

# Collection Klassen JDK 1.0

- Vector
- Stack
- Hashtable

Es können nur Objekte bearbeitet werden ->  
Wrapper-Klassen (Int, Double,...)

# Wrapper-Klassen

Eigene Wrapperklassen für primitive Typen:  
Short, Byte, Integer, Long, Float, Double,  
Character, Boolean

2 Konstruktoren möglich

- Übergabeparameter als Wert des Grundtyps

```
Integer iw1= new Integer(1);
```

- Übergabeparameter als String

```
Integer iw2= new Integer(„1“);
```

# Collection Klassen Erweiterung

## JDK 1.2

- Liste: geordnete Zusammenfassung von Elementen, **Duplikate sind möglich**  
**ArrayList**
- Set: **Es sind keine Duplikate erlaubt**  
**HashSet**
- Map: Zuordnung **Schlüssel -> Element:**  
**HashMap**
- `java.util.Collection`

# Collection Klassen - Beispiel

```
import java.util.*;
public class ArrayList01 {

    public static void main(String[] args) {
        ArrayList al = new ArrayList();
        Integer iw1= new Integer(1);
        Double  dw1= new Double(3.56);

        al.add("Text1");
        al.add(iw1);
        al.add(dw1);
        al.add("Text2");

        for (int i=0;i<al.size();i++){
            System.out.println("Daten der Klasse: " +
                               al.get(i).getClass().getName());
            System.out.println(al.get(i));
        }
    }
}
```

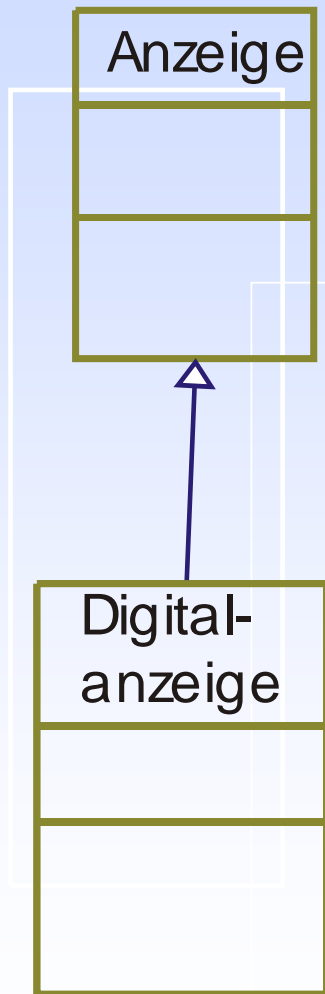
# **Modellierung mit UML (Unified Modeling Language) und Implementierung mit Java**

- Sprachenunabhängige Modellierung der zu erstellenden Software
- Aufbau- und Ablaufbeschreibung
- Implementierung mit geeigneter Programmiersprachen

# Modellierung mit UML und Implementierung mit Java

- Vererbung
- Assoziation
- Aggregation
- Komposition

# Vererbung (1)



**JAVA \*.java**

```
public class Anzeige {
    String hersteller;

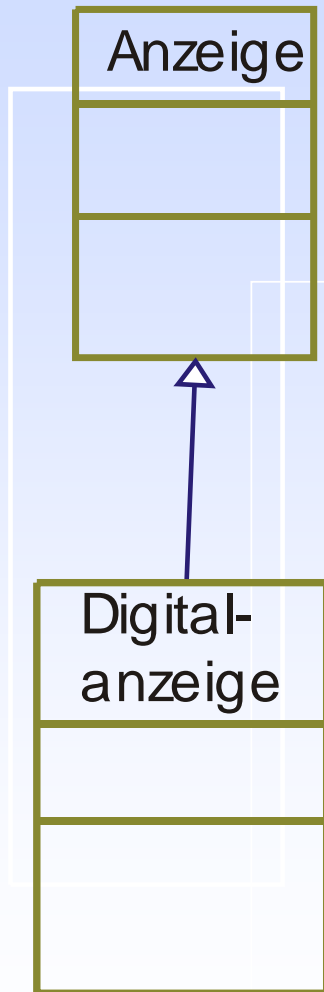
    public Anzeige() {
    }
}

public class Digitalanzeige extends Anzeige{
    int anzahlDerStellen;

    public Digitalanzeige() { }
    public SetAnzahlDerStellen() { }

}
}
```

# Vererbung (2)



JAVA \*.java

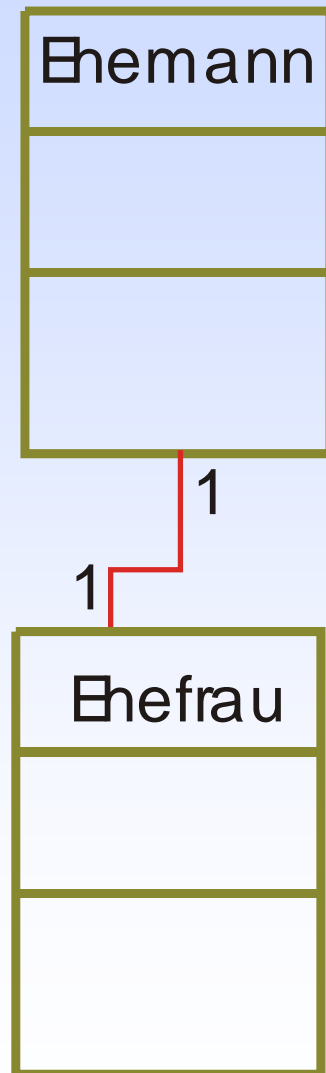


```
public class VererbungBsp {

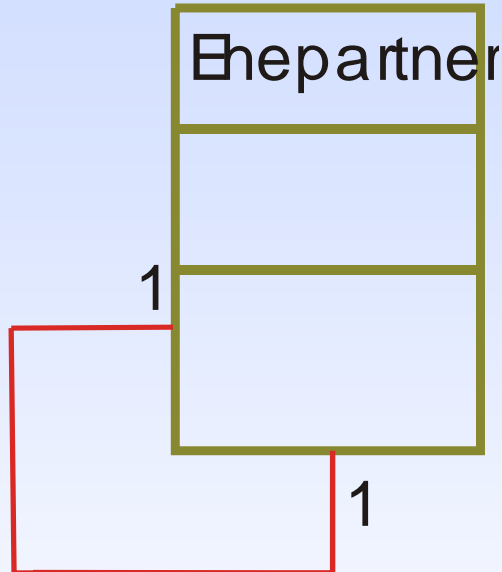
    public static void main(String[] args) {
        Digitalanzeige danzeigeImGeraet1 =
new Digitalanzeige();
        danzeigeImGeraet1.hersteller="MeyerGmbH";
        danzeigeImGeraet1.anzahlDerStellen=5;
    }
}
```



# Assoziation 1:1 (1) - bidirektional



# Assoziation 1:1 (2) - bidirektional



**Zirkuläre reflexive  
Assoziation**

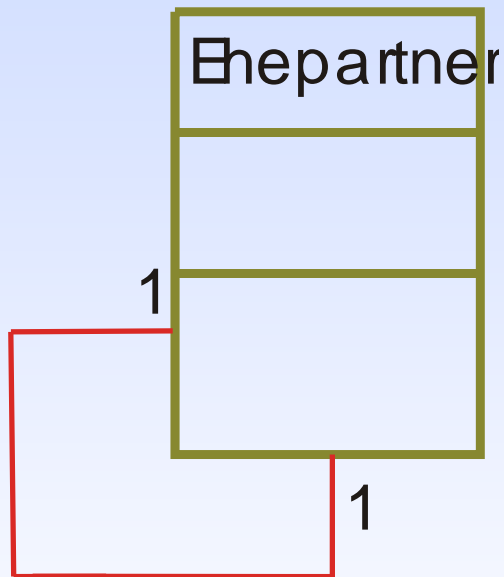
```
JAVA *.java

public class Assoziation1zulbidir {
    public static void main(String[] args) {
        Ehepartner hermannMeyer =
new Ehepartner();
        Ehepartner erikaMeyer =
new Ehepartner();

        hermannMeyer.name="HermannMeyer";
        erikaMeyer.name="ErikaMeyer";
        hermannMeyer.verheiratetMit(erikaMeyer);
        erikaMeyer.verheiratetMit(hermannMeyer);

        hermannMeyer.printName();
        erikaMeyer.printName();
        hermannMeyer.derEhepartner.printName();
        erikaMeyer.derEhepartner.printName();
    }
}
```

# Assoziation 1:1 (3) - bidirektional



```
JAVA *.java

public class Ehepartner {
    private String name;
    private Ehepartner derEhepartner;

    public Ehepartner() {
    }

    public void verheiratetMit(Ehepartner partner){
        this.derEhepartner = partner;
    }

    public void printName(){
        System.out.println(name);
    }
}
```

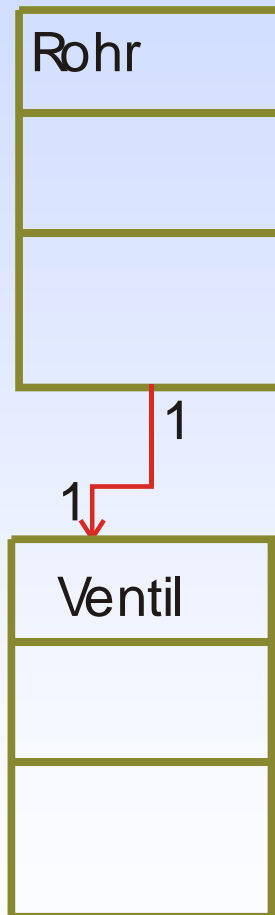
# Instanziierung

```
JAVA *.java
public class Assoziation1zulbidir {
    public static void main(String[] args) {
        Ehepartner hermannMeyer = new Ehepartner();
        Ehepartner erikaMeyer = new Ehepartner();
        ...
    }
}
```

hermannMeyer.Ehepartner

erikaMeyer.Ehepartner

# Assoziation 1:1 (1) - unidirektional



**JAVA \*.java**

```
public class mainAssoziation1zu1UniVentil {

    public static void main(String[] args) {

        Ventil ventill1 = new Ventil();
        Rohr rohr1= new Rohr();

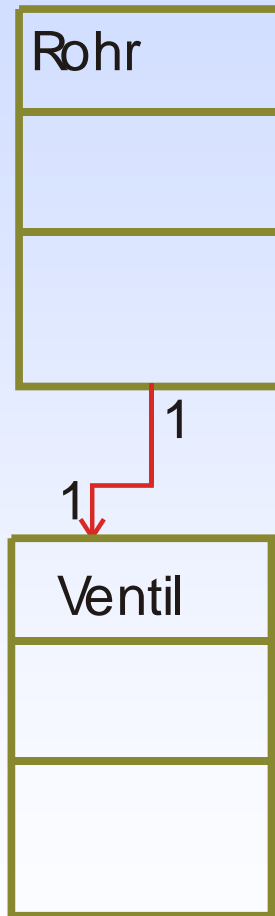
        ventill1.kennung="V1";
        rohr1.kennung="R1";

        rohr1.setVentil(ventill1);
        rohr1.oeffneRohr();

    }

}
```

# Assoziation 1:1 (2) - unidirektional



**JAVA \*.java**

```
public class Rohr {

    String kennung;
    private Ventil meinVentil;

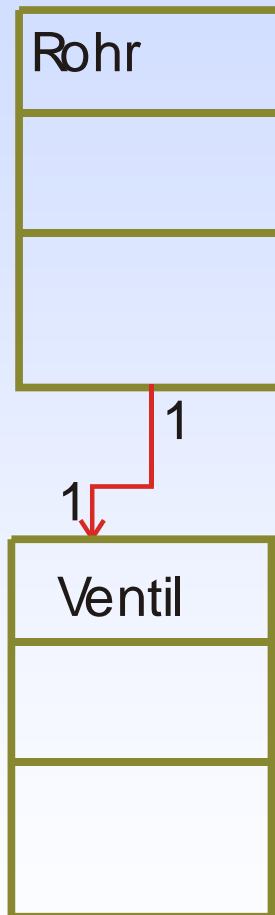
    public Rohr() {
    }

    void setVentil(Ventil dasVentil){
        this.meinVentil=dasVentil;
    }

    void oeffneRohr(){
        meinVentil.oeffnen();
    }

}
```

# Assoziation 1:1 (3) - unidirektional



JAVA \*.java



```
public class Ventil {

    private int durchfluss;
    String kennung;

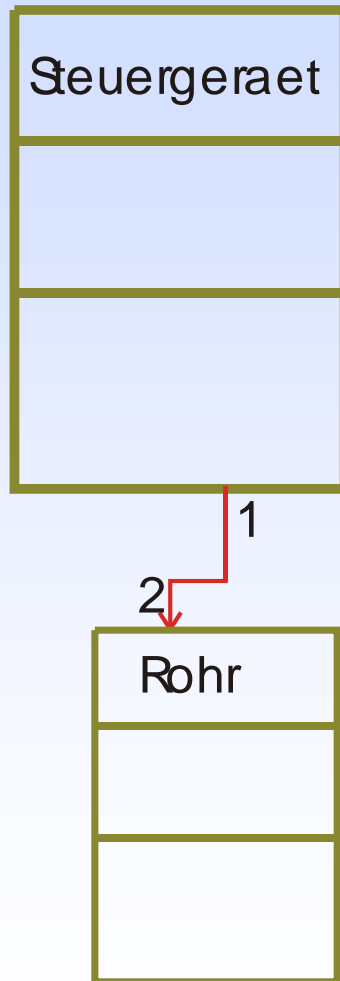
    public Ventil() {
    }

    public void oeffnen() {
        this.durchfluss=100;
    }

    public void schliessen() {
        this.durchfluss=0;
    }

}
```

# Assoziation 1:n (1) - unidirektional



```
JAVA *.java
public class Assoziation1zuNSteuergeraetRohre {

    public static void main(String[] args) {

        Steuergeraet Steuergeraet1=
new Steuergeraet();
        Rohr rohr1=new Rohr();
        Rohr rohr2=new Rohr();

        Steuergeraet1.setRohr1(rohr1);
        Steuergeraet1.setRohr2(rohr2);

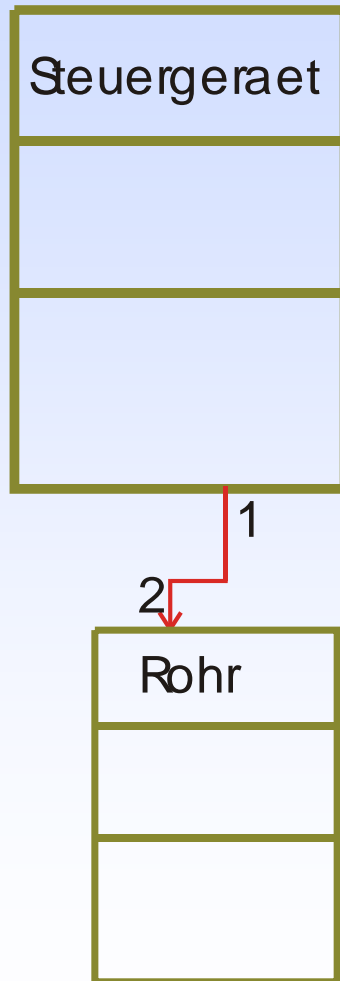
        Steuergeraet1.oeffneAlleRohre();

    }

}
```



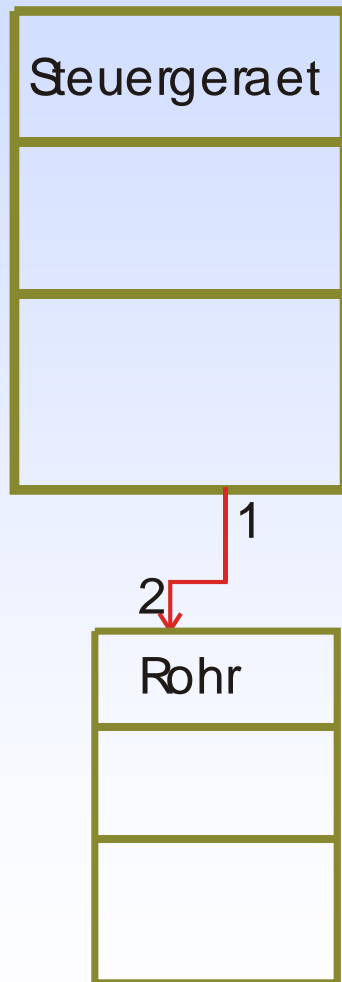
# Assoziation 1:n (2) - unidirektional



**JAVA \*.java**

```
public class Steuergeraet {
    Rohr rohr1;
    Rohr rohr2;
    public void setRohr1(Rohr rohr1){
        this.rohr1=rohr1;
    }
    public void setRohr2(Rohr rohr2){
        this.rohr2=rohr2;
    }
    public void oeffneAlleRohre(){
        rohr1.oeffneRohr();
        rohr2.oeffneRohr();
    }
    public Steuergeraet() {
        rohr1.kennung="R1";
        rohr2.kennung="R2";
    }
}
```

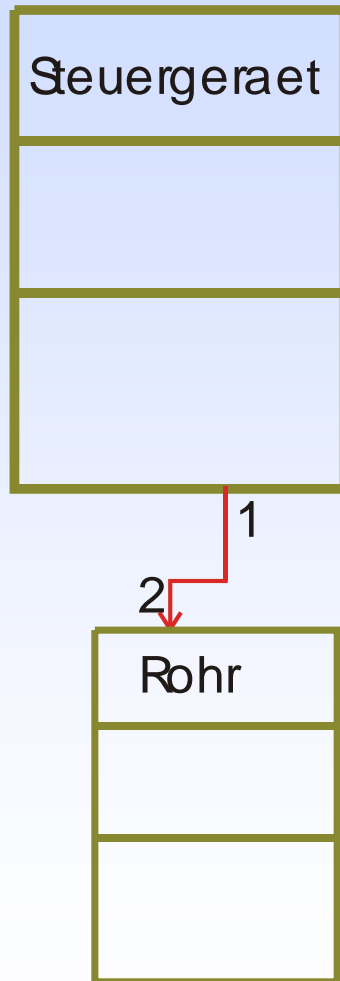
# Assoziation 1:n (3) - unidirektional



**JAVA \*.java**

```
public class Rohr {  
  
    String kennung;  
    meinVentil= new Ventil();  
  
    public Rohr() {  
    }  
  
    void oeffneRohr(){  
        meinVentil.oeffnen();  
    }  
}
```

# Assoziation 1:n (4) - unidirektional



**JAVA \*.java**

```
public class Ventil {

    private int durchfluss;
    String kennung;

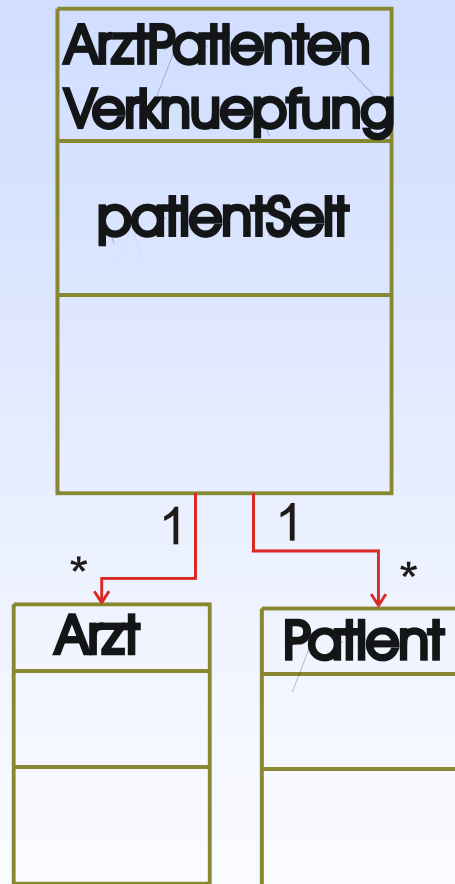
    public Ventil() {
    }

    public void oeffnen() {
        //entsprechender Befehl an Mechanik
        this.durchfluss=100;
    }

    public void schliessen() {
        //entsprechender Befehl an Mechanik
        this.durchfluss=0;
    }

}
```

# Assoziation n:m (1)



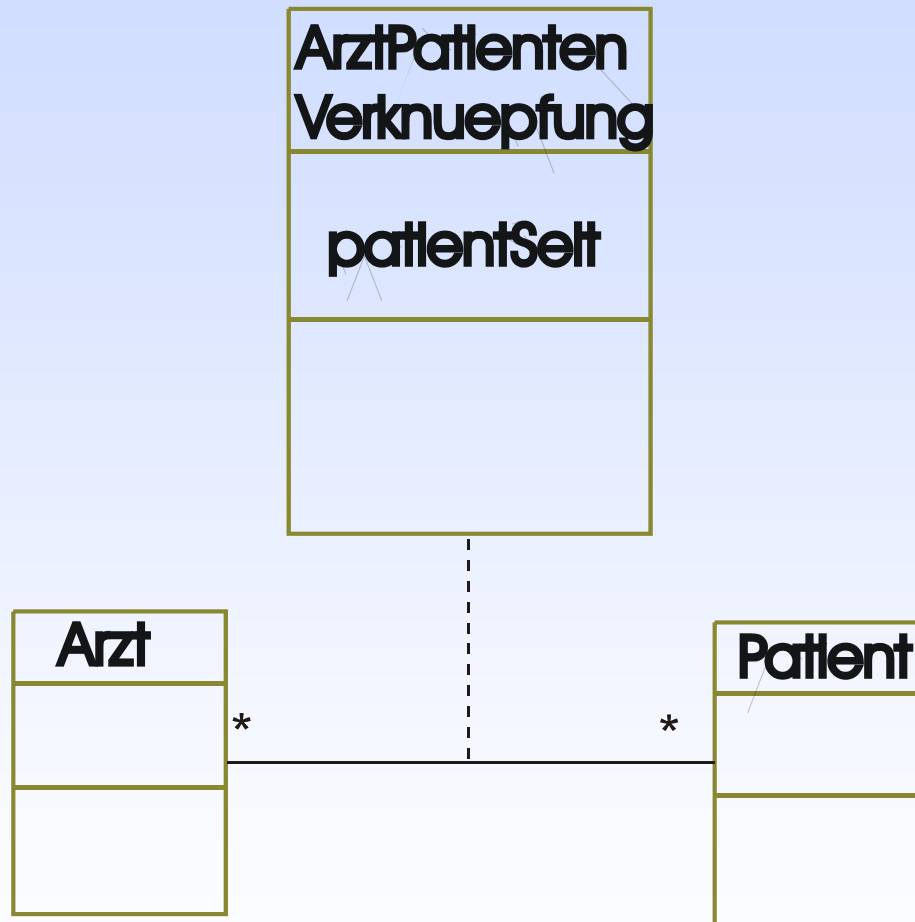
```
JAVA *.java

public class AssoziationNzuM {

    public static void main(String[] args) {
        Arzt arzt1= new Arzt();
        Patient patient1 = new Patient();
        arztPatientenVerknuepfung arztPatientenListe=
            new ArztPatientenVerknuepfung();
        int index;
        arzt1.Name="Meyer";
        arzt1.Nummer=1234;
        patient1.Name="Mueller";
        patient1.Nummer=4321;
        arztPatientenListe.VerbindeArztPatienten(
            arzt1, patient1);

        index=0;
        arztPatientenListe.AusgabeDaten(index);
    }
}
```

# Assoziation n:m (2)



```
JAVA *.java
```

```
public class Arzt {

    String name;
    int nummer;
    public Arzt() {
    }

}

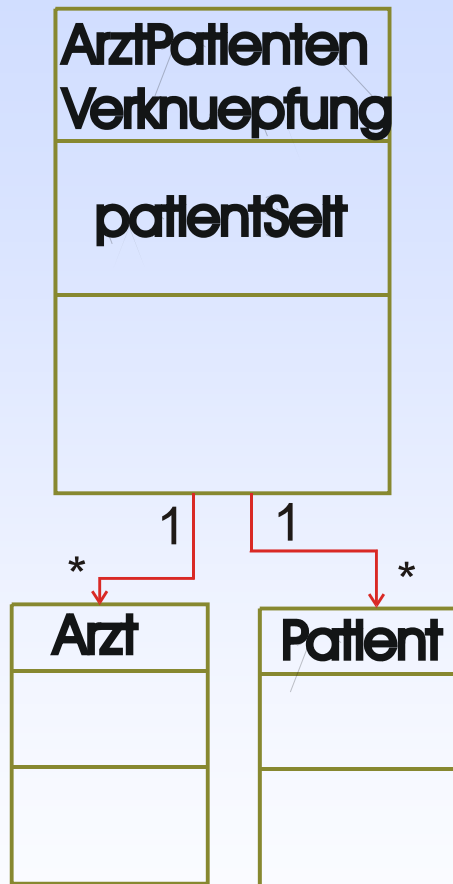
public class Patient {

    String name;
    int nummer;
    public Patient() {
    }

}
```

# Assoziation n:m (3)

Mit attributierter Verknüpfungsklasse



```
JAVA *.java

public class ArztPatientenVerknuepfung {
    static int index=0;
    protected Arzt arztliste[] = new Arzt[10];
    protected Patient patientenliste[] = new Patient[10];
    Date patientSeit;

    public void VerbindeArztPatienten(Arzt neuerArzteintrag,
    Patient neuerPatienteneintrag){
        arztliste[index] = neuerArzteintrag;
        patientenliste[index] = neuerPatienteneintrag;
        index++;
    }

    public void AusgabeDaten(int index){
        System.out.println(arztliste[index].name);
        System.out.println(arztliste[index].nummer);
        System.out.println(patientenliste[index].name);
        System.out.println(patientenliste[index].nummer);
    }
    public ArztPatientenVerknuepfung() {
    }
}
```

# Aggregation und Komposition

**Aggregationen und Kompositionen** sind spezielle Assoziationen, die "Teile/Ganzes"-Beziehungen bzw. "Hat-eine"-Beziehungen oder „besteht aus“ – Beziehungen darstellen.

Aggregation IST eine spezielle Art der Assoziation. Die Modellierung ist nicht immer abgrenzbar

Bei der Aggregation können die "Teile" des "Ganzen" auch einzeln existieren.

Bei der Komposition können Teile nur existieren, wenn auch das "Ganze" existiert. Komposition ist daher angebracht, wenn das Teil nur einem Objekt zugeordnet ist und ohne dieses nicht leben kann.

# Aggregation (1)



**JAVA \*.java**

In Main:  
//Erzeugen der Instanz außerhalb der Klasse  
Messkabel Messkabel1= new Messkabel();

```
public class Messgeraet {
    If Messkabel1.istAngeschlossen()==true
        {...}
    public Messgeraet() {
    }
}

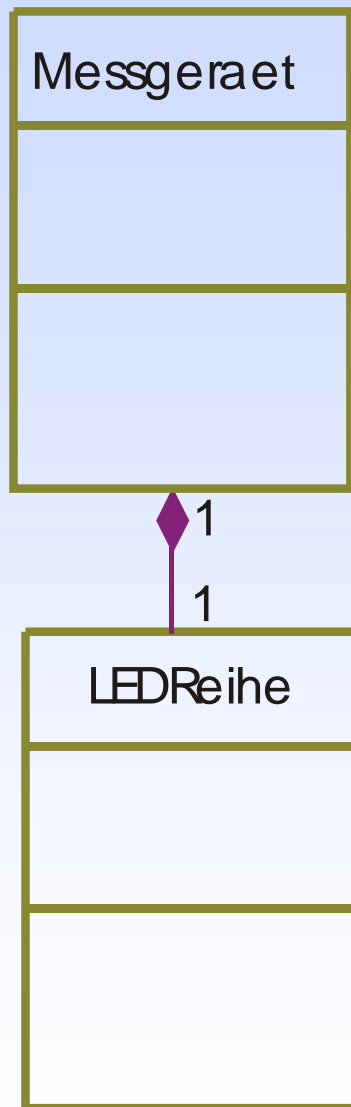
public class Messkabel {
    public Messkabel(){

    }

    public boolean istAngeschlossen(){
        return true;
    }
}
```



# Komposition (1)



**JAVA \*.java**

```
public class Messgeraet {
    //Erzeugen der Instanz in der Klasse
    LEDReihe mgLedReihe= new LEDReihe();

    public Messgeraet() {
    }
}

public class LEDReihe {

    public LEDReihe(){
    }

    public void ansteuern(int nummerLED){
        //spezifische Ansteuererroutine
    }
}
```

# **Aufgabe**

## **UML-Diagramme**

# Tätigkeiten der Implementierung

- Umsetzung der Modelle mit einer Programmiersprache
- Iterative Verfeinerung
- Test
- Fortlaufende Dokumentation

# Prinzipien der Implementierung

- Verbalisierung
- Problemadäquate Datentypen
- Schrittweise Verfeinerung
- Integrierte Dokumentation

# Verbalisierung

## Aussagekräftige Namensgebung

// wenig aussagekräftige Bezeichner  
feld1,  
liste,  
zähler

← **problemfreie,  
technische Bezeichner**

// besser  
messreihe1,  
bestellPositionen,  
anzahlAnZeichen

← **problembezogene  
Bezeichner**

# Verbalisierung

## Aussagekräftige Namensgebung

```
// Berechnung einer Prämie
```

```
p = g1 + z * d;
```



Ohne Aussagekraft,  
zu kurz

```
// besser
```

```
prämie = grundprämie1 +  
zulage * dienstjahre;
```



klar

# Verbalisierung

## Symbolische Konstanten

```
// Konstante
```

```
prämie =
```

```
50.0 +
```

```
10.0 * dienstjahre;
```



**unverständliche  
Konstante**

```
// besser
```

```
final float grundprämie = 50.0;
```

```
final float zulage = 10.0;
```

```
prämie = grundprämie +
```

```
zulage * dienstjahre;
```



**klar**

# Verbalisierung integrierte Kommentare

```
// i um eins erhöhen
```

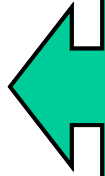
```
i = i + 1;
```



sagt bereits der  
Programcode

```
// besser
```

```
menge = menge + 1;
```

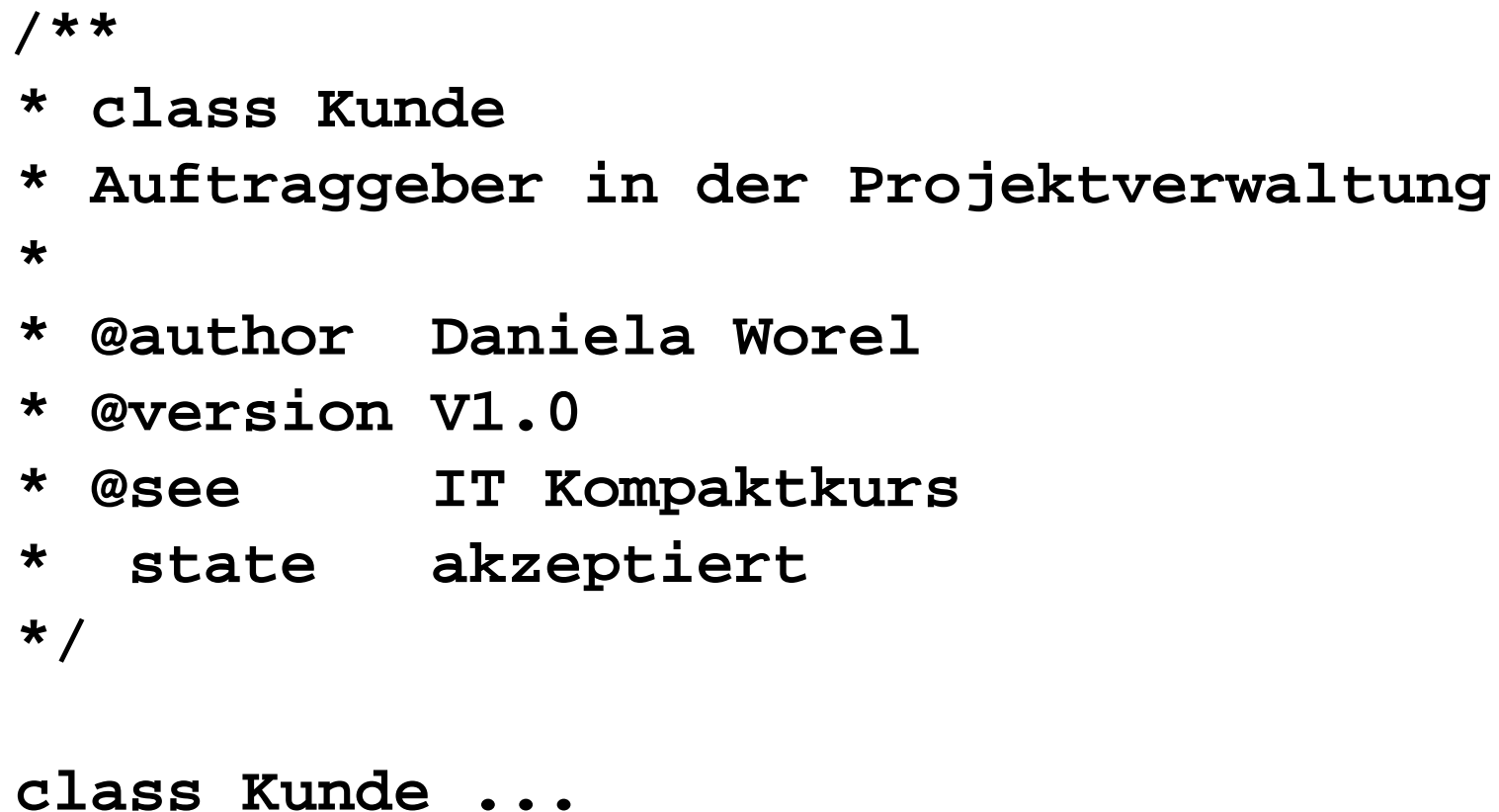


dokumentierender  
Programcode



# Integrierte Dokumentation

## Programmkopf



```
/**  
 * class Kunde  
 * Auftraggeber in der Projektverwaltung  
 *  
 * @author Daniela Worel  
 * @version V1.0  
 * @see IT Kompaktkurs  
 * state akzeptiert  
 */  
  
class Kunde ...
```