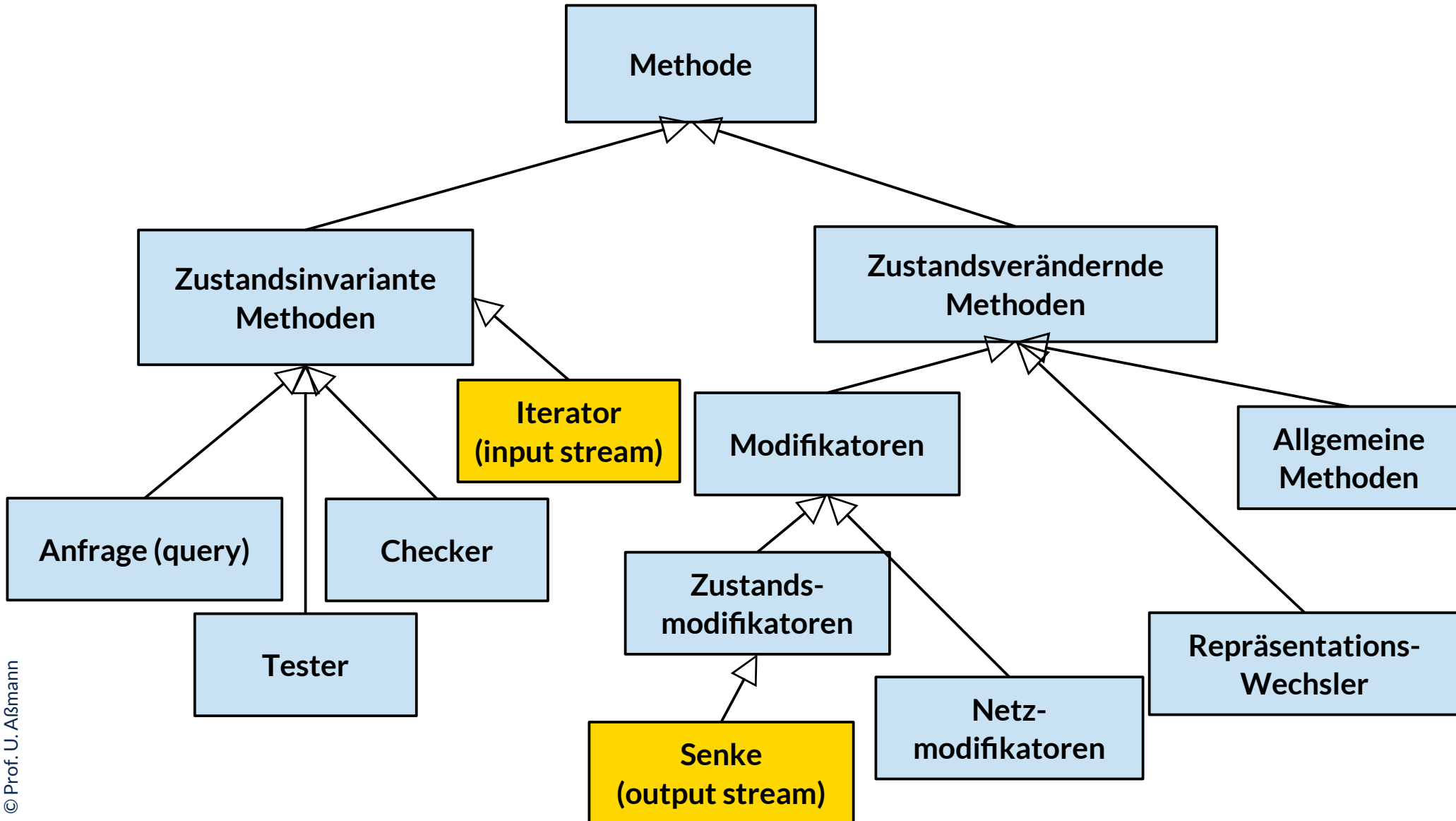
- ▶ Name: **Senke** (auch: Ablage, sink, **output stream**, belt, output-socket)
- ▶ Problem: Ablage eines beliebig großen Datenstromes.
  - push
  - ggf. mit Abfrage, ob noch freier Platz in der Ablage vorhanden
- ▶ Senken (sockets) organisieren den Datenverkehr zum Internet



# Erweiterung: Begriffshierarchie der Methodenarten

30

Softwaretechnologie (ST)



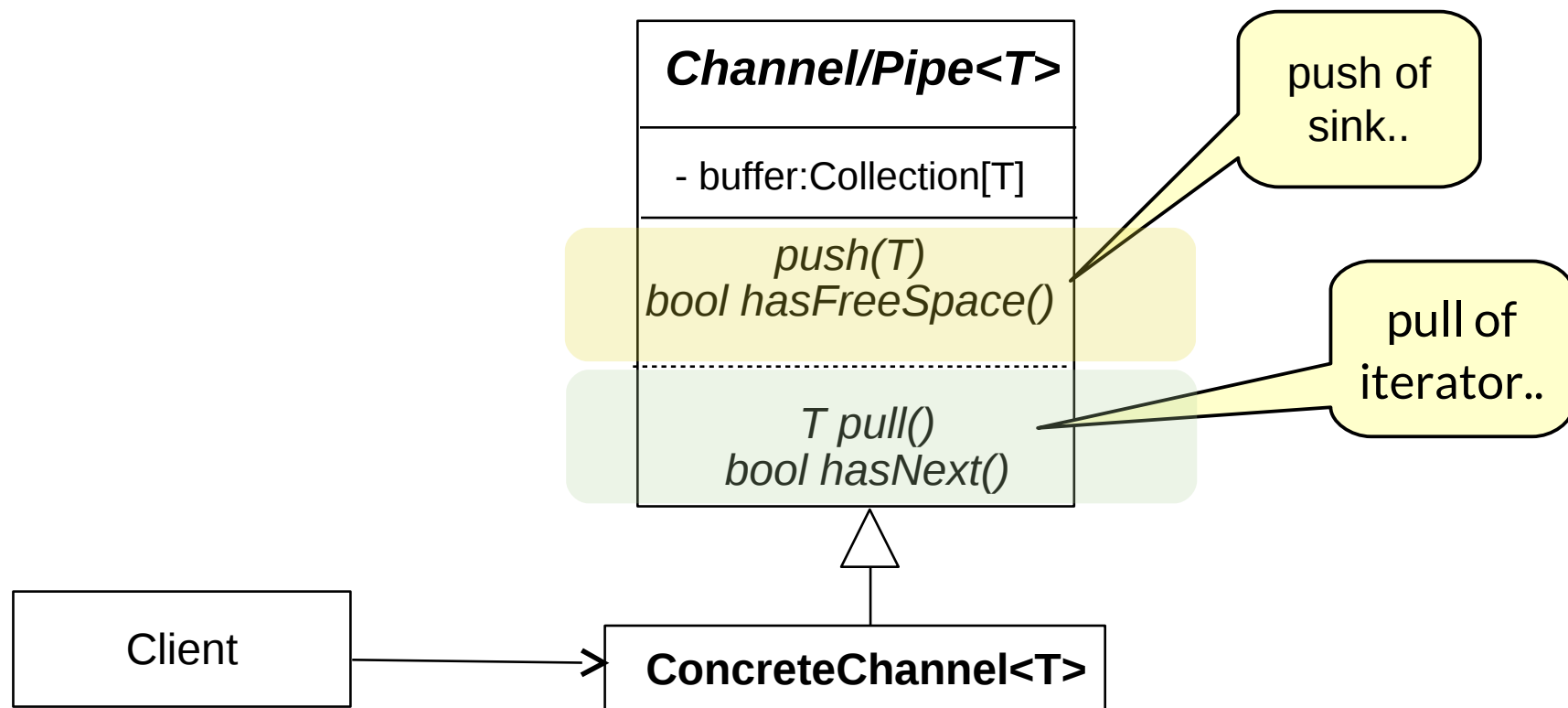
## 23.3.3 Channels (Pipes)

- ▶ Die Kombination aus Senken und Iteratoren, ggf. mit beliebig großem Datenspeicher
- ▶ Eine Pipe ist ein einfacher Konnektor



# Entwurfsmuster Channel und Pipe (Implementierungsmuster)

- ▶ **Name:** Ein **Channel** (Kanal, full stream) ist ein Konnektor, der die gerichtete Kommunikation (Datenfluss) zwischen Produzenten und Konsumenten organisiert. Er kombiniert einen Iterator mit einer Senke.
- ▶ **Zweck:** asynchrone Kommunikation mit Hilfe eines internen Puffers buffer
- ▶ Wir sprechen von einer **Pipe (Puffer, buffer)**, wenn die Kapazität des Kanals endlich ist, d.h. `hasFreeSpace()` irgendwann `false` liefert.



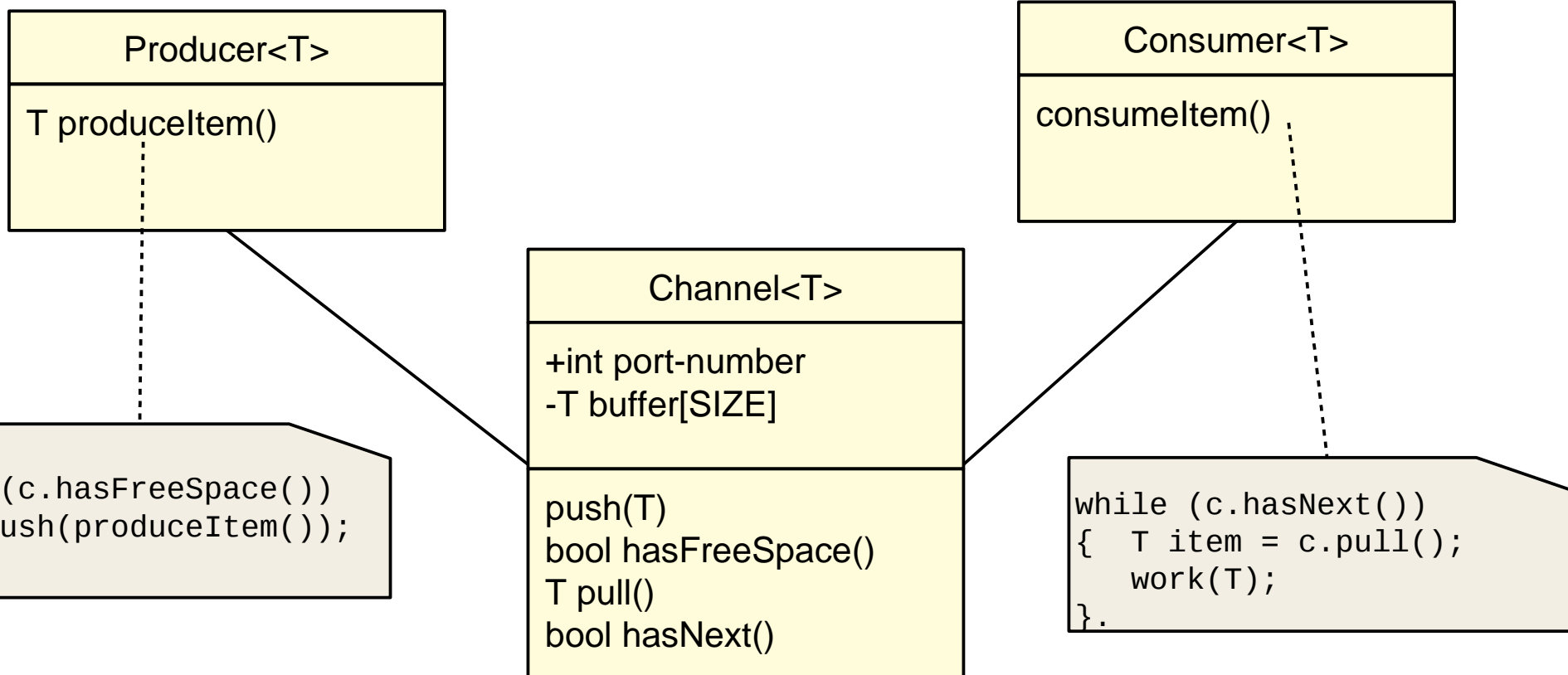


# Channels in anderen Programmiersprachen

- ▶ Channels (pipes) kommen in vielen Sprachen als Konstrukte vor
  - **Shell-Skripte in Linux** (Operator für pipes: “|”)
    - `ls | wc`
    - `cat file.txt | grep “Rechnung”`
    - `sed -e “s/Rechnung/Bestellung/” < file.txt`
  - **Communicating Sequential Processes** (CSP [Hoare], Ada, Erlang):
    - Operator für pull: “?”
    - Operator für push: “!”
  - **C++**: Eingabe- und Ausgabestream `stdin`, `stdout`, `stderr`
    - Operatoren “<<” (read) und “>>” (write)
  - Architectural Description Languages (ADL, Kurs CBSE)
- ▶ Sie sind ein elementares Muster für die Kommunikation von parallelen Prozessen (producer-consumer-Muster)

# Wie organisiere ich die Kommunikation zweier Aktoren?

- ▶ Einsatzzweck: Ein **Aktor (Prozess)** ist ein parallel arbeitendes Objekt. Zwei Aktoren können mit Hilfe eines Kanals kommunizieren und lose gekoppelt arbeiten
- ▶ Bsp.: Pipes mit ihren Endpunkten (Sockets) organisieren den Verkehr auf dem Internet; sie bilden Kanäle zur Kommunikation zwischen Prozessen (Producer-Consumer-Muster)



# Konnektoren als Verallgemeinerung von Kanälen

**Def.:** Ein **Konnektor** ist eine technische Klasse, die zur Kommunikation von Mitgliedern eines Teams dient.

- ▶ Konnektoren sind bi- oder multidirektional, sie fassen zwei oder mehrere Kanäle zusammen; Kanäle bilden spezielle *gerichtete Konnektoren*
- ▶ Kommunikation über Konnektoren muss nicht gerichtet sein; es können komplexe Protokolle herrschen
- ▶ Konnektoren können mehrere Input und Output Streams koppeln



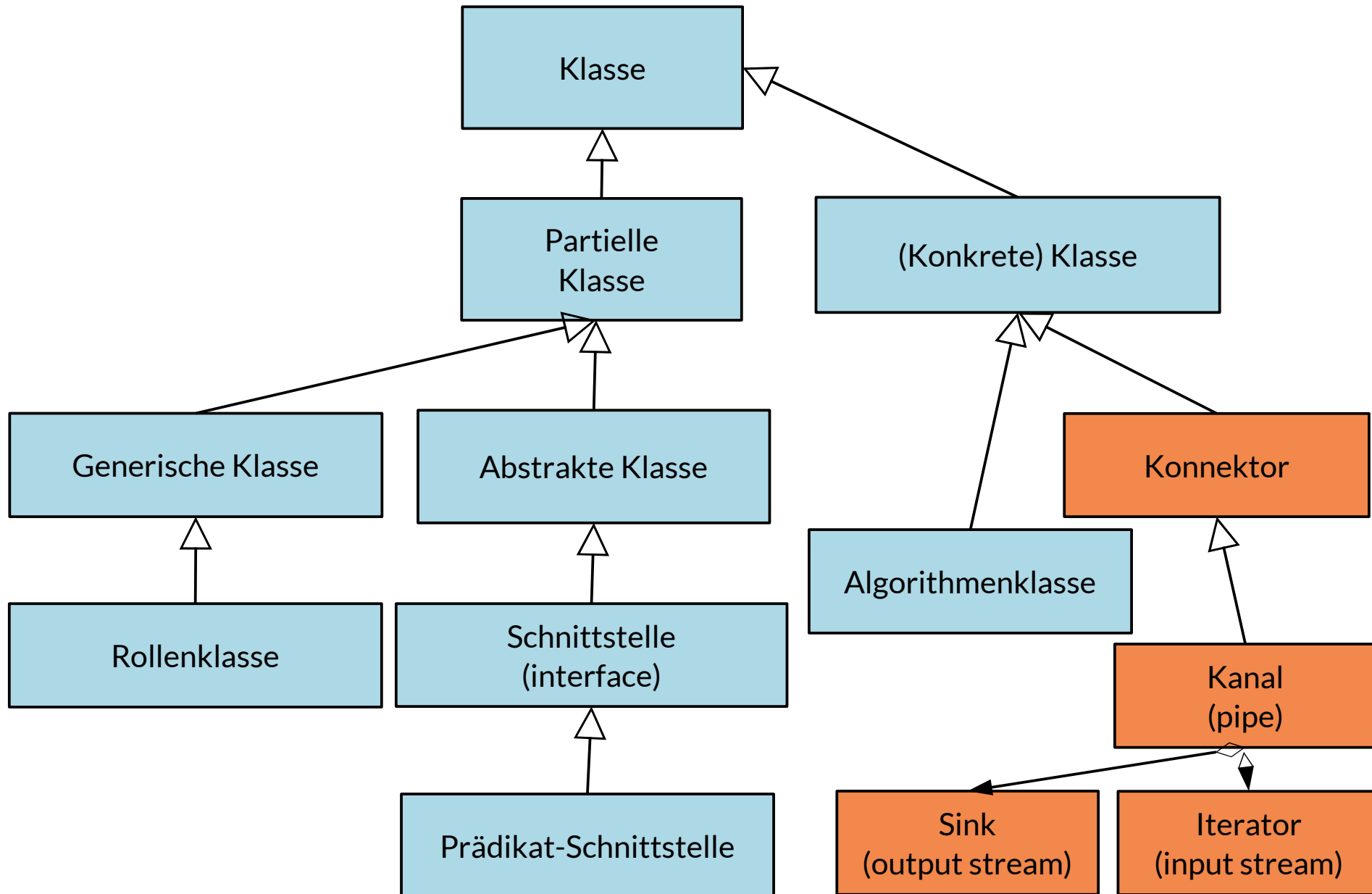
```
while (c.hasFreeSpace())
{   c.push(produceItem());
}.
```

```
while (c.hasNext())
{   T item = c.pull();
    work(T);
}.
```

## Q2: Begriffshierarchie von Klassen (Erweiterung)

36

Softwaretechnologie (ST)



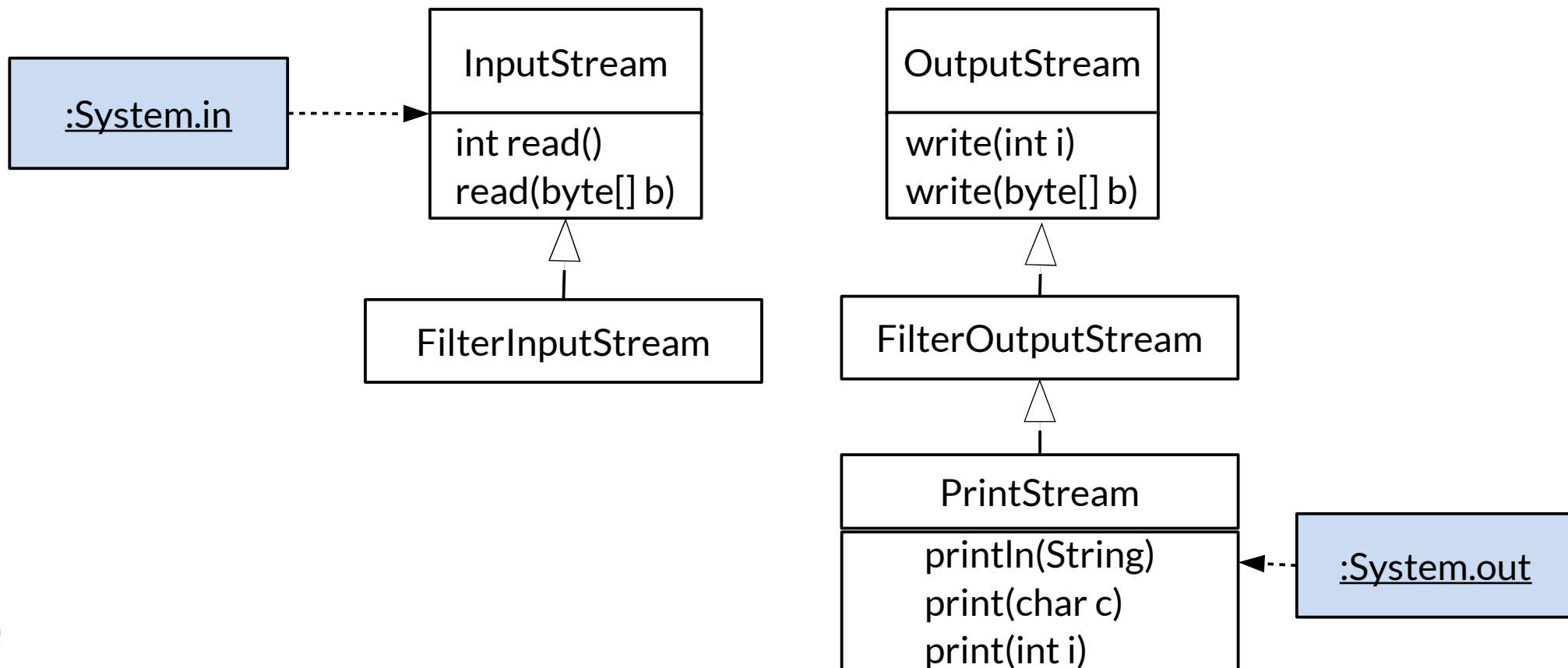
## 23.4 Strombasierte Ein- und Ausgabe (Input/Output) und persistente Datenhaltung

- ▶ In der Regel wird in einer Programmiersprache Ein- und Ausgabe über Ströme bzw. Konnektoren realisiert.
- ▶ Das JDK nutzt Iteratoren/Streams an verschiedenen Stellen



## 23.4.1 Ein- und Ausgabe in Java

- ▶ Die Klasse **java.io.InputStream** stellt einen Iterator/Stream in unserem Sinne dar. Sie enthält Methoden, um Werte einzulesen
- ▶ **java.io.OutputStream** stellt eine Senke dar. Sie enthält Methoden, um Werte auszugeben
- ▶ Die statischen Objekte `in`, `out`, `err` bilden die Sinks und Streams in und aus einem Programm, d.h. die Schnittstellen zum Betriebssystem

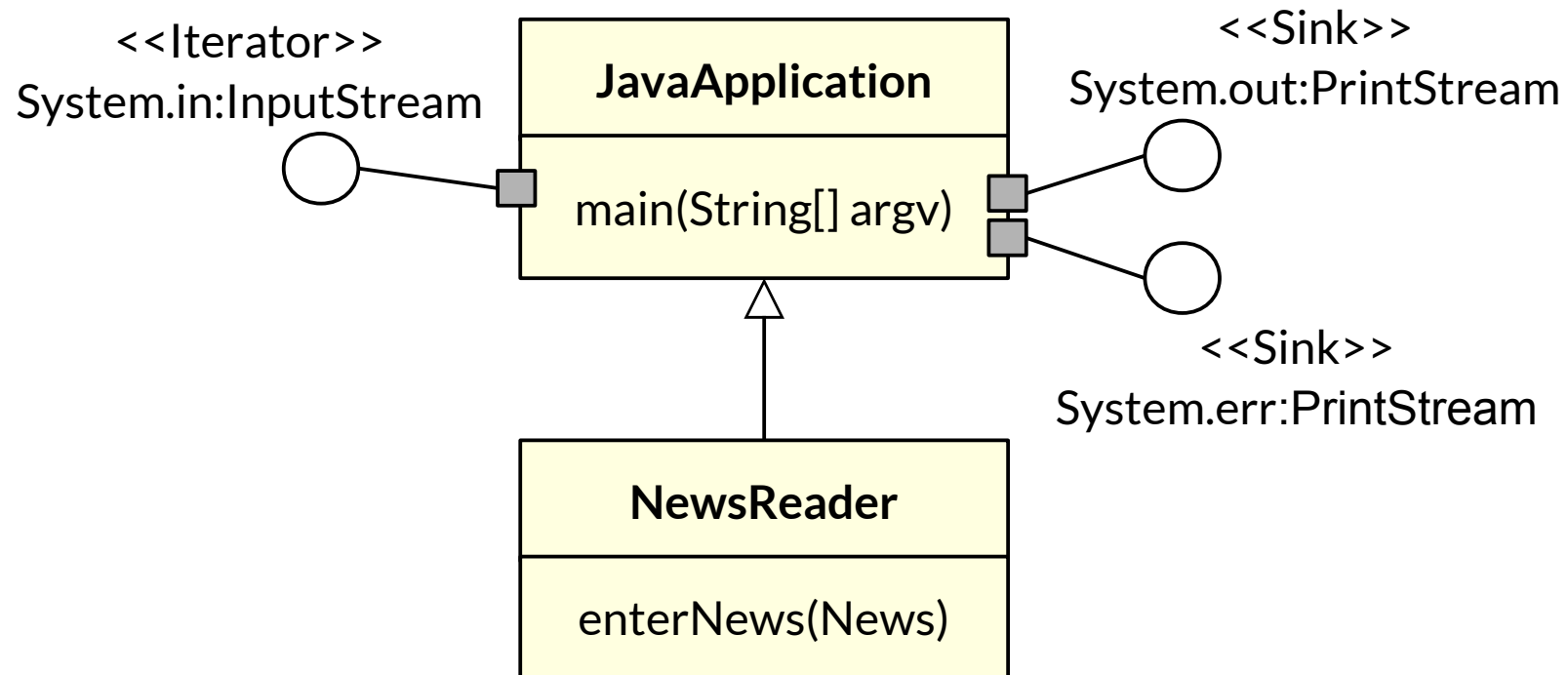


# Java-Anwendungen mit ihren Standard-Ein/Ausgabe-Strömen

39

Softwaretechnologie (ST)

- ▶ Ein Programm in Java hat 3 Standard-Ströme
  - Entwurfsidee stammt aus dem UNIX/Linux-System
- ▶ Notation: UML-Komponenten



## 23.4.2 Temporäre und persistente Daten mit Streams organisieren

- ▶ Daten sind
  - **temporär**, wenn sie mit Beendigung des Programms verloren gehen, das sie verwaltet;
  - **persistent**, wenn sie über die Beendigung des verwaltenden Programms hinaus erhalten bleiben.
    - Steuererklärungen, Bestellungen, ...
- ▶ Objektorientierte Programme benötigen Mechanismen zur Realisierung der *Persistenz von Objekten*.
  - Einsatz eines Datenbank-Systems
    - Objektorientiertes Datenbank-System
    - Relationales Datenbank-System  
Java: Java Data Base Connectivity (JDBC)
    - Zugriffsschicht auf Datenhaltung  
Java: Java Data Objects (JDO)
  - Speicherung von Objektstrukturen in Dateien mit Senken und Iteratoren
    - Objekt-Serialisierung (*Object Serialization*)
    - Die Dateien werden als Channels benutzt:
    - Zuerst schreiben in eine Sink
    - Dann lesen mit Iterator



# Objekt-Serialisierung in Java, eine einfache Form von persistenten Objekten

- ▶ Die Klasse `java.io.ObjectOutputStream` stellt eine Senke dar
  - Methoden, um ein Geflecht von Objekten linear darzustellen (zu *serialisieren*) bzw. aus dieser Darstellung zu rekonstruieren.
  - Ein `OutputStream` entspricht dem Entwurfsmuster Sink
  - Ein `InputStream` entspricht dem Entwurfsmuster Iterator
- ▶ Eine Klasse, die Serialisierung zulassen will, muß die (leere!) Prädikat-Schnittstelle `java.io.Serializable` implementieren.

```
class ObjectOutputStream {  
    public ObjectOutputStream (OutputStream out)  
        throws IOException;  
    // push Method  
    public void writeObject (Object obj)  
        throws IOException;  
}
```

# Objekt-Serialisierung: Abspeichern

42

Softwaretechnologie (ST)

```
import java.io.*;

class XClass implements Serializable {
    private int x;
    public XClass (int x) {
        this.x = x;
    }
}

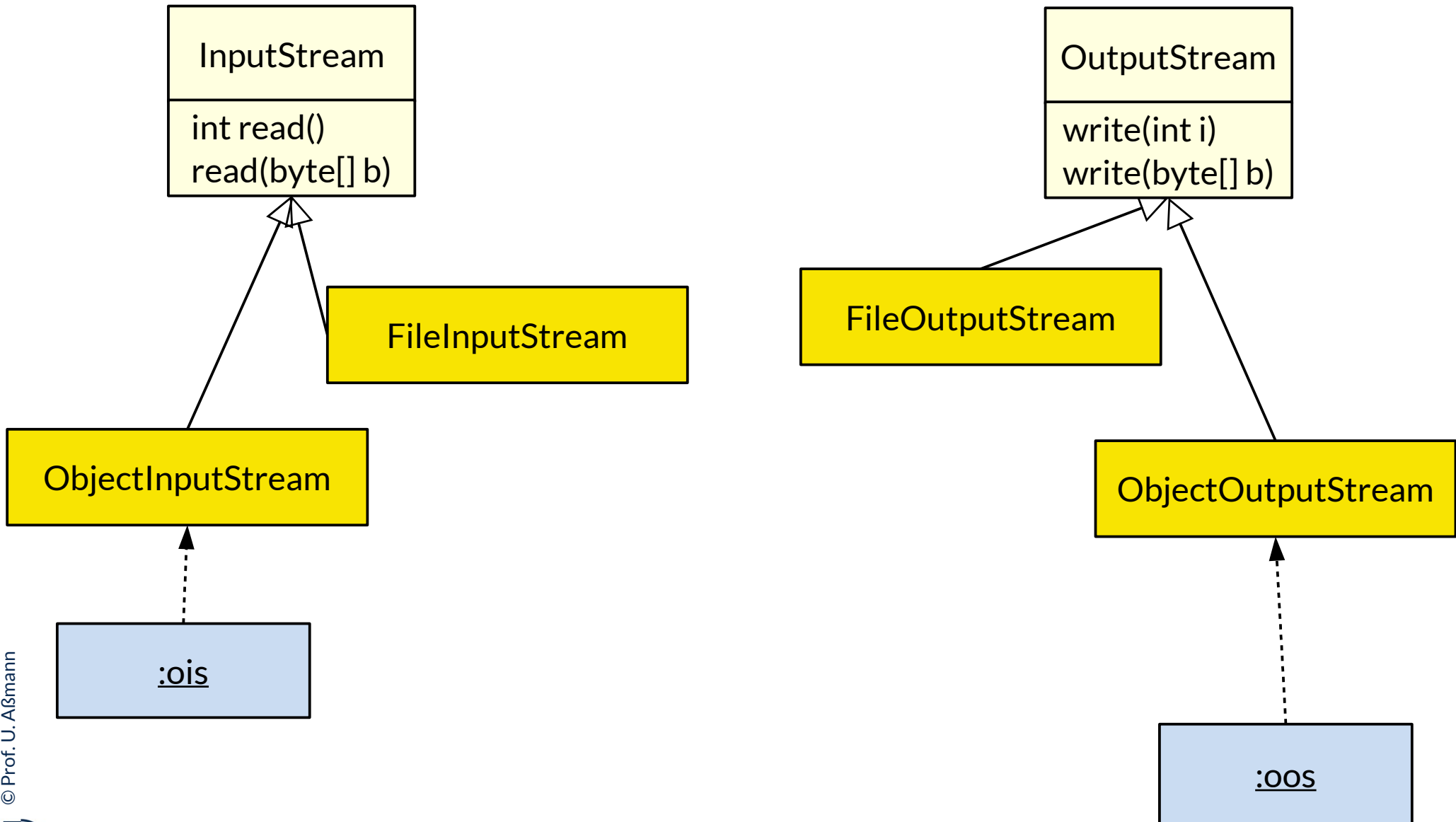
...
XClass xobj;
...
FileOutputStream fos = new FileOutputStream("Xfile.dat");
ObjectOutputStream oos = new ObjectOutputStream(fos);

// internally realized as push for all child objects
oos.writeObject(xobj); // push
...
```

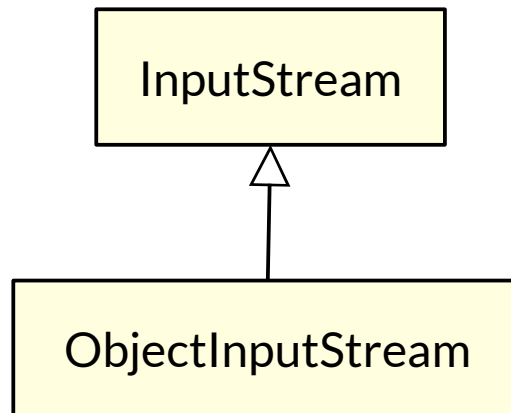
# Input und Output Streams im JDK

43

Softwaretechnologie (ST)



- ▶ **Die Klasse `java.io.ObjectInputStream`** stellt einen Iterator/InputStream in unserem Sinne dar
  - Methoden, um ein Geflecht von Objekten linear darzustellen (zu *serialisieren*) bzw. aus dieser Darstellung zu rekonstruieren (zu *deserialisieren*)
  - Ein OutputStream entspricht dem Entwurfsmuster Sink
  - Ein InputStream entspricht dem Entwurfsmuster Iterator



```
import java.io.*;

class XClass implements Serializable {
    private int x;
    public XClass (int x) {
        this.x = x;
    }
}

...
XClass xobj;
...
FileInputStream fis = new FileInputStream("Xfile.dat");
ObjectInputStream ois = new ObjectInputStream(fis);
// internally realised as pull
xobj = (XClass) ois.readObject(); // pull
```

## 23.4.3 Ereignisse und Kanäle

- ▶ Kanäle (gerichtete Konnektoren) eignen sich hervorragend zur Kommunikation mit der Außenwelt, da sie die Außenwelt und die Innenwelt eines Softwaresystems **entkoppeln**
- ▶ Ereignisse können in der Außenwelt **asynchron** (“losgelöst vom System”) stattfinden und auf einem Kanal in die Anwendung transportiert werden
  - Dann ist der Typ der Daten ein Ereignis-Objekt
  - In Java wird ein externes oder internes Ereignis immer durch ein Objekt repräsentiert

# The End

- ▶ Was unterscheidet einen Iterator von einer Senke?
- ▶ Warum läuft Ein- und Ausgabe in Programmen immer über Ströme?
- ▶ Wie heißen die Java-Stromobjekte für Ein- und Ausgabe?
- ▶ Was unterscheidet ein Team von einer Kollaboration?
- ▶ Warum spielt ein Objekt eines Teams in einer Kollaboration eine Rolle?
- ▶ Einige Folien stammen aus den Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. Used by permission.



# Anhang

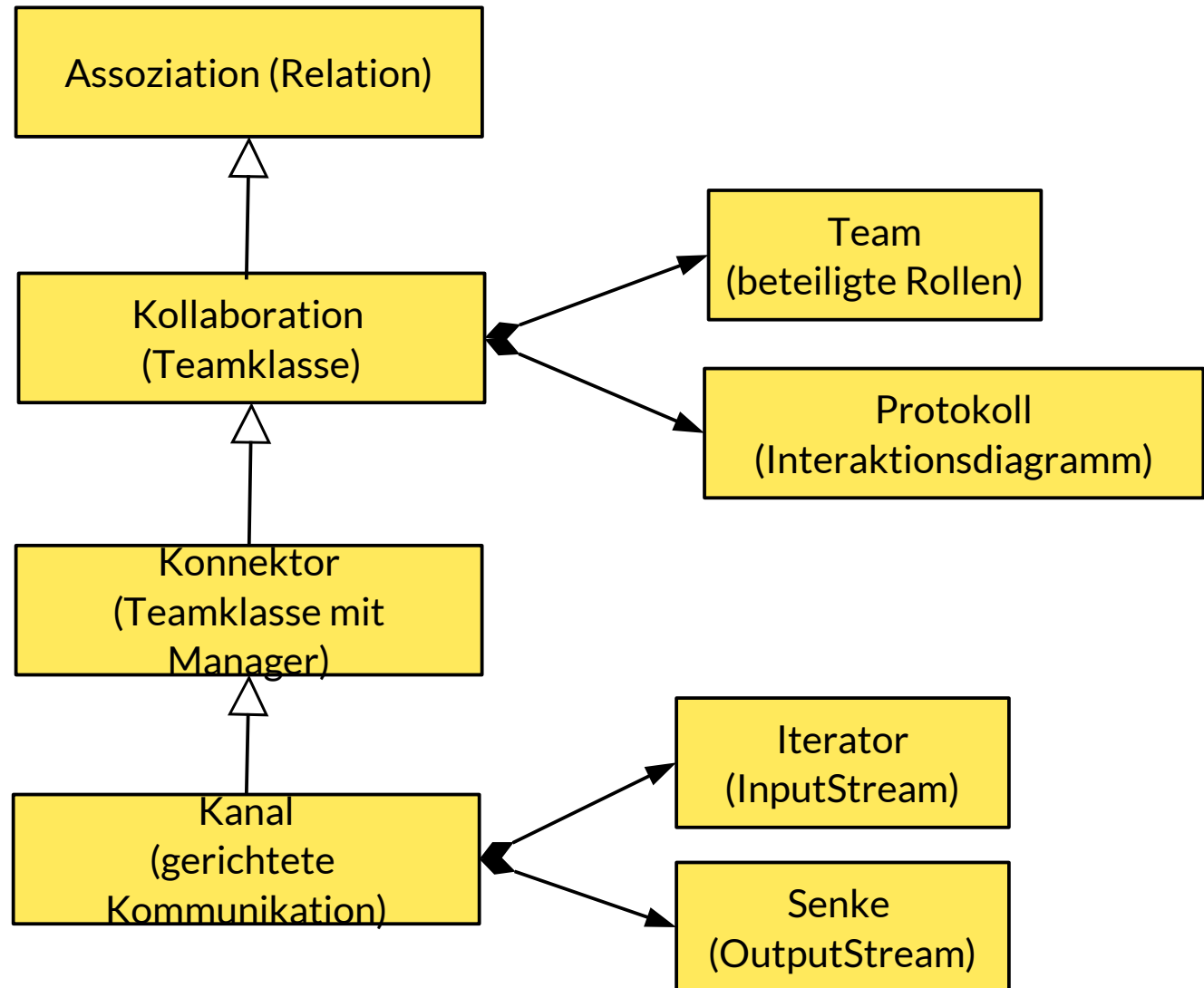




# Relationale Klassen (Konnektoren)

49

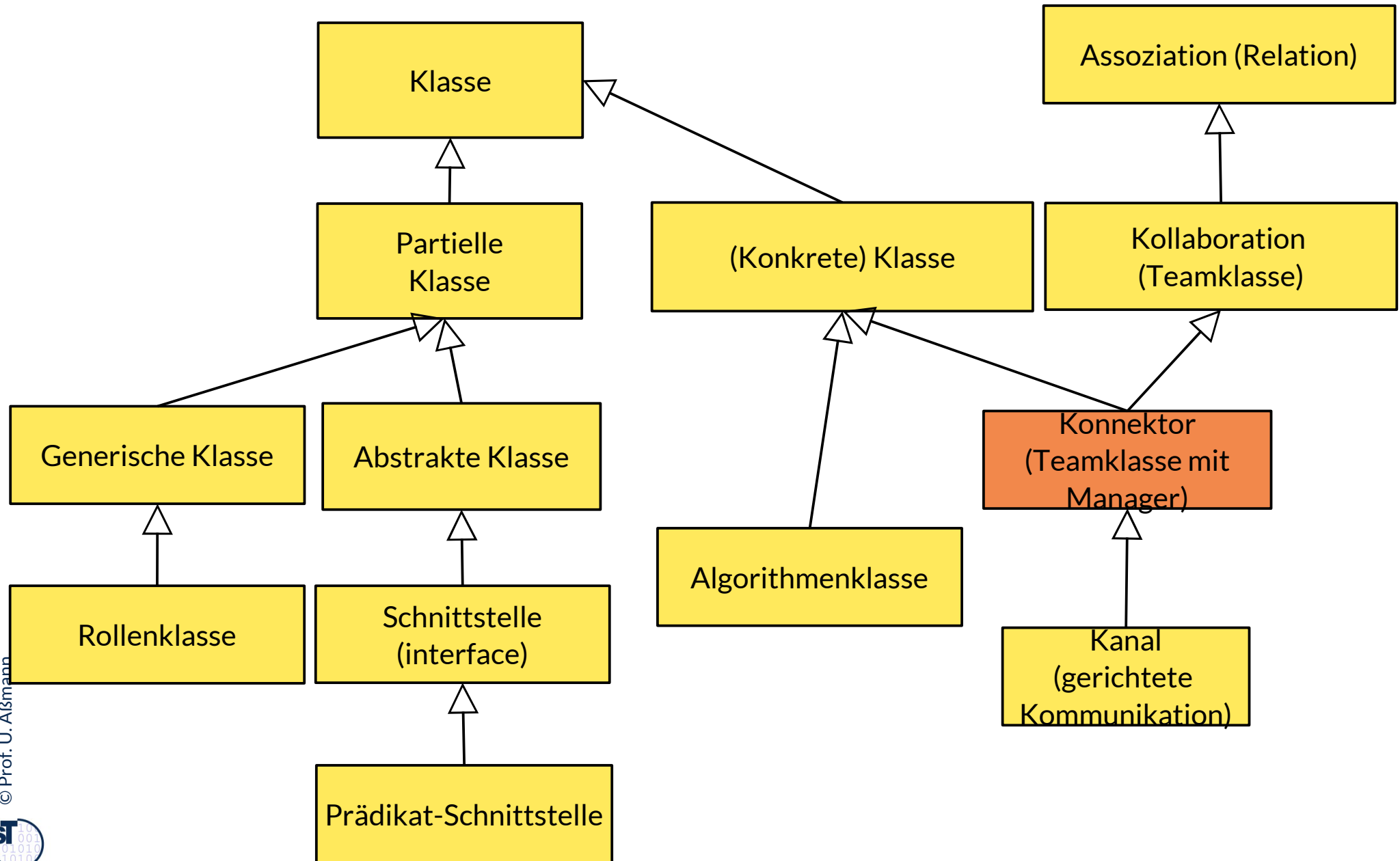
Softwaretechnologie (ST)



## Q2: Begriffshierarchie von Klassen (Erweiterung)

50

Softwaretechnologie (ST)



# Iterator-Implementierungsmuster in modernen Sprachen

51

Softwaretechnologie (ST)

- ▶ In vielen Programmiersprachen (Sather, Scala, Ada) stehen **Iteratormethoden (stream methods)** als spezielle Prozeduren zur Verfügung, die die Unterobjekte eines Objekts liefern können
  - Die **yield**-Anweisung gibt aus der Prozedur die Elemente zurück
  - Iterator-Prozedur kann mehrfach aufgerufen werden und damit als input-stream verwendet werden
  - Beim letzten Mal liefert sie null

```
class bigObject {
  private List subObjects;
  public iterator Object deliverThem() {
    while (i in subObjects) {
      yield i;
      // Dieser Punkt im Ablauf wird sich als Zustand gemerkt
      // Beim nächsten Aufruf wird hier fortgesetzt
    }
  }
}

.. BigObject bo = new BigObject(); ...
.. a = bo.deliverThem();
.. b = bo.deliverThem();..
```

# Kanäle sind technische Kollaborationen

52

Softwaretechnologie (ST)

- ▶ Ein **Konnektor** bildet eine standardisierte technische Kollaboration.
  - Sie beschreiben die standardisierte Interaktion, Nebenläufigkeit und Kommunikation eines Teams von Beteiligten
  - Konnektoren sind besser wiederverwendbar als allgemeine Kollaborationen
- ▶ Bsp: Ein Kanal ist ein einfacher Konnektor

