

# 23. Entwurfsmuster für Kommunikation in Teams – Programmierung von fixen Netzen für die Programmierung der I/O und des Internet Kommunikation mit Iteratoren, Senken, Kanälen und Konnektoren Datenhaltung mit potentiell unendlichen Collections

Prof. Dr. Uwe Aßmann  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
Technische Universität Dresden  
Version 20-0.3, 6/25/20

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



# Betreff: "Softwaretechnologie für Einsteiger" 2. Auflage

- ▶ zur Info: o.g. Titel steht zur Verfügung:
  - 50 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!

Frederic Weatherly (1910)

Text aus Wikipedia/McCourt, Danny Boy, S.  
87 f.

Johnny Cash's classic implementation

<https://www.youtube.com/watch?v=egf4X1UzsAA>

see also seine CD "American IV"



**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.**

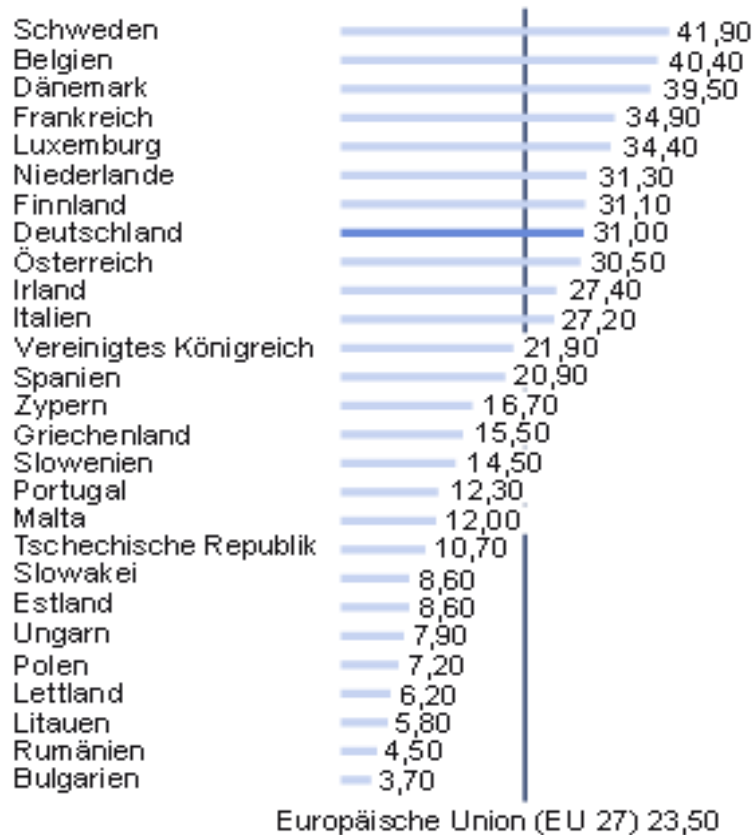
And I shall hear, though soft you tread above me  
And all my grave will warmer, sweeter be  
For you will bend and tell me that you love me,  
And I shall sleep in peace until you come to me.

# Warum müssen Softwareingenieure fortgeschrittenes Wissen besitzen?

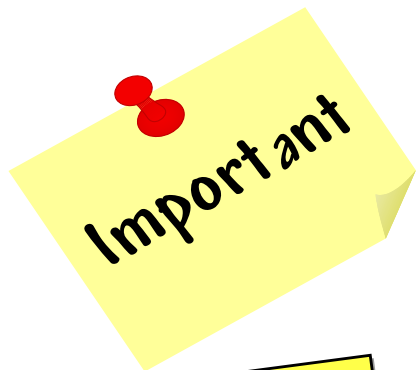
- ▶ Die Konkurrenz ist hart: Zu den Kosten der Arbeit:

## Arbeitskosten in der Privatwirtschaft 2012

je geleistete Stunde in EUR



© Statistisches Bundesamt, Wiesbaden 2013

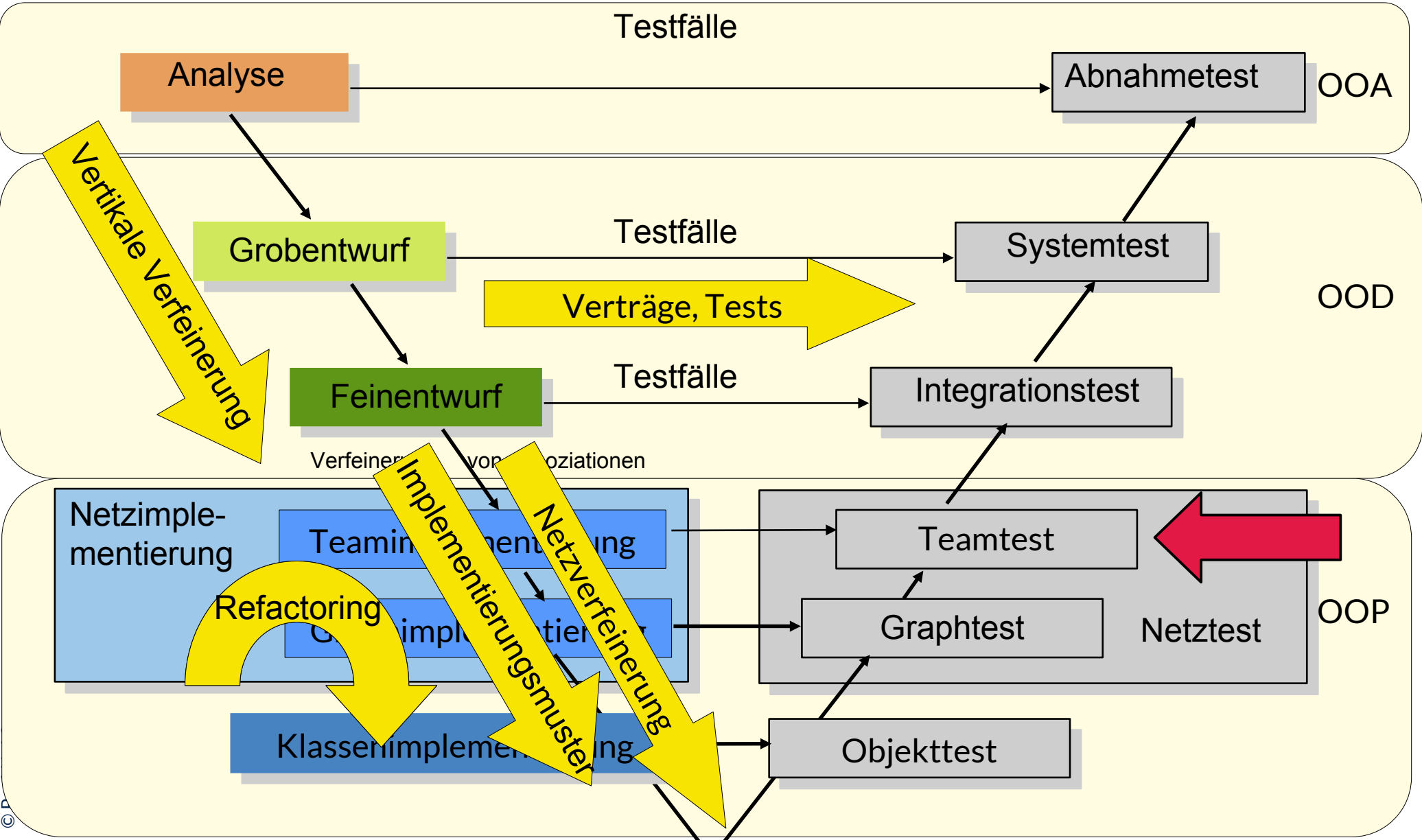


<http://www.heise.de/resale/imgs/17/1/0/0/1/3/4/1/>

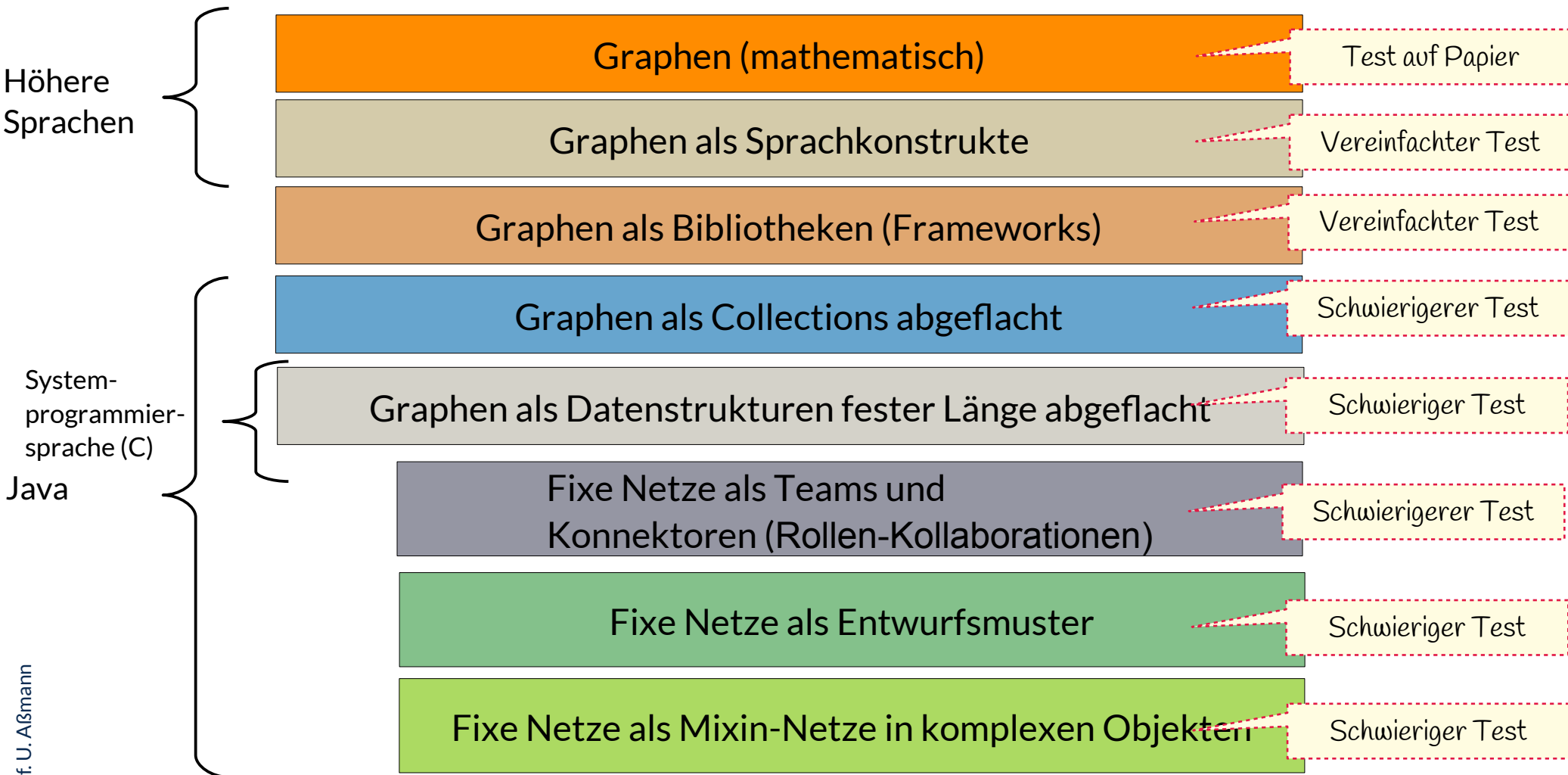
ArbeitskostenEULAenderStart2012-9bb2e8b041f1342e.png

# Q4: Softwareentwicklung im V-Modell

[Boehm 1979]



# Repräsentation von flexiblen und fixen Objektnetzen als Datenstrukturen (Netzverfeinerung)

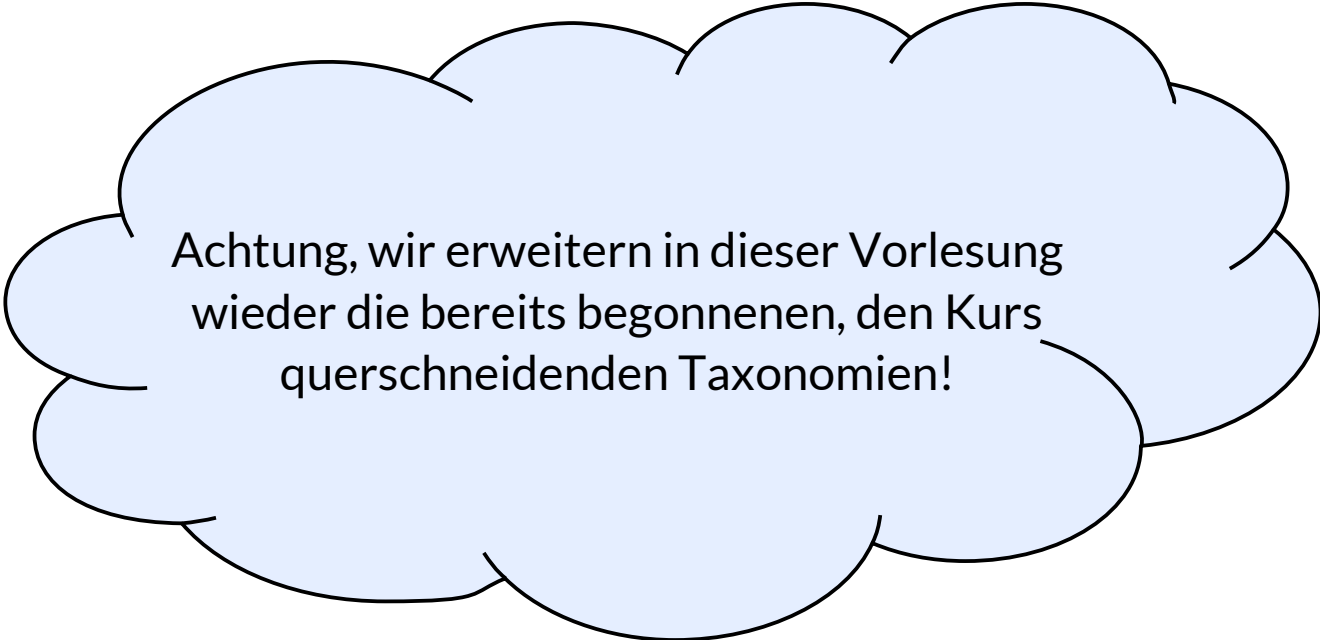


# E.23.1 Lernen mit Begriffshierarchien, die die Vorlesung querschneiden



# Querschneidende Begriffshierarchien

- ▶ 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 für fixe Netze

- ▶ 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



# Teams (fixe Objektnetze)

- ▶ **Problem:**
  - Das Management von Objekt-Netzen ist eine der schwierigsten und fehleranfälligen Aufgaben im objektorientierten Programmieren.
  - Objektnetz-Programmierung ist sehr wichtig für Softwarequalität
- ▶ Neue Sprachen:
  - Java gehört zur 1. Generation von objektorientierten Programmiersprachen.
  - Die 2. Generation verbessert die Beschreibung von Teams, Netzen von Objekten:
    - Object Teams [Www.objectteams.org](http://www.objectteams.org)
    - SCROLL <https://github.com/max-leuthaeuser/SCROLL>
- ▶ Wir lernen in diesem Kapitel, wie man mit Bibliotheken und Entwurfsmustern das Programmieren von Objektnetzen auch in Java verbessern kann!

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

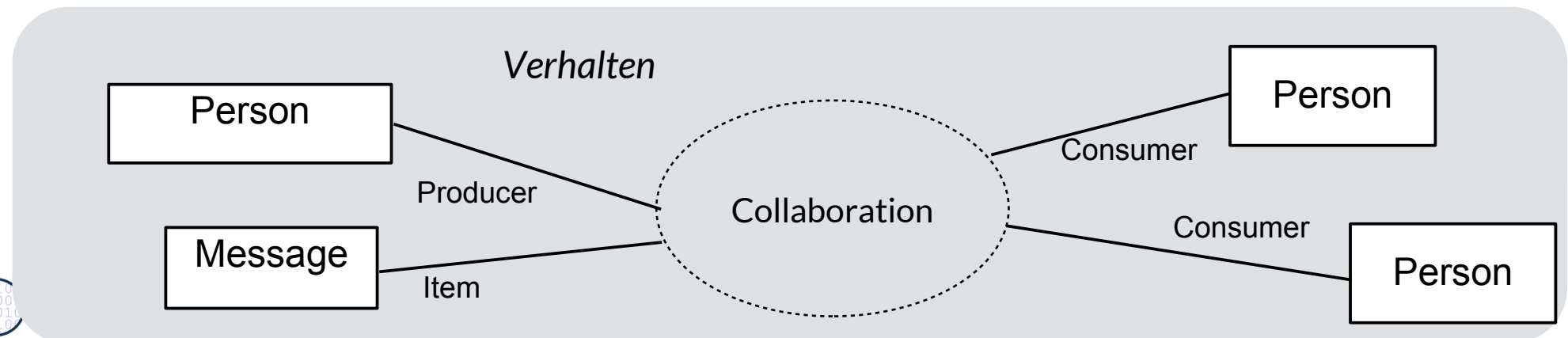
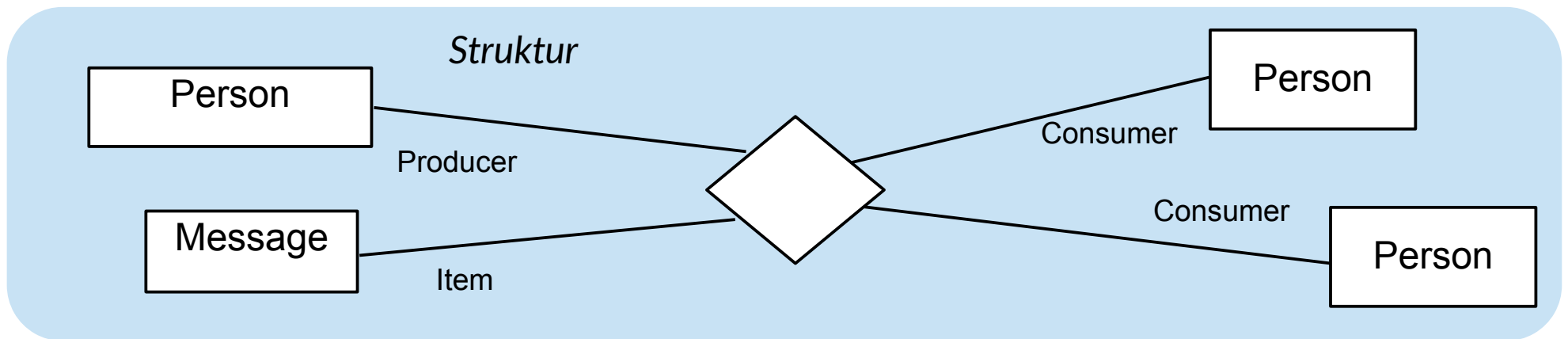
Die Interaktion eines Teams wird in UML durch eine **Kollaboration** beschrieben.

- ▶ Historik: Der Begriff des Teams wurde von der TU Berlin im Projekt [www.objectteams.org](http://www.objectteams.org) geprägt.

Was ist der Unterschied zwischen einem Team und einer Kollaboration?

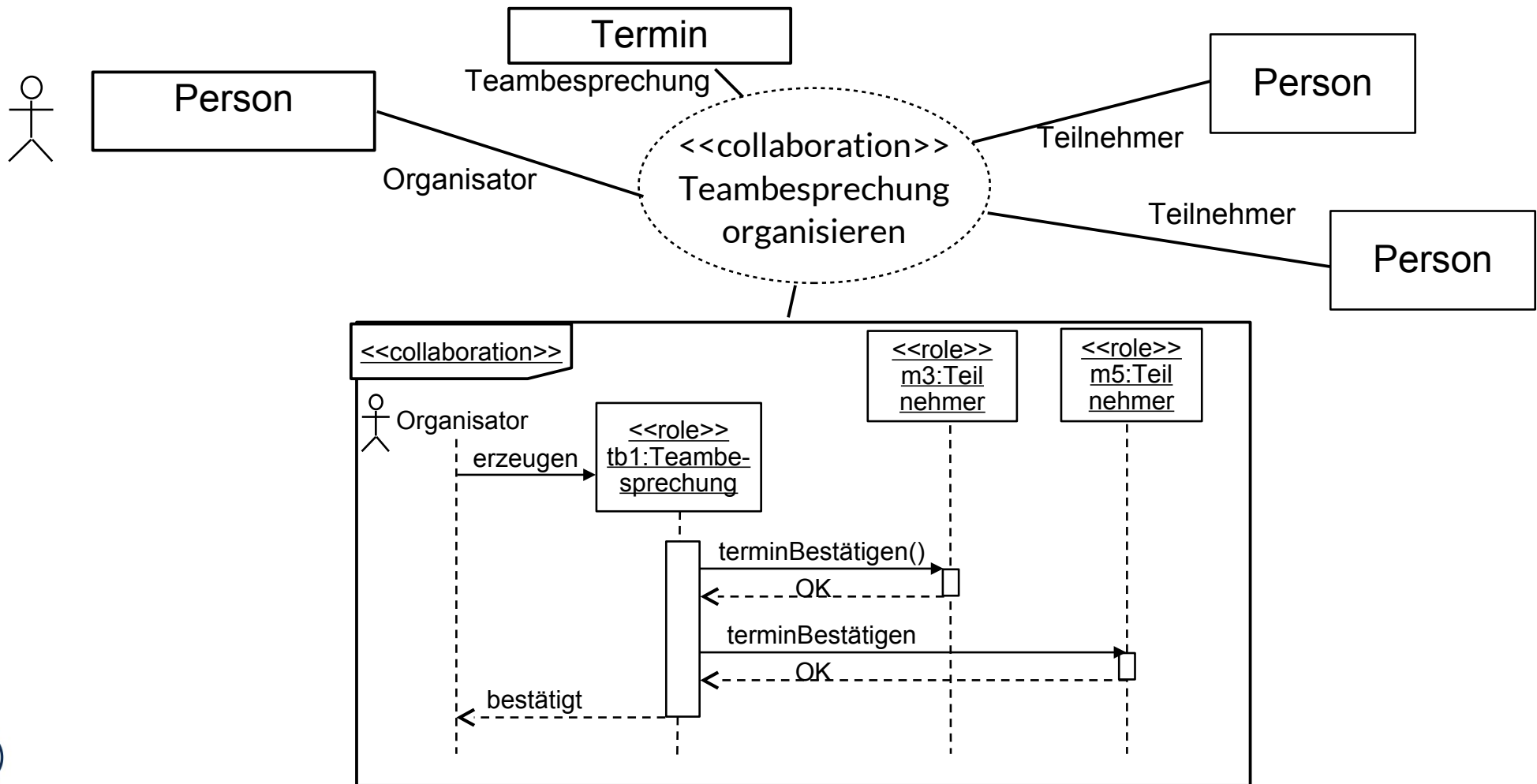
# Kollaborationen kapseln das Verhalten von fixen Netzen (Teams)

- ▶ Die *Struktur* von Teams (fixes Netz mit n Mitgliedern) wird in UML durch n-stellige Assoziationen dargestellt
- ▶ Das *Verhalten* eines fixen Netzes und seine Kommunikation durch Kollaborationen
- ▶ Def.: Eine **Kollaboration (collaboration, Kollaborationsklasse)** realisiert die Kommunikation eines Teams mit einem festen anwendungsspezifischen Protokoll



# Kollaborationen kapseln Verhalten eines Teams durch Interaktionsprotokolle

- ▶ Eine **Kollaboration** beschreibt die anwendungsspezifische Interaktion, Nebenläufigkeit und Kommunikation eines Teams, d.h. einer fixen Menge von beteiligten Objekten.
- ▶ Die Kollaboration beschreibt also ein Szenario querschneidend durch die Lebenszyklen mehrerer Objekte



# Teams und technische Konnektoren

**Def.:** Ein **Teamobjekt** ist ein Objekt, das die Kommunikation einer Gruppe (feste Anzahl) von interagierenden und kommunizierenden Objekten kapselt. Ist ein Teamobjekt hauptsächlich mit dem technischen Austausch von Daten zwischen den Objekten beschäftigt, heißt es **Konnektor**.

Bsp: Teams: Dynamo Dresden Team, Staatskapelle

Konnektoren: Aldi--Peter Müller:Käufer, Finanzamt--Jenny Klein:Steuerzahler

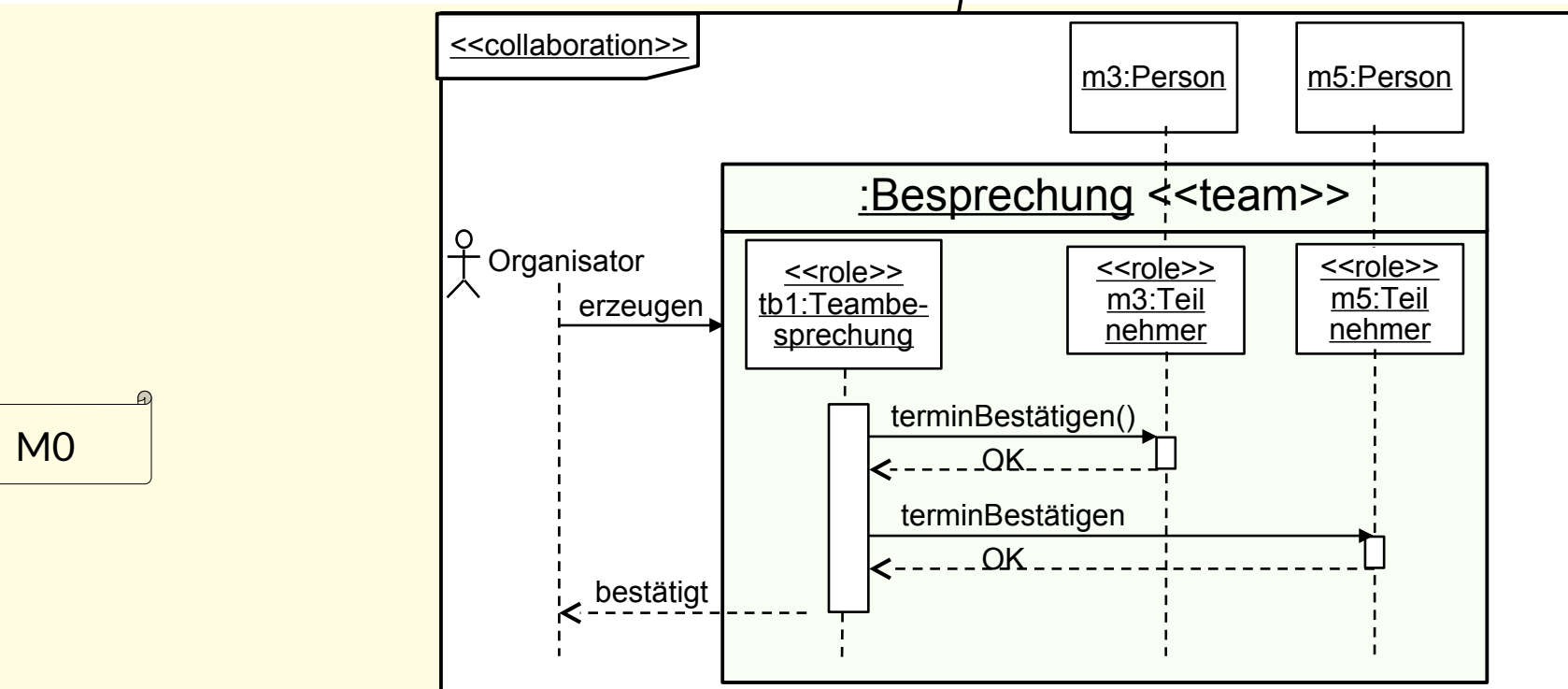
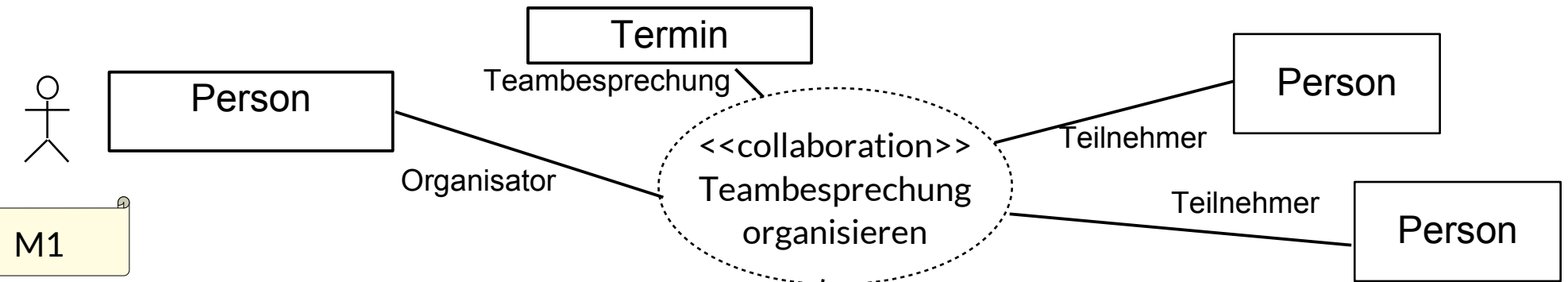
**Def.:** Kann eine Kollaboration durch eine Klasse gekapselt werden, spricht man von einer **Teamklasse**.  
Spezialfall beim Datenaustausch: **Konnektorklasse**, kurz **Konnektor**.

Bsp: Teamklassen: Fußballmannschaft, Kapelle

Konnektorklassen: Produzent-Konsument, Client-Server

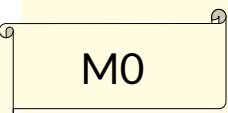
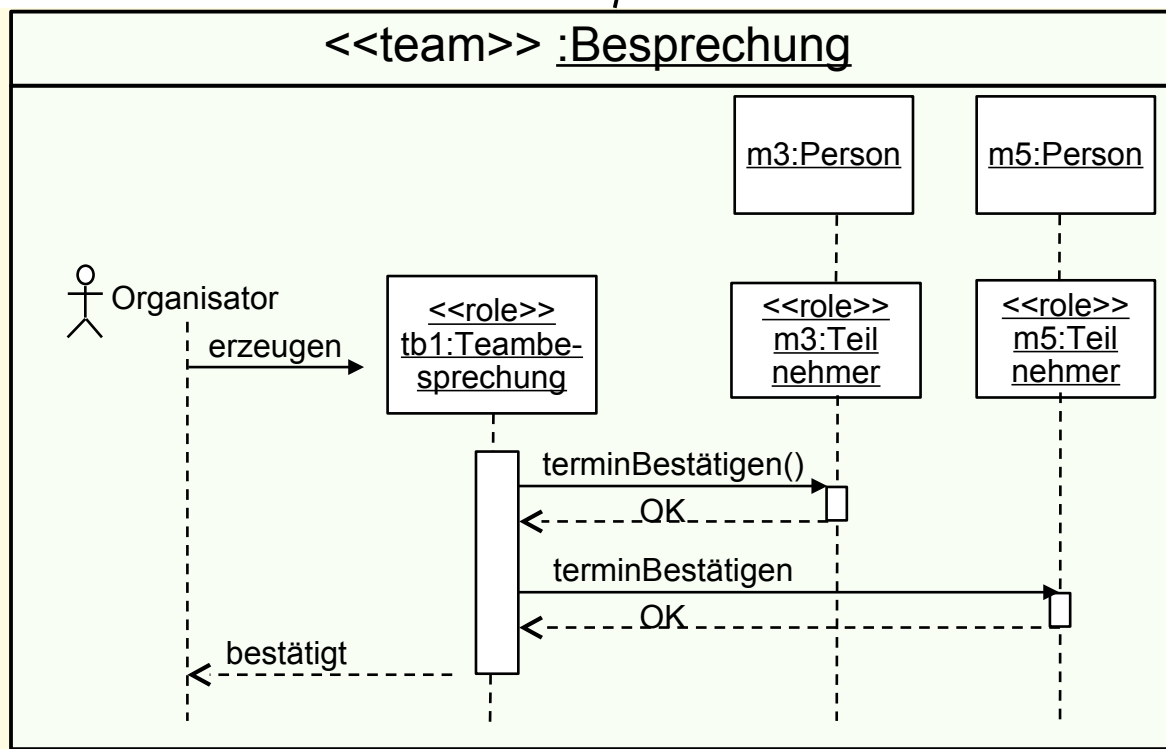
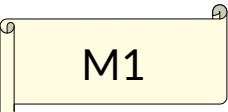
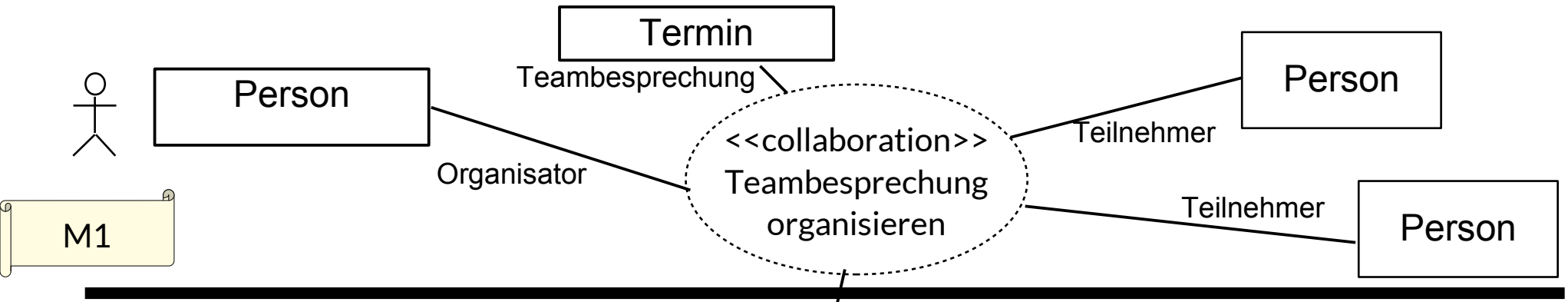
# Teamklassen kapseln Verhalten durch Interaktionsprotokolle

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



# Kurznotation

- Wir schreiben die Kollaboration in den Konnektor



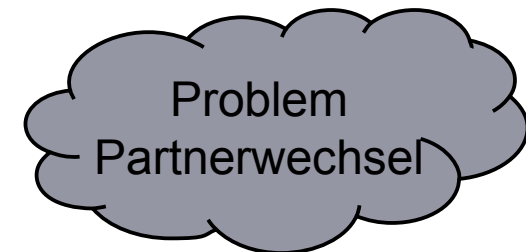


Beobachtung:  
Viele Entwurfsmuster beschreiben  
Schemata für **Teams** und ihre **Kollaborationen**.

## 23.2 Aktoren und Känale (Actors and Channels) für lose gekoppelte Teams

### Wie man Konnektoren für die Programmierung des Internets nutzt

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



**Def.:** Ein **Aktor (actor)** ist ein aktives Objekt mit einem eigenen Lebensfaden (thread).

Ein Aktor besitzt Kanäle, über die er Botschaften empfängt und sendet

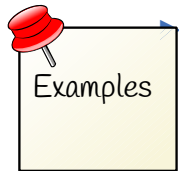
BeEin Webserver ist ein Aktor und besitzt *ports* als Kanäle (http-Port 80)

- Liste der standardisierten Ports

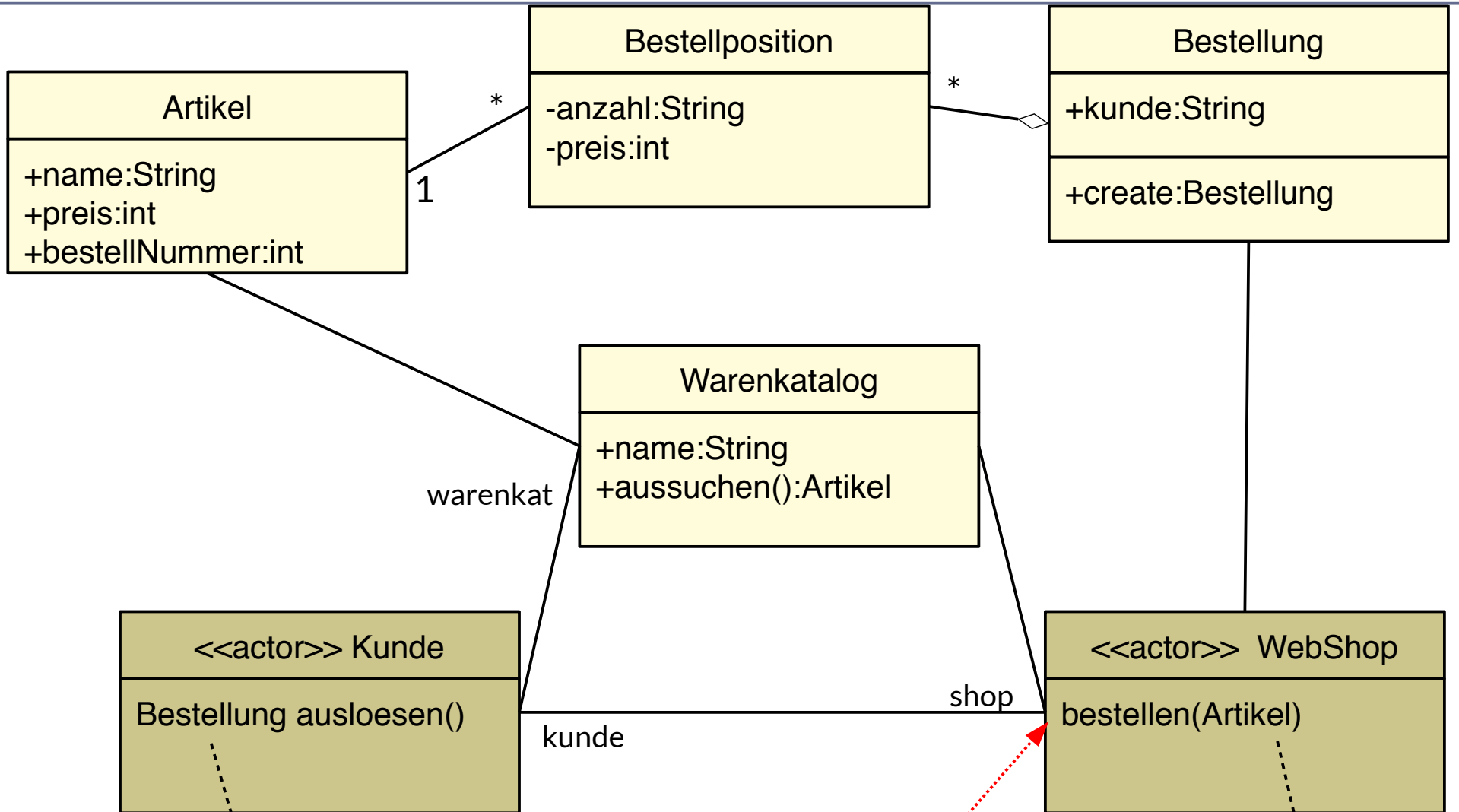
[https://de.wikipedia.org/wiki/Liste\\_der\\_standardisierten\\_Ports](https://de.wikipedia.org/wiki/Liste_der_standardisierten_Ports)

▶ Unter LINUX heißt der Aktor **Prozess** und besitzt die Kanäle

- `stdin` (Standardeingabe)
- `stdout` (Standardausgabe)
- `stderr` (Standardfehlerausgabe)



# Beispiel: Bestellung auf einem Webshop



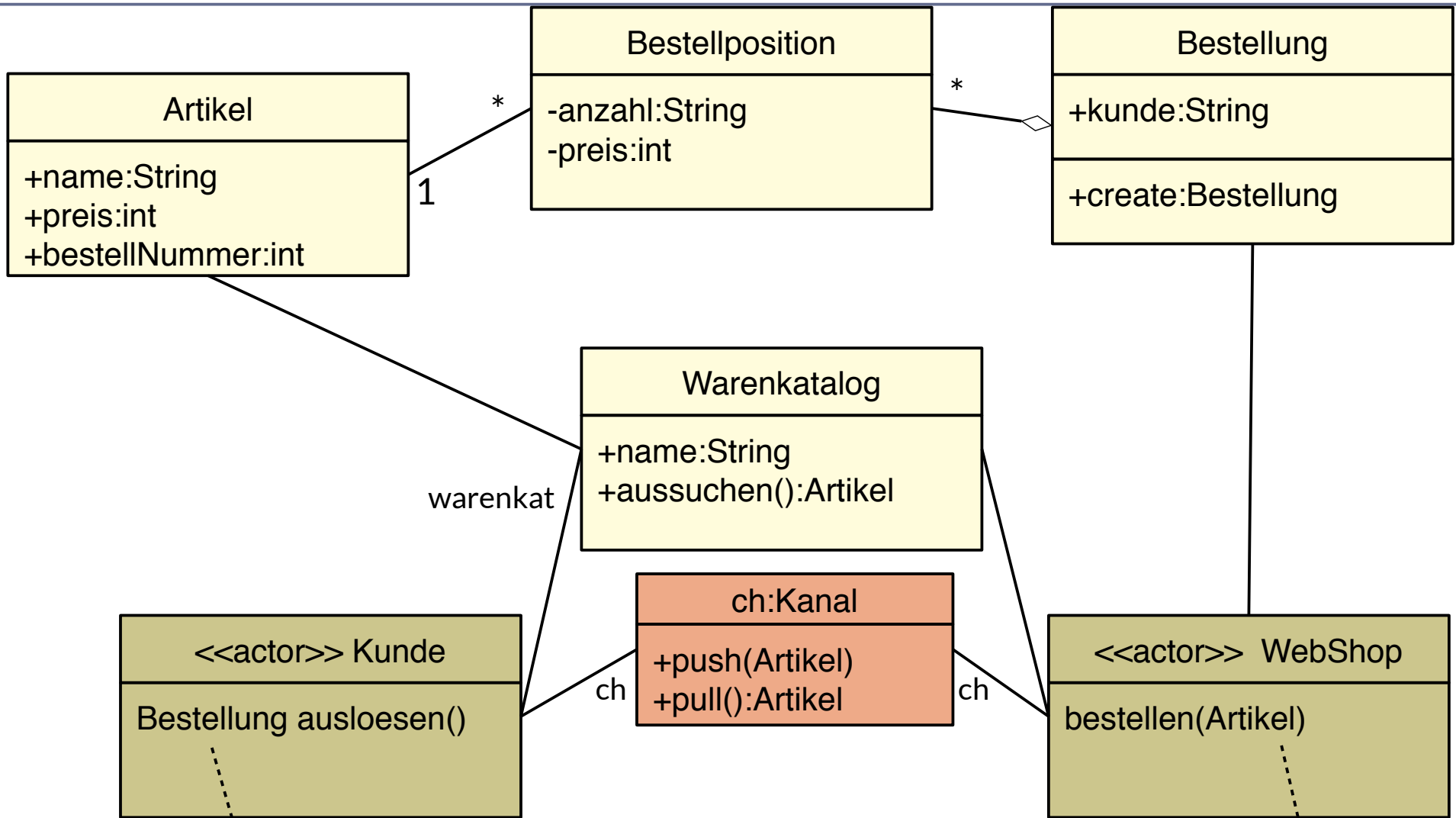
```
while (hatBedarf())
{ artikel = warenkat.aussuchen();
shop.bestellen(artikel);
}
```

Enge Kopplung!

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



# Beispiel: Bestellung auf einem Webshop; nun lose gekoppelt



```

while (hatBedarf())
{
    artikel = warenkat.aussuchen();
    ch.push(artikel);
}
    
```

Lose Kopplung!

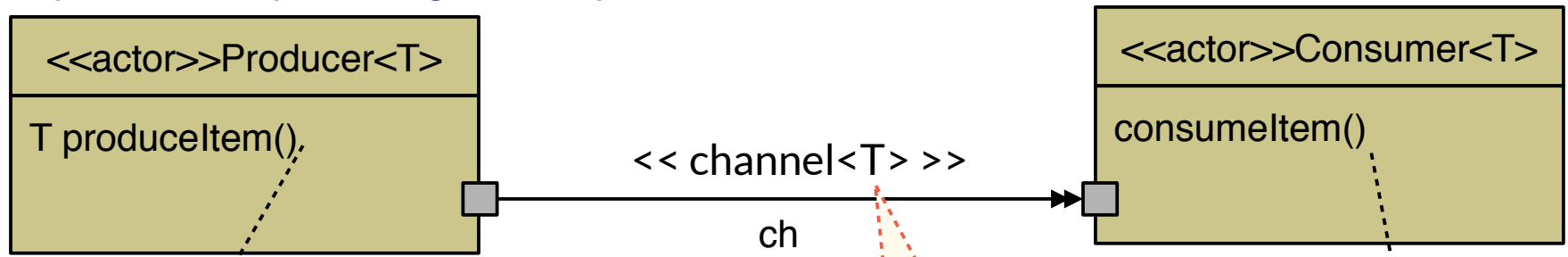
```

while (true)
{
    b = ch.pull();
    b.workOn();
    b.ship();
}
    
```

# Kanäle bilden Netze mit Datenfluss

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

- ▶ UML Notation: Andocken eines Kanals an sockets (*pins, 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))



```
while ()  
{  
  ch.push(produceItem());  
}
```

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

Kanäle bilden oft generische Klassen

Lose Kopplung!

# Anwendung von Kanälen: lose-gekoppelte Kommunikation, auch im Internet

Problem  
Partnerwechsel

23 Softwaretechnologie (ST)

- ▶ Webshops dürfen die konkreten Objekte ihrer Kunden nicht kennen
- ▶ **Kanäle erlauben, die Partner zu wechseln**, ohne Kenntnis des Netzes
  - Ideal für fixe 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.
- ▶ Channel sind technische Objekte, zum olympischen Ring “Infrastruktur” gehörend
- ▶ Darstellung von potentiell unendlichen Collections



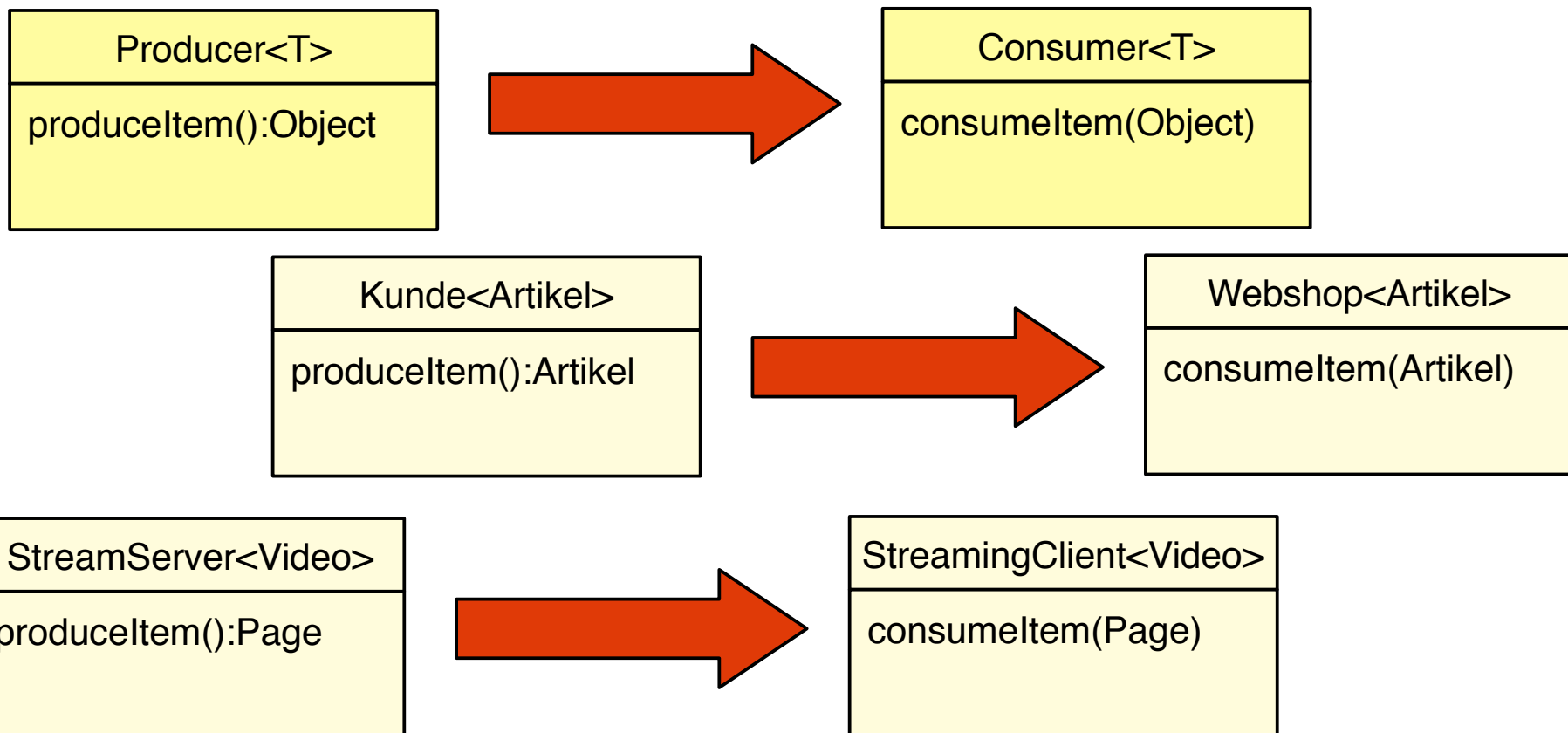


# Wie organisiere ich die “unendlich lange” Kommunikation zweier Aktoren?

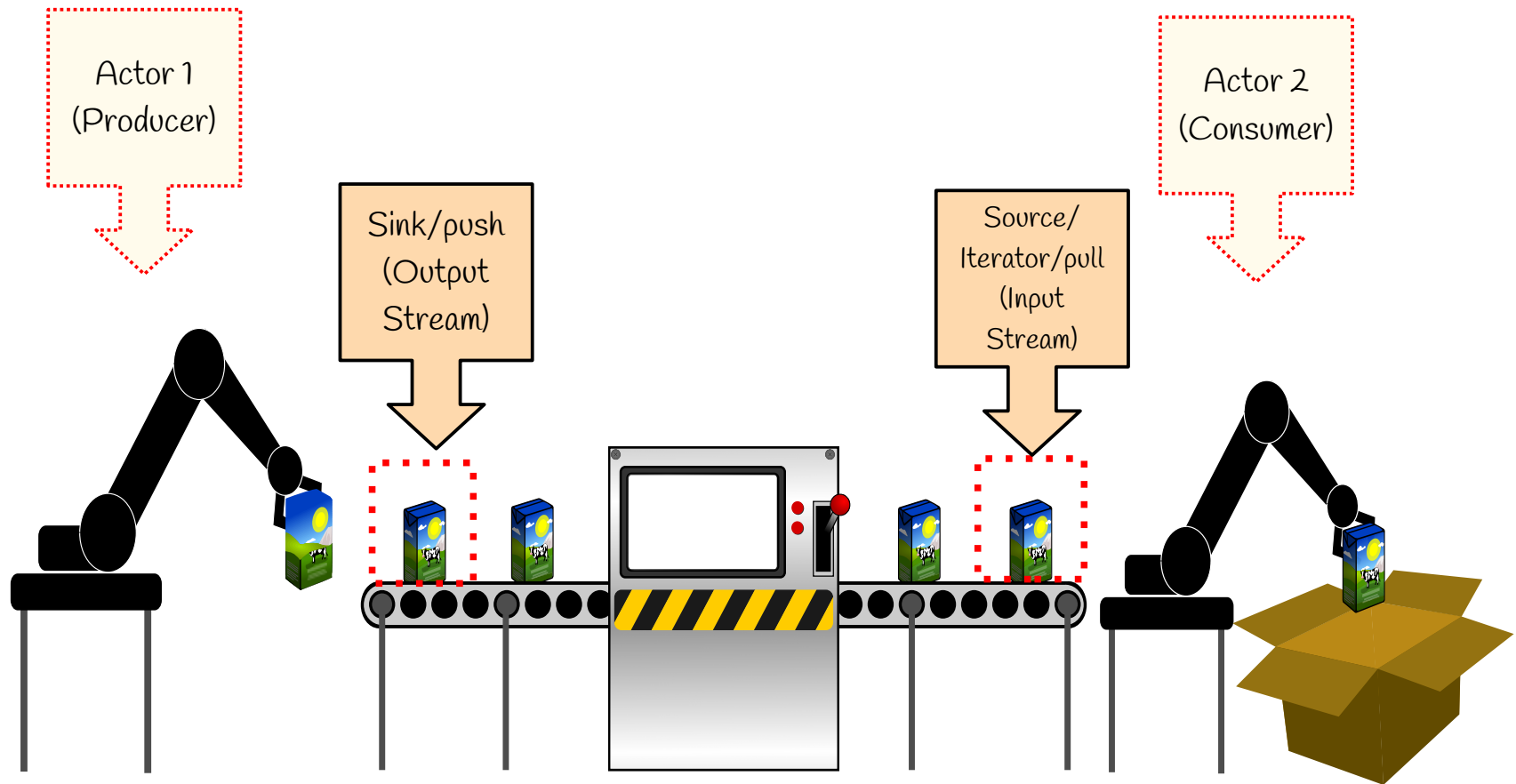
25

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, unbestimmte Collections?
- ▶ **Antwort:** mit Kanälen.

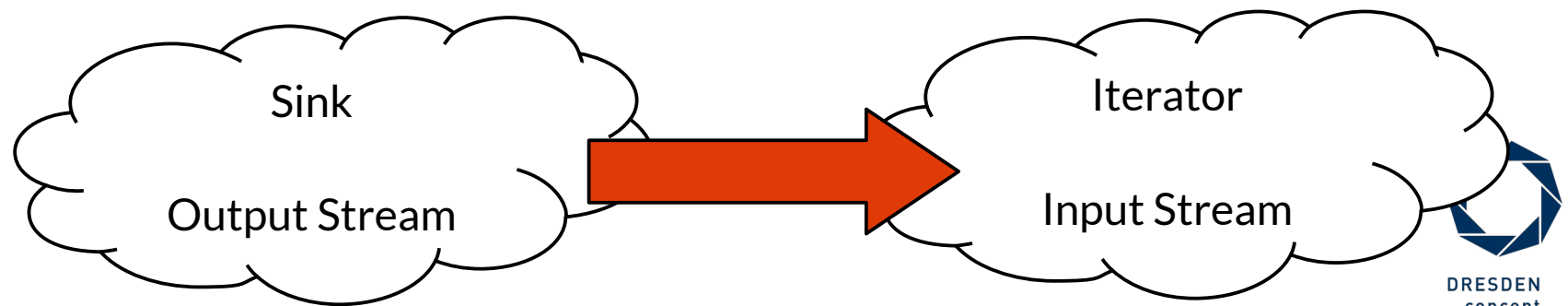


# Actors and Channels with Sinks and Iterators



## 23.3.1 Entwurfsmuster Iterator (Eingabestrom, input stream)

Kanäle bestehen aus mit einander verbundenen Enden, mindestens zweien

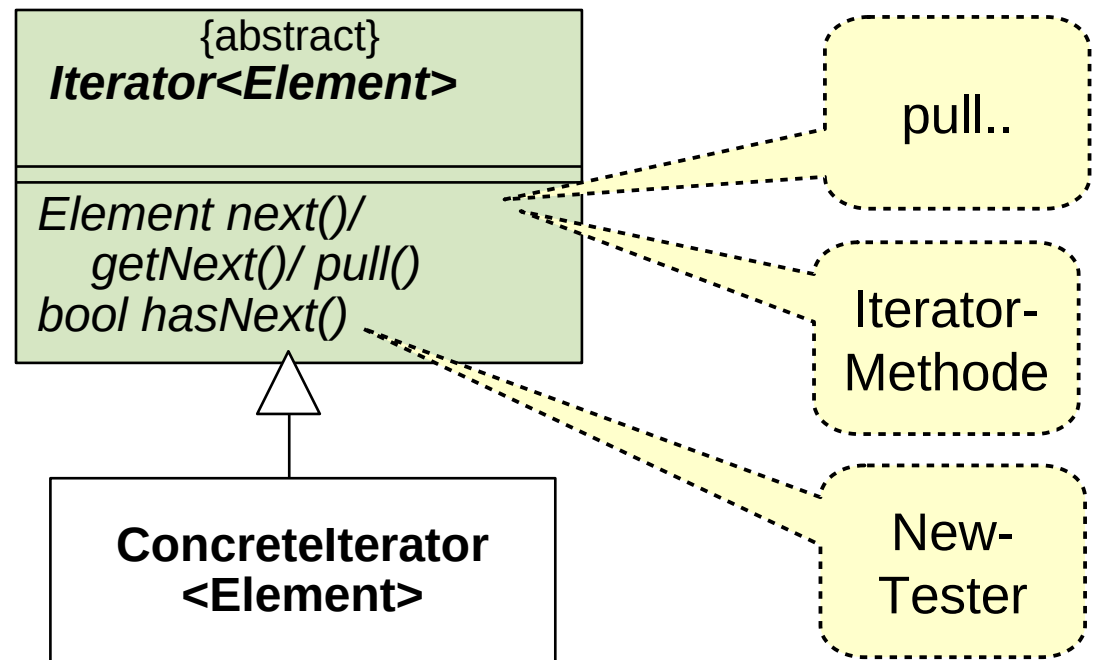


# Entwurfsmuster Iterator (Input Stream) (Implementierungsmuster)

28

Softwaretechnologie (ST)

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

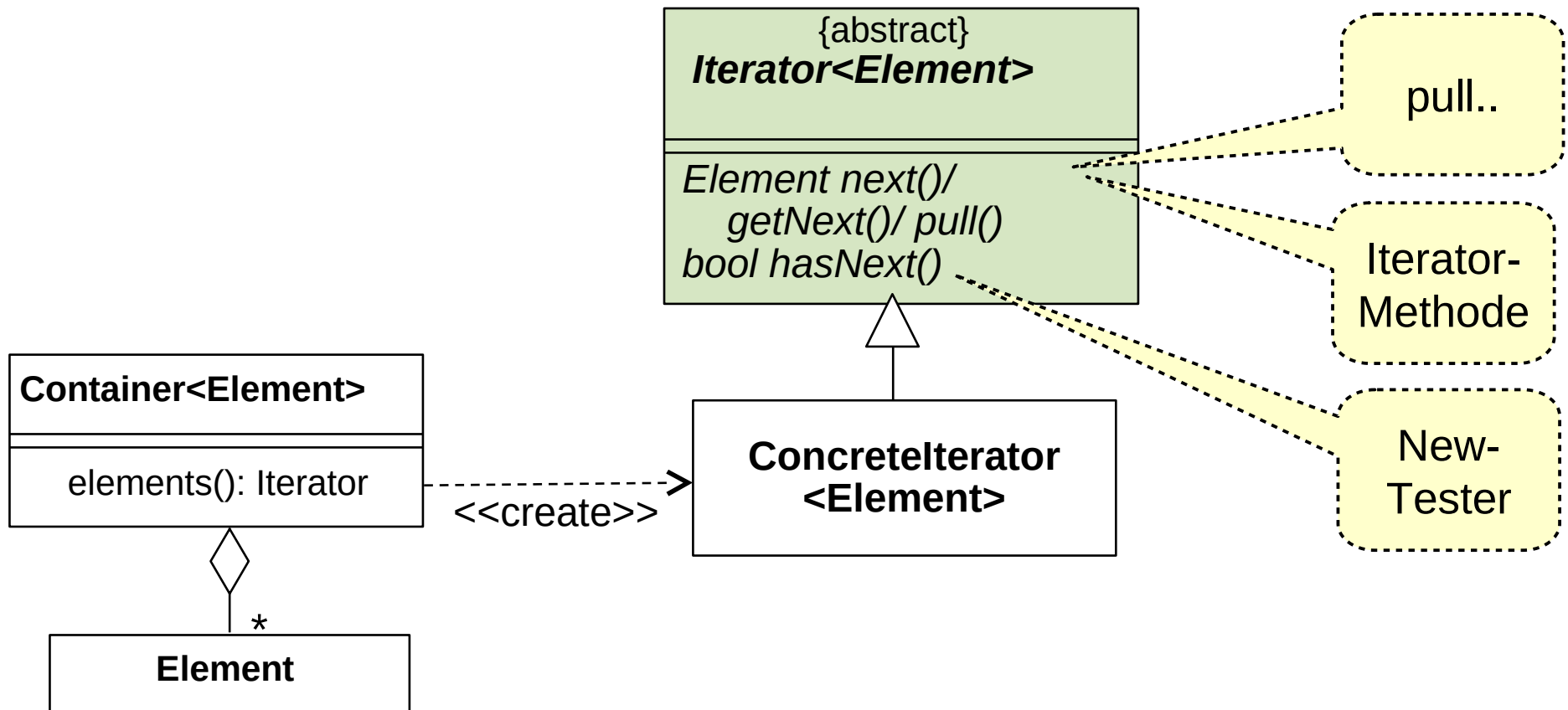


# Entwurfsmuster Iterator (Input Stream) (Implementierungsmuster)

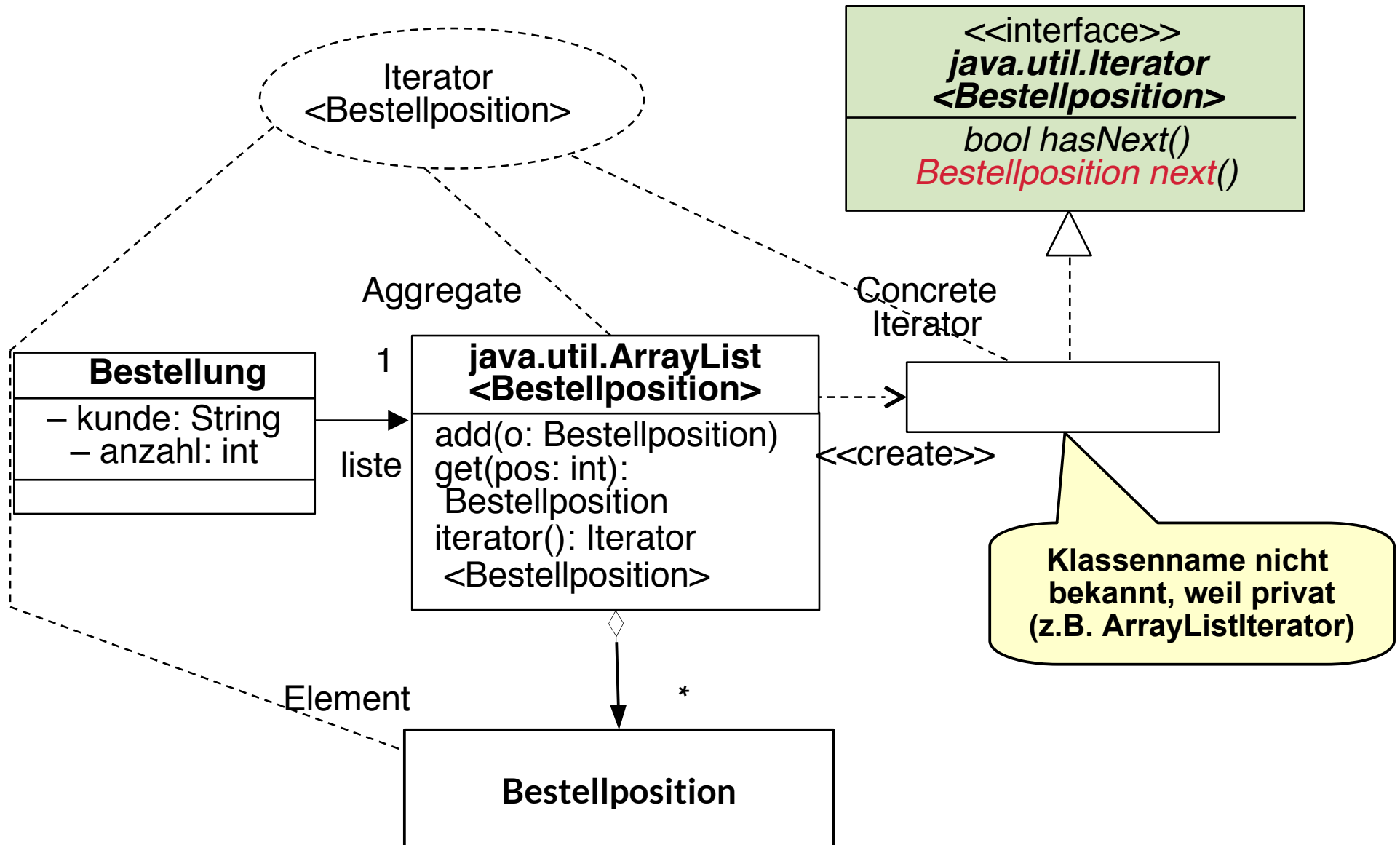
29

Softwaretechnologie (ST)

- ▶ Containerklassen haben Iteratoren



# Iterator-Beispiel in der JDK (ArrayList)



# Implementierungsmuster Iterator

► Verwendungsbeispiel:

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

► Einsatzzwecke:

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



beachte

# Anwendungsbeispiel mit Iteratoren

32

Softwaretechnologie (ST)

```
import java.util.Iterator;
...
class Bestellung {

    private String kunde;
    private List<Bestellposition> liste;
    ...
    public int auftragssumme() {
        Iterator<Bestellposition> i = liste.iterator();
        Bestellposition b;

        int s = 0;
        while (i.hasNext()) { // Test whether next element
                               // (on the source) is available
            b = i.next();      // pull the next element
            s += b.positionspreis();
        }
        return s;
    }
    ...
}
```

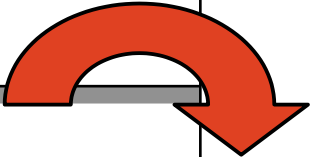
Online:  
Bestellung2.java



# For-Schleifen auf Iterable-Prädikatschnittstellen

- ▶ 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;
}
```



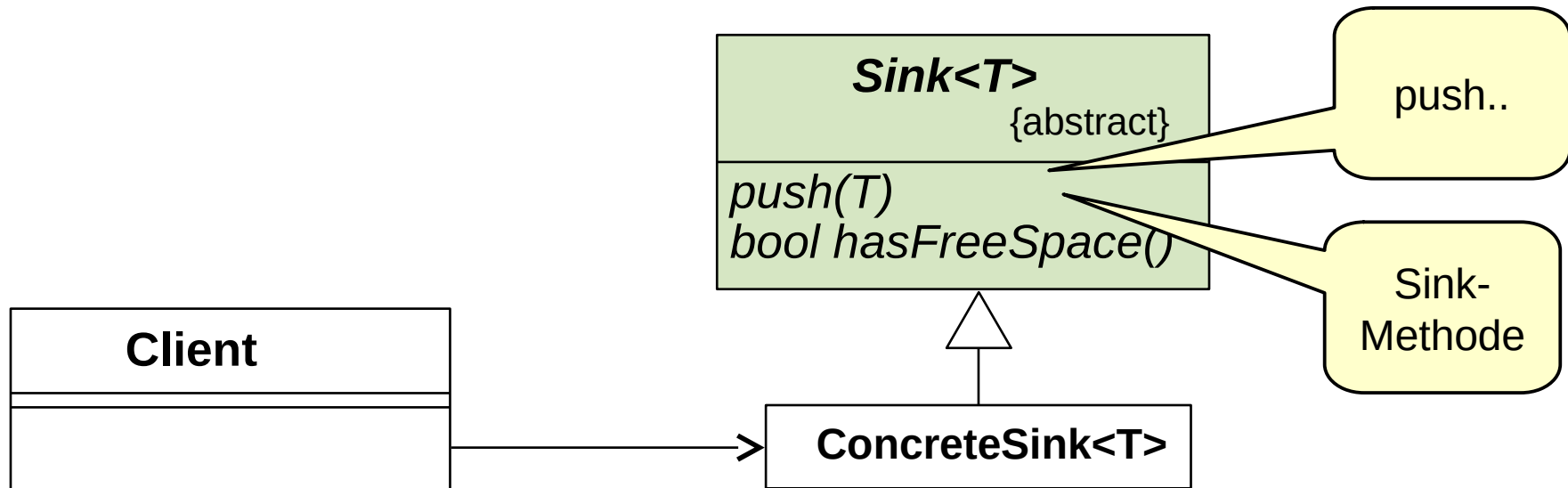
```
class BillItem { int price; }
class Bill {
    int sum = 0;
    private List billItems;
    public void sumUp() {
        for (Iterator i = billItems.iterator();
            i.hasNext(); ) {
            item = i.next();
            sum += item.price;
        }
    }
    return sum;
}
```

## 23.3.2 Senken (Sinks, OutputStream)

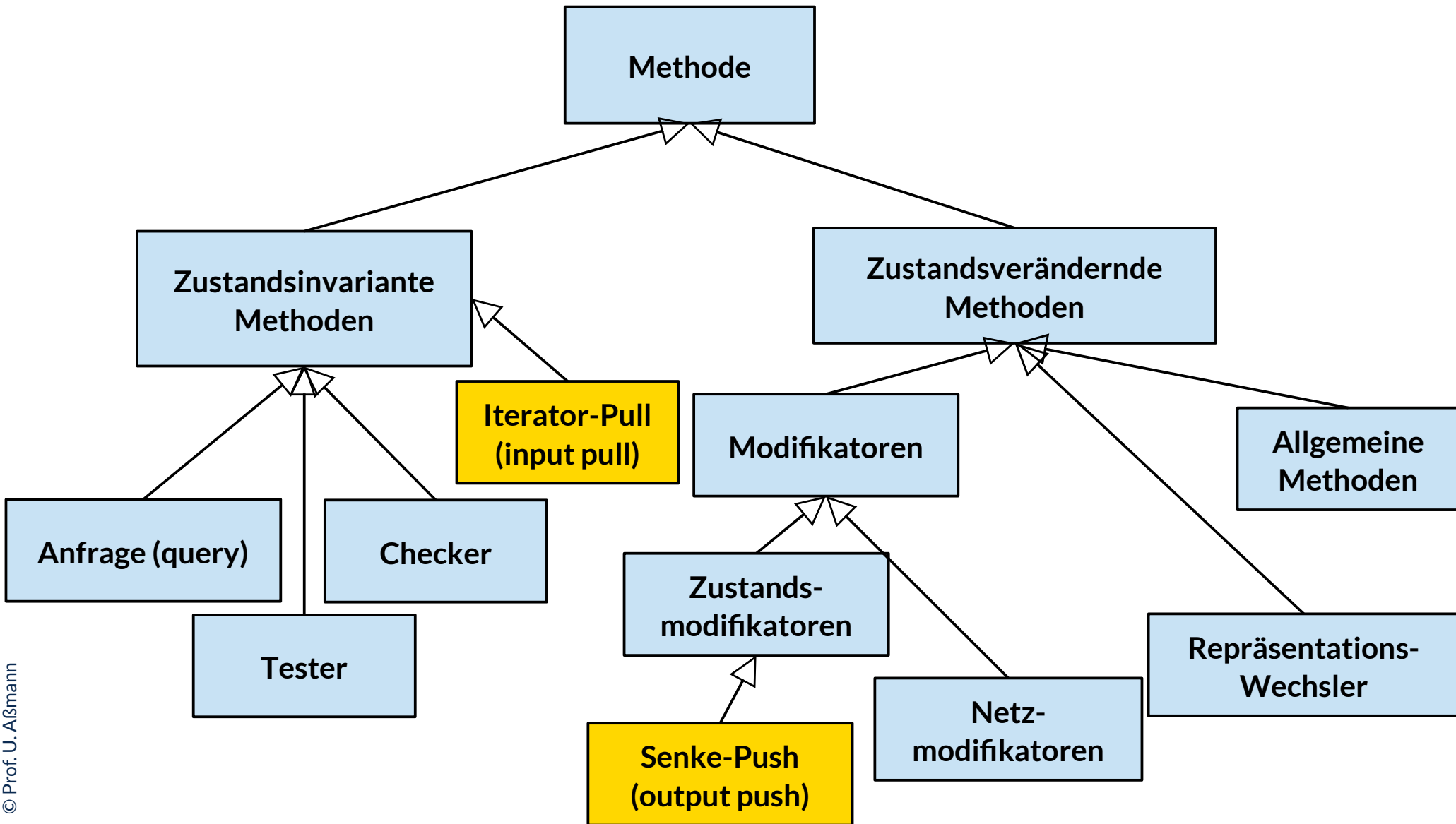


# Entwurfsmuster Senke (Implementierungsmuster)

- ▶ 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



# Q3: Erweiterung: Begriffshierarchie der Methodenarten



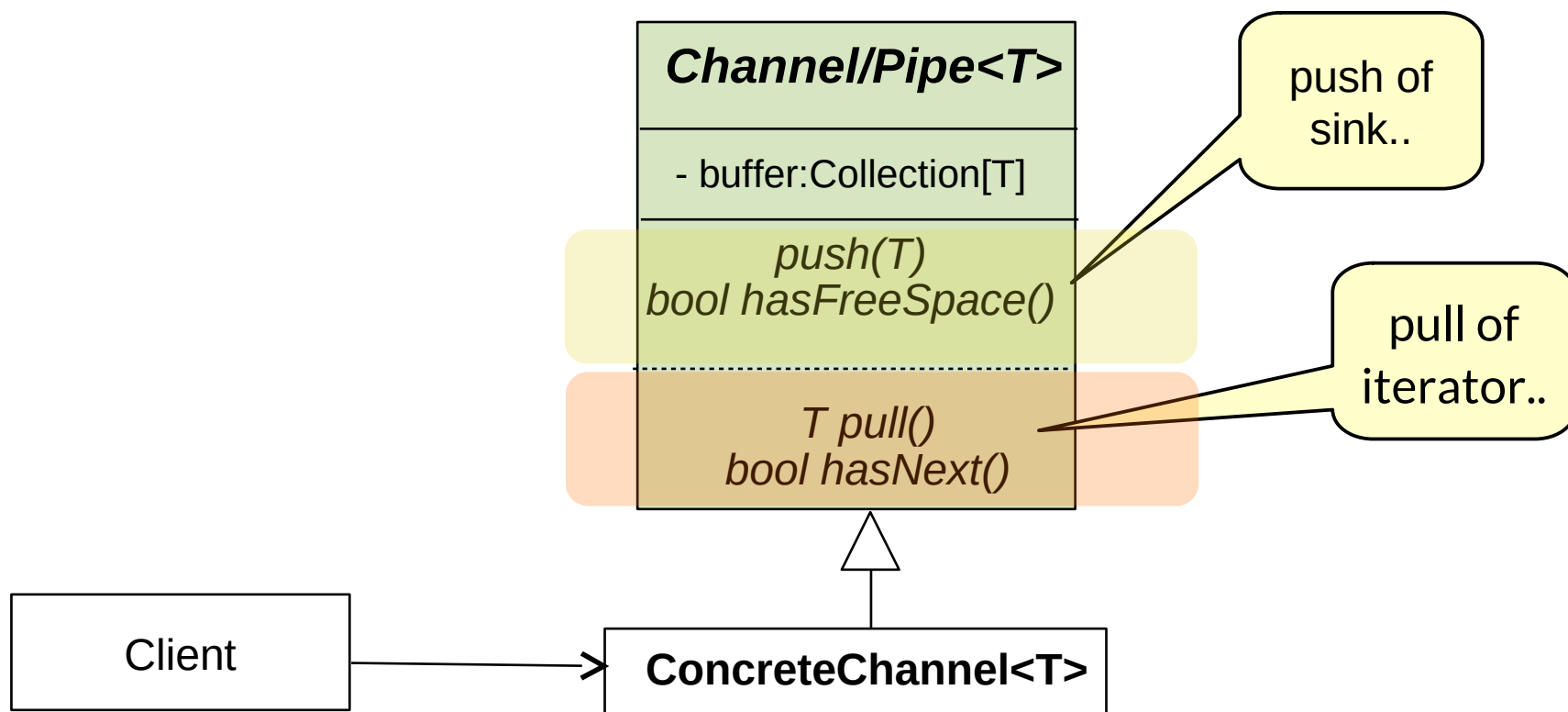
## 23.4 Channels (Pipes)

- ▶ 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.

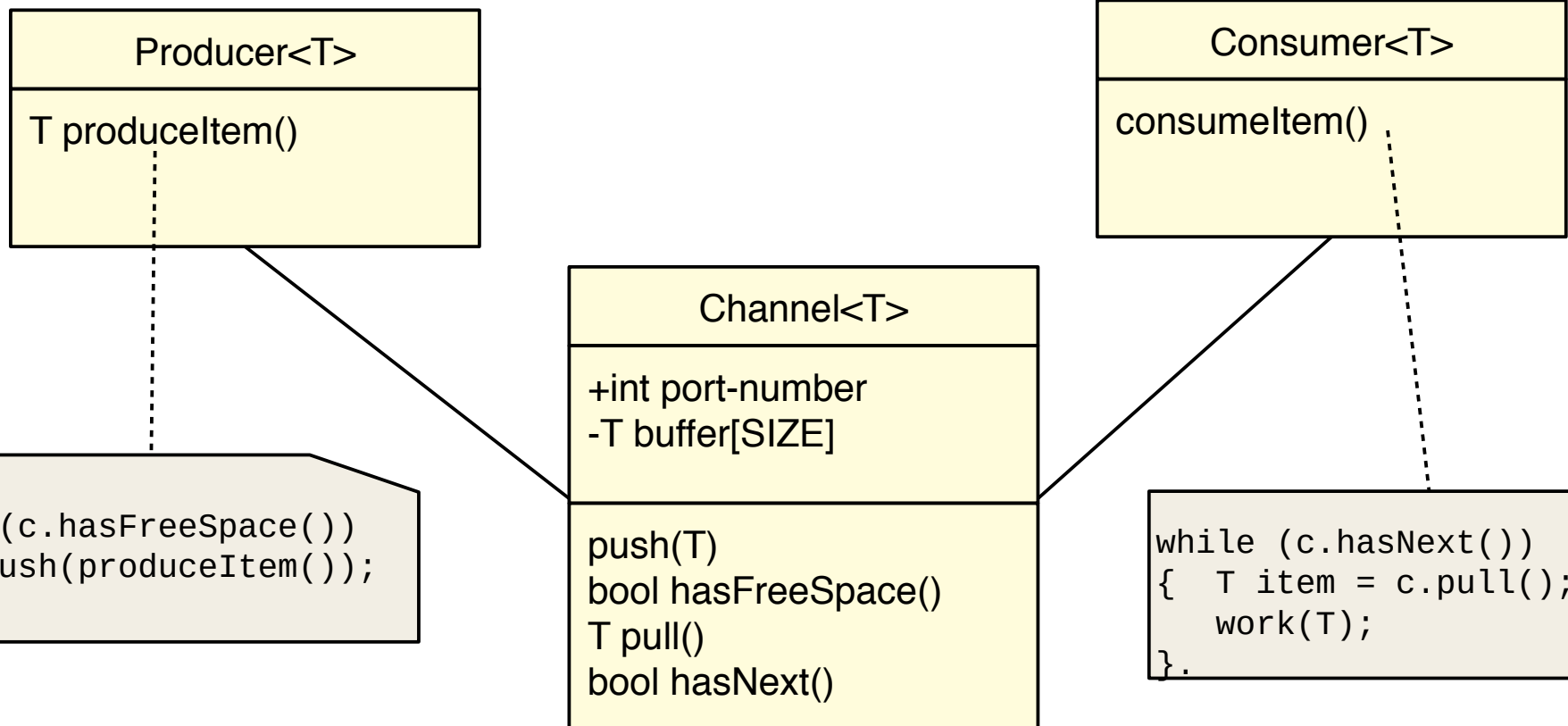


# Channels in anderen Programmiersprachen

- ▶ Channels (pipes) kommen in vielen Sprachen als Konstrukte vor
- ▶ Linux versieht jeden Prozess mit 3 Streams (stdin, stdout, stderr), zwischen denen Kanäle gezogen werden können.
- ▶ Textuelle Syntax:
  - **Shell-Skripte in Linux** bestehen oft aus Produzenten-Konsumenten-Teams, die mit Kanal kommunizieren (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, pull) und ">>" (push, 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)



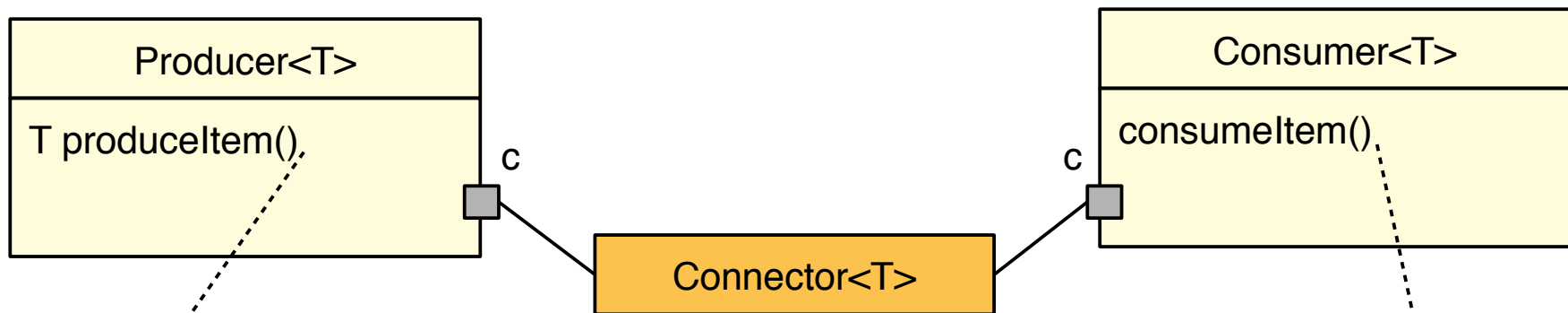


# Konnektoren als Verallgemeinerung von Kanälen

**Wdh.:**

**Def.:** Eine **Konnektorklasse** ist eine Teamklasse, 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



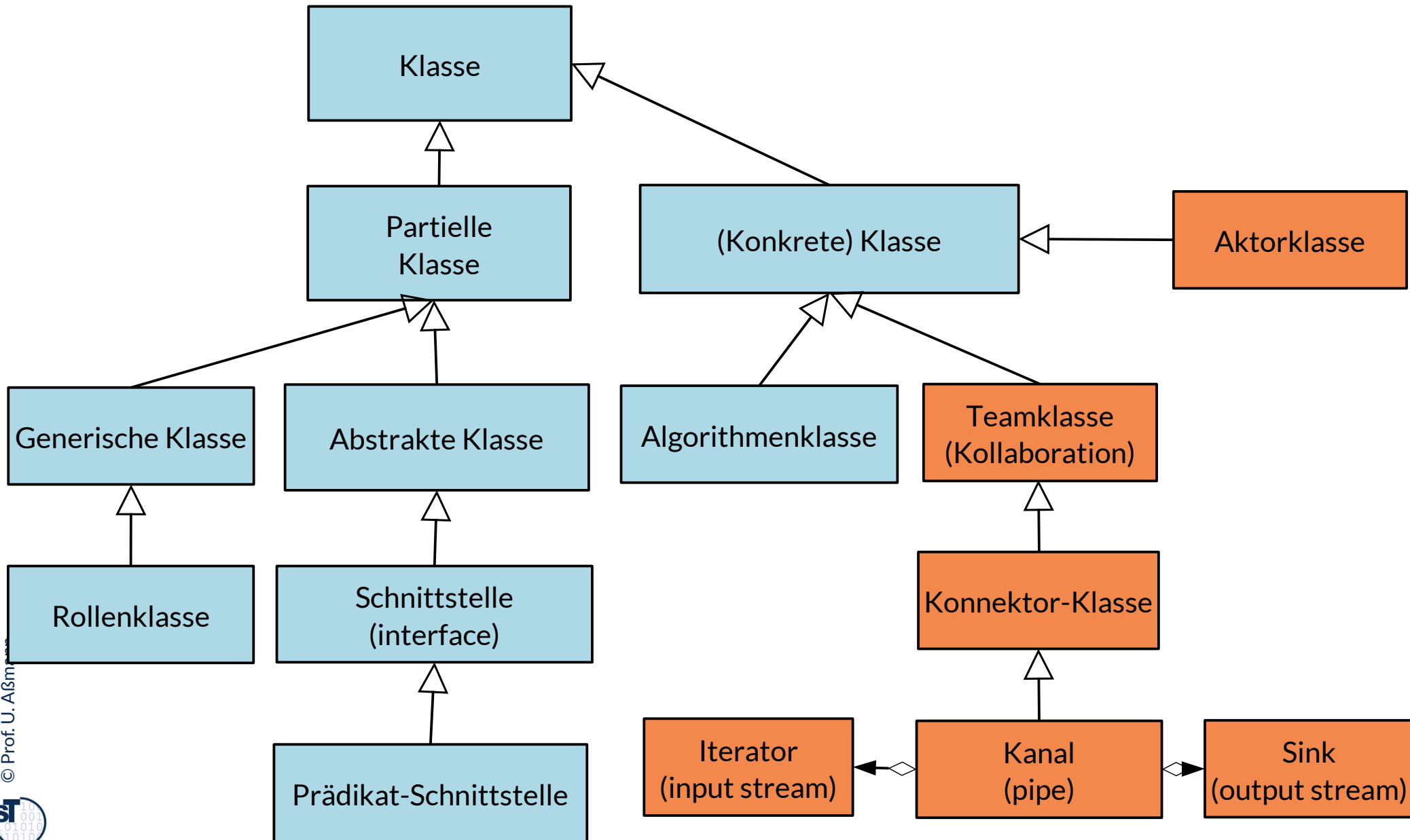
```
while (c.hasFreeSpace())
{   c.push(produceItem());
}.
```

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

**Trenne Konnektoren von Anwendungsklassen ab, um die Wiederverwendung zu erhöhen**

- ▶ Konnektoren gehören zum olympischen Ring “Infrastruktur”

# Q2: Begriffshierarchie von Klassen (Erweiterung)



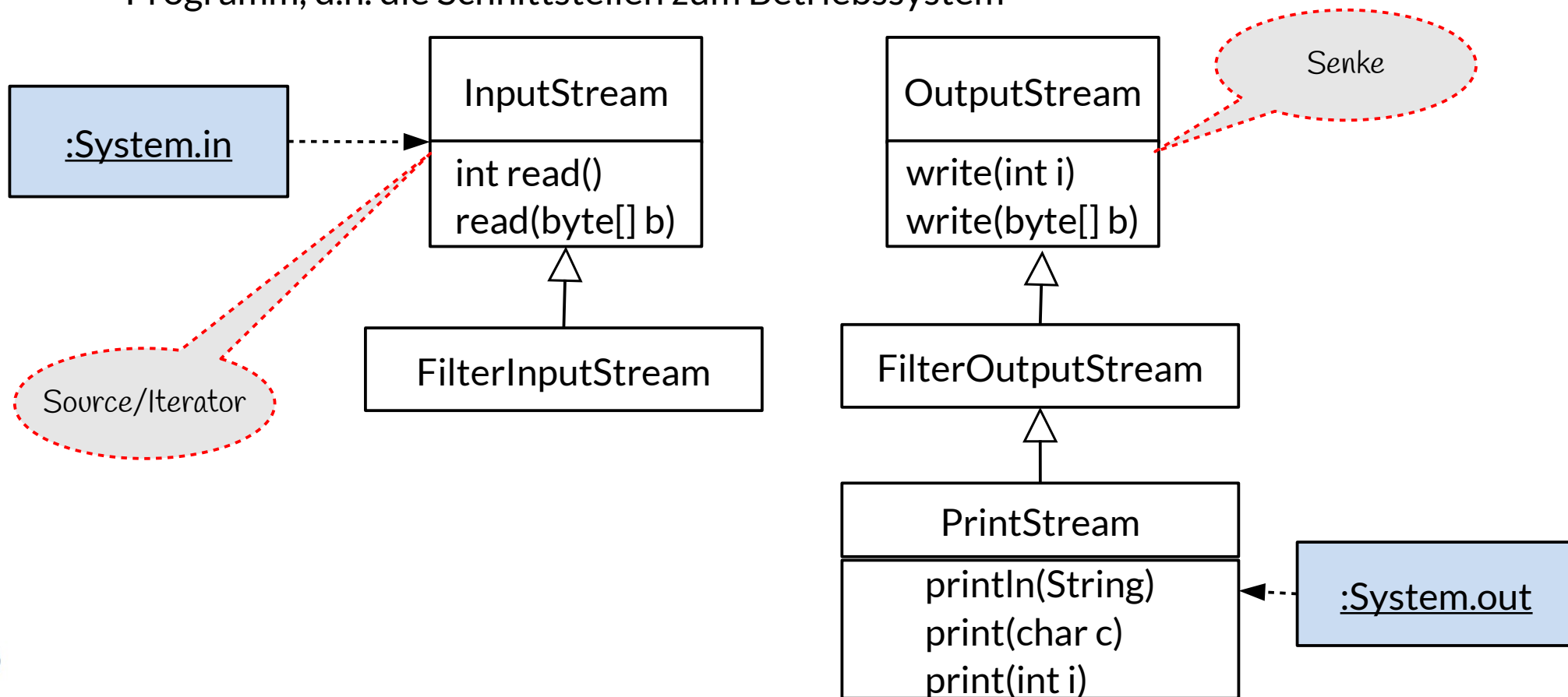
## 23.4 Channels und Streams sind überall: Strombasierte Ein- und Ausgabe (Input/Output) und persistente Datenhaltung

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



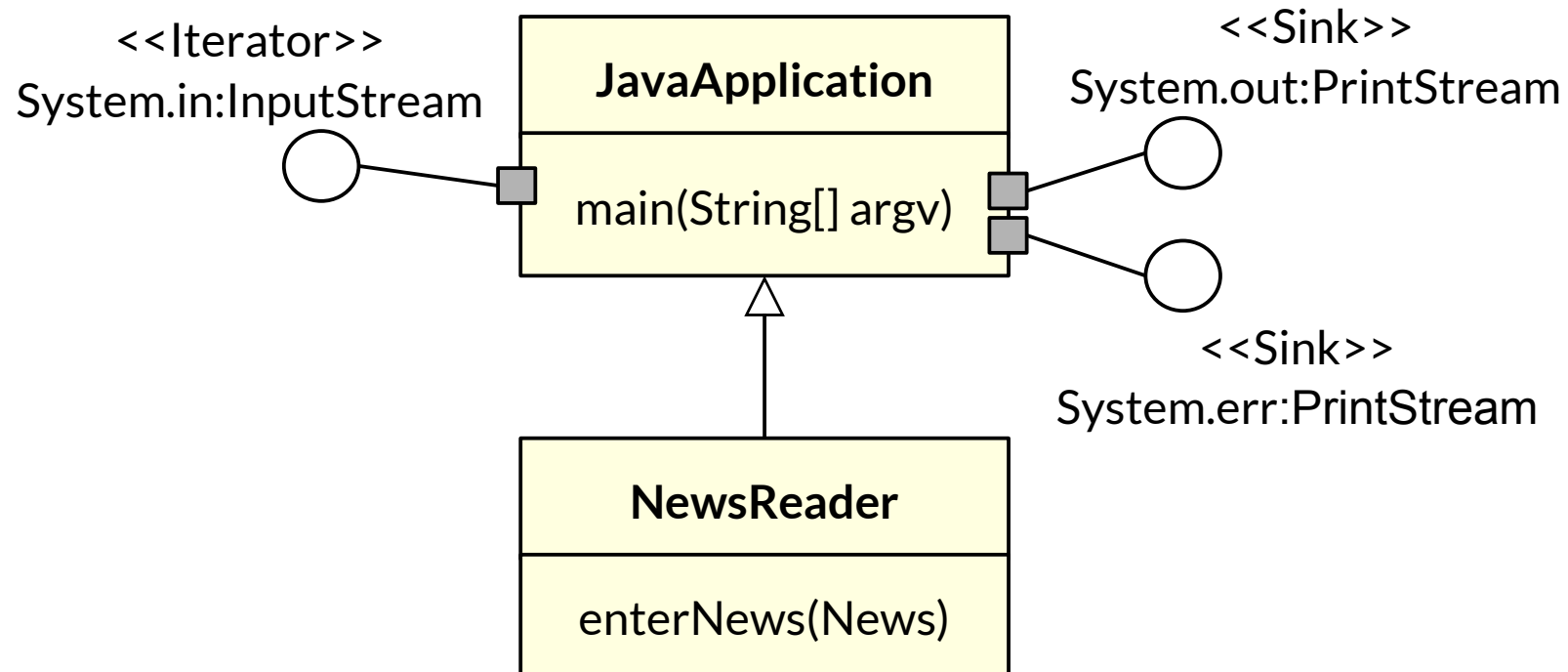
## 23.4.1 Ein- und Ausgabe in Java mit Strömen und Senken

- ▶ 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

- ▶ Ein Programm in Java hat 3 Standard-Ströme `in`, `out`, `err`
  - 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 (Siehe Vorlesung “Datenbanken”)
    - 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

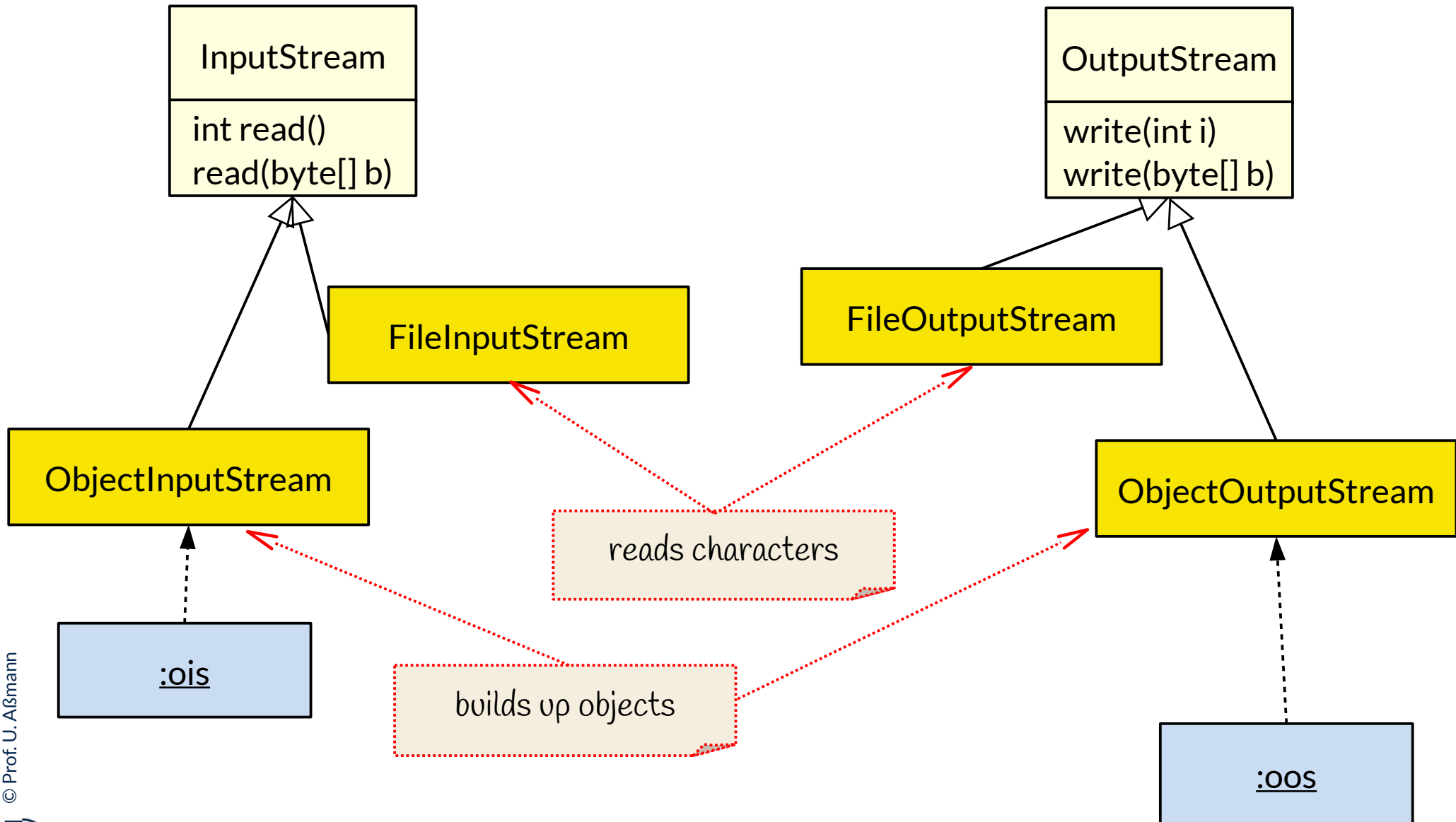
```
import java.io.*;

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);

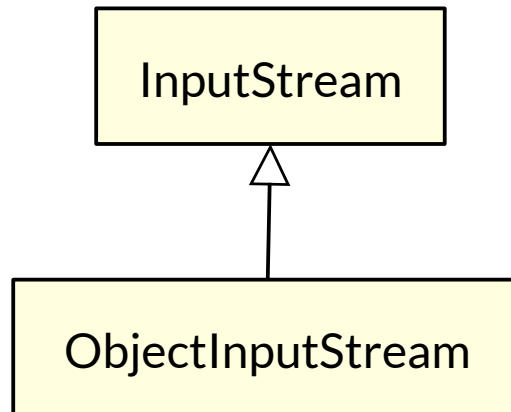
// internally realized as push for all child objects
oos.writeObject(xobj); // push
...
```

# Verschiedene Arten von Input und Output Streams im JDK



# ObjectInputStream aus dem JDK

- ▶ 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



# Objekt-Serialisierung: Einlesen mit `ObjectInputStream` aufsetzend auf `FileInputStream`

```
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

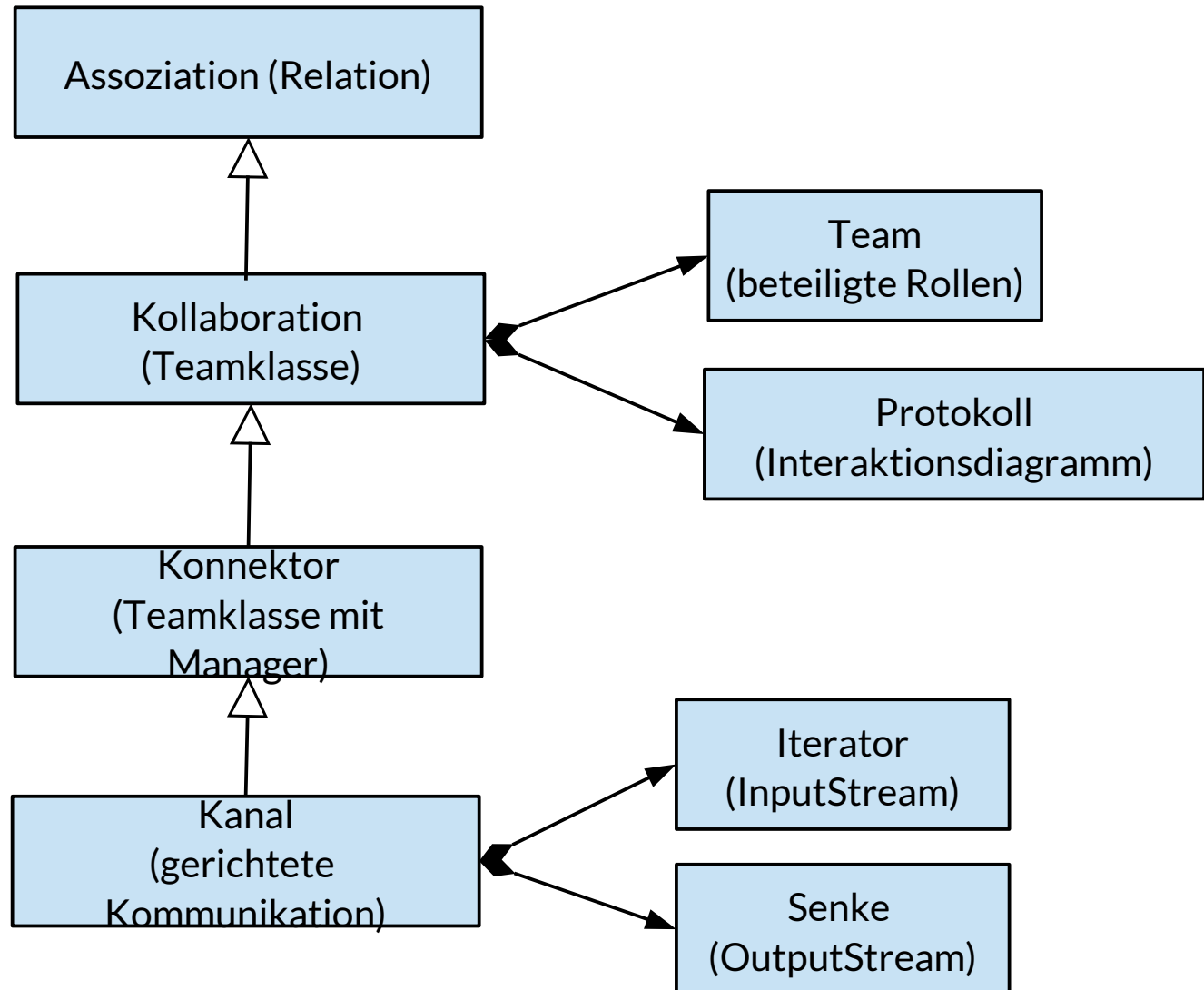
- ▶ Was unterscheidet ein Team von einer Kollaboration?
- ▶ Was unterscheidet einen Iterator von einer Senke?
- ▶ Was ist der Unterschied zwischen einer fixen und einer unbestimmt großen Datenstruktur?
- ▶ Warum läuft Ein- und Ausgabe in Programmen immer über Ströme?
- ▶ Wie heißen die Java-Stromobjekte für Ein- und Ausgabe?
- ▶ Was unterscheidet ein Team von einer Kollaboration?
- ▶ Wann macht es Sinn, die Kollaboration eines Teams durch ein Objekt, den Konnektor, zu kapseln?
- ▶ Warum ist ein Kanal ein spezieller Konnektor?
- ▶ Was ist besonders genial am UNIX/Linux Ein-/Ausgabesystem?
- ▶ Einige Folien stammen aus den Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. Used by permission.

# Anhang



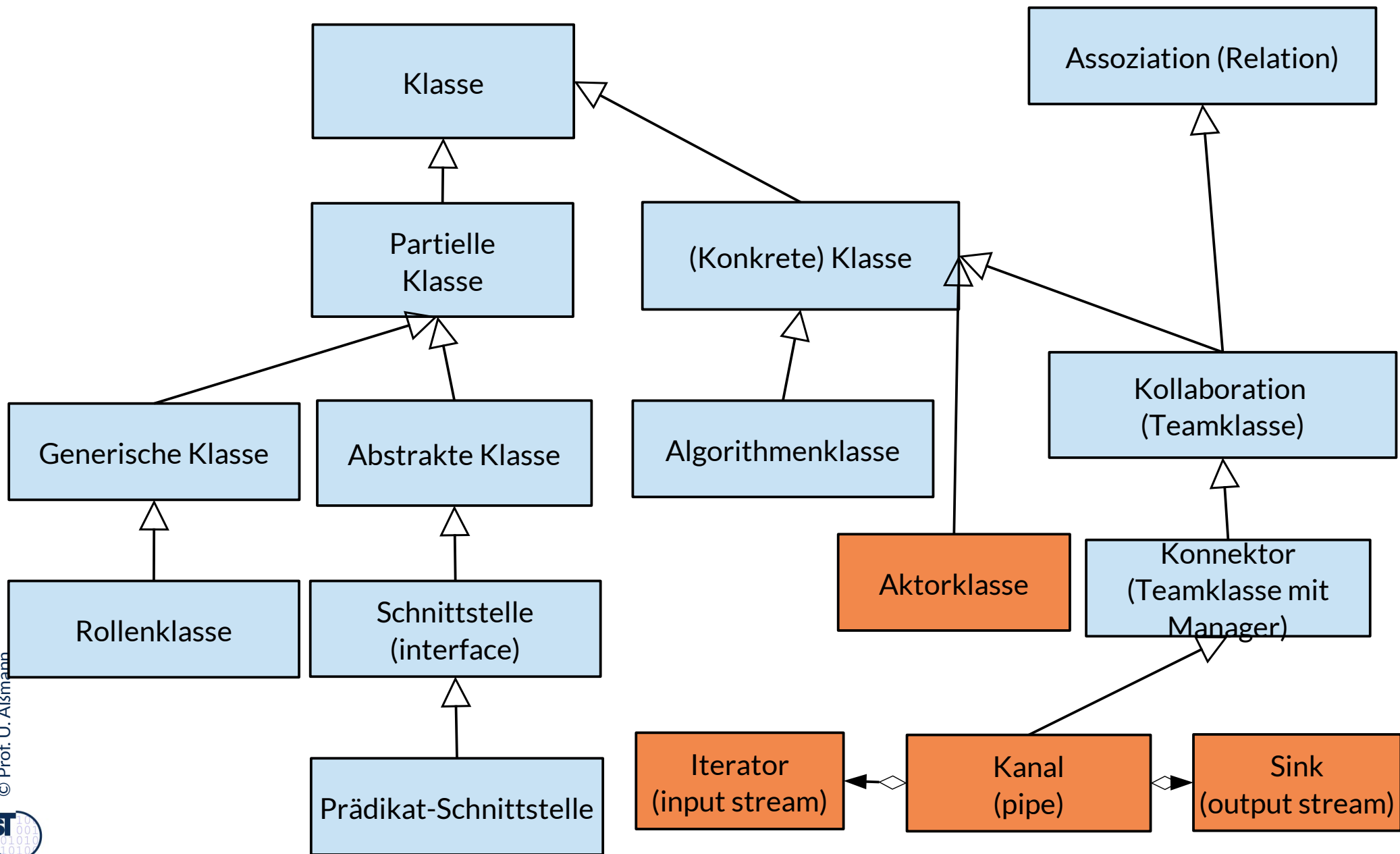
DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

# Relationale Klassen (Konnektoren)





# Q2: Begriffshierarchie von Klassen (Erweiterung)



# Iterator-Implementierungsmuster in modernen Sprachen

- ▶ 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();..
```

## 23. Entwurfsmuster für Kommunikation in Teams – Programmierung von fixen Netzen für die Programmierung der I/O und des Internet Kommunikation mit Iteratoren, Senken, Kanälen und Konnektoren Datenhaltung mit potentiell unendlichen Collections

Prof. Dr. Uwe Aßmann  
Lehrstuhl Softwaretechnologie  
Fakultät für Informatik  
Technische Universität Dresden  
Version 20-0.3, 6/25/20

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



DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

## Betreff: "Softwaretechnologie für Einsteiger" 2. Auflage

- ▶ zur Info: o.g. Titel steht zur Verfügung:
  - 50 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!

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

Johnny Cash's classic implementation  
<https://www.youtube.com/watch?v=egf4X1UzsAA>

see also seine CD "American IV"



**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.

© Prof. U. A. G. Mann



# 50. Dixieland-Festival in Dresden wegen Corona verschoben auf 2021!

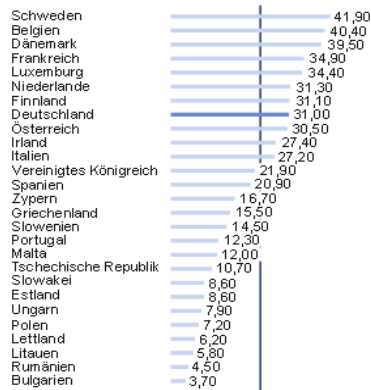
## Warum müssen Softwareingenieure fortgeschrittenes Wissen besitzen?

4 Softwaretechnologie (ST)

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



Arbeitskosten in der Privatwirtschaft 2012  
je geleistete Stunde in EUR



Europäische Union (EU 27) 23,50

© Statistisches Bundesamt, Wiesbaden 2013

© Prof. U. A. Gmann



<http://www.heise.de/resale/imgs/17/1/0/0/1/3/4/1/ArbeitskostenEULaenderStart2012-9bb2e8b041f1342e.png>

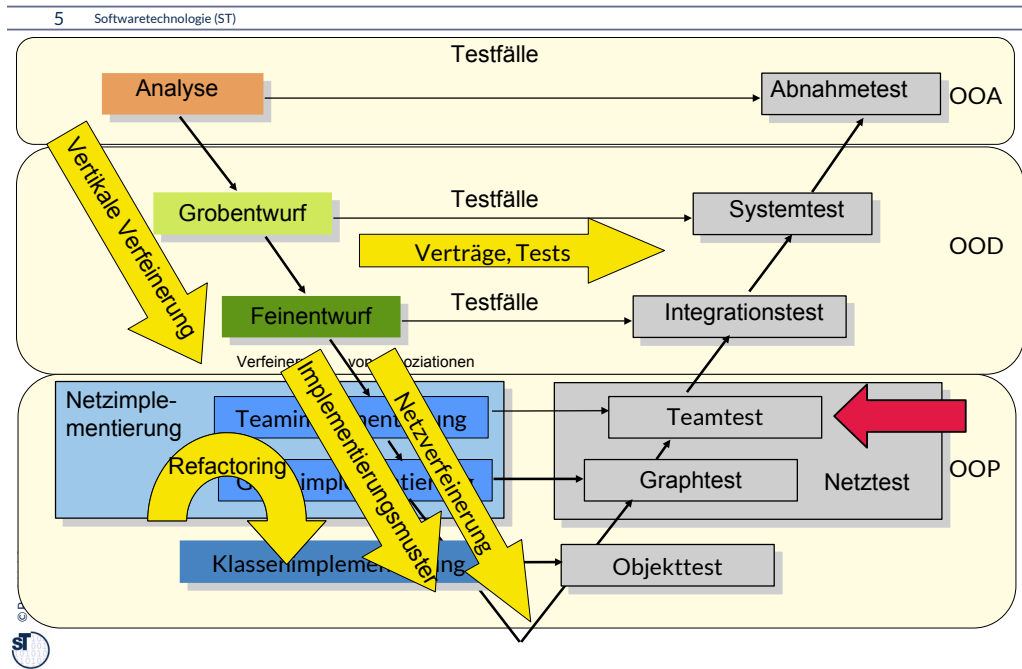
Für einfache Programmierer ohne Architektur-Knowhow ist die Welt nicht mehr so einfach, weil es viel Konkurrenz gibt.

Die Lohnunterschiede zu Schwellenländern sind signifikant und machen es den Unternehmen leicht, das Programmieren auszulagern.

Beschaffen Sie sich also Architektur-Knowhow, um Software-Architekt zu werden.

## Q4: Softwareentwicklung im V-Modell

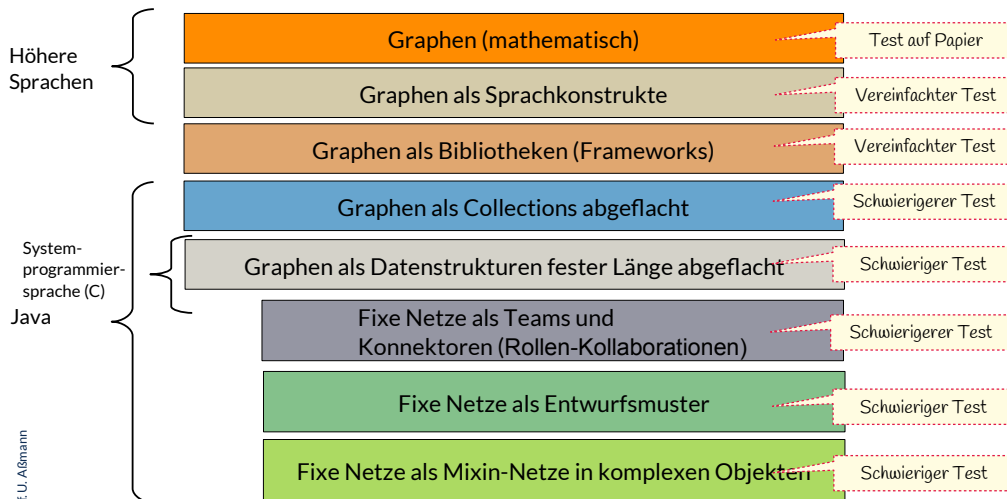
[Boehm 1979]



Die stufenweise Verfeinerung von dem Pflichtenheft, dem Ergebnis der Analyse aus, hin zur Implementierung beinhaltet Netzverfeinerung und Teamverfeinerung.

## Repräsentation von flexiblen und fixen Objektnetzen als Datenstrukturen (Netzverfeinerung)

6 Softwaretechnologie (ST)



© Prof. U. Altmann



Auf Ebene der Anforderungen werden **flexible Objektnetze** als math. Graphen dargestellt. Im Grobentwurf und Feinentwurf werden sie repräsentiert durch (verfeinert zu)

- Graphen als **Sprachkonstrukte** (für Sprachen, die das eingebaut haben)
  - Rapid Application Development (RAD)
- Graphen aus Java-Graph-**Bibliotheken** (jgrapht)
- **Implementierungsmuster** wie **Collections**, nach dem Abflachen/Flachklopfen von bidirektionalen Assoziationen in gerichteten Links
- **maschinennahe Implementierungsmuster** wie **Datenstrukturen** fester Länge (Arrays, Matrizen) (speicher-bewusstes Programmieren)

Wohl dem, der eine gute Testsuite für flexible Objektnetze hat!

- Jgrapht, unser Beispiel-Framework für Graphen, hat Generatoren für Graphen, die die Konstruktion von Testsuiten unterstützen.

**Fixe Netze** mit statisch festem  $n$ ,  $m$  (z.B. 1:1-Assoziationen) können durch **Teams** (Kollaborationen, Konnektoren) verfeinert werden durch

- **Konnektorklassen** realisieren Assoziationen und tragen Rollentypen als Assoziationsenden
- **Entwurfsmuster** wie Decorator, Chain, Composite
- interne Subobjektnetze großer Objekte (bobs), Endo-Assoziationen

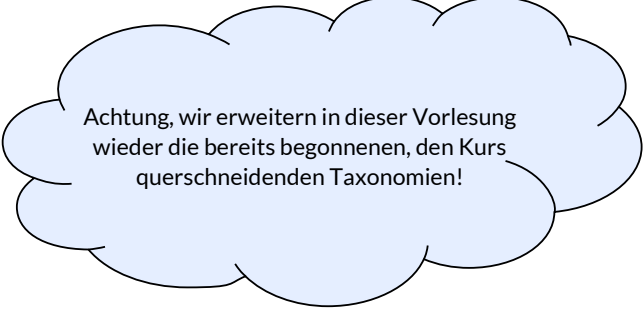




## E.23.1 Lernen mit Begriffshierarchien, die die Vorlesung querschneiden



- ▶ 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!

Eine Vorlesung kann nur eine Teilmenge des Stoffes aufbereiten; übergreifende Zusammenhänge von Begriffen müssen oft selbst erarbeitet werden, um die Vorlesungen im Kopf zusammensetzen.

Viele Begriffshierarchien querschneiden mehrere Vorlesungen.

## 23.1 Teams für fixe Netze

- ▶ 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



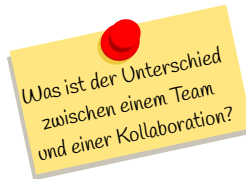
DRESDEN  
concept  
Exzellenz aus  
Wissenschaft  
und Kultur

- ▶ **Problem:**
  - Das Management von Objekt-Netzen ist eine der schwierigsten und fehleranfälligen Aufgaben im objektorientierten Programmieren.
  - Objektnetz-Programmierung ist sehr wichtig für Softwarequalität
- ▶ Neue Sprachen:
  - Java gehört zur 1. Generation von objektorientierten Programmiersprachen.
  - Die 2. Generation verbessert die Beschreibung von Teams, Netzen von Objekten:
    - Object Teams [Www.objectteams.org](http://www.objectteams.org)
    - SCROLL <https://github.com/max-leuthaeuser/SCROLL>
- ▶ Wir lernen in diesem Kapitel, wie man mit Bibliotheken und Entwurfsmustern das Programmieren von Objektnetzen auch in Java verbessern kann!

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

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

- ▶ Historik: Der Begriff des Teams wurde von der TU Berlin im Projekt [www.objectteams.org](http://www.objectteams.org) geprägt.

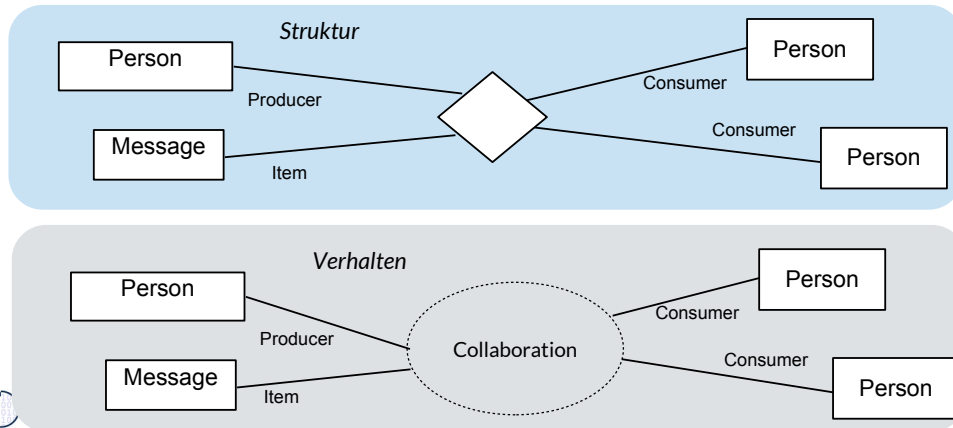


Objektgruppen sind vernetzt, können aber nicht zusammen als Ganzes angesprochen werden – dann spricht man von einem Team.

## Kollaborationen kapseln das Verhalten von fixen Netzen (Teams)

12 Softwaretechnologie (ST)

- ▶ Die *Struktur* von Teams (fixes Netz mit n Mitgliedern) wird in UML durch n-stellige Assoziationen dargestellt
- ▶ Das *Verhalten* eines fixen Netzes und seine Kommunikation durch Kollaborationen
- ▶ Def.: Eine **Kollaboration** (*collaboration*, **Kollaborationsklasse**) realisiert die Kommunikation eines Teams mit einem festen anwendungsspezifischen Protokoll

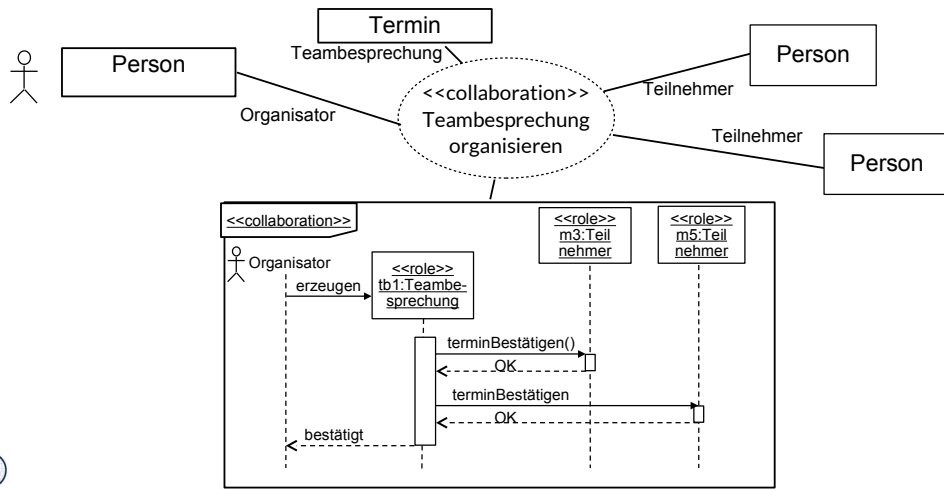


Kollaborationen sind Zusammenarbeitsformen von Objektgruppen, also fixen Objektnetzen.

In UML kann man sie durch Collaboration beschreiben – in Java nicht.

# Kollaborationen kapseln Verhalten eines Teams durch Interaktionsprotokolle

- ▶ Eine **Kollaboration** beschreibt die anwendungsspezifische Interaktion, Nebenläufigkeit und Kommunikation eines Teams, d.h. einer fixen Menge von beteiligten Objekten.
- ▶ Die Kollaboration beschreibt also ein Szenario querschneidend durch die Lebenszyklen mehrerer Objekte



**Def.:** Ein **Teamobjekt** ist ein Objekt, das die Kommunikation einer Gruppe (feste Anzahl) von interagierenden und kommunizierenden Objekten kapselt. Ist ein Teamobjekt hauptsächlich mit dem technischen Austausch von Daten zwischen den Objekten beschäftigt, heißt es **Konnektor**.

Bsp: Teams: Dynamo Dresden Team, Staatskapelle  
Konnektoren: Aldi--Peter Müller:Käufer, Finanzamt--Jenny Klein:Steuerzahler

**Def.:** Kann eine Kollaboration durch eine Klasse gekapselt werden, spricht man von einer **Teamklasse**.  
Spezialfall beim Datenaustausch: **Konnektorklasse**, kurz **Konnektor**.

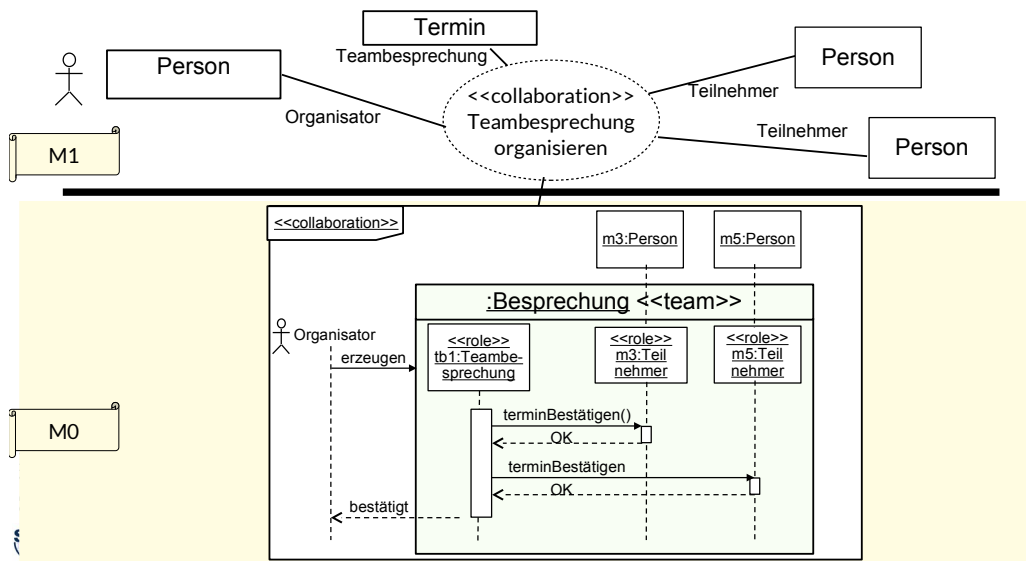
Bsp: Teamklassen: Fußballmannschaft, Kapelle  
Konnektorklassen: Produzent-Konsument, Client-Server

Kollaborationen können in UML, aber nicht in Java beschrieben werden; Teams und Konnektoren schon, denn sie bilden Objekte bzw. Klassen.

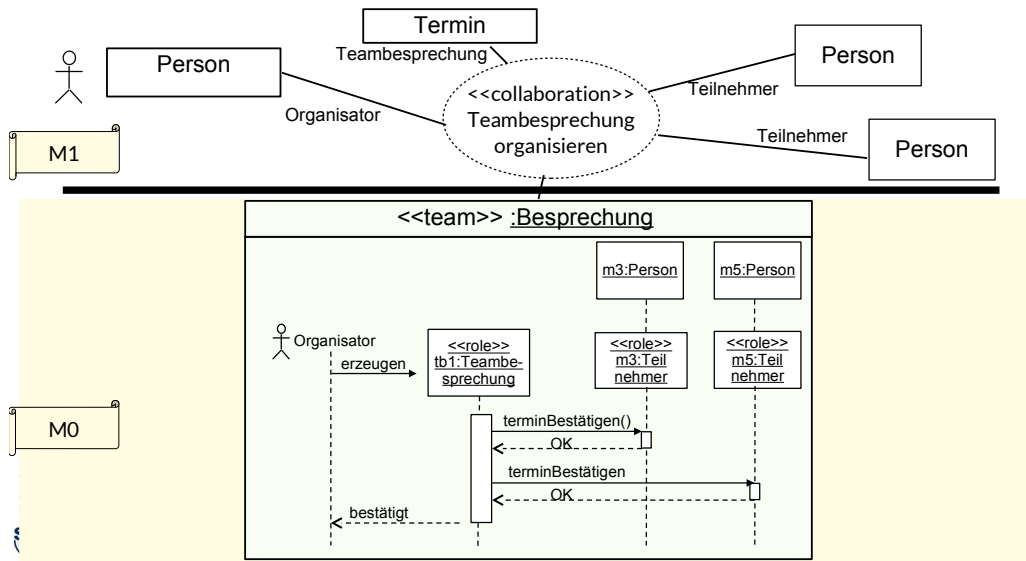


# Teamklassen kapseln Verhalten durch Interaktionsprotokolle

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



- Wir schreiben die Kollaboration in den Konnektor



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

Die Programmierung von Teams besteht aus:

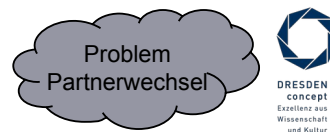
- Verknüpfen des Objektnetzes
- Angabe einer Kollaboration, die bestimmt, wie das Team kommunizieren und kollaborieren soll
- Vernetzung der Objekte des Teams mit dem Teamobjekt (dem "Ganzen")

In der gleichen Weise bestimmen Entwurfsmuster Teams.

## 23.2 Aktoren und Kanäle (Actors and Channels) für lose gekoppelte Teams

### Wie man Konnektoren für die Programmierung des Internets nutzt

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



**Def.:** Ein **Aktor (actor)** ist ein aktives Objekt mit einem eigenen Lebensfaden (thread).  
Ein Aktor besitzt Kanäle, über die er Botschaften empfängt und sendet



BeEin Webserver ist ein Aktor und besitzt *ports* als Kanäle (http-Port 80)

- Liste der standardisierten Ports  
[https://de.wikipedia.org/wiki/Liste\\_der\\_standardisierten\\_Ports](https://de.wikipedia.org/wiki/Liste_der_standardisierten_Ports)

▶ Unter LINUX heißt der Aktor **Prozess** und besitzt die Kanäle

- `stdin` (Standardeingabe)
- `stdout` (Standardausgabe)
- `stderr` (Standardfehlerausgabe)



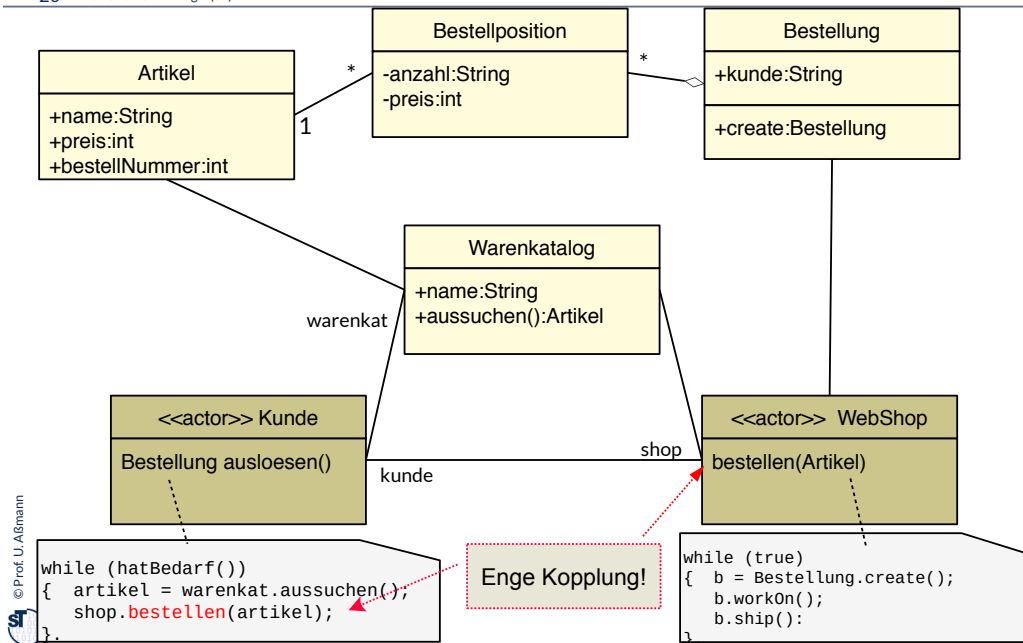
Die Programmierung von Teams besteht aus:

- Verknüpfen des Objektnetzes
- Angabe einer Kollaboration, die bestimmt, wie das Team kommunizieren und kollaborieren soll
- Vernetzung der Objekte des Teams mit dem Teamobjekt (dem "Ganzen")

In der gleichen Weise bestimmen Entwurfsmuster Teams.

## Beispiel: Bestellung auf einem Webshop

20 Softwaretechnologie (ST)



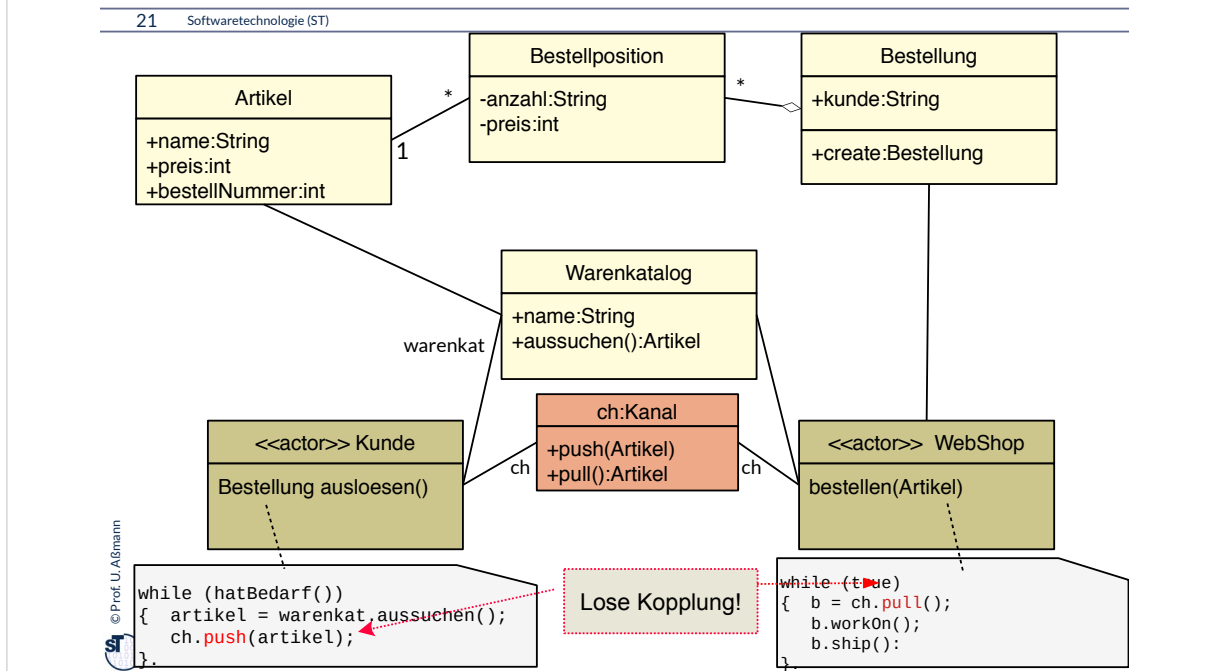
Kanäle sind eines der elementarsten Entwurfsmuster für das Web und das Internet. Sie begegnen dem Problem,

- dass ein Client die Details eines Servers kennen muss
- dass ein Produzent die Details eines Konsumenten kennen muss

Idee: man führt ein Zwischenobjekt ein, das nur der Kommunikation dient

- dieses hat standardisierte Schnittstellen und “versteckt” den Server bzw. Konsumenten
- der Channel gehört zum olympischen Ring “Infrastruktur”, denn es dient ausschließlich der Kommunikation

## Beispiel: Bestellung auf einem Webshop; nun lose gekoppelt



Kanäle sind eines der elementarsten Entwurfsmuster für das Web und das Internet. Sie begegnen dem Problem,

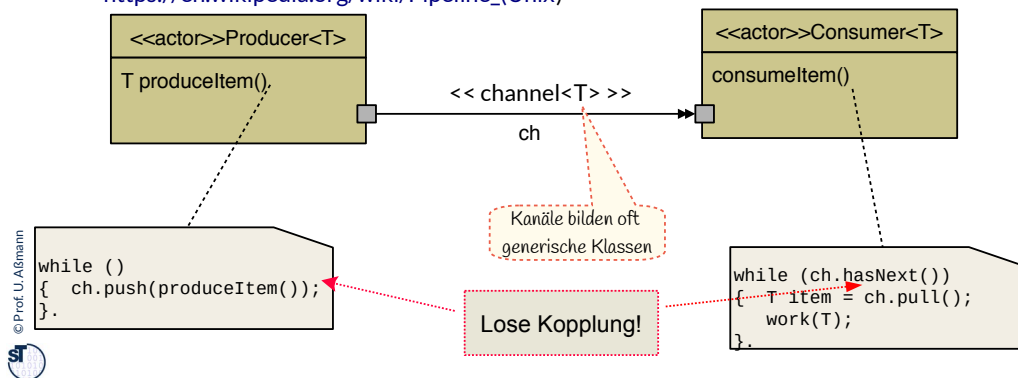
- dass ein Client die Details eines Servers kennen muss
- dass ein Produzent die Details eines Konsumenten kennen muss

Idee: man führt ein Zwischenobjekt ein, das nur der Kommunikation dient

- dieses hat standardisierte Schnittstellen und "versteckt" den Server bzw. Konsumenten
- der Channel gehört zum olympischen Ring "Infrastruktur", denn es dient ausschließlich der Kommunikation

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

- ▶ UML Notation: Andocken eines Kanals an *sockets (pins, 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))



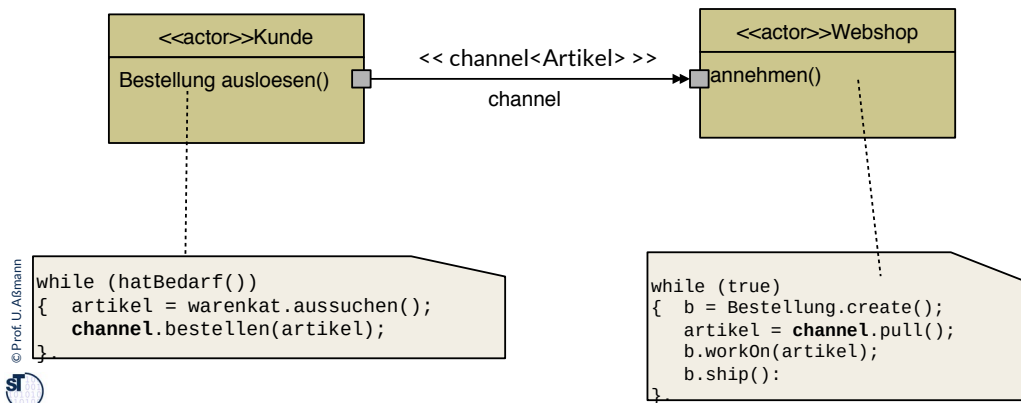
Kanäle (Pipes) und Streams wurden in UNIX erfunden (ca. 1970). Sie sind nicht verschwunden, denn sie bilden eine zweite Art des Verknüpfens von Komponenten:

- Algebraik: Programmieren mit Funktionsaufrufen
- Co-Algebraik: Programmieren mit Streams

In UML werden Kanäle mit Pfeilen zwischen Pins (ports) von Aktoren dargestellt



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



## 23.3 Entwurfsmuster Channel

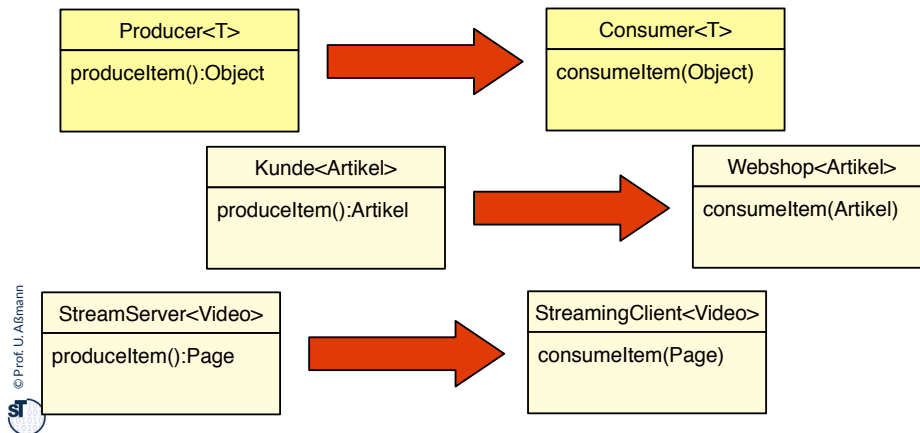
- ▶ Eines der wesentlichen Entwurfsmuster für Internet- und Webprogrammierung.
- ▶ Channel sind technische Objekte, zum olympischen Ring “Infrastruktur” gehörend
- ▶ Darstellung von potentiell unendlichen Collections



## Wie organisiere ich die "unendlich lange" Kommunikation zweier Akteure?

25 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, unbestimmte Collections?
- ▶ **Antwort:** mit Kanälen.



**Problem:** Über der Zeit laufen in einem Webshop eine Menge von Bestellungen auf

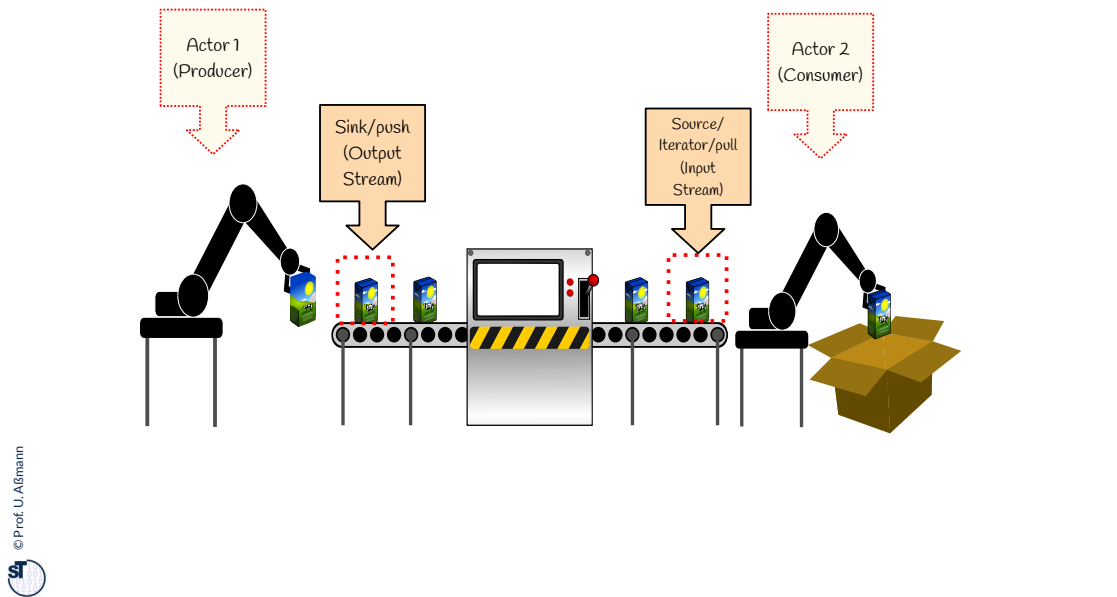
- Sie sind aber nicht in endlicher Form in Mengen oder Collections zu repräsentieren

**Frage:** Wie repräsentiert man potentiell unendliche Collections?

**Antwort:** mit Kanälen.

Viele Prozesse in Softwaresystemen benötigen den Austausch von unendlichen Mengen von Daten.

Wie repräsentiert man potentiell unendliche Mengen und Collections?

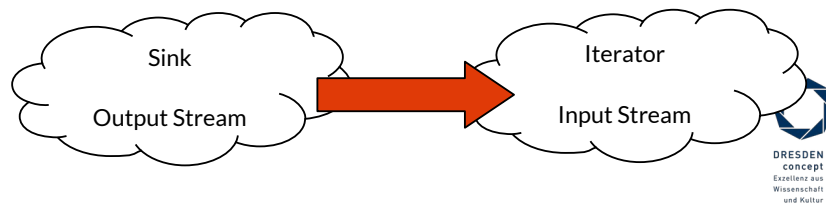


Ein Kanal ist einem Förderband vergleichbar mit

- einer Eingabestelle (sink)
- einer Ausgabestelle (source, iterator)
- Zwei Aktoren, die die Sink “pushen” und die Source “pullen”

## 23.3.1 Entwurfsmuster Iterator (Eingabestrom, input stream)

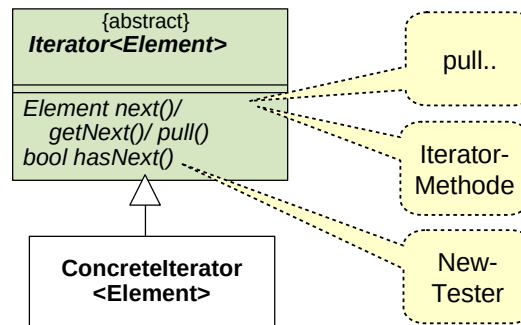
Kanäle bestehen aus mit einander verbundenen Enden, mindestens zweien



## Entwurfsmuster Iterator (Input Stream) (Implementierungsmuster)

28 Softwaretechnologie (ST)

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



Das Entwurfsmuster Iterator wird benutzt, um potentiell unendliche Folgen von Eingaben in eine Klasse oder Komponente zu realisieren.

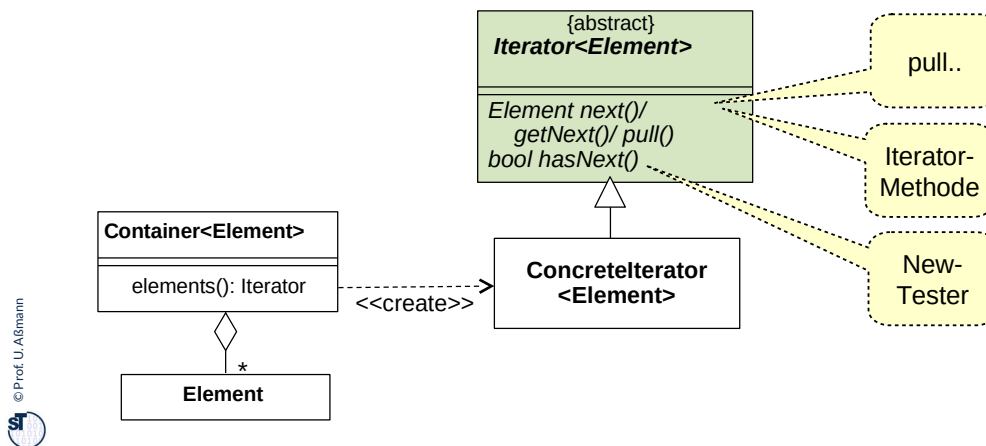
Merkmale Entwurfsmuster **Iterator** (auch: **input stream**, source, Cursor, Enumeration, pull-socket)

- Sequentielles, polymorphes Durchlaufen der Elemente eines strukturieren Objekts oder einer Collection (Navigation). Typisierung des Inhalts (der Elemente)
- Aufzählen der in einem “Behälter” befindlichen Elemente durch *Herausziehen (pull)* mit Hilfe einer *Iteratormethode (next, getNext, pull)*
- Keine Aussage über die Reihenfolge im Container möglich
- Merken des Zustandes der Navigation

## Entwurfsmuster Iterator (Input Stream) (Implementierungsmuster)

29 Softwaretechnologie (ST)

- ▶ Containerklassen haben Iteratoren

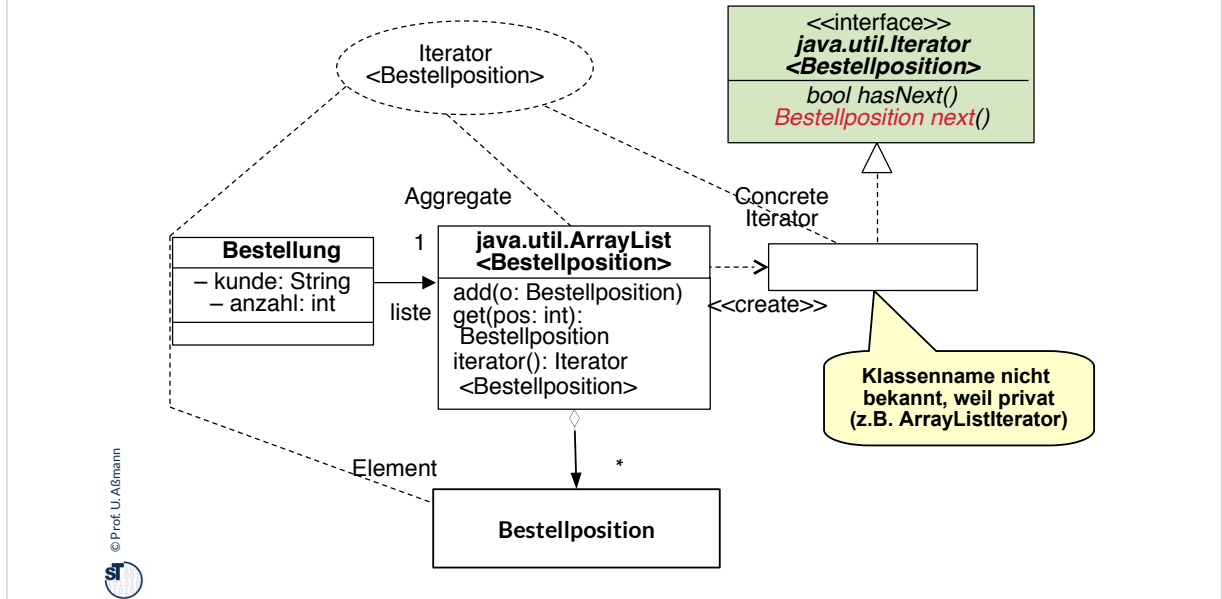


Das Entwurfsmuster Iterator wird benutzt, um potentiell unendliche Folgen von Eingaben in eine Klasse oder Komponente zu realisieren.

Merkmale Entwurfsmuster **Iterator** (auch: **input stream**, **Cursor**, **Enumeration**, **pull-socket**)

- Sequentielles, polymorphes Durchlaufen der Elemente eines strukturieren Objekts oder einer Collection (Navigation). Typisierung des Inhalts (der Elemente)
- Aufzählen der in einem “Behälter” befindlichen Elemente durch *Herausziehen (pull)* mit Hilfe einer *Iteratormethode (next, getNext, pull)*
- Keine Aussage über die Reihenfolge im Container möglich
- Merken des Zustandes der Navigation

# Iterator-Beispiel in der JDK (ArrayList)



Collections haben Iteratoren zugeordnet. Man kann mit dem Iterator Stück für Stück aus einer Collection alle Elemente herausholen, sodass transparent bleibt, wie viele Elemente die Collection hat.



- ▶ Verwendungsbeispiel:

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

- ▶ Einsatzzwecke:

- Verbergen der inneren Struktur einer Datenstruktur
- **bedarfsgesteuerte Berechnungen** auf der Struktur
- "unendliche" Datenstrukturen

beachte

### Einsatzzwecke:

- Iteratoren und Iteratormethoden erlauben, die Elemente einer komplexen Datenstruktur nach außen hin bekannt zu geben, aber ihre Struktur zu verbergen (Geheimnisprinzip)
- Sie erlauben **bedarfsgesteuerte Berechnungen (processing on demand, lazy processing)**, weil sie nicht alle Objekte der komplexen Datenstruktur herausgeben, sondern nur die, die wirklich benötigt werden
- Iteratoren können "unendliche" Datenstrukturen repräsentieren, z.B. unendliche Mengen wie die natürlichen Zahlen, oder "unendliche" Graphen wie die Karte der Welt in einem Navigator

## Anwendungsbeispiel mit Iteratoren

```
import java.util.Iterator;
...
class Bestellung {

    private String kunde;
    private List<Bestellposition> liste;
    ...
    public int auftragssumme() {
        Iterator<Bestellposition> i = liste.iterator();
        Bestellposition b;

        int s = 0;
        while (i.hasNext()) { // Test whether next element
                               // (on the source) is available
            b = i.next();     // pull the next element
            s += b.positionspreis();
        }
        return s;
    }
    ...
}
```



## For-Schleifen auf Iterable-Prädikatschnittstellen

- ▶ 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;
}
```



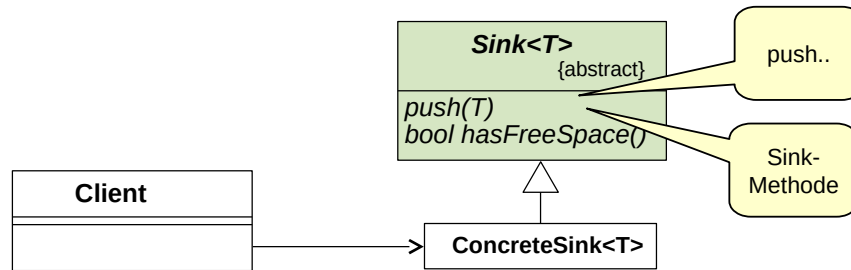


## 23.3.2 Senken (Sinks, OutputStream)

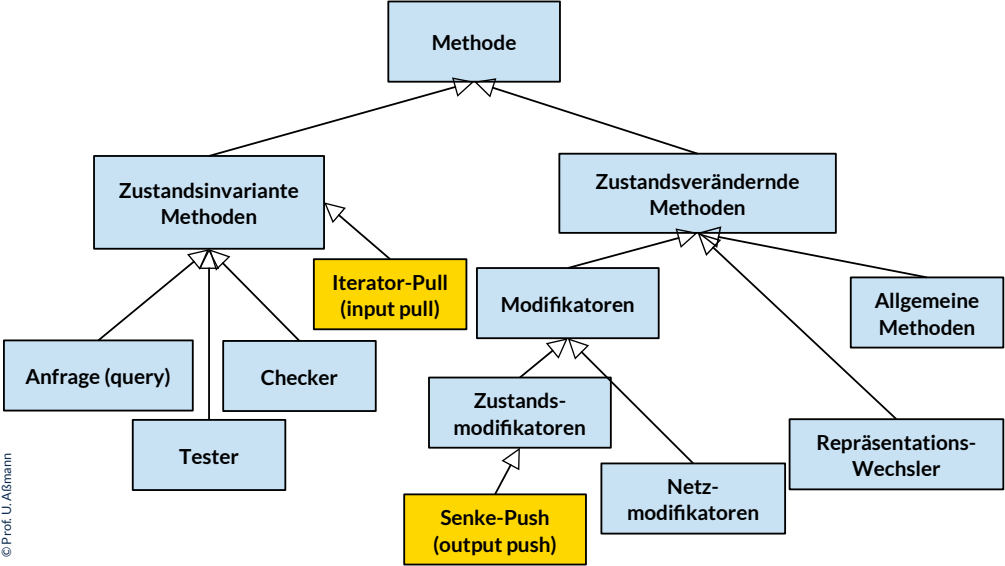


## Entwurfsmuster Senke (Implementierungsmuster)

- ▶ 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



# Q3: Erweiterung: Begriffshierarchie der Methodenarten





## 23.4 Channels (Pipes)

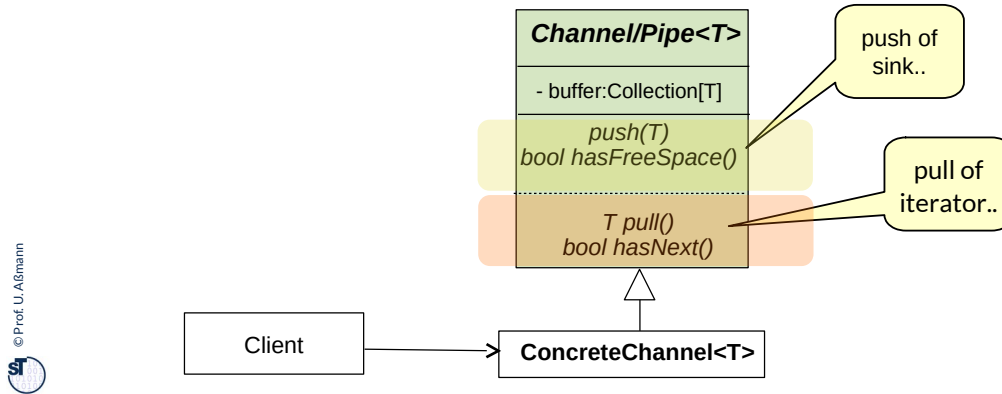
- ▶ Die Kombination aus Senken und Iteratoren, ggf. mit beliebig großem Datenspeicher
- ▶ Eine Pipe ist ein einfacher Konnektor



## Entwurfsmuster Channel und Pipe (Implementierungsmuster)

38 Softwaretechnologie (ST)

- ▶ **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.



Zweck: asynchrone Kommunikation zwischen zwei Komponenten durch Pufferung eines beliebig großen Datenstromes mit Hilfe eines Puffers buffer

- Kombination von push- und pull-Methoden (Iterator- und Ablagemethoden)
- ggf. mit Abfrage, ob noch freier Platz in der Ablage vorhanden
- ggf. mit Abfrage, ob noch Daten im Kanal vorhanden

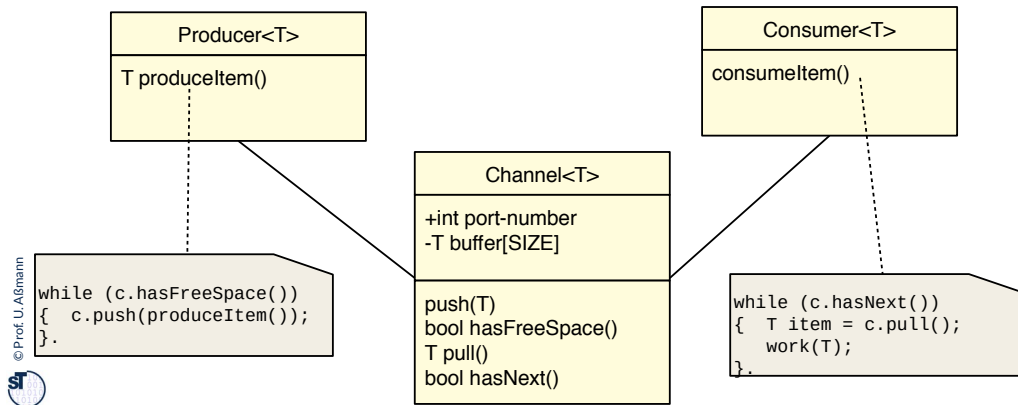


- ▶ Channels (pipes) kommen in vielen Sprachen als Konstrukte vor
- ▶ Linux versteht jeden Prozess mit 3 Streams (stdin, stdout, stderr), zwischen denen Kanäle gezogen werden können.
- ▶ Textuelle Syntax:
  - **Shell-Skripte in Linux** bestehen oft aus Produzenten-Konsumenten-Teams, die mit Kanal kommunizieren (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, pull) und ">>" (push, 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 den Internet; sie bilden Kanäle zur Kommunikation zwischen Prozessen (Producer-Consumer-Muster)



Viele Prozesse in Softwaresystemen benötigen den Austausch von unendlichen Mengen von Daten.

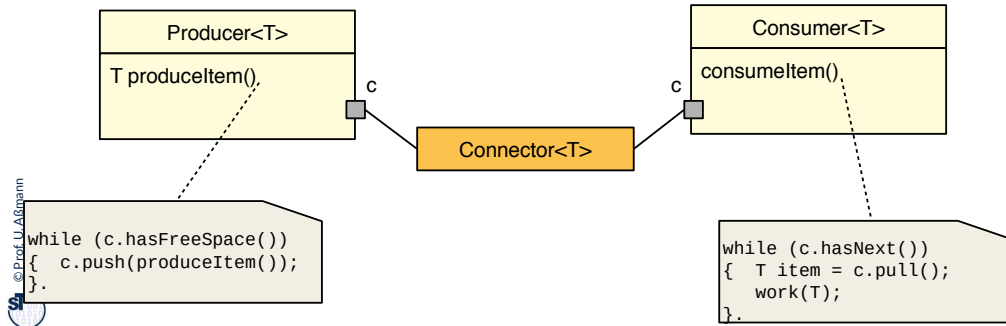
Wie repräsentiert man potentiell unendliche Mengen?

## Konnektoren als Verallgemeinerung von Kanälen

### Wdh.:

**Def.:** Eine **Konnektorklasse** ist eine Teamklasse, 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



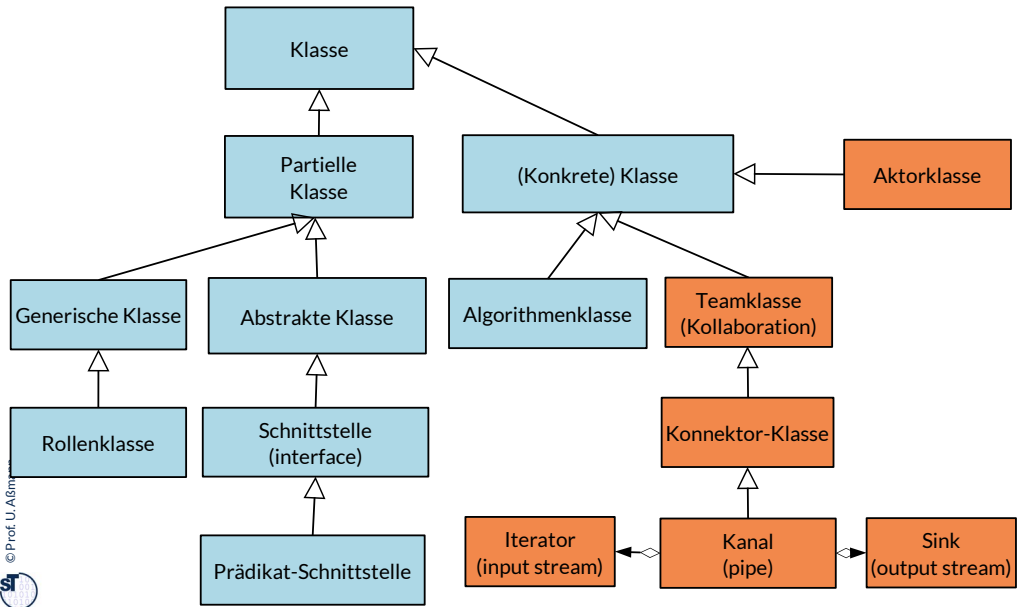
Konnektoren müssen nicht bilateral sein, sondern können n (fix) Partner verbinden

**Trenne Konnektoren von Anwendungsklassen ab, um die Wiederverwendung zu erhöhen**

- ▶ Konnektoren gehören zum olympischen Ring "Infrastruktur"



## Q2: Begriffshierarchie von Klassen (Erweiterung)



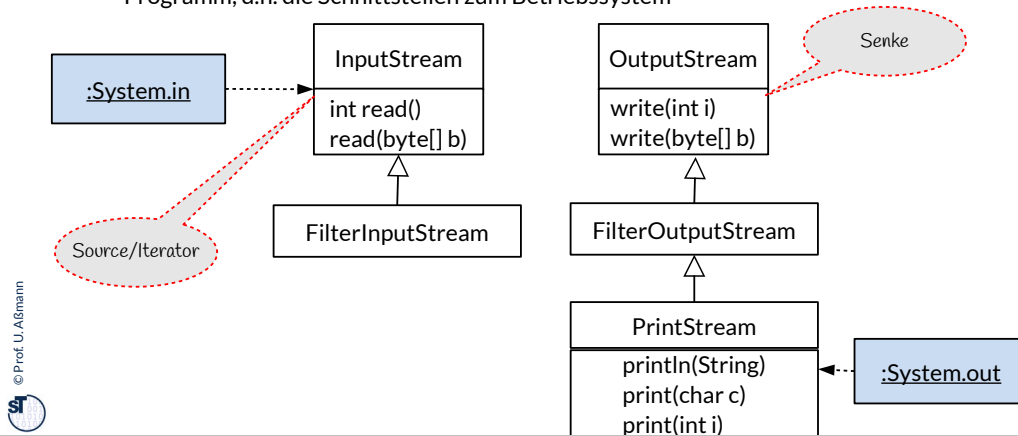
## 23.4 Channels und Streams sind überall: Strombasierte Ein- und Ausgabe (Input/Output) und persistente Datenhaltung

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



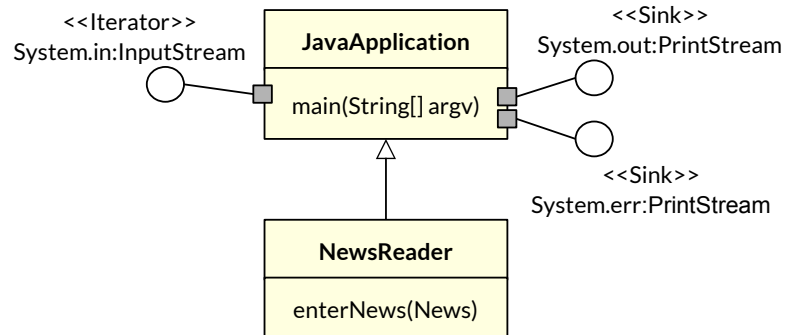
## 23.4.1 Ein- und Ausgabe in Java mit Strömen und Senken

- ▶ 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

- ▶ Ein Programm in Java hat 3 Standard-Ströme in, out, err
  - 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 (Siehe Vorlesung "Datenbanken")
    - 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;
}
```



```
import java.io.*;

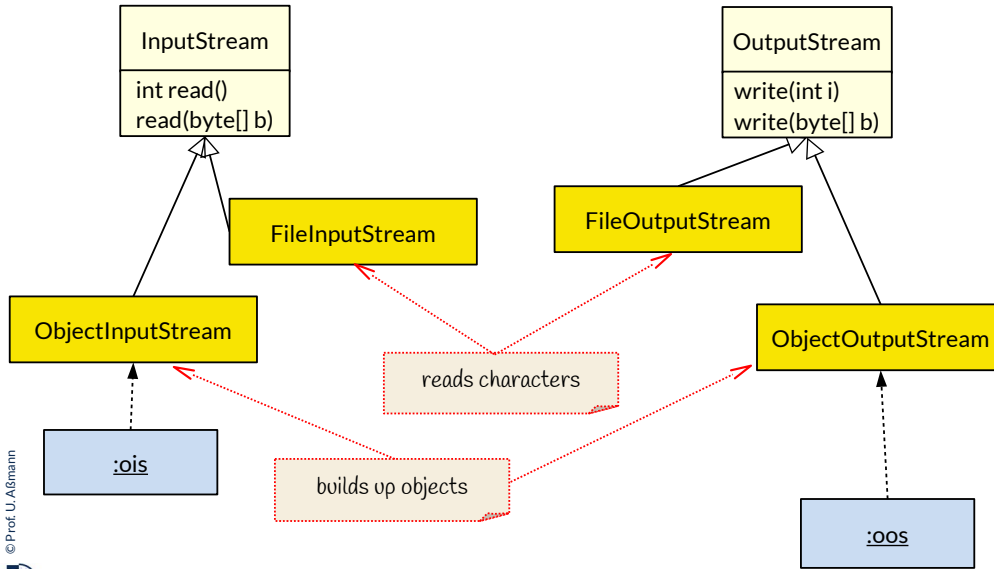
class XClass implements Serializable {
    private int x;
    public XClass (int x) {
        this.x = x;
    }
}

...
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
...
```

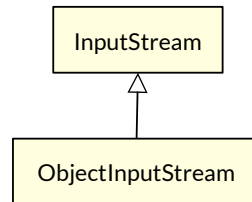


# Verschiedene Arten von Input und Output Streams im JDK



## ObjectInputStream aus dem JDK

- ▶ 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



## Objekt-Serialisierung: Einlesen mit `ObjectInputStream` aufsetzend auf `FileInputStream`

```
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 ein Team von einer Kollaboration?
- ▶ Was unterscheidet einen Iterator von einer Senke?
- ▶ Was ist der Unterschied zwischen einer fixen und einer unbestimmt großen Datenstruktur?
- ▶ 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?
- ▶ Wann macht es Sinn, die Kollaboration eines Teams durch ein Objekt, den Konnektor, zu kapseln?
- ▶ Warum ist ein Kanal ein spezieller Konnektor?
- ▶ Was ist besonders genial am UNIX/Linux Ein-/Ausgabesystem?
- ▶ Einige Folien stammen aus den Vorlesungsfolien zur Vorlesung Softwaretechnologie von © Prof. H. Hussmann. Used by permission.

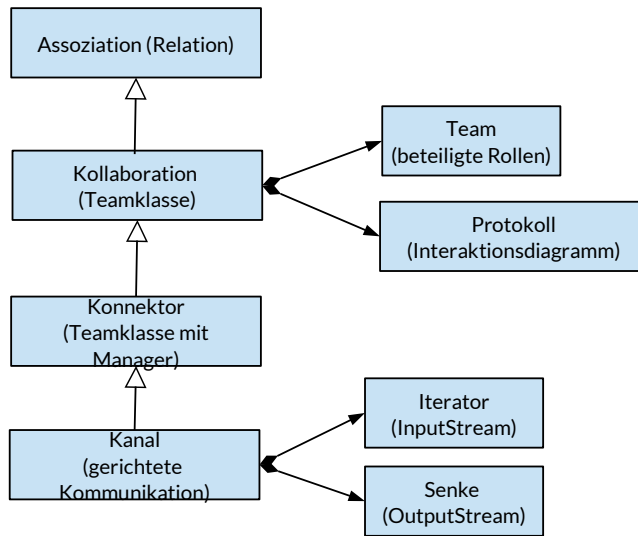




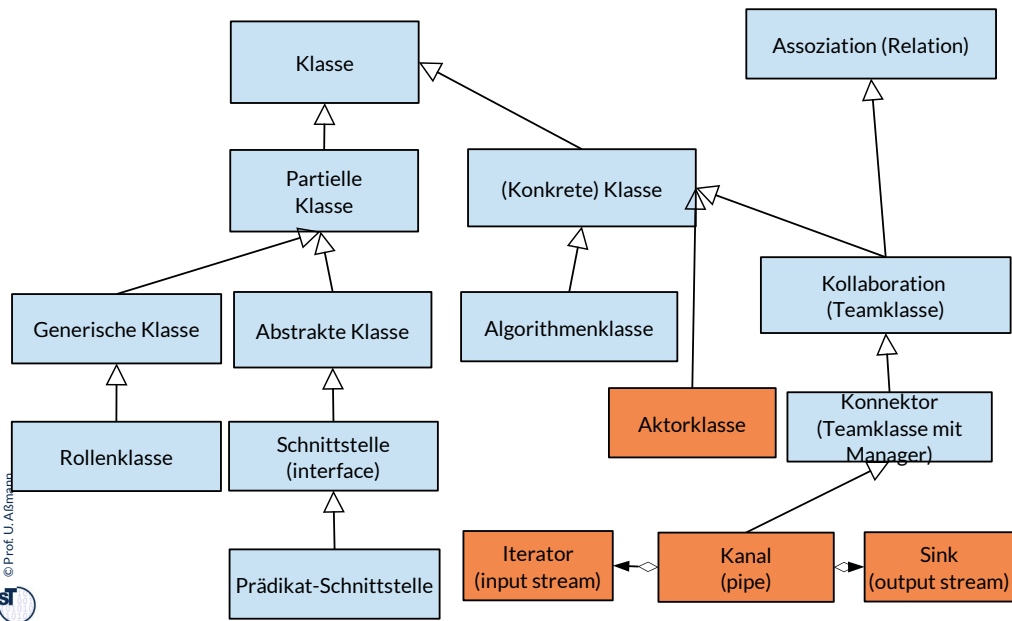
# Anhang



# Relationale Klassen (Konnektoren)



## Q2: Begriffshierarchie von Klassen (Erweiterung)



“Jetzt geht's rund”:

Was bedeuten alle diese Begriffe?

Achtung: Sie sollten nicht bis zur Klausur warten, die Bedeutung erklären zu können!

## Iterator-Implementierungsmuster in modernen Sprachen

- ▶ 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();..
```

