

Grundlagen der Programmierung

Dr. Christian Herzog
Technische Universität München

Wintersemester 2002/2003

Kapitel 3: Klassen und Objekte

Überblick über Kapitel 3 der Vorlesung

- ❖ Klasse
 - Attribute, Operationen
- ❖ Instanz einer Klasse (Objekt vs. Klasse)
- ❖ Definition Instanzendiagramm, Klassendiagramm
 - Ein System ist eine Menge von kooperierenden Klassen oder Objekten
- ❖ Objekte und Klassen in Java
- ❖ Die Vererbungs-Beziehung
 - Beispiele von Vererbungen
- ❖ Vererbung in Java
- ❖ Kombination von Aggregation und Vererbung
- ❖ Das Kompositions-Muster

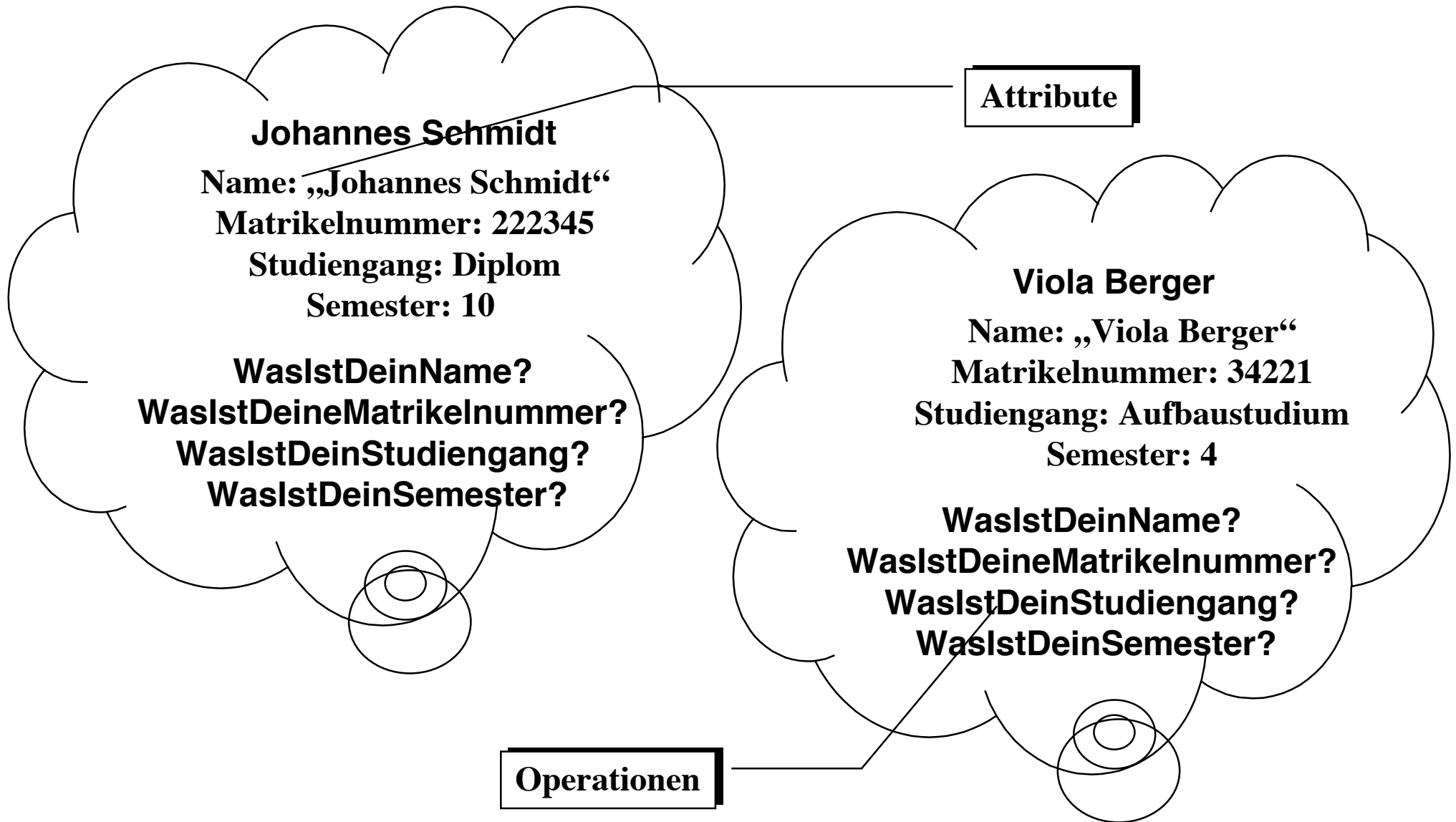
Unser Grundmodell der Modellierung

- ❖ Ein System besteht aus Subsystemen, die wieder aus Subsystemen bestehen, und diese dann letztendlich aus Gegenständen.
 - Diese Gegenstände bezeichnen wir auch als **Objekte**.
- ❖ Alle zu verarbeitende Informationen in einem System ist auf diese Objekte verteilt.
 - Die Verarbeitung von Information geschieht entweder innerhalb der Objekte oder durch Kommunikation von Nachrichten zwischen zwei Objekten.

Objekt

- ❖ **Definition Objekt:** Ein Objekt ist ein elementares Teilsystem. Es repräsentiert einen beliebigen Gegenstand in einem System.
- ❖ Ein Objekt ist durch seinen **Zustand** und seine **Funktionalität** gegeben.
- ❖ Zustand und Funktionalität setzen sich im allgemeinen aus Teilzuständen und einzelnen Operationen zusammen.
 - wir nennen die Teilzustände auch **Attribute**
 - wir nennen die einzelnen Operationen auch **Methoden** des Objektes
- ❖ Wir nennen die Operationen eines Objektes, die von anderen Objekten aufgerufen werden können, die **Schnittstelle** des Objektes.

Zwei Beispiels-Objekte



Attribute, Operationen, Merkmale

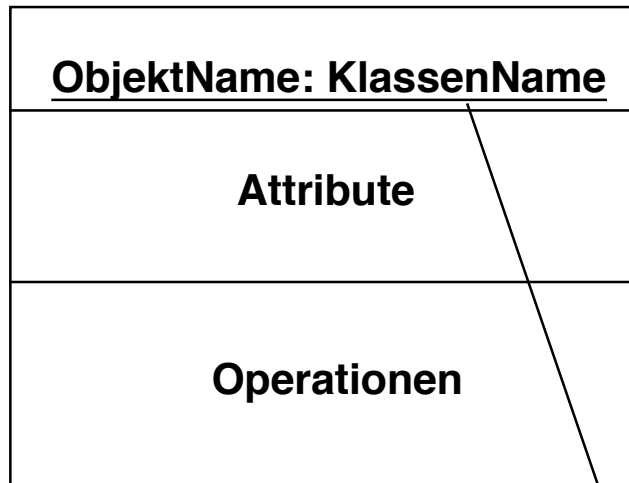
- ❖ Ein Objekt besitzt Attribute und Operationen
 - **Definition Attribut:** Messbare, durch Werte erfassbare Eigenschaft des Objektes.
 - **Definition Operation:** Eine Tätigkeit, die ein Objekt ausführen kann, um Berechnungen durchzuführen, Ereignisse auszulösen sowie Botschaften zu übermitteln.
- ❖ **Definition Merkmale:** Die zu einem Objekt gehörigen Attribute und Operationen.
- ❖ **Definition Schnittstelle:** Die Menge der Operationen eines Objektes, die von anderen Objekten aufgerufen werden können.



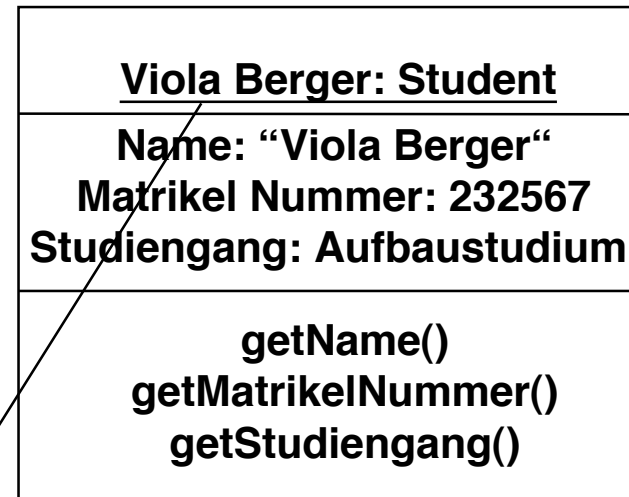
**Nicht alle
Operationen
müssen zur
Schnittstelle
gehören!**

Graphische Darstellung: Objekt

Allgemein:



Beispiel:



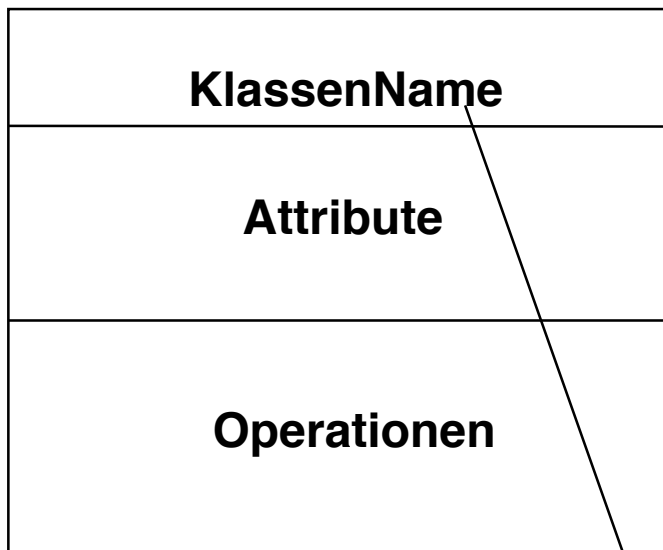
Die Unterstreichung ist wichtig!

Klasse und Instanz

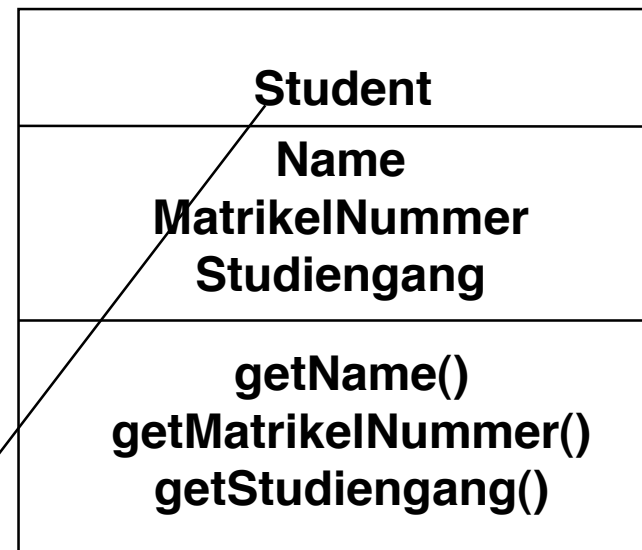
- ❖ Wir können Objekte mit gleichen Merkmalen zusammenfassen bzw. Objekte nach ihren Merkmalen *klassifizieren*:
- ❖ **Definition Klasse:**
 - Die Menge aller Objekte mit gleichen Merkmalen, d.h. mit gleichen Attributen und Operationen.
- ❖ **Definition Instanz:**
 - Ein Objekt ist eine Instanz einer Klasse K, wenn es Element der Menge aller Objekte der Klasse K ist.
- ❖ Künftig werden werden wir die Klasse weniger als Menge von Objekten auffassen sondern als *Beschreibung der Merkmale* ihrer Objekte.
 - Dabei wirkt sie wie eine Schablone zur Generierung (*Instantiierung*) von ihr zugeordneten Objekten (Instanzen).

Graphische Darstellung von Klassen

Allgemein:

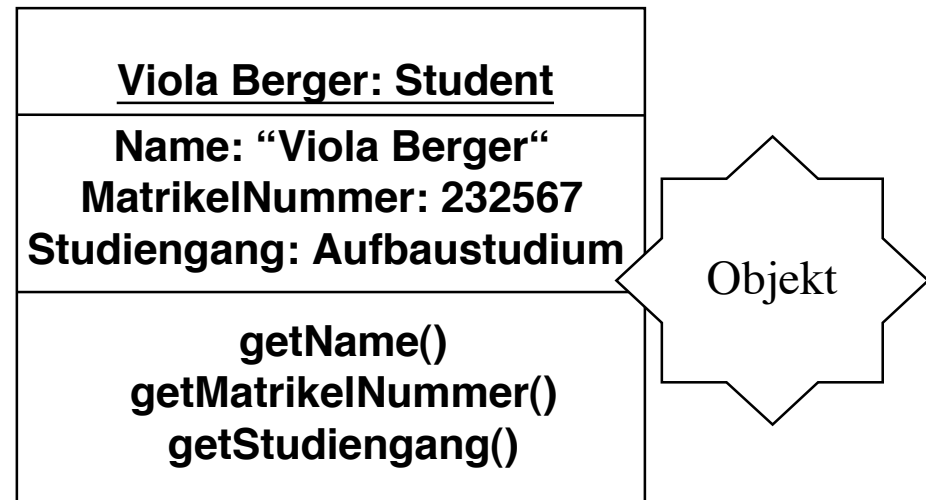
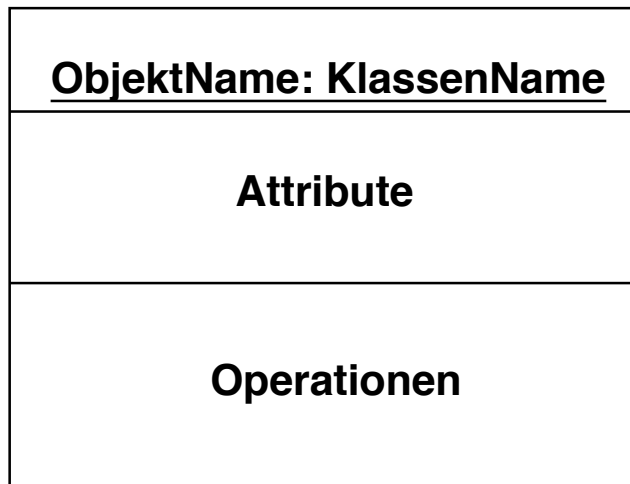
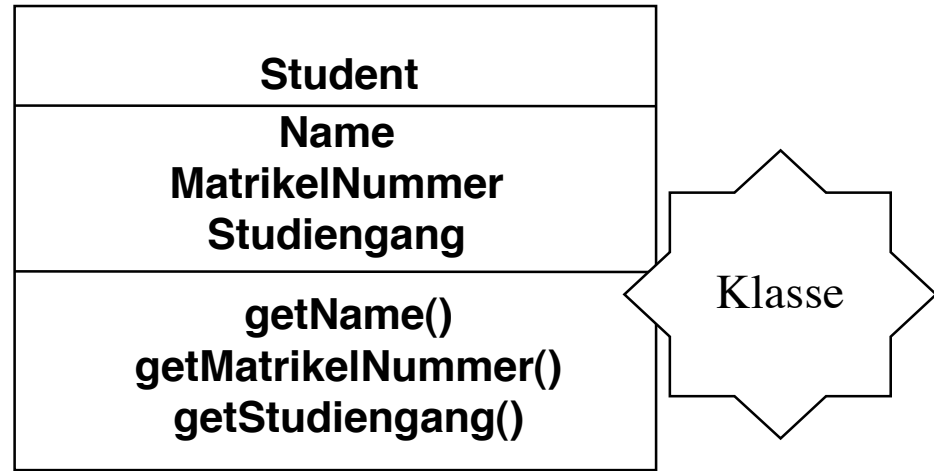
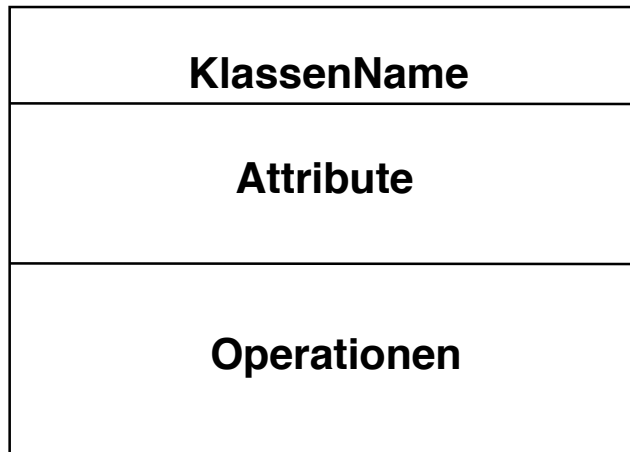


Beispiel:



Hier wird nicht unterstrichen!

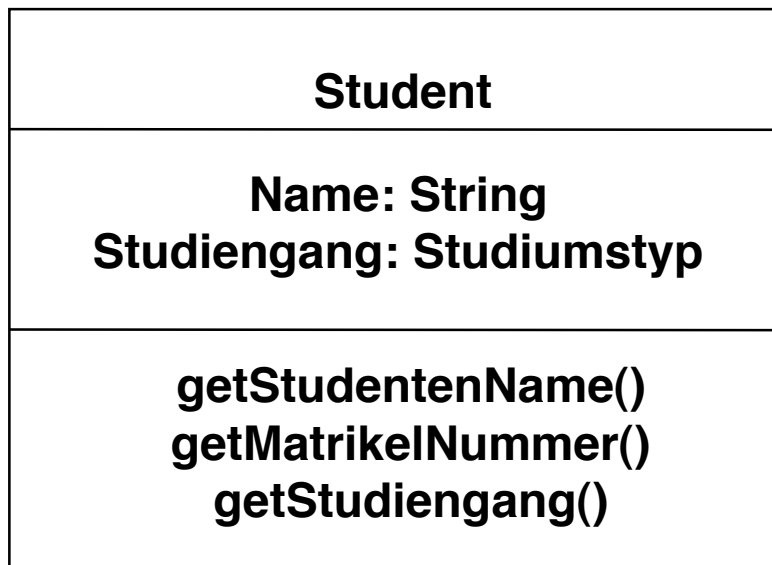
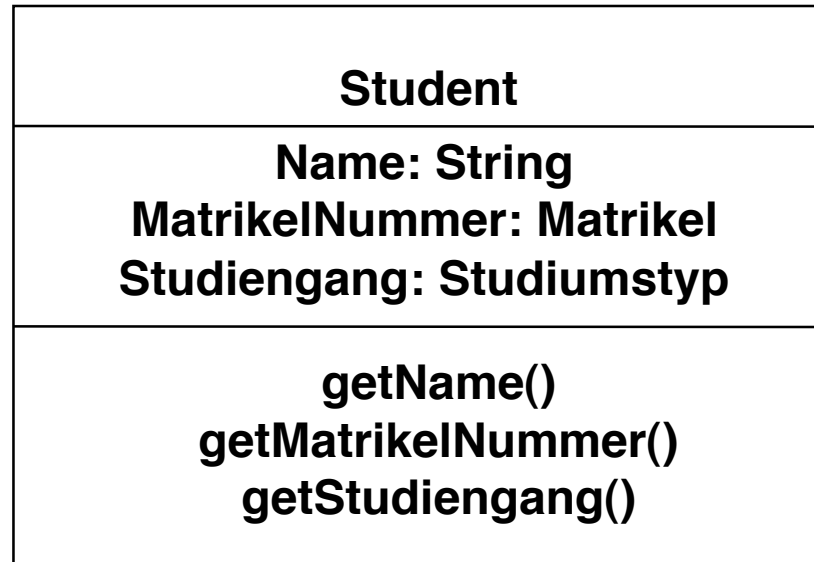
Graphische Darstellung: Objekt vs. Klasse



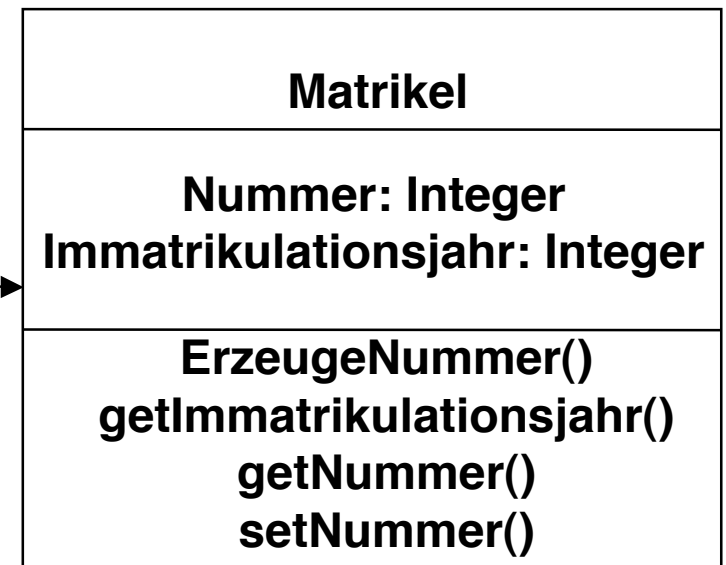
Attribute

- ❖ Ein **Attribut** einer Klasse wird in der Attributliste der Klasse als *Name* (während der Analyse) oder als *Name: Typ* (während des detaillierten Entwurfs) aufgelistet. Beispiele von Typen sind
 - **String**: Die Menge aller Zeichenketten
 - **Integer**: Die Menge aller ganzen Zahlen
 - **Boolean**: Die Menge der Wahrheitswerte {Wahr, Falsch}
 - **Studiumstyp**: Die Menge der Studiengänge {Diplom, Bachelor, Aufbaustudium}
- ❖ Ein Attribut kann auch als Beziehung zu einer Klasse gezeichnet werden, insbesondere wenn die Beziehung zwischen beiden Klassen klargemacht werden soll.
 - Der Name des Attributes ist dann der Name der Beziehung.

Attribut vs. Klasse



MatrikelNummer



Operation

- ❖ Die **Operationen** arbeiten auf Attributen der Klasse und anderen Objekten, mit denen die Klasse eine Beziehung hat.
- ❖ Die Menge von Operationen, die eine Klasse oder eine Menge von Klassen (Subsystem) zur Verfügung stellt, bezeichnen wir als **Schnittstelle** der Klasse (des Subsystems).

Einschub: Zwei Prinzipien der Didaktik

❖ Erstes Prinzip:

- Verwende keine Konzepte, die du nicht gründlich eingeführt hast!

❖ Zweites Prinzip:

- Programmieren lernt man nur durch Programmieren!
- Also: so früh wie möglich programmieren!

❖ **Problem:**

- Bereits das kleinste Java-Programm verwendet komplizierte Konzepte.

❖ **Kompromiss:**

- Wir fangen früh an, in Java zu programmieren.
- Wir betrachten jetzt einiges als „*so ist es halt*“ und freuen uns auf das Aha-Erlebnis, wenn wir später die Konzepte kennenlernen.



**Augen zu
und durch!**

Klassendefinition in Java

Student
Name: String Matrikelnummer: Integer
getName() getMatrikelnummer()

Was ist denn das???

```
public class Student {  
  
    // Konstruktor:  
    public Student() {}  
  
    // Attribute:  
    protected String name;  
    protected int matrikelnummer;  
  
    // Operationen (Methoden):  
    public String getName() {  
        return name;  
    }  
    public int getMatrikelnummer() {  
        return matrikelnummer;  
    }  
}
```

❖ GrProg-Konventionen:

- Klassennamen groß geschrieben, Merkmalsnamen klein geschrieben
- Attribute sind geschützt (protected), d.h. sie gehören nicht zur Schnittstelle

Besetzung der Attribute: Initialisierung

```
public class Student {  
  
    // Konstruktor:  
    public Student() {}  
  
    // Attribute:  
    protected String name = "Viola";  
    protected int matrikelNummer = 1234567;  
  
    // Operationen (Methoden):  
    public String getName() {  
        return name;  
    }  
    public int getMatrikelNummer() {  
        return matrikelNummer;  
    }  
}
```

- ❖ Initialisierung der Attribute in der Klassendefinition
 - Default-Werte

Besetzung der Attribute im Konstruktor

```
public class Student {  
  
    // Konstruktor:  
    public Student() {  
        name = "Viola";  
        matrikelNummer = 1234567;  
    }  
  
    // Attribute:  
    protected String name;  
    protected int matrikelNummer;  
  
    // Operationen (Methoden):  
    public String getName() {  
        return name;  
    }  
    public int getMatrikelNummer() {  
        return matrikelNummer;  
    }  
}
```

- ❖ Der Konstruktor ist eine spezielle Methode.
- ❖ Sein Name ist identisch mit dem Klassennamen.
- ❖ Der Konstruktor wird ausgeführt, wenn ein Objekt der Klasse instantiiert wird. (Objekte müssen explizit instantiiert werden.)
- ❖ Er wird dann (u.a) benutzt, um die Attribute zu initialisieren.
- ❖ Instantiierung eines Objekts der Klasse Student:

```
Student s = new Student();
```

Parametrisierte Besetzung der Attribute

```
public class Student {  
  
    // Konstruktor:  
    public Student(String n, int m) {  
        name = n;  
        matrikelNummer = m;  
    }  
  
    // Attribute:  
    protected String name;  
    protected int matrikelNummer;  
  
    // Operationen (Methoden):  
    public String getName() {  
        return name;  
    }  
    public int getMatrikelNummer() {  
        return matrikelNummer;  
    }  
}
```

❖ Instantiierung eines Objekts bei einem parametrisierten Konstruktor:

```
Student s =  
    new Student("Viola", 1234);
```

System und Umgebung in Java

```
public class Student {  
    public Student(String n, int m) {  
        name = n, matrikelNummer = m; }  
    protected String name;  
    protected int matrikelNummer;  
    public String getName() {  
        return name; }  
    public int getMatrikelNummer() {  
        return matrikelNummer; }  
}
```

System



```
public class Umgebung {  
    public static void main (String[] args) {  
        Student s = new Student ("Viola", 1234);  
        System.out.println(s.getName());  
        System.out.println(s.getMatrikelNummer());  
    }  
}
```

Schnittstelle

Umgebung

Editieren, Compilieren, Programmlauf

❖ **Editieren:**

- Klassendefinition Student in Datei **Student.java**
- Klassendefinition Umgebung in Datei **Umgebung.java**

❖ **Kompilieren:**

- **javac Student.java** liefert Datei **Student.class**
- **javac Umgebung.java** liefert Datei **Umgebung.class**
- (Es hätte auch nur der Befehl **javac Umgebung.java** genügt. Das Java-System ist so „intelligent“, alle zusätzlich benötigten Klassen mit zu kompilieren.)

❖ **Ablauf des Programms (Exekutieren):**

- **java Umgebung** (ohne Extension **.class**)
- Das Java-System führt die **main**-Methode der Klasse Umgebung aus

System und Umgebung mit dem Java-System

```
public class Student {  
    public Student(String n, int m) {  
        name = n; matrikelNummer = m; }  
    protected String name;  
    protected int matrikelNummer;  
    public String getName() {  
        return name; }  
    public int getMatrikelNummer() {  
        return matrikelNummer; }  
}
```

```
public class Umgebung {  
    public static void main (String[] args) {  
        Student s = new Student ("Viola", 1234);  
        System.out.println(s.getName ());  
        System.out.println(s.getMatrikelNummer ());  
    }  
}
```

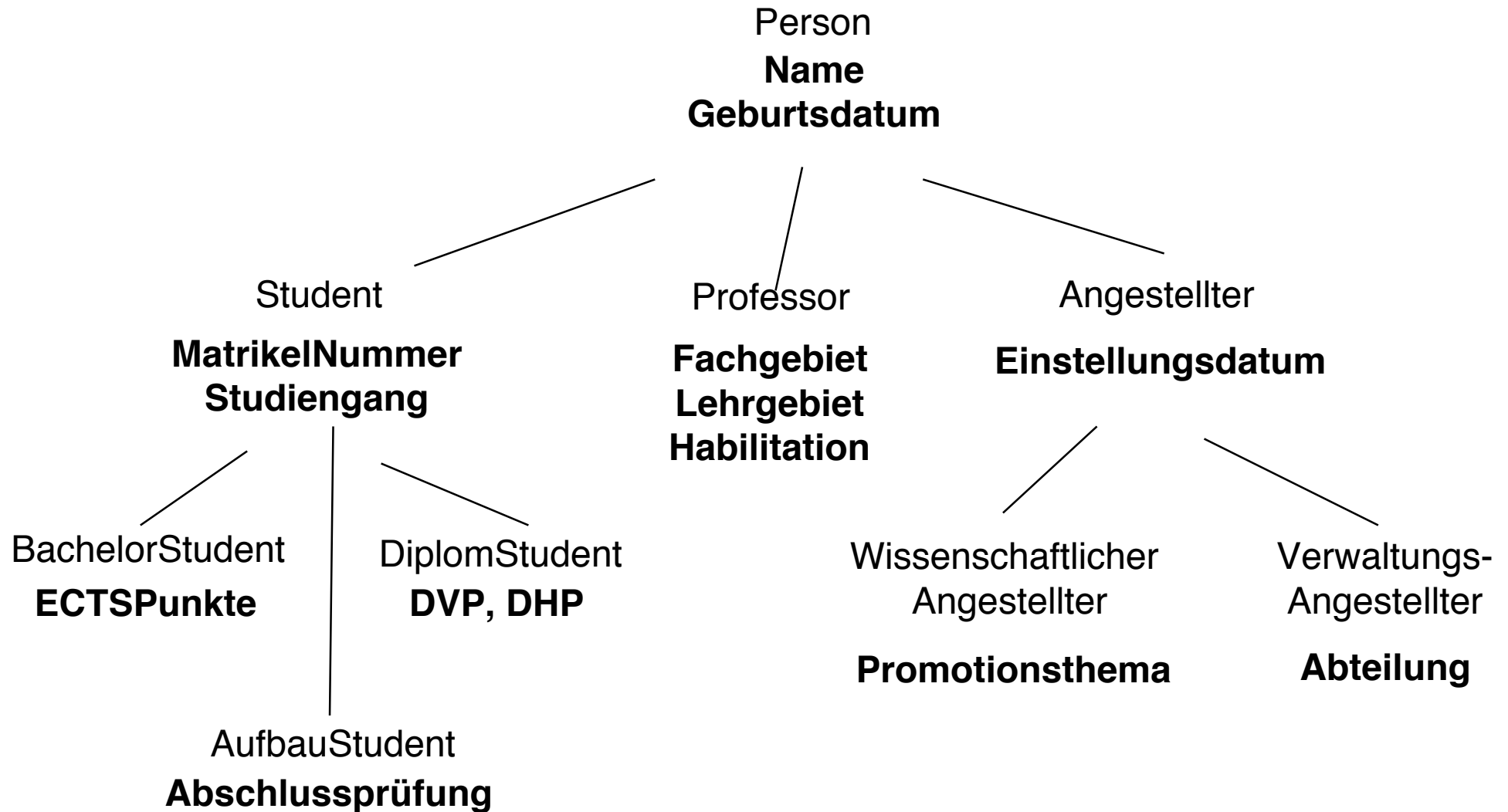
Java-
System

Schnittstelle zum
Java-System

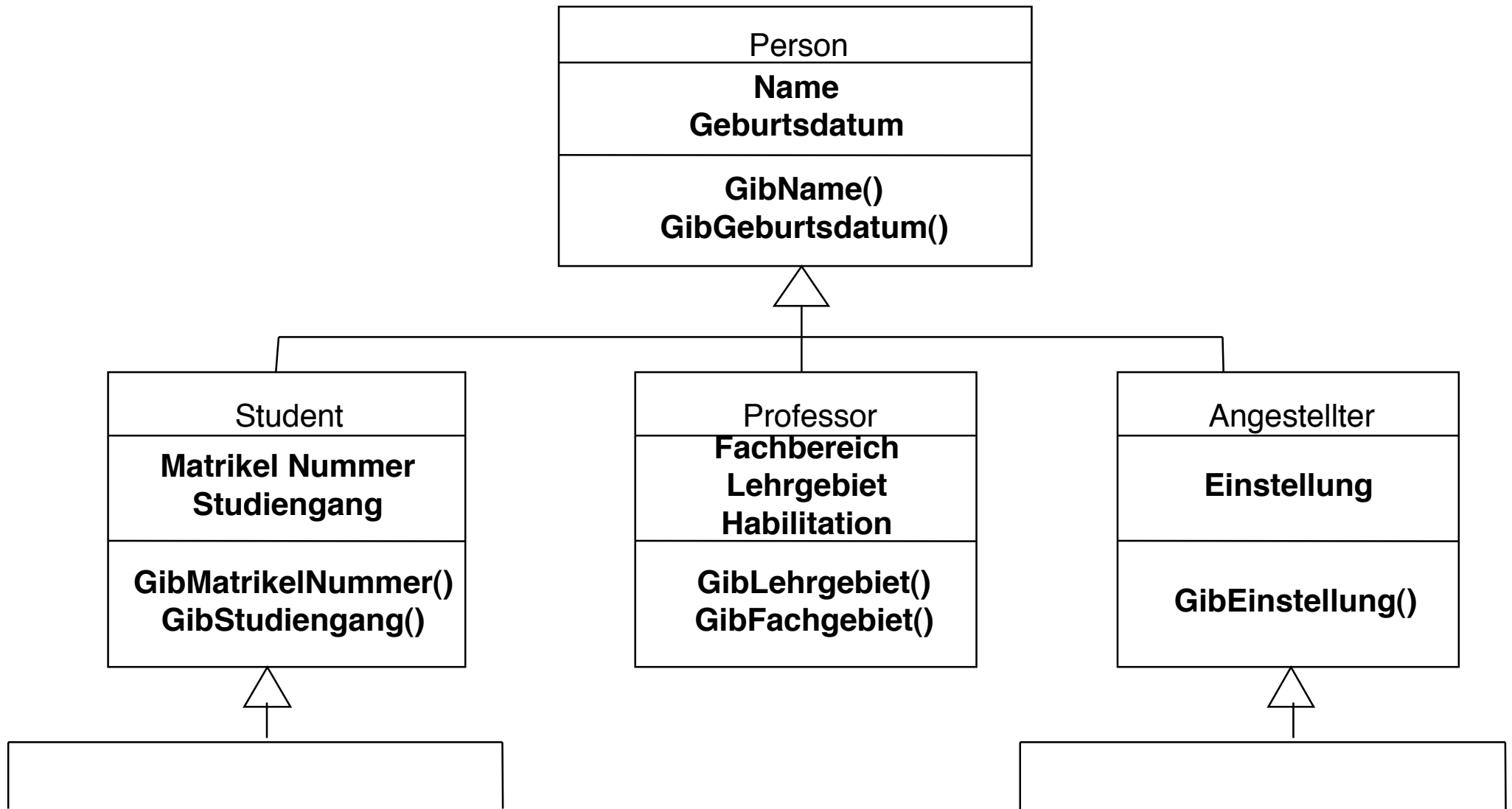
Zwei wichtige Prinzipien der Modellierung

- ❖ **Informationskapselung** (*information hiding*): Objekte können auf andere Objekte nur über deren Schnittstelle zugreifen.
 - Ein Objekt kann also nicht direkt auf die Attribute eines anderen Objektes zugreifen.
- ❖ **Klassifikation**: Komponenten können nach ihren Merkmalen klassifiziert werden.
 - Beispiel: Zwei Objekte Obj1 und Obj2 können zusammengefasst werden, wenn sie dieselbe Operation *print()* verstehen.
- ❖ **Klassifikationen kann man benutzen, um Mengen von Objekten hierarchisch zu strukturieren.**
 - Beispiel: Die Personengruppen an einer Universität.

Klassifikation von Personengruppen an einer Universität



Klassifikation von Personengruppen an einer Universität



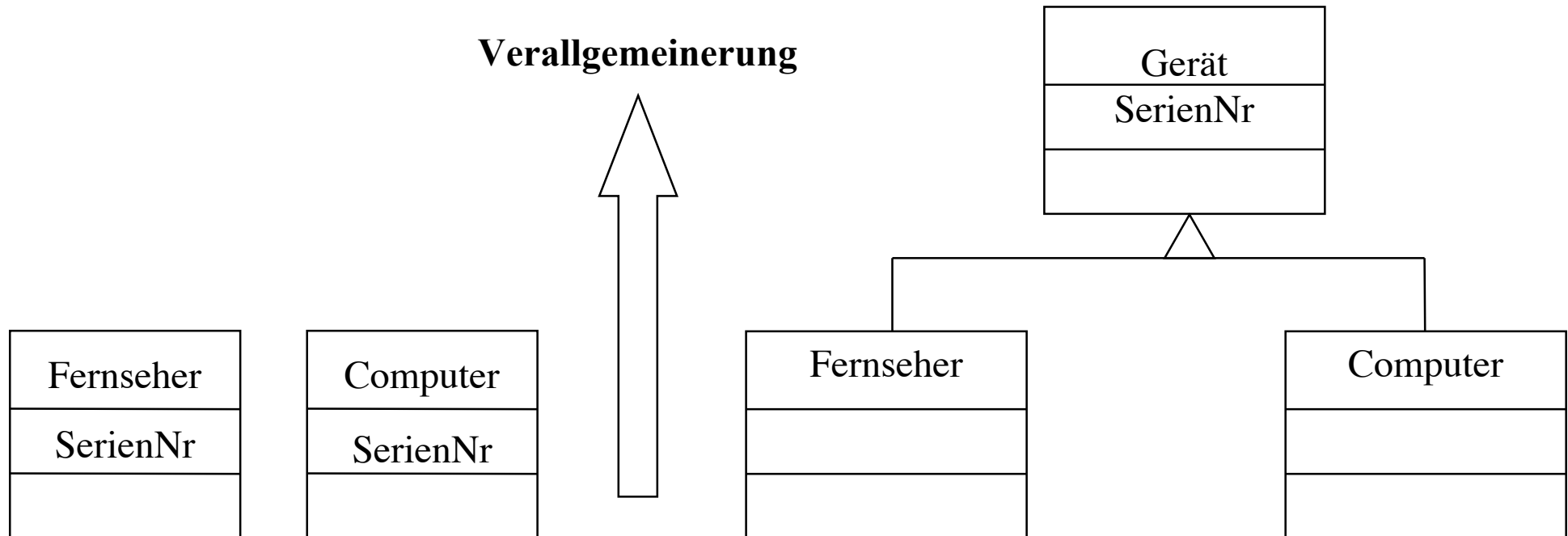
Die Vererbungsbeziehung

- ❖ Zwei Klassen stehen in einer **Vererbungsbeziehung** (*inheritance relationship*) zueinander, falls die eine Klasse, auch **Unterklasse** (Subklasse) genannt, alle Merkmale der anderen Klasse, auch **Oberklasse** genannt, besitzt, und darüber hinaus noch zusätzliche Merkmale.
- ❖ Es gilt somit für die Mengen A_U , A_O der Attribute und die Mengen O_U , O_O der Operationen der Unterklasse U und Oberklasse O :
 - $A_O \sqsubseteq A_U$ und $O_O \sqsubseteq O_U$
- ❖ Eine Unterklasse wird also durch Hinzufügen von Merkmalen **spezialisiert**.
- ❖ Umgekehrt verallgemeinert die Oberklasse die Unterklasse dadurch, dass sie spezialisierende Eigenschaften weglässt. Wir nennen das auch **Verallgemeinerungsbeziehung** (*generalization relationship*).

Vererbungsbeispiel

❖ Eine Firma stellt sowohl Fernsehgeräte als auch Computer her. Beide haben Seriennummern.

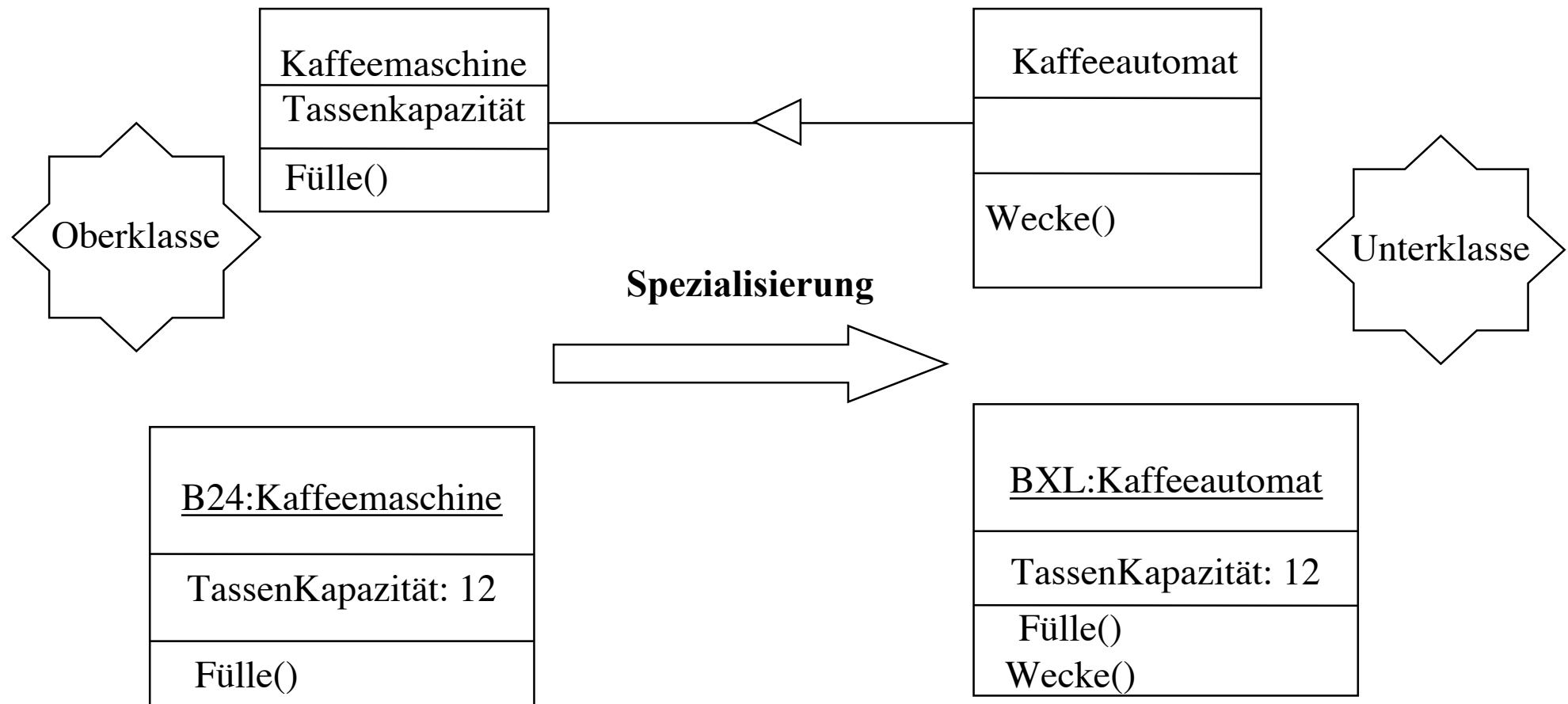
❖ Die Seriennummer wird in einer Oberklasse Gerät angeführt und von dort vererbt.



Noch ein Vererbungsbeispiel

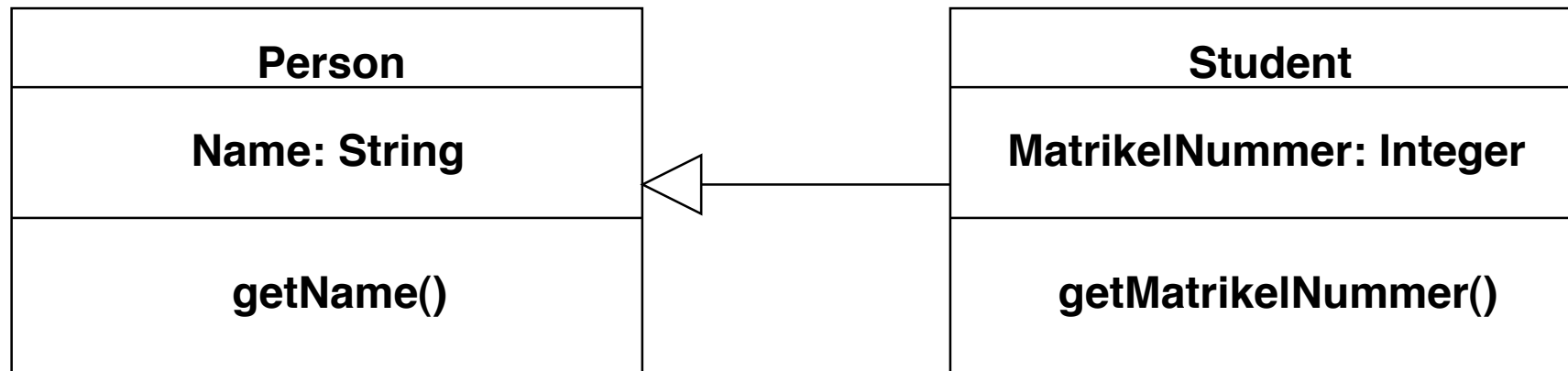
- ❖ Eine Kaffeemaschine kann eine Anzahl von Tassen füllen.

- ❖ Das Luxusmodell hat noch eine Weckfunktion



Vererbung in Java

❖ Klasse Student jetzt als Unterklasse von Person:



❖ `public class Person { ... }`

❖ `public class Student extends Person { ... }`

System und Umgebung in der veränderten Situation

```
public class Person {  
    public Person {}  
  
    protected String name;  
  
    public String getName() {  
        return name; }  
}
```

```
public class Student extends Person{  
    public Student(String n, int m) {  
        name = n; matrikelNummer = m; }  
  
    protected int matrikelNummer;  
  
    public int getMatrikelNummer() {  
        return matrikelNummer; }  
}
```

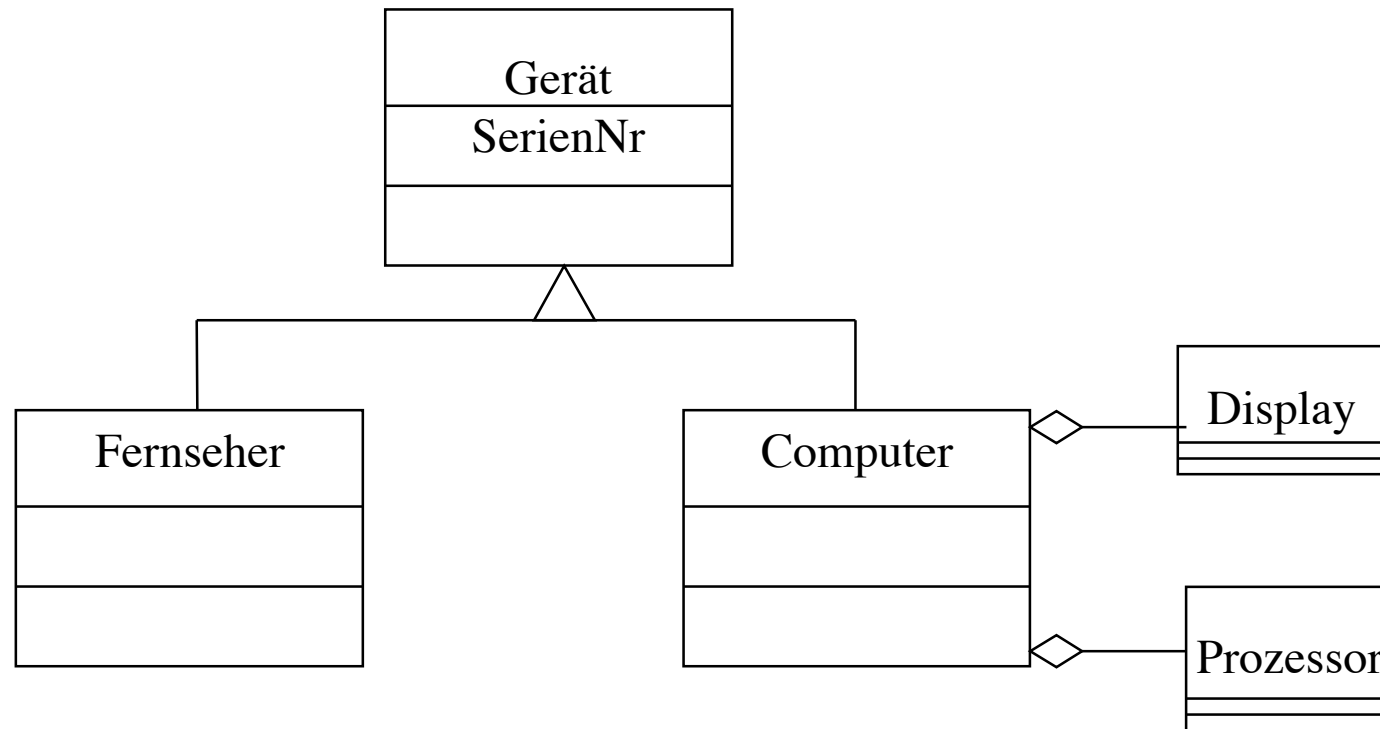


Schnittstelle

```
public class Umgebung {  
    public static void main (String[] args) {  
        Student s = new Student("Viola", 1234);  
        System.out.println(s.getName());  
        System.out.println(s.getMatrikelNummer());  
    }  
}
```

Aggregation und Vererbung lassen sich kombinieren

- ❖ In der Modellierung tritt oft der Fall auf, dass wir Gegenstände klassifizieren müssen, aber gleichzeitig auch deren Struktur erkenntlich machen wollen.

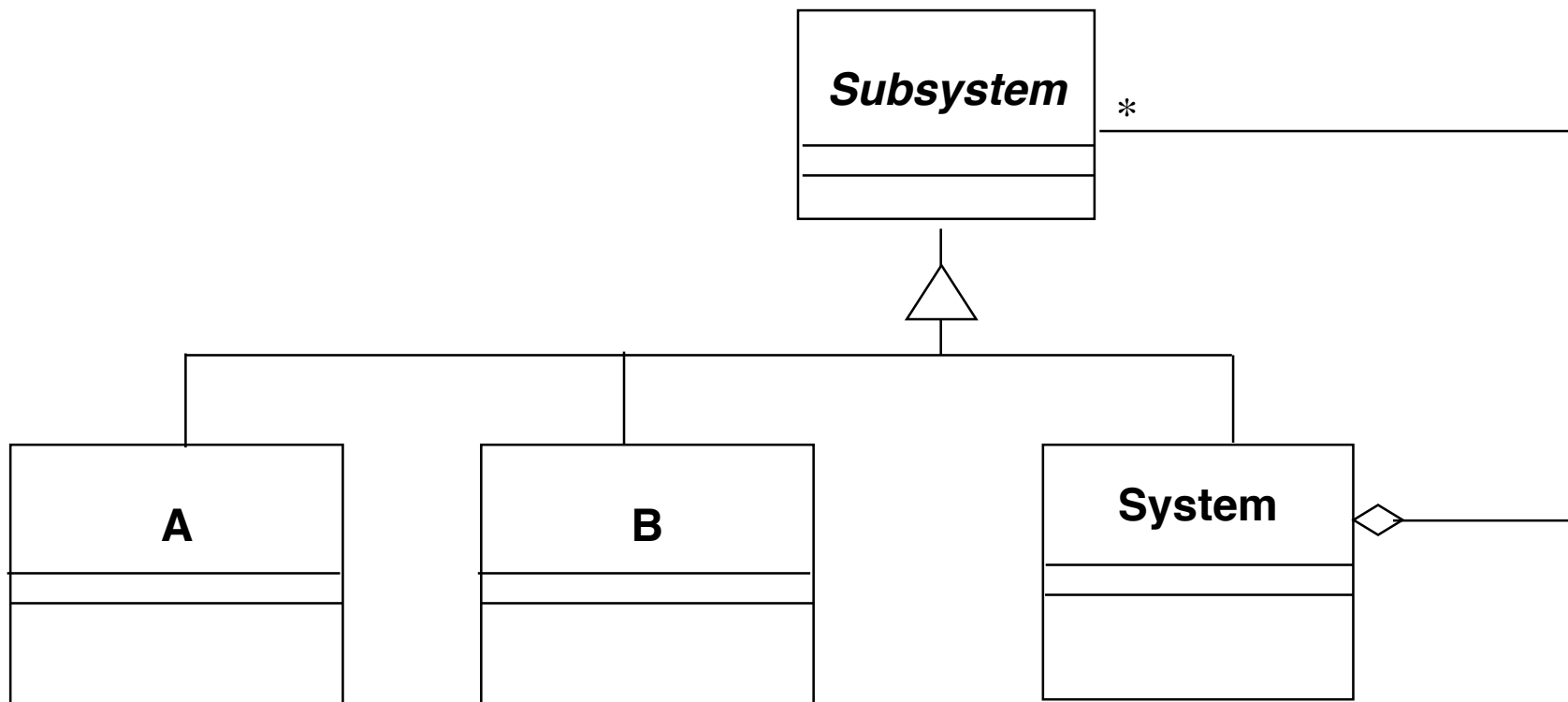


Modellierung unserer Systemdefinition

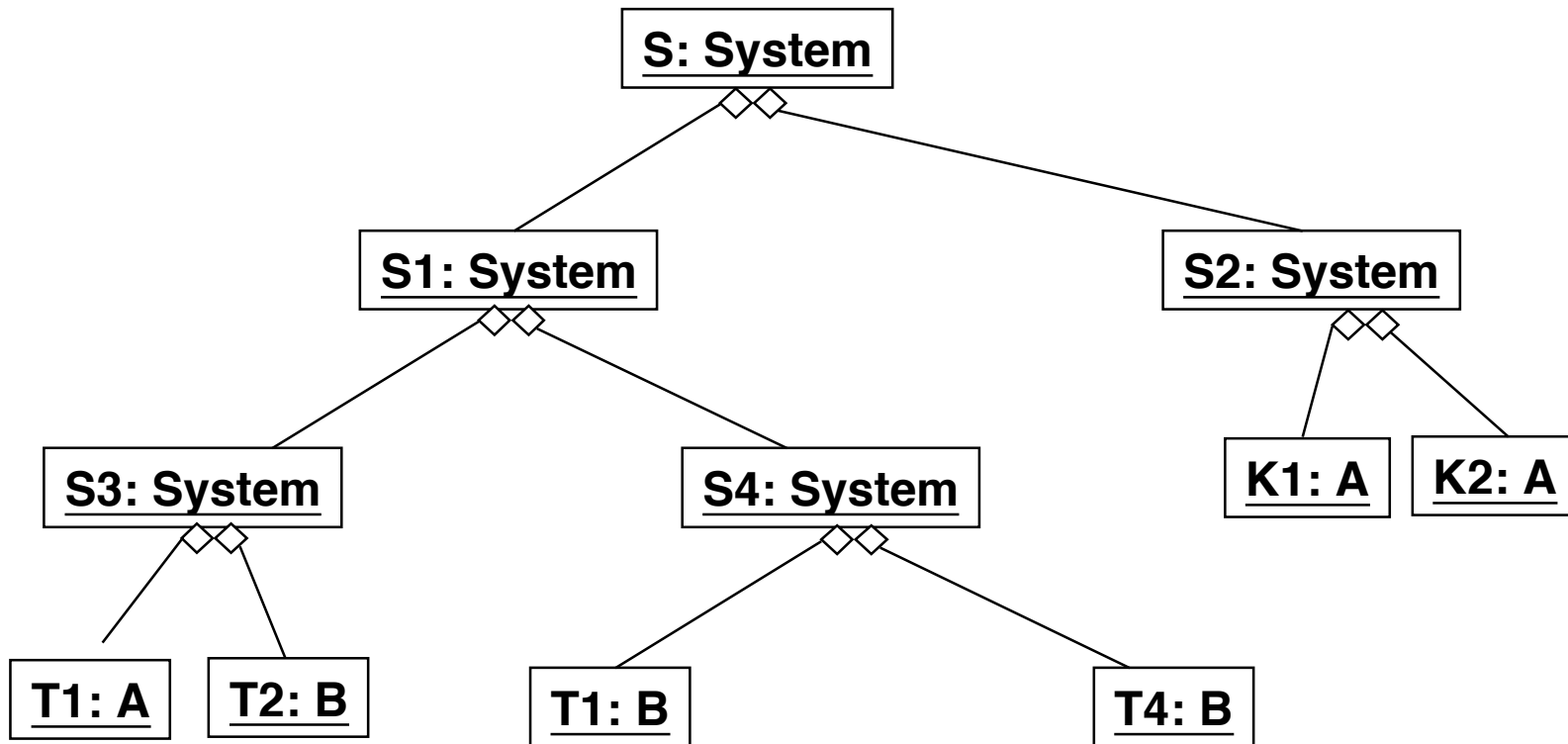
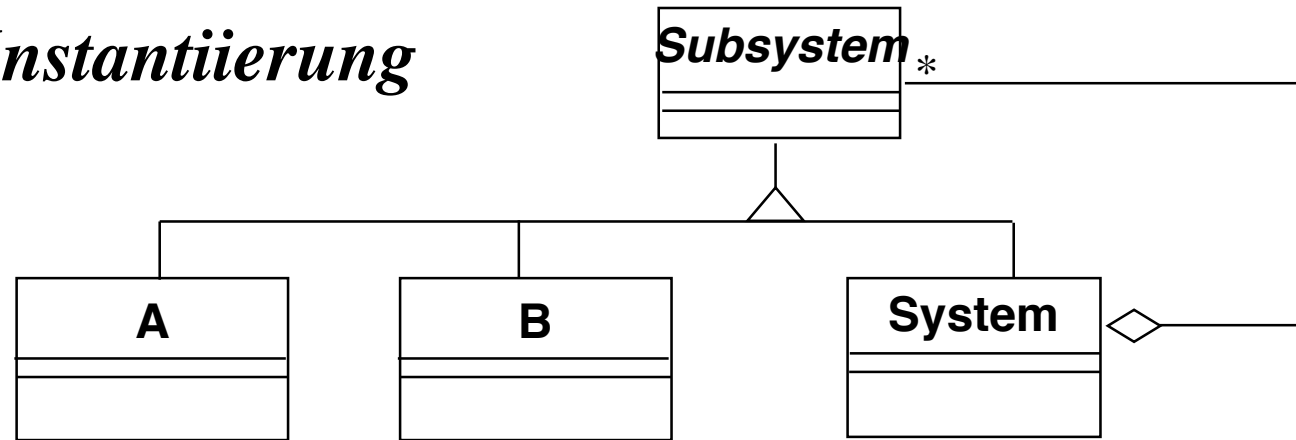
- ❖ **Wiederholung (Definition eines Systems):** Unter einem System versteht man eine *Menge von Komponenten (Gegenständen)*, die in einem gegebenen Bezugssystem in einem Zusammenhang stehen, und die *Beziehungen zwischen diesen Komponenten*.
- ❖ Die Komponenten eines Systems können selbst wieder (Sub-)Systeme sein.

Modellierung des Systembegriffs

- ❖ **Beispiel:** Ein System kann beliebig viele Subsysteme und 2 Arten von Komponenten A und B haben.

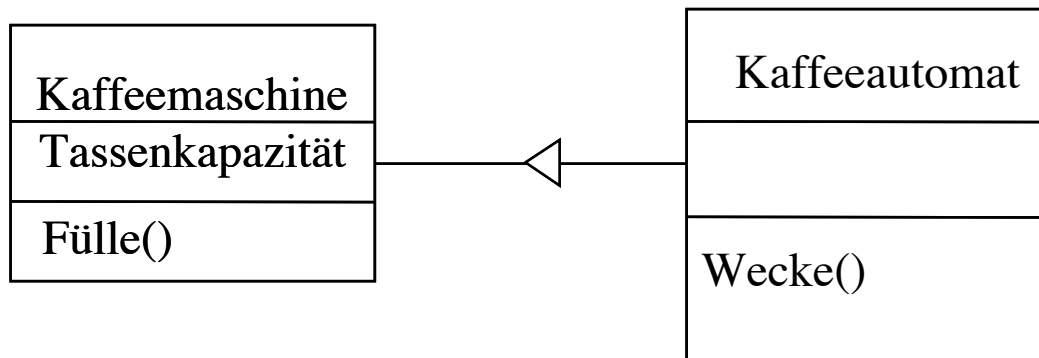


Beispiel einer Instantiierung

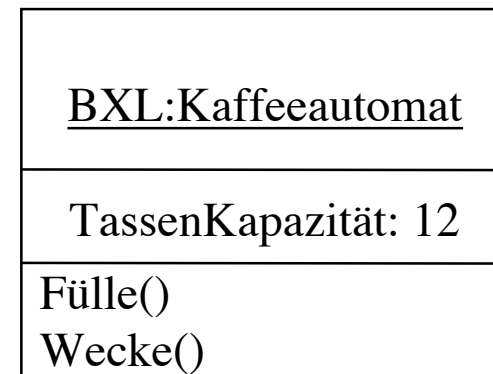


Was passiert mit der Vererbung bei Instanzdiagrammen?

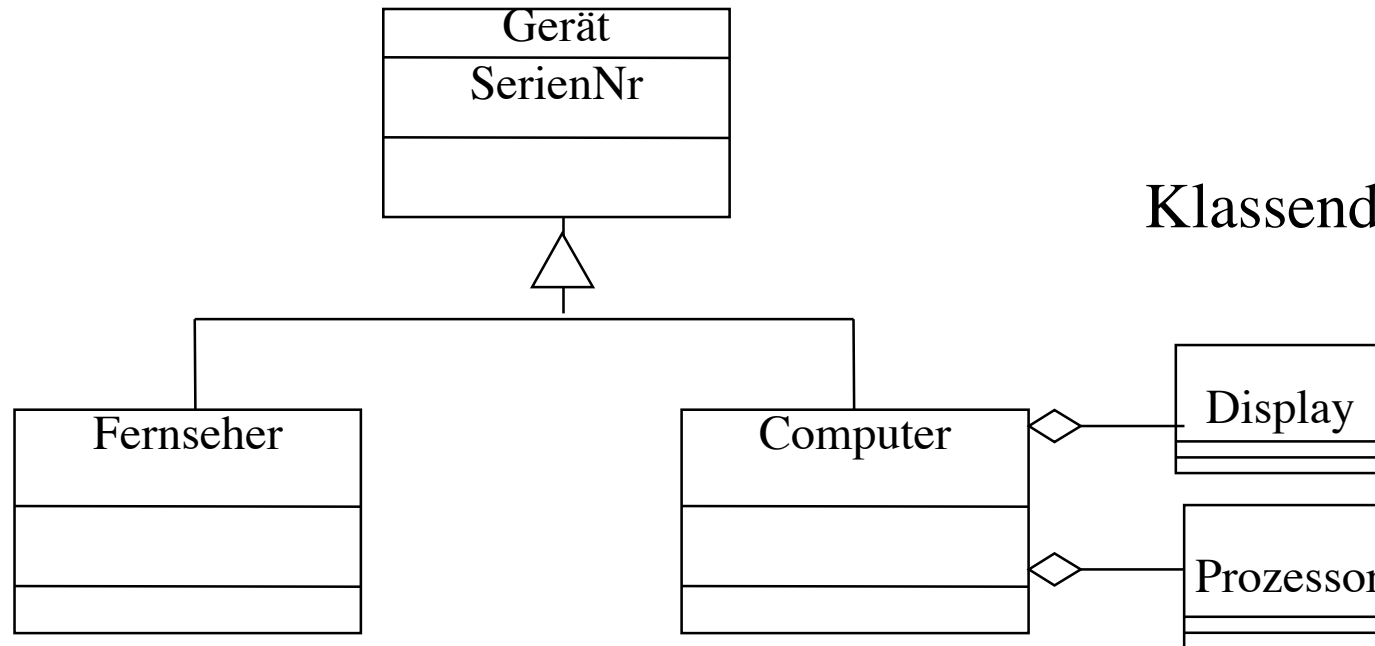
Klassendiagramm



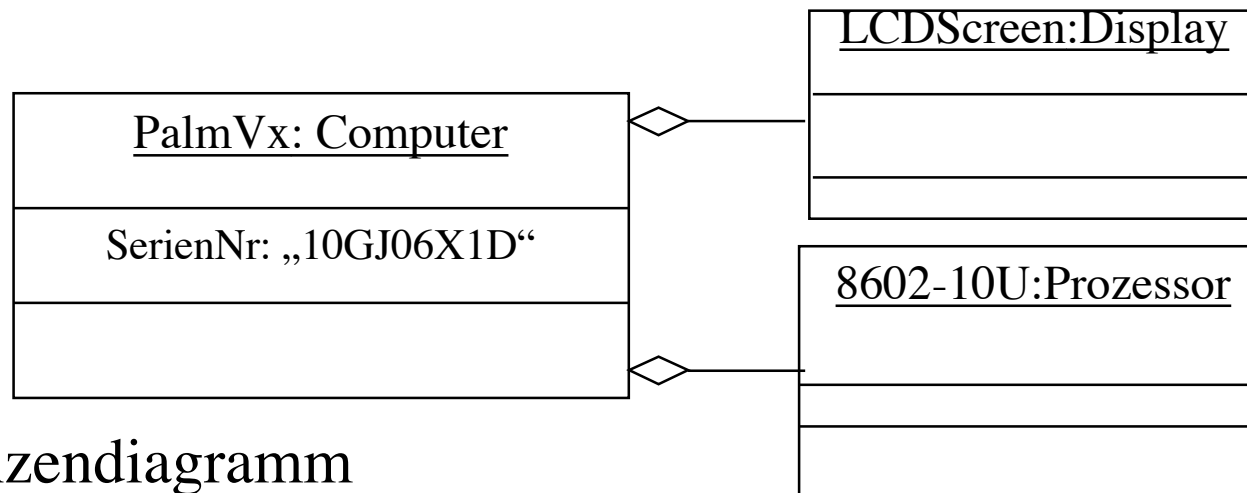
Objektdiagramm



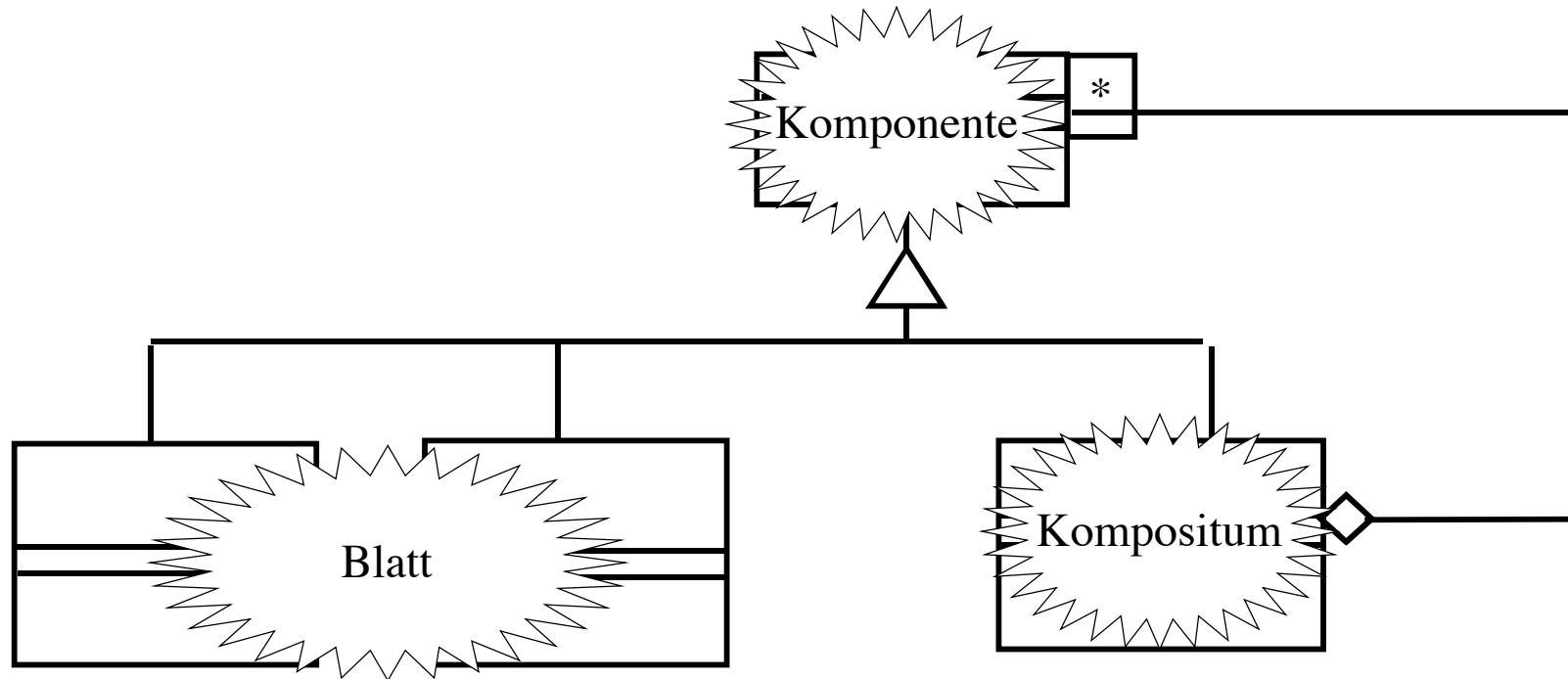
Klassendiagramm



Instanzendigramm

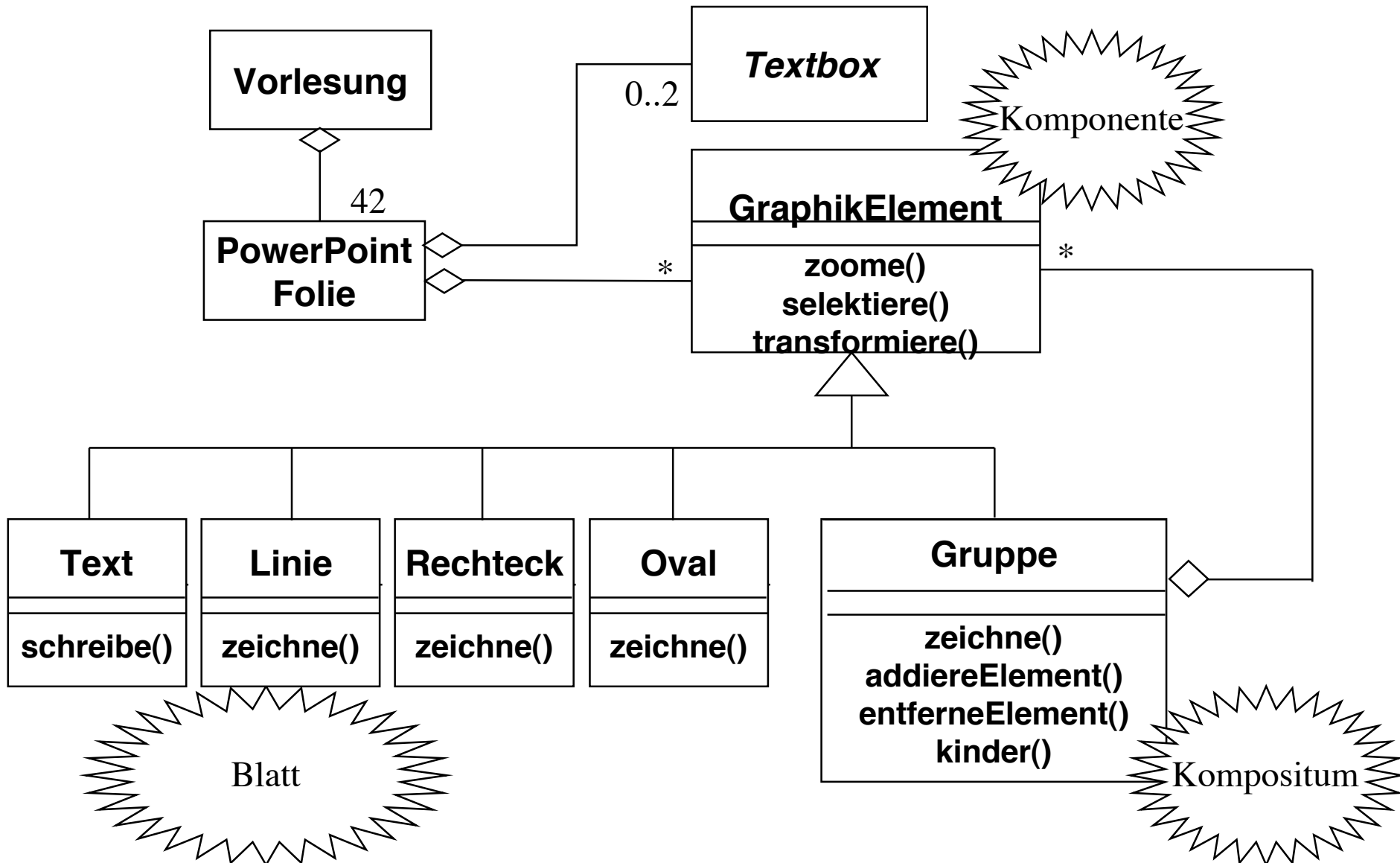


Kompositionsmuster



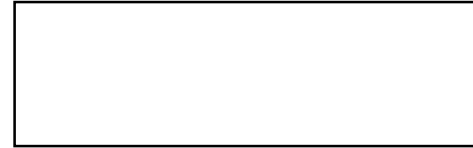
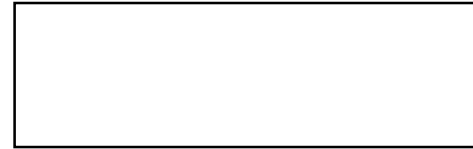
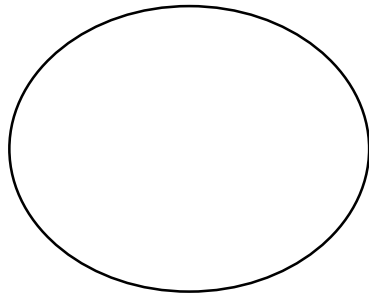
Gamma et. al: Composite Pattern,
Kompositum

Modellierung der heutigen Vorlesung

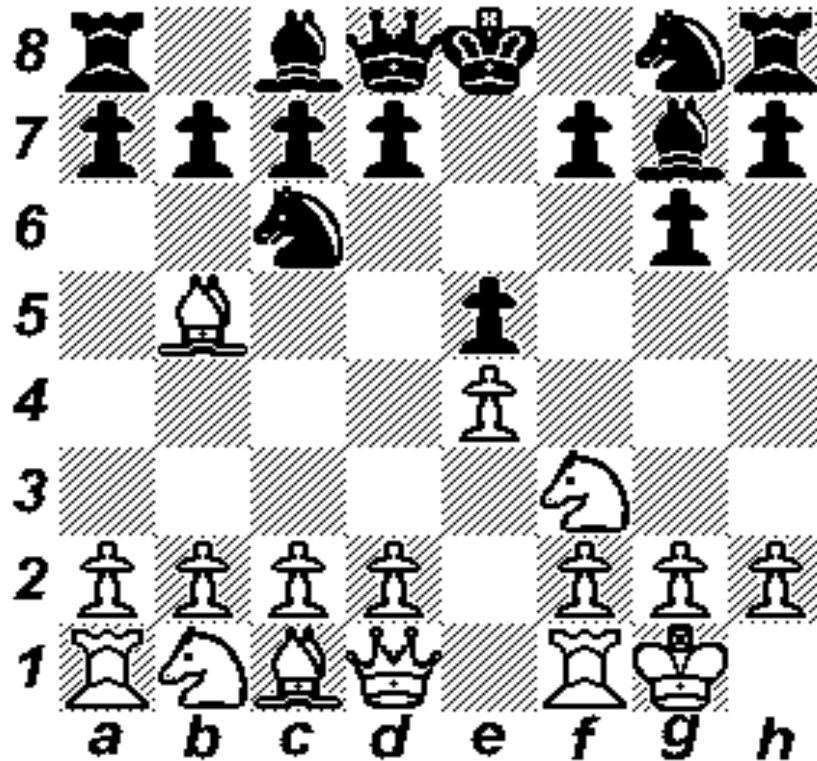


Dies ist ein Beispiel

❖ Eine äußerst interessante Folie



Muster im Schach



- ❖ Spanische Eröffnung
- ❖ Schwarz: Läufer Fianchetto
- ❖ Weiss: Kurze Rochade

Nützlichkeit von Entwurfsmustern

- ❖ Entwurfsmuster sind wieder verwendbares Wissen bei der Entwicklung von Informatik-Systemen, vor allem bei der Analyse und beim Systementwurf.
- ❖ Entwurfsmuster lassen sich zu einem Gesamtentwurf kombinieren, der dann als Grundlage für ein Informatik-System dienen kann.
- ❖ Wir werden noch weitere Entwurfsmuster kennen lernen:
 - Beobachter-Muster (Observer Pattern): Applets
 - Adapter-Muster (auch „Wrapper“ genannt): Zum Aufruf von alten, nicht mehr änderbaren Schnittstellen

Zusammenfassung

- ❖ **Objekt: Attribute, Operationen, Merkmale**
- ❖ **Objekt als Instanz einer Klasse**
- ❖ **Instanzendiagramm, Klassendiagramm**
- ❖ **Klassen und Instanzen in Java**
 - die main-Methode
 - Konstruktoren
- ❖ **Kanonische Beziehungen in der Modellierung:**
 - Aggregation
 - Vererbung
- ❖ **Vererbung in Java**
- ❖ **Kompositionsmuster**

Organisatorisches

- ❖ Ab Donnerstag, 31.10.2002, findet die Donnerstagsvorlesung im FMI-Gebäude, Hörsaal 3, statt.
 - Zeit: 11:15 bis 12:00 Uhr
- ❖ Am Dienstag, dem 12. November 2002, entfällt die Vorlesung, da zeitgleich die Studentenvollversammlung stattfindet.
- ❖ Am Donnerstag, dem 5. Dezember 2002, entfällt die Vorlesung wegen des Dies Academicus

- ❖ Am Donnerstag, dem 14. November 2002, wird das FMI-Gebäude eingeweiht. Vermutlich muss auch dann die Vorlesung entfallen.