Methodische und Praktische Grundlagen der Informatik (MPGI 3)

WS 2008/09

Softwaretechnik

Steffen Helke

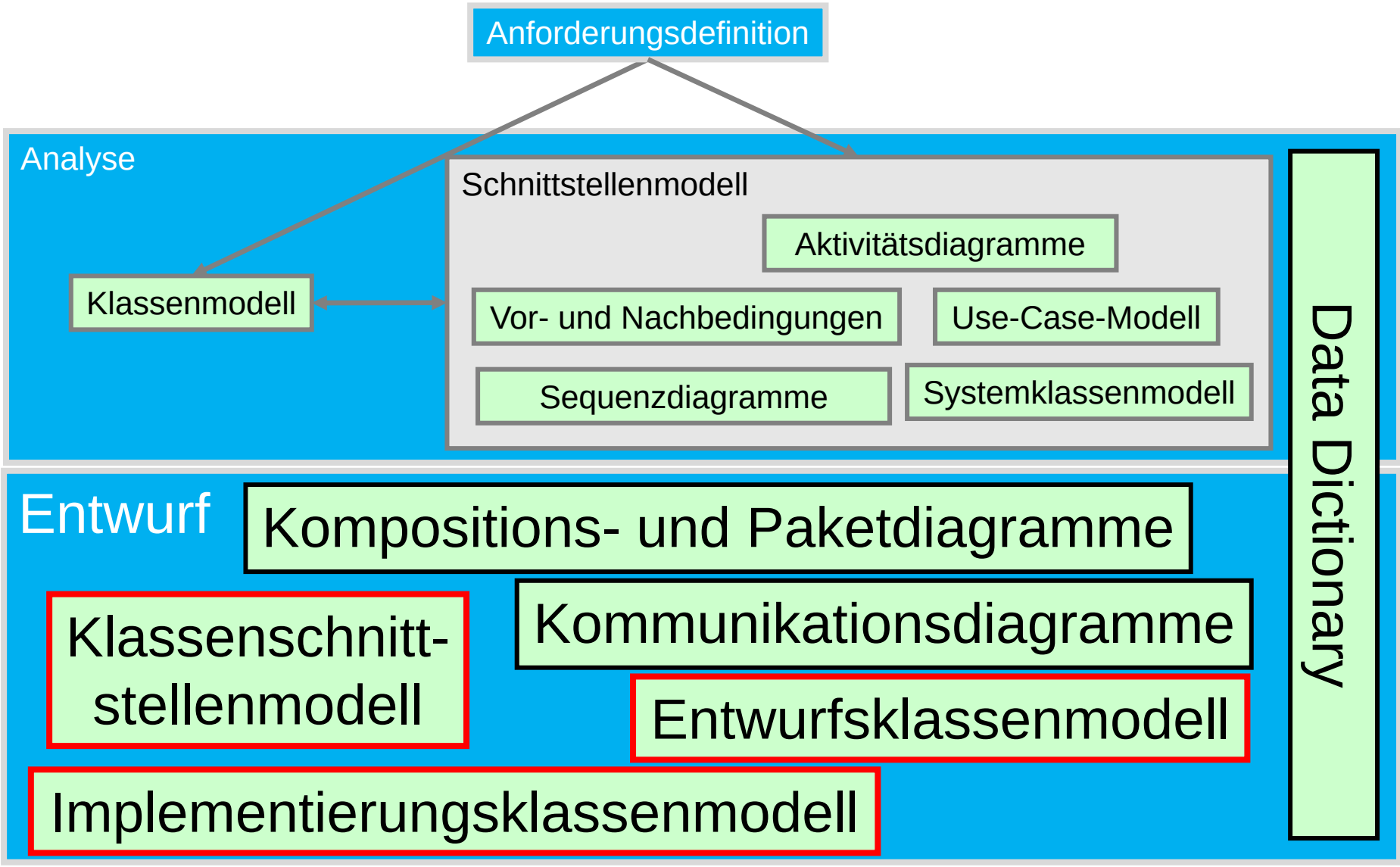
Andreas Mertgen (Organisation)

Rojahn Ahmadi, Georgy Dobrev, Daniel Gómez Esperón, Simon Rauterberg, Jennifer Ulrich (Tutoren)

Was machen wir heute?

- **Wiederholung**
  - Ableiten von Klassenschnittstellen
- **Implementierungskonzepte**
  - **Assoziationen**
  - **Vererbung**

Rückblick: Vom Entwurf zum Code



Abgrenzung der Klassendiagramme

1. **Analyseklassenmodell** (Systemklassenmodell)
   - Modell am Ende der Analysephase
2. **Entwurfsklassenmodell** (verfeinertes Systemklassenmodell)
   - Ableitung während der Entwicklung der Kommunikationsdiagramme
   - Attribute und Assoziationen (ungerichtet)
3. **Implementierungsklassenmodell** (Klassenschnittstellenmodell)
   - Methoden und Sichtbarkeiten für Attribute und Methoden
   - Hinzufügen von Referenzen (gerichtete Assoziationen)

Ableiten des Implementierungsklassenmodells

1. Vervollständigen der Attribute

- Prüfen, ob neue Attribute erforderlich sind

2. Ableiten von Referenzierungsattributen

- Kodieren von Assoziationen

3. Einführen von Methoden

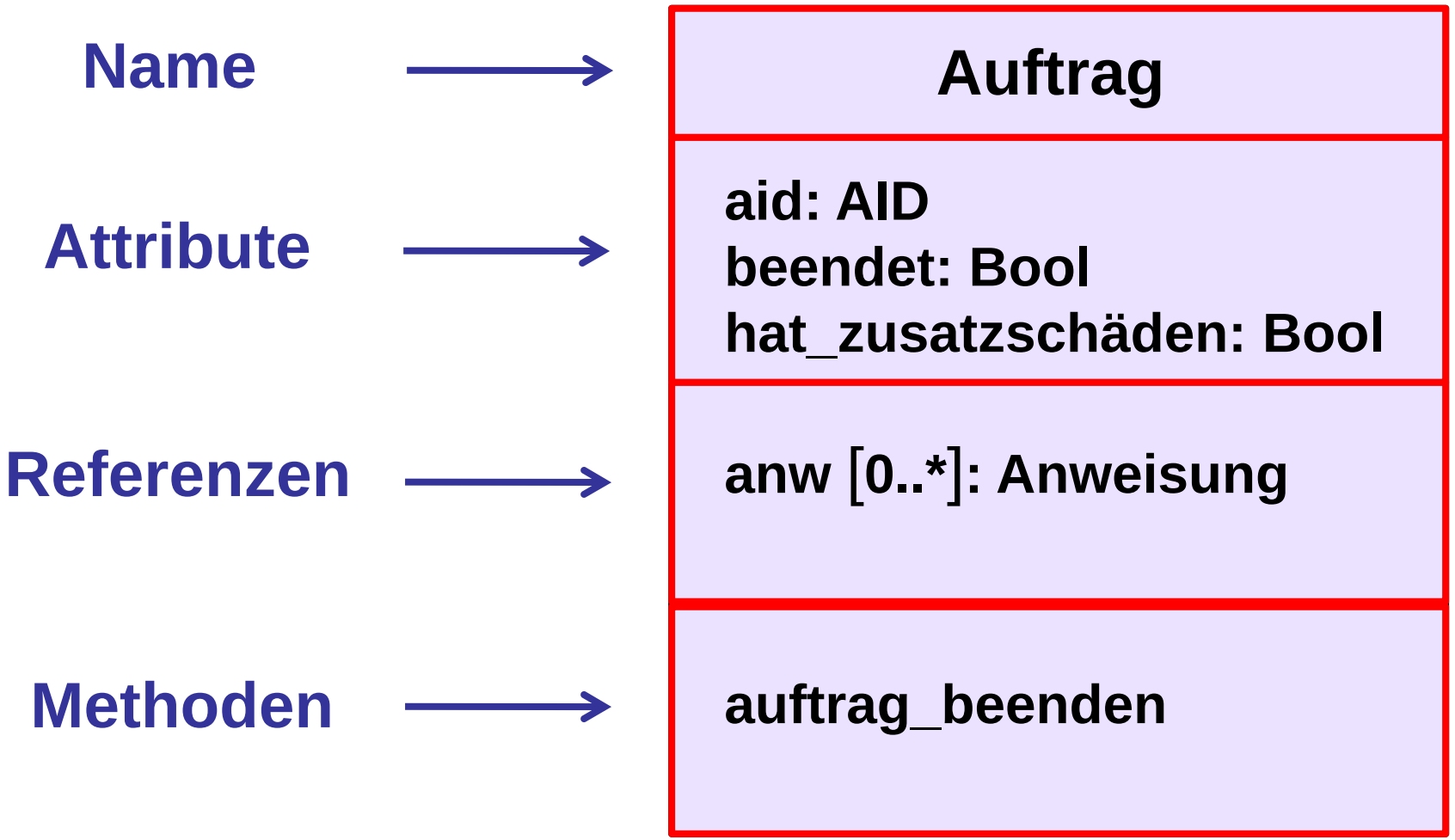
- für Systemoperationen und systeminterne Methoden

4. Vergeben von Sichtbarkeiten

- für Attribute und Methoden

Verfeinerung von Klassen

Beispiel: Klasse Auftrag



Ableiten von Referenzierungsattributen

1. Identifizieren von Klient-Klassen

- Überprüfen der Kommunikationsdiagramme
- Finden aller Server-Klassen zu einem Klient

2. Untersuchen der Referenz-Beziehungen

- **Referenzierungsdauer** durch den Klienten
- **Sichtbarkeit** des Servers für seine Klienten
- **Änderbarkeit** der Referenzierung auf den Server
- **Multiplizität** des Servers aus Sicht des Klienten

3. Einführen von Referenzierungsattributen

- Klient-Klassen erhalten Attribute mit Referenzen auf Server, annotiert mit den gewonnen Informationen

Einführen von erforderlichen Methoden

1. Implementierung von Systemoperationen

- Prüfen, welche Control-Klassen mit der Boundary-Klasse eines Akteurs verbunden sind
- Einführen von **Methoden für Systemoperationen in dafür zuständigen Control-Klassen**

2. Auffinden systeminterner Methoden

- Prüfen aller Nachrichtenflüsse im Kommunikations-Diagramm
- Einführen von **Methoden in Server-Klassen zur Bearbeitung von** systeminternen **Anfragen**

Anfragen an Kollektionen von Objekten

Kodierungsvarianten für Referenzen auf Kollektionen

1. Referenzierungsattribut in Klient-Klasse

- Abbildung des Typs auf eine Bibliotheksklasse, z.B. Bag, Set oder Seq
- bei wenig komplexen Anfragen an die Kollektion
- Anfrage wird durch Methode in der Klient-Klasse umgesetzt

2. Eigenständige Server-Klasse für Kollektion

- bei komplexen Anfragen
- Anfrage wird durch Methode in der Server-Klasse umgesetzt

Vergeben von Sichtbarkeiten

1. Attribute

- als **privat** deklarieren
- Lese- und Schreibzugriff mit **get** und **set-Methoden**

2. Methoden

- Methoden zur Bearbeitung von Anfragen an Server-Objekte müssen für Klient-Objekte **öffentlich** sein
- Löst ein Klient-Objekt eine Aufgabe selbst, so ist die Methode dafür als **privat** zu markieren

Wie kommt man zu einer guten Implementierung?

- **Einsetzen von Entwurfsmustern**
- **Verwenden von Case-Tools**
- **Umsetzen von etablierten Implementierungsstrategien**

Verwendung von Entwurfsmustern

„Katalog“ der „Gang of Four“

Gamma, Helm, Johnson, Vlissides

„Design Patterns – Elements of Reusable Object- Oriented Software“

➡ **Einteilung in …**

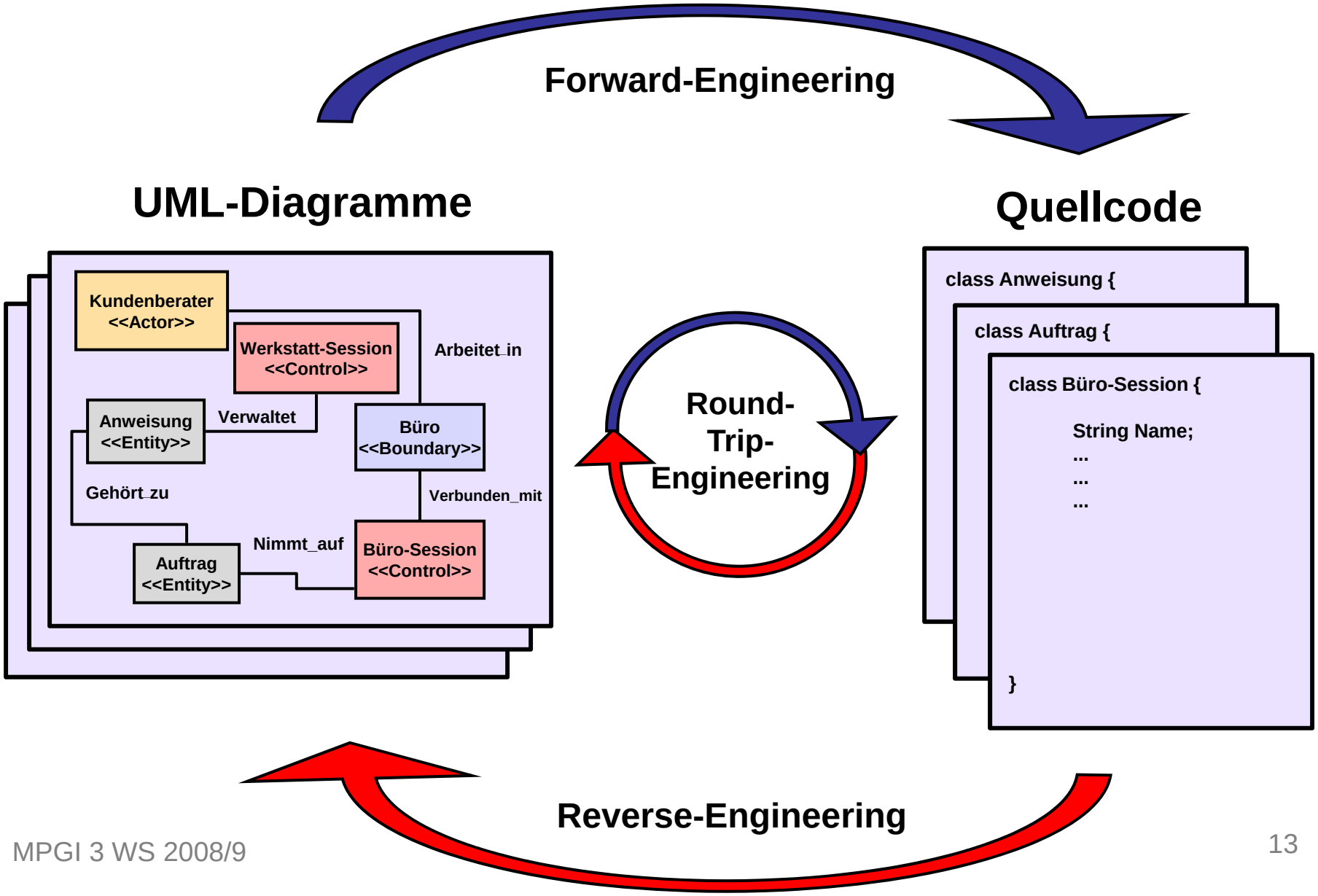
- Erzeugungsmuster
- Strukturmuster
- Verhaltensmuster

- *Abstract Factory*
- *Builder*
- *Factory Method*
- *Prototype*
- *Singleton*

- *Adapter*
- *Bridge*
- ***Composite***
- *Decorator*
- *Facade*
- *Flyweight*
- *Proxy*

- *Chain of Responsibility*
- *Command*
- *Interpreter*
- *Iterator*
- *Mediator*
- *Memento*
- ***Observer***
- *State*
- *Strategy*
- *Template Method*
- *Visitor*

Entwicklung durch den Einsatz von Case-Tools



Implementierungsstrategien für UML-Modelle

- **Nicht alle UML-Konstrukte können in einer Programmiersprache direkt umgesetzt werden.**
- **Beispiele:**
  - Direkt sind keine Assoziationen darstellbar.
  - Nicht alle OO-Konzepte werden immer unterstützt (z.B. keine Mehrfachvererbung in Java).

➡ Martin Fowler: Analysis Patterns, Addision Wesley 1997

Hinweis: Die Darstellung der folgenden Folien basiert auf Ideen zur VL von Gregor Engels (Paderborn)

Implementierung von Assoziationen

Beispiel: Beziehung zwischen Auftrag und Anweisung

Die Assoziation enthält folgende Informationen:

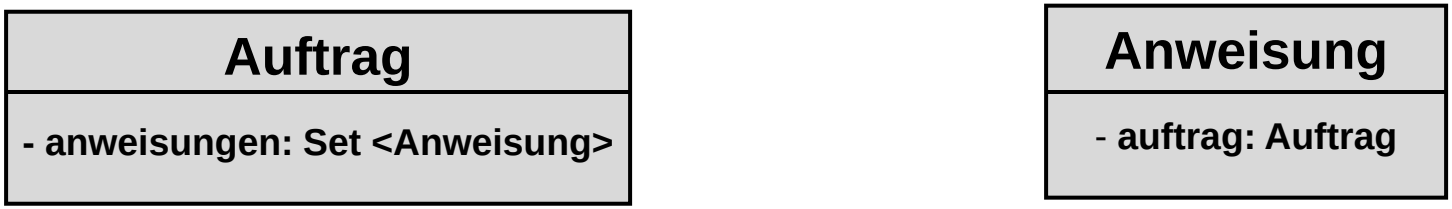
- Ein Auftrag kann (beliebig viele) Anweisungen haben.
- Ein Auftrag kennt diese Anweisungen.
- Eine Anweisung gehört immer zu genau einem Auftrag.
- Eine Anweisung kennt den Auftrag zu dem sie gehört.
- Wenn eine Anweisung zu einem Auftrag gehört, so ist sie auch nur in diesem Auftrag enthalten.

Implementierung von Assoziationen in Java

- **Ausdrucksmittel**
  - Attribute
  - Operationen
  - neue Klassen
- **Implementierungsvarianten**
  - Gegenseitige Referenzierung
  - Einfache Referenzierung
  - Assoziationsklassen

Implementierung von Assoziationen in Java

Gegenseitige Referenzierung (Attribute)



- Anweisungsobjekte haben eine private Objektreferenz auf Auftrag
  - Assoziation mit Multiplizität 1 durch ein Referenzierungsattribut darstellbar
- Auftragsobjekte haben eine private Menge von Objektreferenzen auf Anweisungen
  - Assoziation mit Multiplizität * durch ein als Set typisiertes Referenzierungsattribut darstellbar
  - „ordered“ entsprechend als Liste

Implementierung von Assoziationen in Java

Gegenseitige Referenzierung (Zugriff)

- Zugriffmethoden zum Abfragen und Ändern der Referenzen
- Methoden stellen Einhaltung der Multiplizitäten sicher

Referenzielle Integrität muss bei gegenseitiger Referenzierung garantiert werden!

Implementierung von Assoziationen in Java

Referentielle Integrität in setAuftrag

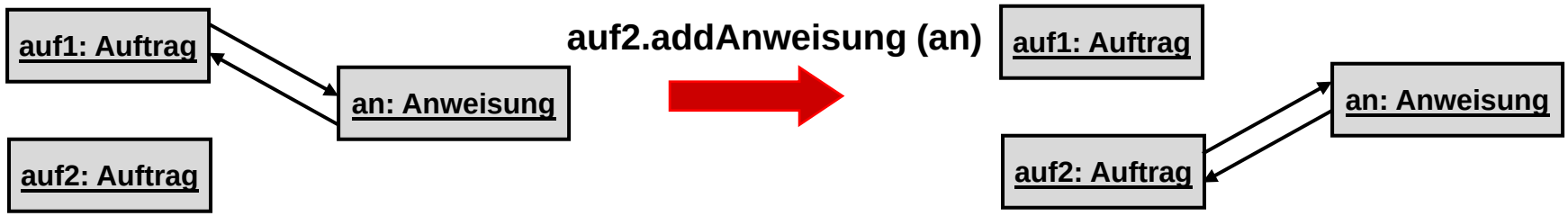
an.setAuftrag (auf2) enthält drei Schritte

1. **auf1** benachrichtigen, dass **an** entfernt wird
2. Attribut **auftrag** von **an** neu setzen
3. **auf2** benachrichtigen, dass **an** neu dazukommt

So wird garantiert, dass eine Anweisung immer nur zu genau einem Auftrag gehört!

Implementierung von Assoziationen in Java

Referentielle Integrität in addAnweisung



auf2.addAnweisung (an) enthält zwei Schritte

1. **an** zur Menge in **auf2** hinzufügen
2. **an** über Änderung informieren

an.setAuftrag (auf2) ruft **auf2.addAnweisung (an)** auf, und
auf2.addAnweisung (an) ruft **an.setAuftrag (auf2)** auf

➡ Bei naiver Programmierung, Gefahr von unendlichen Schleifen!

Implementierung von Assoziationen in Java

Korrekte Implementierung für setAuftrag


```
setAuftrag (Auftrag newAuftrag)
 { if (this.auftrag != newAuftrag) // nur für neue Aufträge setzen
   { if (this.auftrag != null)          // alter Auftrag noch nicht gelöscht
        { Auftrag oldAuftrag = this.auftrag; // Zwischenspeichern
          this.auftrag = null;           // Abruchkriterium
          oldAuftrag.deleteAnweisung(this); // Austragen
         }
      this.auftrag = newAuftrag; // neuen Auftrag setzen
      newAuftrag.addAnweisung(this); // Eintragen
   }
 }
```

Implementierung von Assoziationen in Java
Gegenseitige Referenzierung (Bewertung)

Vorteile

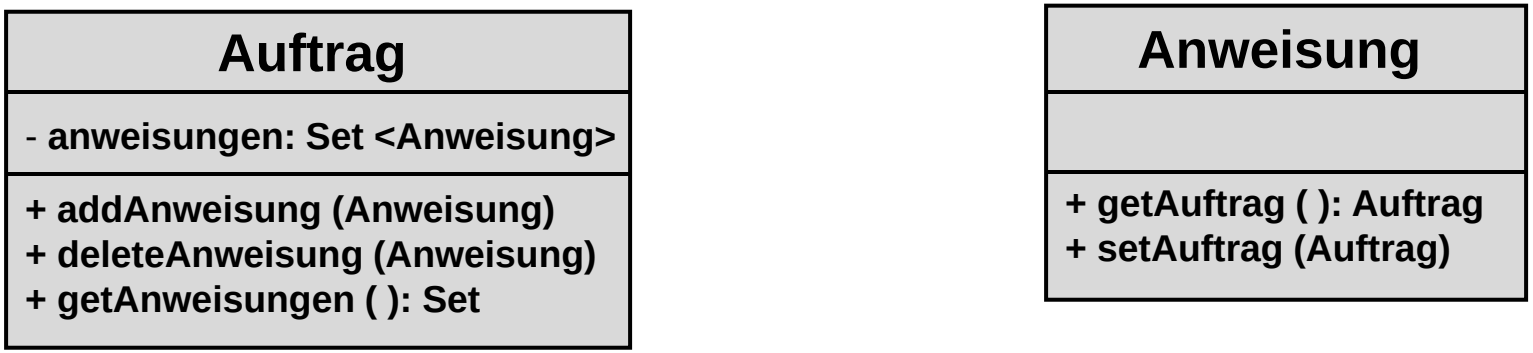
- Schnelle Navigation in beiden Richtungen
- Es existieren Standardimplementierungen

Nachteile

- Redundante Informationshaltung
- Updates kosten Zeit und sind nicht ganz leicht zu implementieren

Implementierung von Assoziationen in Java

Einfache Referenzierung



- direkte Navigation nur in einer Richtung
- indirekte Navigation in die andere Richtung sehr aufwendig, z.B.
  - Anfrage über Control-Objekt, das alle Auftragsobjekte kennt.
  - Durchsuchen aller Auftragsobjekte nötig

Implementierung von Assoziationen in Java
Einfache Referenzierung (Bewertung)

Vorteile

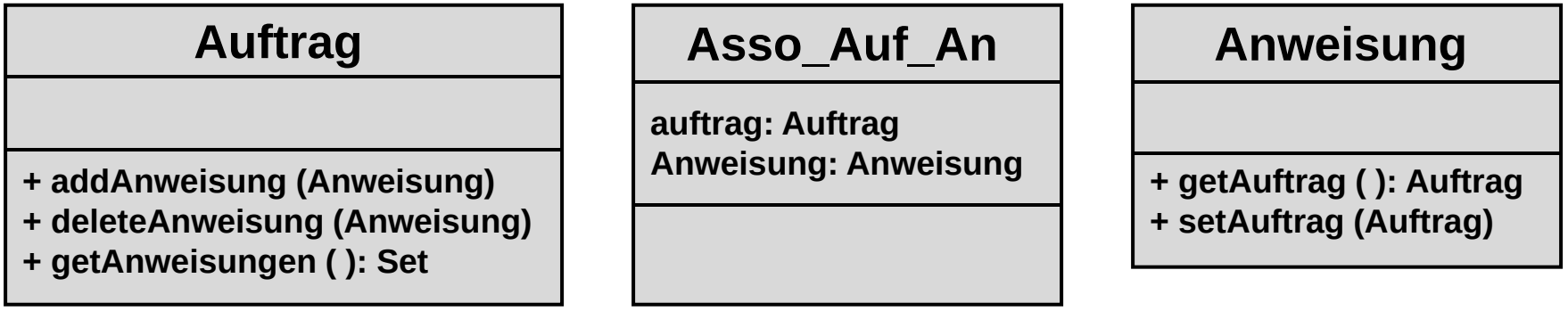
- schnelle Navigation in eine Richtung
- keine redundante Datenhaltung
- einfache und schnelle Updates

Nachteile

- Navigation in der Gegenrichtung ist sehr aufwendig

Implementierung von Assoziationen in Java

Assoziationsklassen



- Für jeden neuen Link zwischen Auftrag und Anweisung wird ein neues Objekt der Assoziationsklasse erzeugt.
- zentrale Verwaltung aller Assoziationsobjekte
- Navigation erfolgt über Selektion aller Assoziationsobjekte des anfragenden Elements

Implementierung von Assoziationen in Java
Assoziationsklassen (Bewertung)

Vorteile

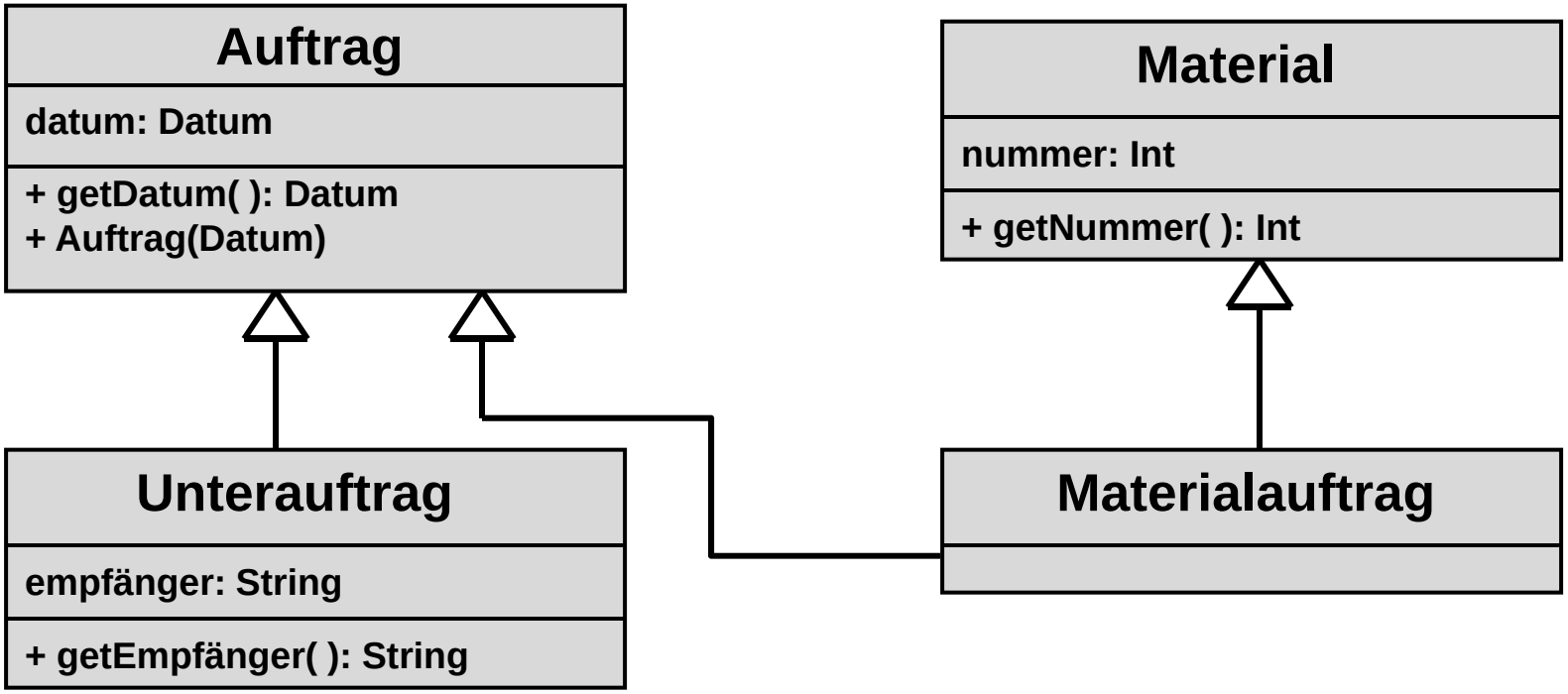
- automatische Konsistenzhaltung
- kein Platzbedarf für nicht existente Beziehungen
- keine Modifikation in den Datenstrukturen der beteiligten Objekte

Nachteile

- Navigation in beiden Richtungen aufwendig
- nur bei wenig Objektverbindungen sinnvoll einsetzbar

Implementierung von Vererbung

Beispiel: Unterauftrag und Materialauftrag



Eigenschaften von Materialauftrag

- besitzt alle Attribute/Operationen von Auftrag und Material
- kann als Auftrag oder Material auftreten
- kann neue Eigenschaften definieren (**Namenskonflikte**)

Probleme beim Implementieren von Vererbung

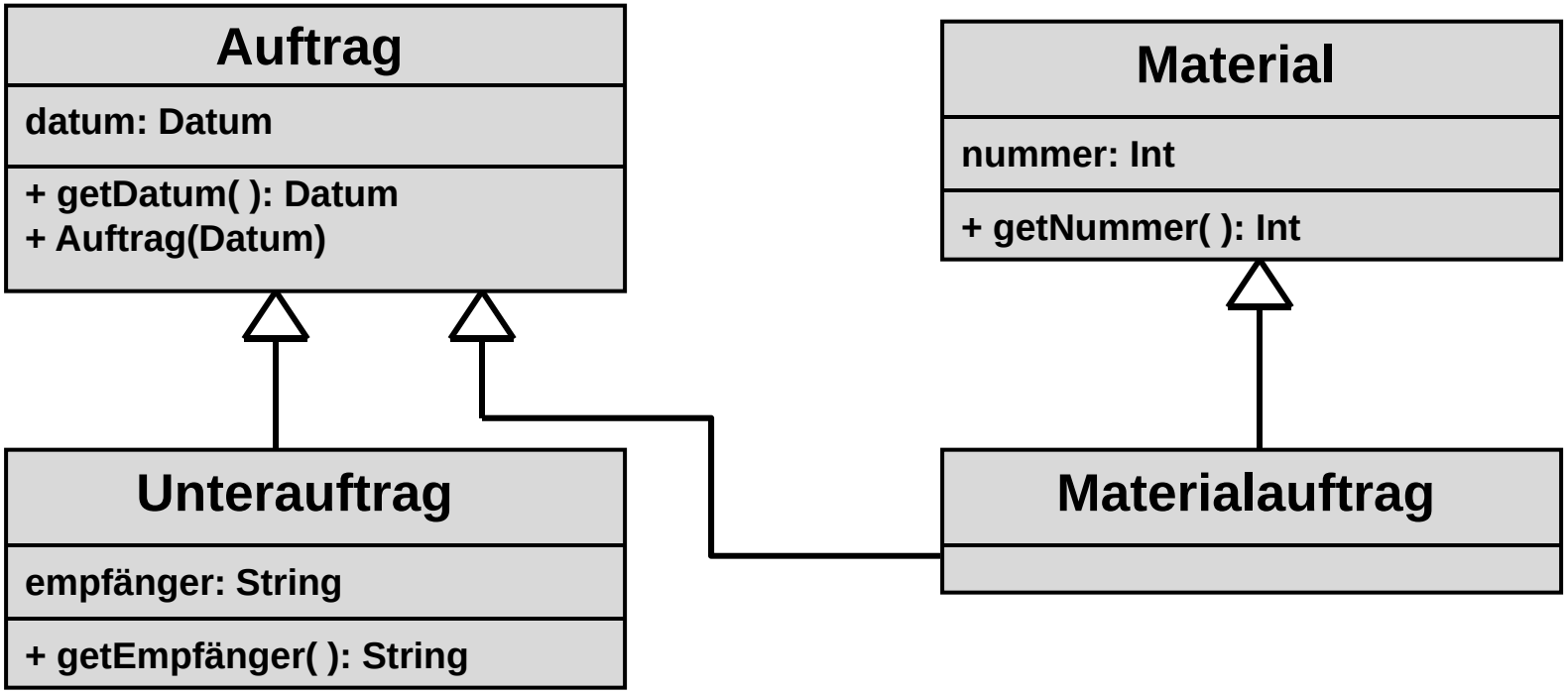
Mächtigkeit vom Programmiersprachen

- gar keine Vererbung (z.B. Pascal)
- bieten nur Einfachvererbung (z.B. Simula, Smalltalk)
- teilweise Mehrfachvererbung (Java)
- volle Mehrfachvererbung (C++)

Manchmal will man auch aus Effizienzgründen Vererbung vermeiden!

Implementierung von Vererbung

Beispiel: Strategien

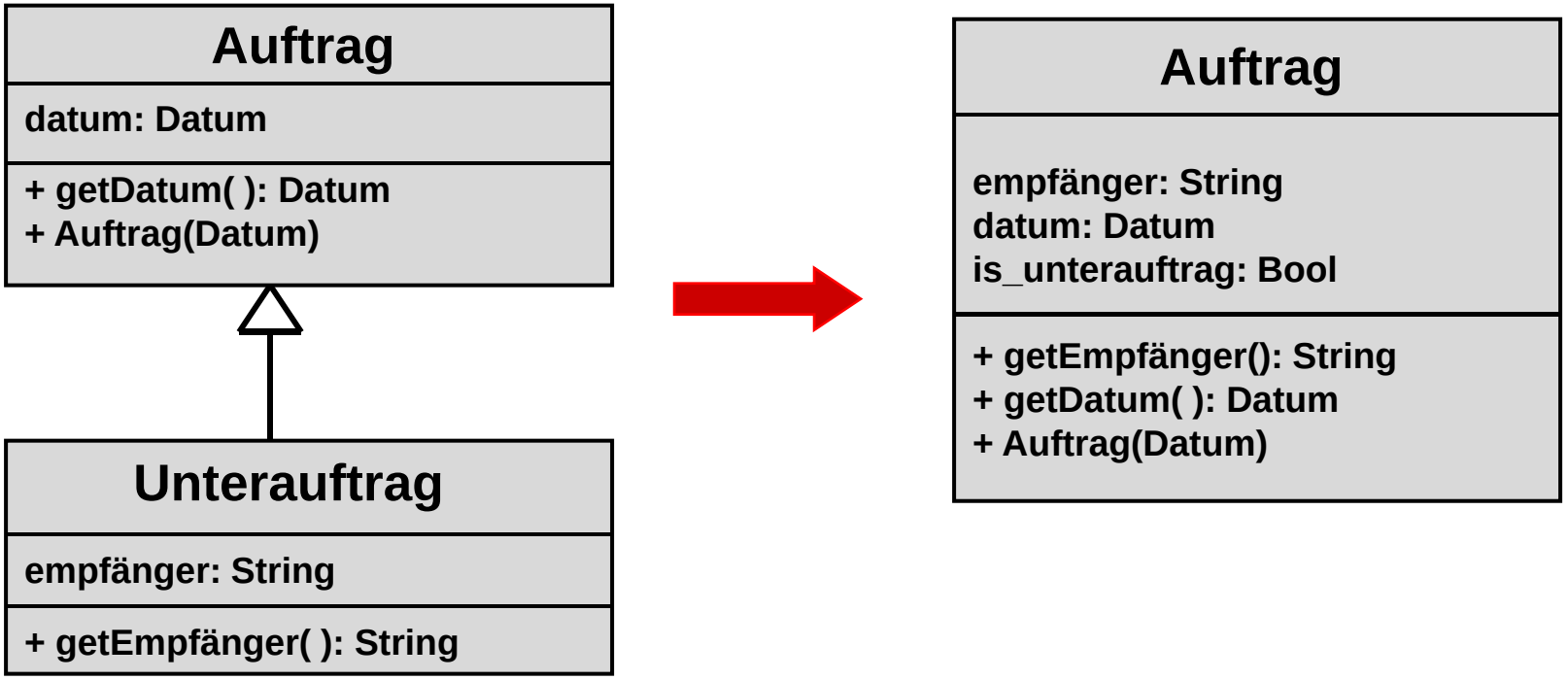


Ansätze zum Implementieren von Vererbung

- Flags
- Delegation (aufwärts und abwärts)
- Interfaces (+ Delegation)

Implementierung von Vererbung

Beispiel: mit Hilfe von Flags



- Superklasse enthält alle Informationen der Unterklasse
- Flag entscheidet über konkreten Typ der Klasse
- Operationen wie getEmpfänger sind nur beim richtigen Typ anwendbar
- Verhalten der Polymorphie muss selbst implementiert werden

Implementierung von Vererbung
mit Hilfe von Flags (Bewertung)

Vorteile

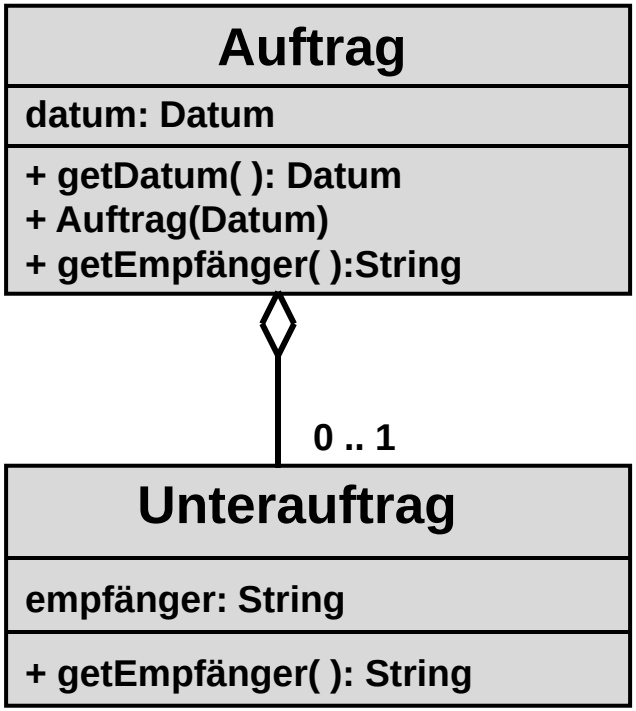
- man kommt ohne Vererbungsmechanismen aus
- intuitives Konzept

Nachteile

- aufwendige Implementierung
- schlecht wartbare Klassen (groß und unübersichtlich)
- Typfehler fallen erst zur Laufzeit auf
- Speicherverschwendung, da Objekte der Oberklasse eigentlich keinen Platz für Felder der Unterklasse brauchen

Implementierung von Vererbung

Beispiel: durch Abwärtsdelegation



- Superklasse deklariert Operationen der Unterklasse
- Wenn die Superklasse als Unterklasse fungiert, enthält sie das Objekt von Unterauftrag, aber gibt es nicht raus
- Methodenaufrufe der Unterklasse werden per Delegation an das Objekt Unterauftrag versendet

Implementierung von Vererbung
durch Abwärtsdelegation (Bewertung)

Vorteile

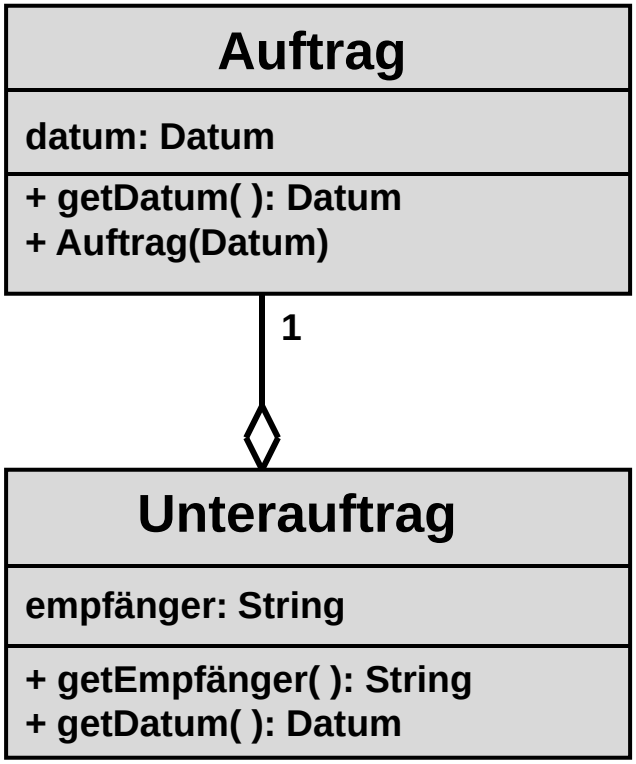
- man kommt ohne Vererbungsmechanismen aus
- Implementierung der Unterklassen findet in den Unterklassen statt
- Referenz auf die Unterklasse kann als Flag fungieren

Nachteile

- Superklasse muss alle Methoden der Unterklasse deklarieren
- Typfehler fallen erst zur Laufzeit auf

Implementierung von Vererbung

Beispiel: durch Aufwärtsdelegation



- Unterklasse deklariert Operationen der Superklasse
- Die Unterklasse besitzt ein Objekt der Oberklasse, das für andere Objekte nicht sichtbar ist
- Methodenaufrufe für die Superklasse werden per Delegation an das Objekt Auftrag delegiert

Implementierung von Vererbung
durch Aufwärtsdelegation (Bewertung)

Vorteile

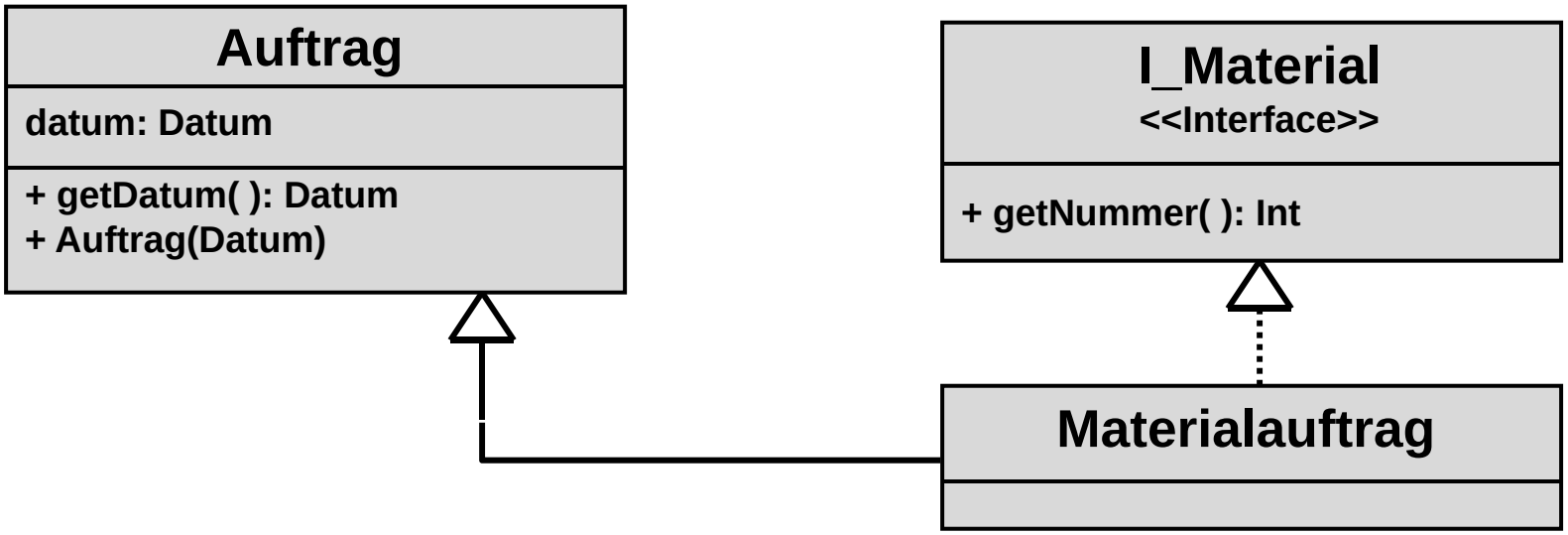
- man kommt ohne Vererbungsmechanismen aus
- Implementierung der Unterklassen findet in den Unterklassen statt
- Superklasse muss nicht geändert werden

Nachteile

- Unterklasse muss alle Methoden der Superklasse deklarieren und Aufrufe delegieren
- Polymorphie nicht möglich: Unterklassen können nicht in der Rolle der Superklassen auftreten

Implementierung von Vererbung

Beispiel: durch Einfachvererbung und Interfaces



- Java hat nur Einfachvererbung
- andere Vererbungen nur durch Implementierung von Interfaces möglich
- die Implementierung der Methode **getNummer** muss in der Klasse Materialauftrag erfolgen

Implementierung von Vererbung
durch Einfachvererbung und Interfaces (Bewertung)

Vorteile

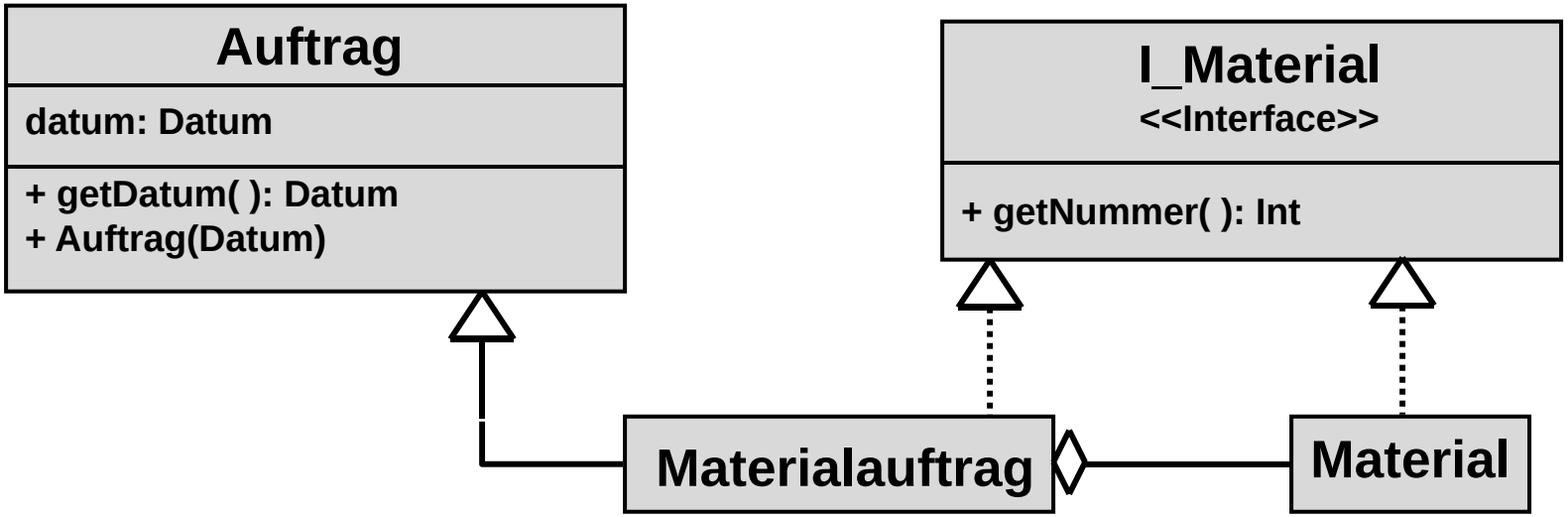
- eventuelle Konflikte können in der Implementierung von Materialauftrag gelöst werden
- Klassen die als Superklassen in einer Mehrfachvererbung auftauchen sollten als Interface dargestellt werden (sehr übersichtliches Vorgehen)

Nachteile

- Implementierungen von Methoden können automatisch nur von einer Superklasse geerbt werden

Implementierung von Vererbung

Beispiel: durch Interfaces und Delegation



- Materialauftrag hat ein privates Objekt der Klasse Material
- die Methode **getNummer** ist in der Klasse Material implementiert und wird per Delegation von der Klasse Materialauftrag erreicht

Implementierung von Vererbung
durch Interfaces und Delegation (Bewertung)

Vorteile

- Konflikte bei Mehrfachvererbung können gelöst werden
- Kein redundanter Code
- alle Vererbungsfeatures können unterstützt werden
- die Superklasse muss nicht modifiziert werden

Nachteile

- Unterklasse muss alle Methoden des Interfaces mit einem Delegationscode kodieren

Achtung Raumänderung

Weihnachtsvorlesung

am

18.12.08

im

H 2013