FernUniversität in Hagen

FAKULTÄT FÜR
MATHEMATIK UND INFORMATIK

Lehrgebiet Programmiersysteme
Prof. Dr. Friedrich Steimann

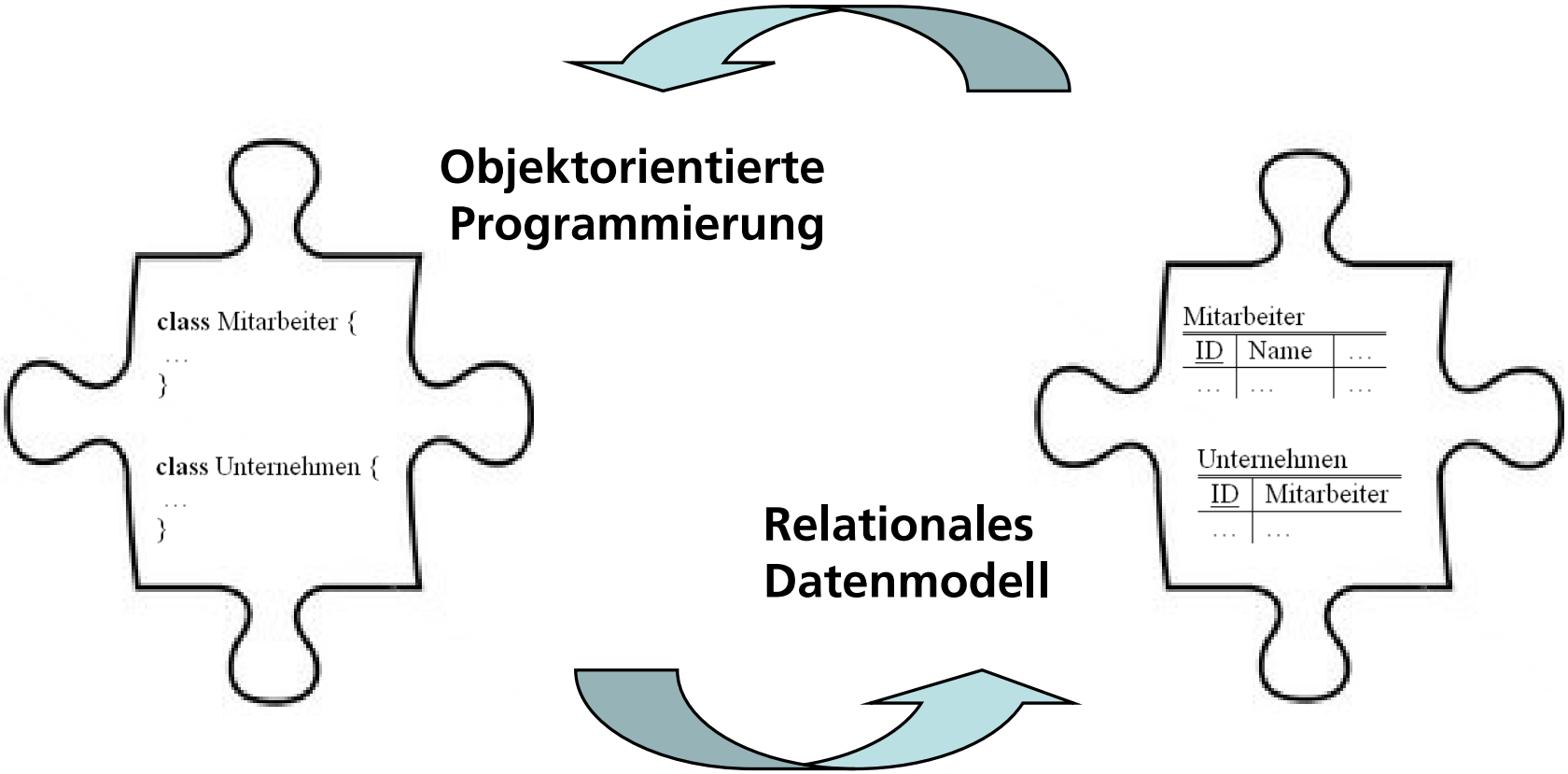
Objektrelationale Programmierung

Dilek Stadtler
Friedrich Steimann

Inhalt

- **Motivation und Problembeschreibung**
  - **Das Problem der Unidirektionalität**
  - **Das Problem der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen**
- **Die Idee der Objektrelationalen Programmierung**
- **Implementierungsansätze**
- **Zusammenfassung**

OOP vs. RDM

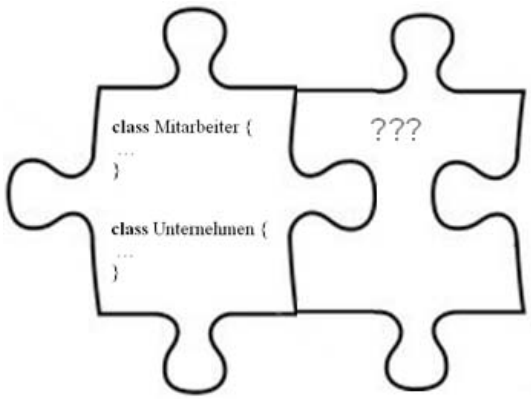

Objektorientierte Programmierung
class Mitarbeiter {
...
}
class Unternehmen {
...
}
Relationales Datenmodell
Mitarbeiter
ID | Name | ...
... | ... | ...
Unternehmen
ID | Mitarbeiter
... | ...

Viele Ansätze mit unterschiedlichen Zielen

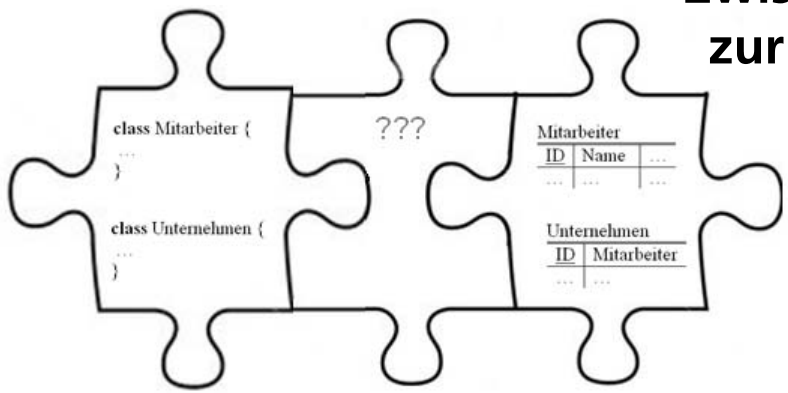
- **Objekt-relationale Datenbanken (z.B. Oracle Database)**
- **Objekt-relationales Mapping (z.B. Hibernate)**
- **Integrierte Systeme (z.B. Tycoon-2: Schmidt, Matthes et al. …)**
- **Erweiterungen von OO-Sprachen um Relationen (z.B. Rumbaugh 1987, Bierman&Wren 2005, Østerbye 2007, …)**
- **Umsetzung von Relationen durch automatische Codegenerierung (z.B. Amelunxen 2004)**
- **Umsetzung von Relationen durch Pattern (Genova 2003, Gessenharter 2009)**
- **Relationale Abfragesprache (auch) für Collection-basierte Datenstrukturen (z.B. LINQ)**
- **…**

Klassifikation verschiedenster Ansätze

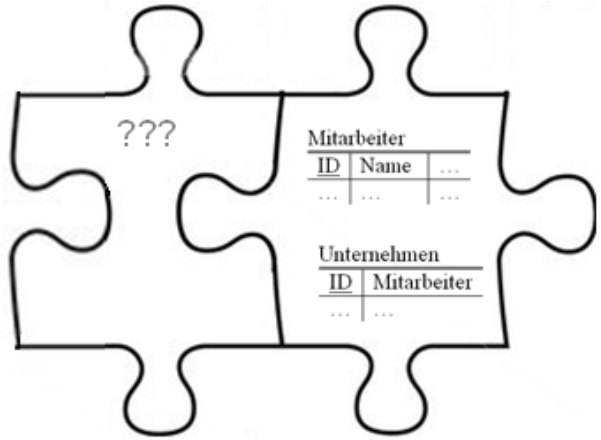
Erweiterung der Objektorientierung



Zwischenschicht zur Umsetzung

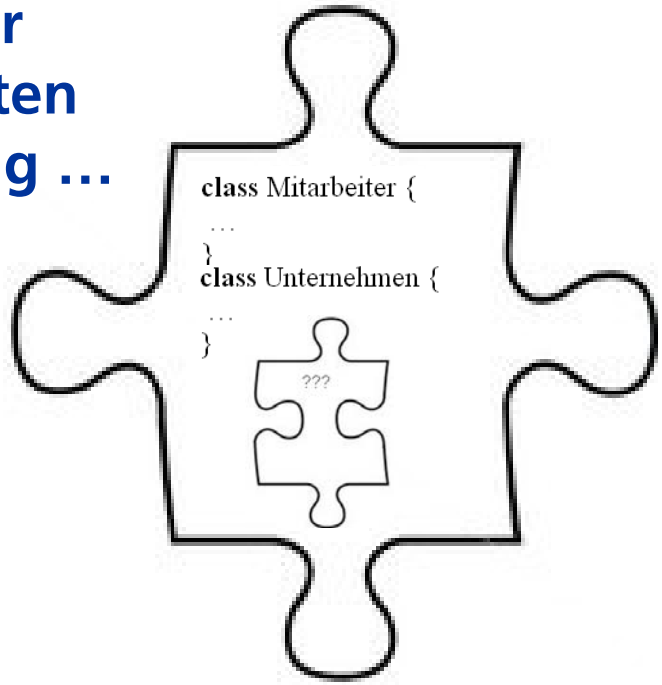


Erweiterung des relationalen Datenmodells



Unser Ziel

Minimal-invasive Erweiterung der objektorientierten Programmierung …



… so dass relationale Elemente integriert werden …

… und die Vorteile einer relationalen Sicht (weitestgehend) übernommen werden

Integration bereits weit fortgeschritten

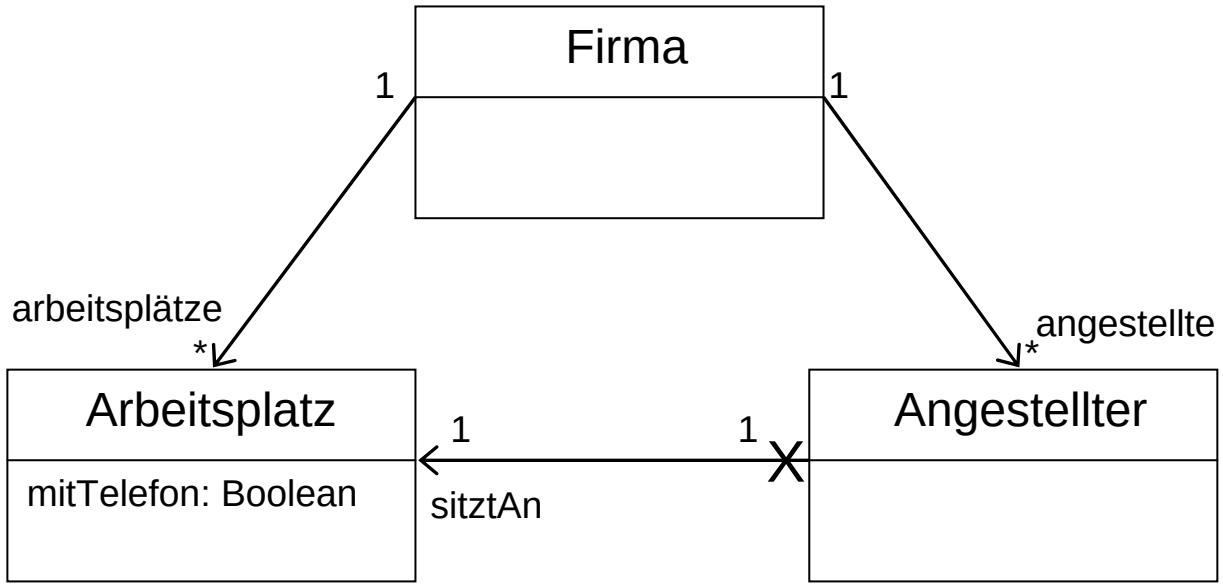
- **z. B. LINQ - Language INtegrated Query in .NET**
- **deklarative, relationale Abfragen auf Mengen in der Syntax der Programmiersprache**
- **Abfragen gegen Datenbanken (LINQ to SQL, LINQ to Entities)**

aber auch:

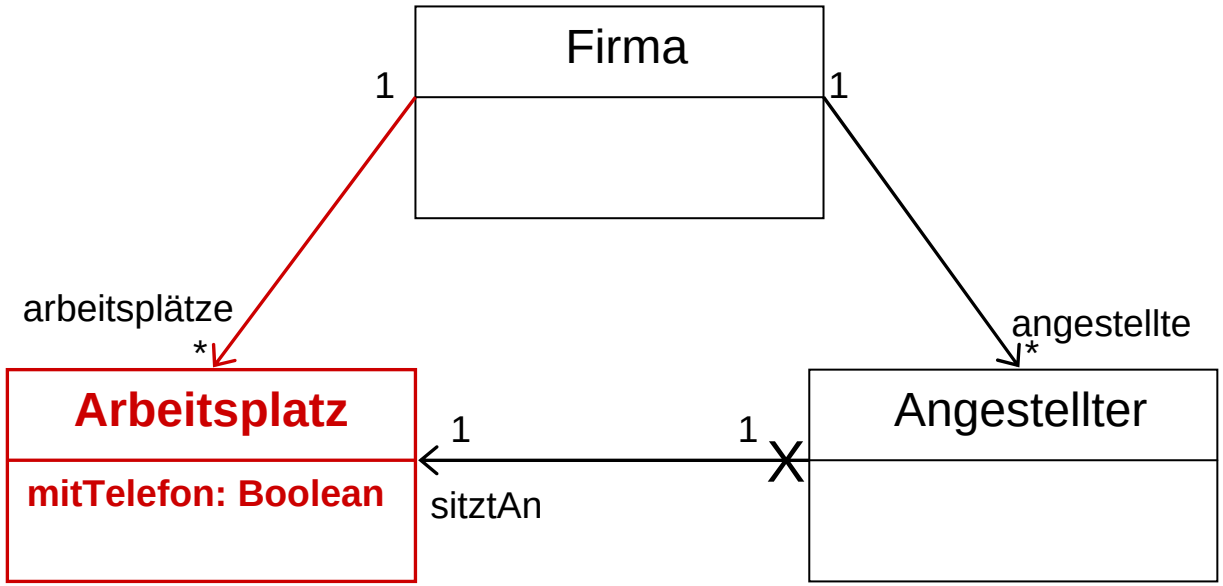
- **LINQ to Objects: Sprachinterne Abfragen nicht gegen Datenbanken, sondern gegen speicherresidente Collections**

  (gegen alle Objekte die die Schnittstelle IEnumerable **implementieren)**

Ein Beispiel-Modell



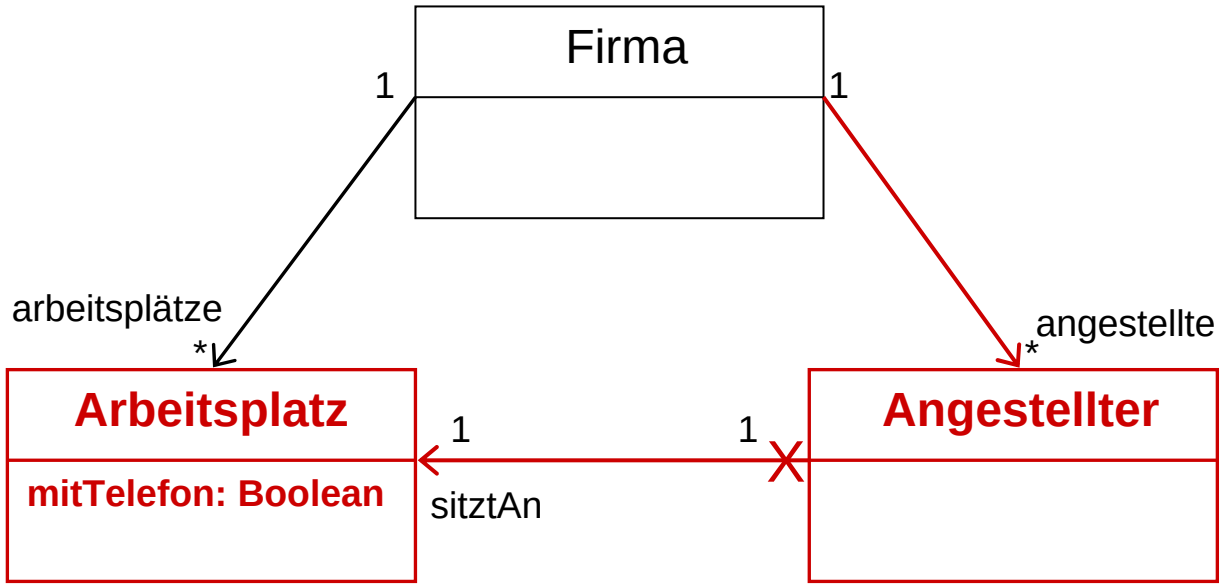
Relationale Abfragen mit LINQ



Welche Arbeitsplätze mit Telefon hat eine Firma?

```
var aplzMitTel = from a in arbeitsplätze
                 where a.mitTelefon
                 select a;
```

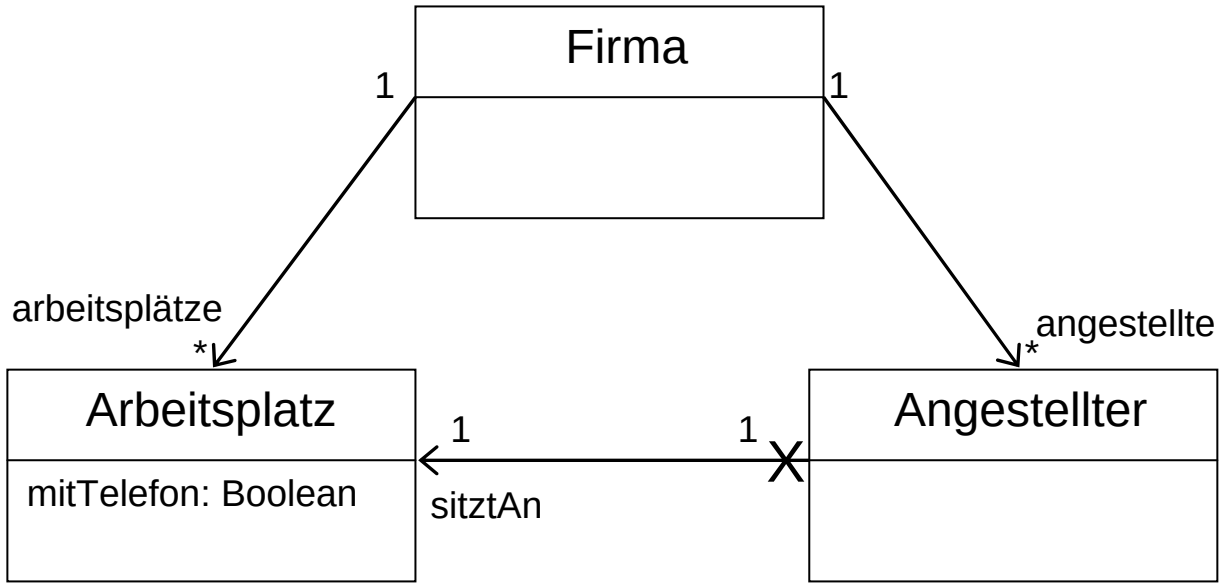
Relationale Abfragen mit LINQ



Welche Angestellte einer Firma haben einen Arbeitsplatz mit Telefon?

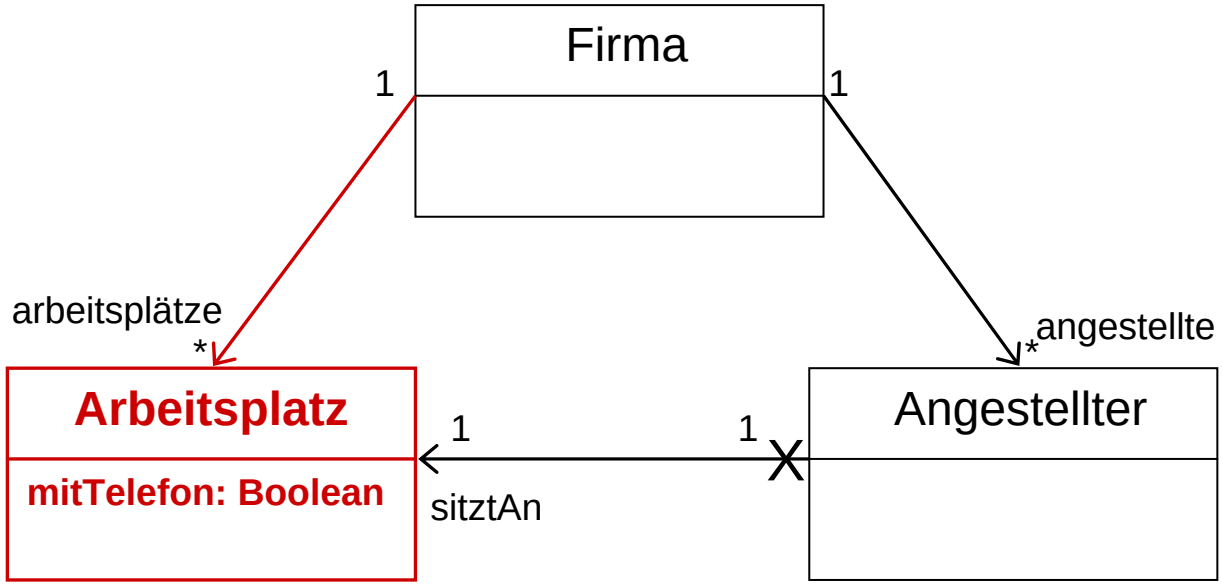
```
var agstMitTel =  from a in angestellte
                  where a.sitztAn.mitTelefon
                  select a;
```

Das Problem der Unidirektionalität



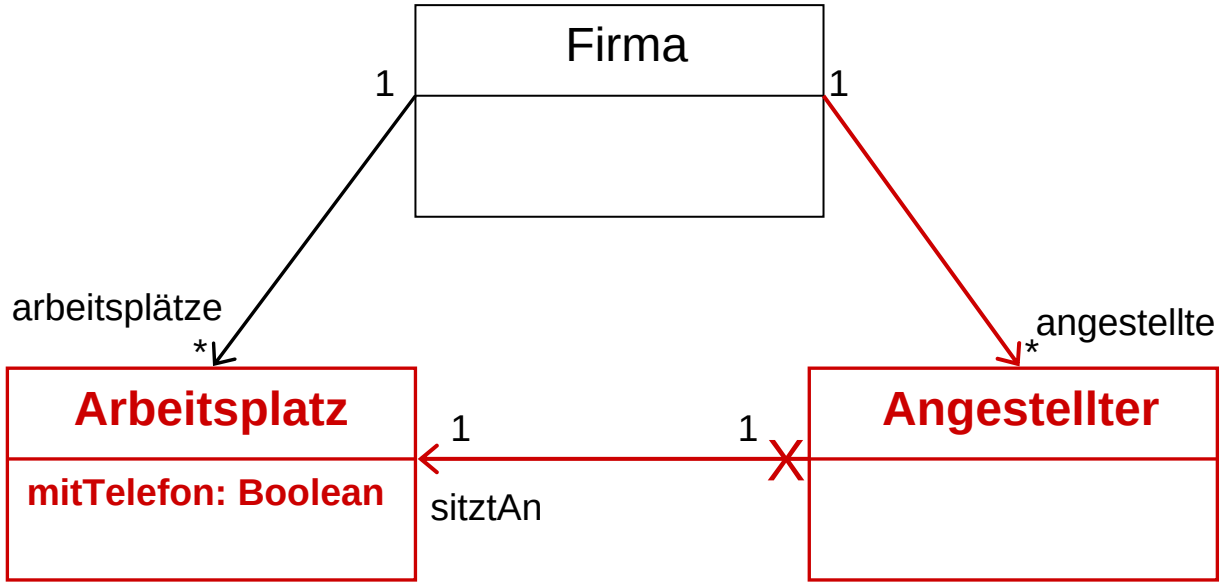
Welche *freie* Arbeitsplätze mit Telefon hat eine Firma?

Das Problem der Unidirektionalität



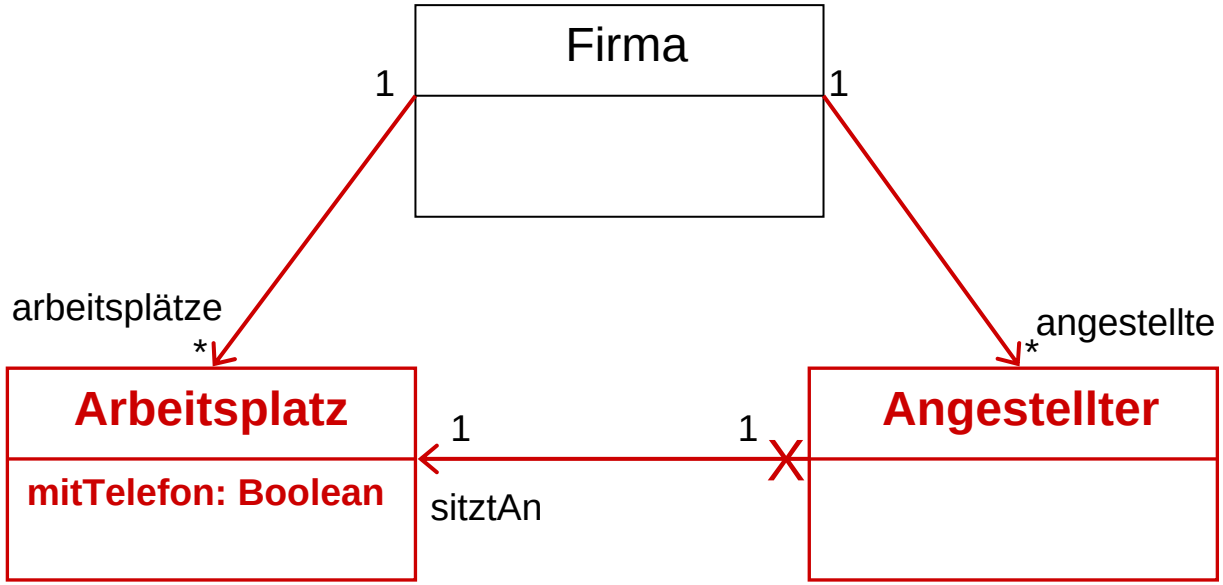

```
var aplzMitTel =      from a in arbeitsplätze
                      where a.mitTelefon
                      select a;

var agstArbt =        from a in angestellte
                      where a.sitztAn.mitTelefon
                      select a.sitztAn;

var freiAplzMitTel =  aplzMitTel.Except(agstArbt);
```

Das Problem der Unidirektionalität



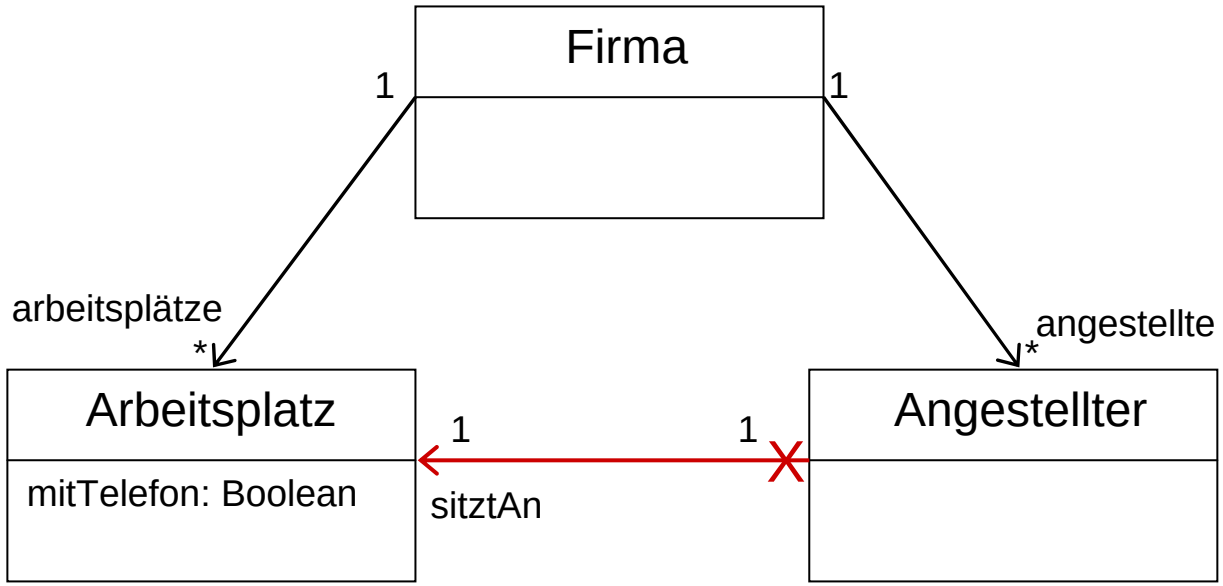

```
var aplzMitTel =      from a in arbeitsplätze
                      where a.mitTelefon
                      select a;

var agstArbt =        from a in angestellte
                      where a.sitztAn.mitTelefon
                      select a.sitztAn;

var freiAplzMitTel =  aplzMitTel.Except(agstArbt);
```

Das Problem der Unidirektionalität

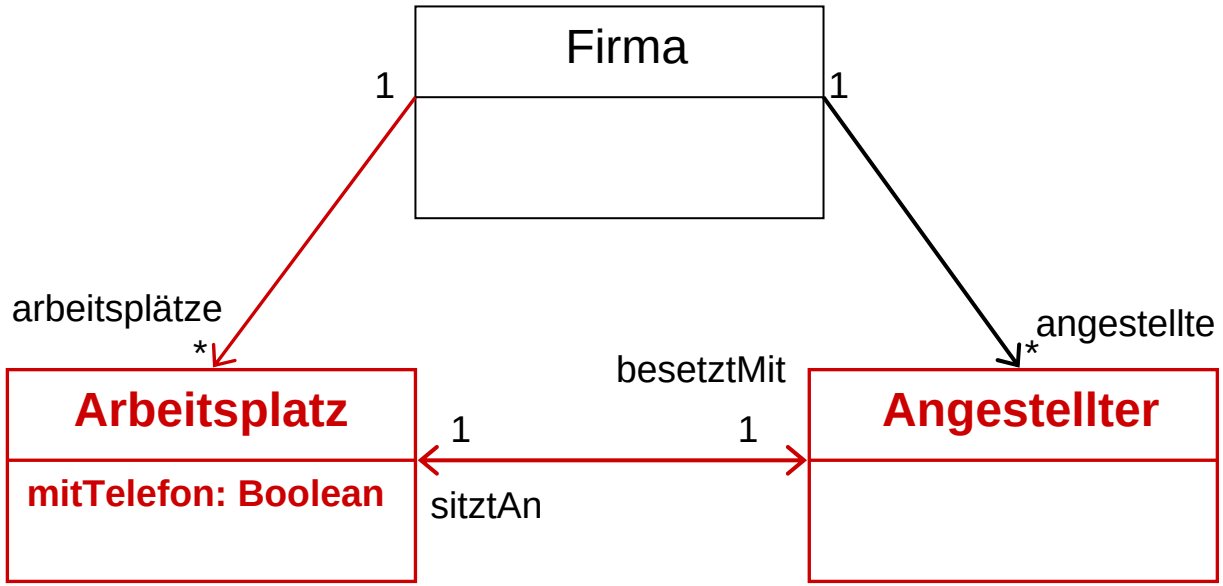



```
var aplzMitTel =      from a in arbeitsplätze
                      where a.mitTelefon
                      select a;

var agstArbt =        from a in angestellte
                      where a.sitztAn.mitTelefon
                      select a.sitztAn;

var freiAplzMitTel =  aplzMitTel.Except(agstArbt);
```

Das Problem der Unidirektionalität



Das Problem der Unidirektionalität



Welche *freie* Arbeitsplätze mit Telefon hat eine Firma?

```
var freiAplzMitTel =  from a in arbeitsplätze
                      where a.mitTelefon && a.besetztMit == null
                      select a;
```

Das Problem der Unidirektionalität

```
class Firma { … }

class Angestellter {
    Arbeitsplatz sitztAn;
}

class Arbeitsplatz {
    bool mitTelefon;
    Angestellter besetztMit;
}
```

```
Angestellter a = new Angestellter();
Arbeitsplatz b = new Arbeitsplatz();

a.sitztAn = b;
b.besetztMit= a;
```

~~
```
void setArbeitsplatz(Arbeitsplatz a) {
    this.sitztAn = a;
    a.setBesetztMit(this); }

void setAngestellter(Angestellter a) {
    this.besetztMit = a;
    a.setSitztAn(this);}
```
~~

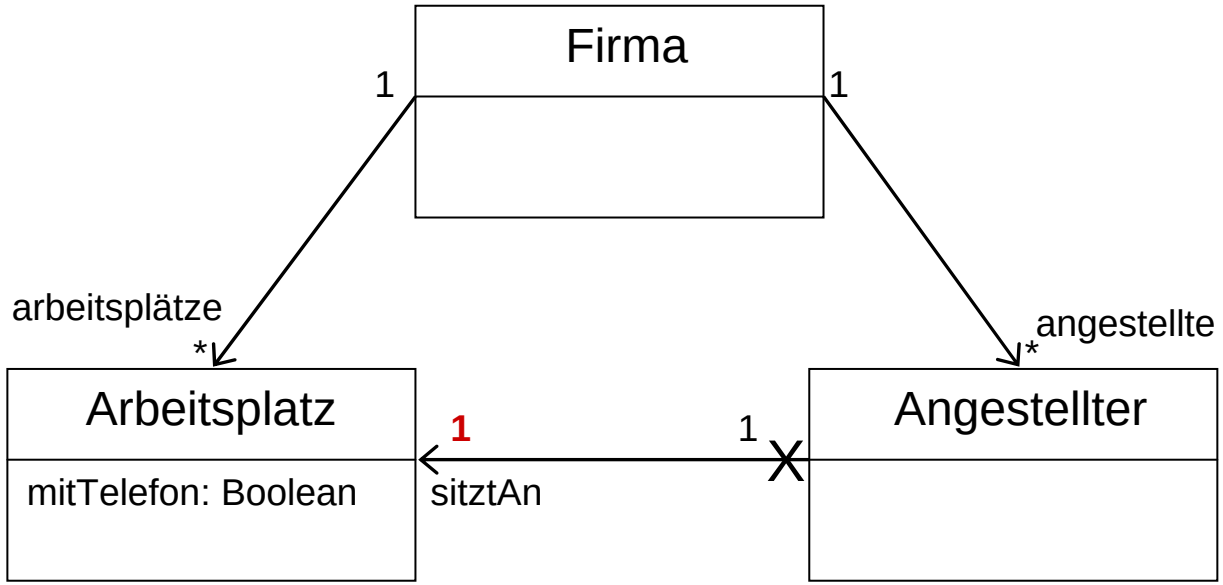
Anforderungen an eine Erweiterung der objektorientierten Programmierung

1. **Einführung von bidirektionalen Beziehungen (grundsätzliche Navigierbarkeit in beide Richtungen wie bei Relationen)**

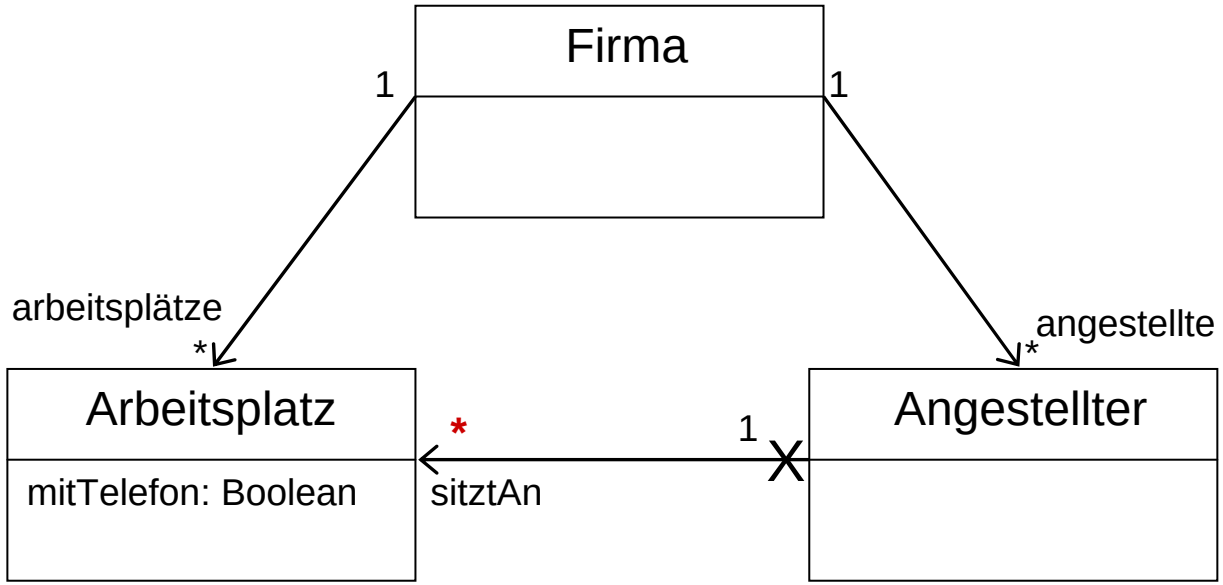
Inhalt

- **Motivation und Problembeschreibung**
  - ✓ **Das Problem der Unidirektionalität**
  - **Das Problem der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen**
- **Die Idee der Objektrelationalen Programmierung**
- **Implementierungsansätze**
- **Zusammenfassung**

Das Problem der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen



Das Problem der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen



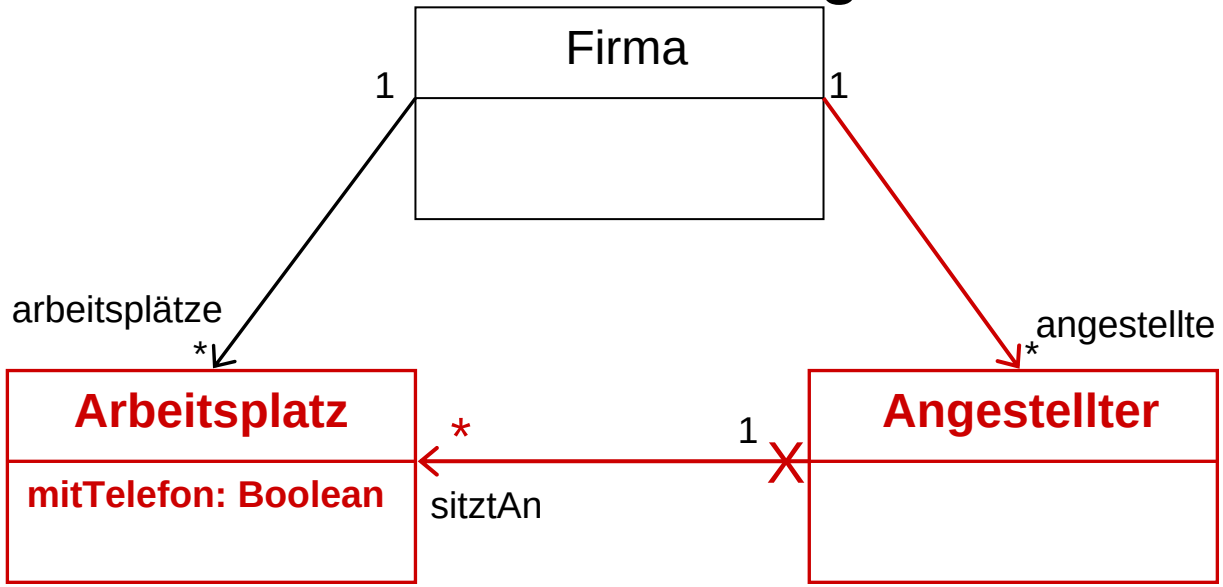
Das Problem der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen

```
class Firma { … }

class Angestellter {
    Arbeitsplatz sitztAn;
}

class Arbeitsplatz { … }
```

Das Problem der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen

```
class Firma { … }

class Angestellter {
    ICollection<Arbeitsplatz> sitztAn =
        new List<Arbeitsplatz>();
}

class Arbeitsplatz { … }
```

Das Problem der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen



Welche Angestellte einer Firma haben einen Arbeitsplatz mit Telefon?

```
var agstMitTel =  from a in angestellte
                  where a.sitztAn.mitTelefon
                  select a;
```

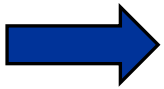
Anforderungen an eine Erweiterung der objektorientierten Programmierung

1. **Einführung von bidirektionalen Beziehungen (grundsätzliche Navigierbarkeit in beide Richtungen wie bei Relationen)**

2. **Aufhebung der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen**

Anforderungen

1. **Einführung von bidirektionalen Beziehungen (Relationen)**
2. **Aufhebung der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen**



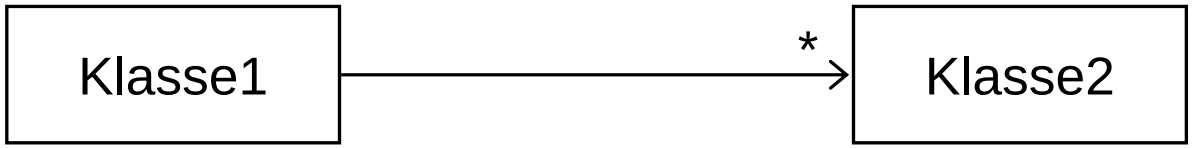
Trotz weitestgehender Anbindung des RDM an die Objektorientierung:

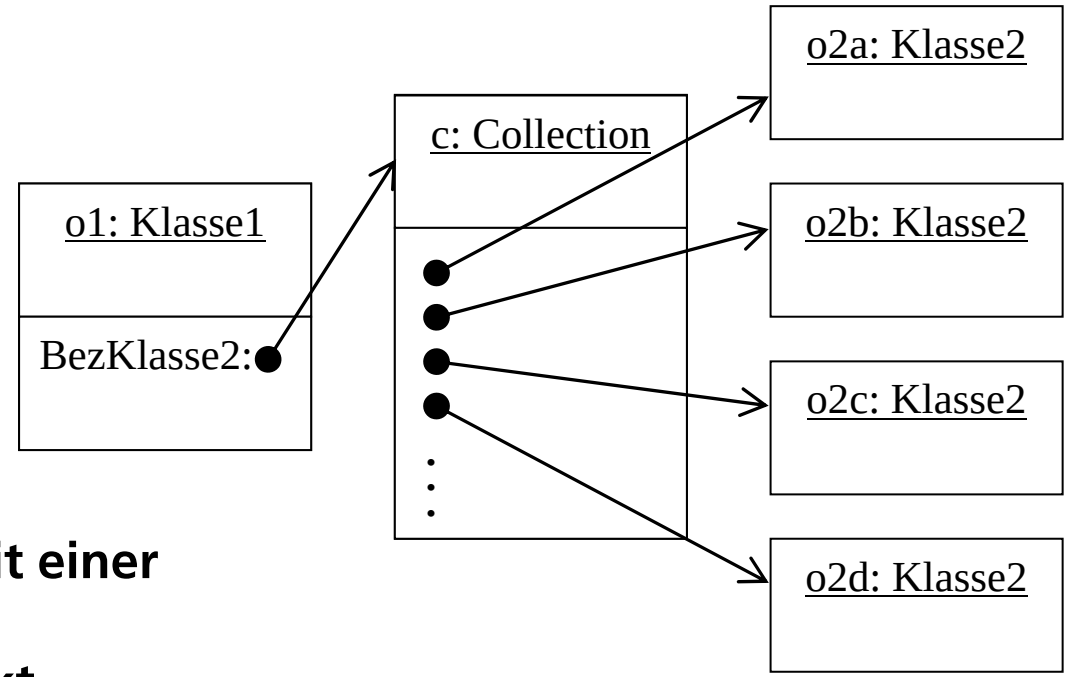
Es bleiben mindestens zwei Brüche zwischen der Verwendung von externen relationalen Datenquellen und speicherresidenten Datenstrukturen (Zeigern und Collections)

Inhalt

- ✓ **Motivation und Problembeschreibung**
  - ✓ **Das Problem der Unidirektionalität**
  - ✓ **Das Problem der Unterscheidung von Zu-1- und Zu-*n*-Beziehungen**
- **Die Idee der Objektrelationalen Programmierung**
- **Implementierungsansätze**
- **Zusammenfassung**

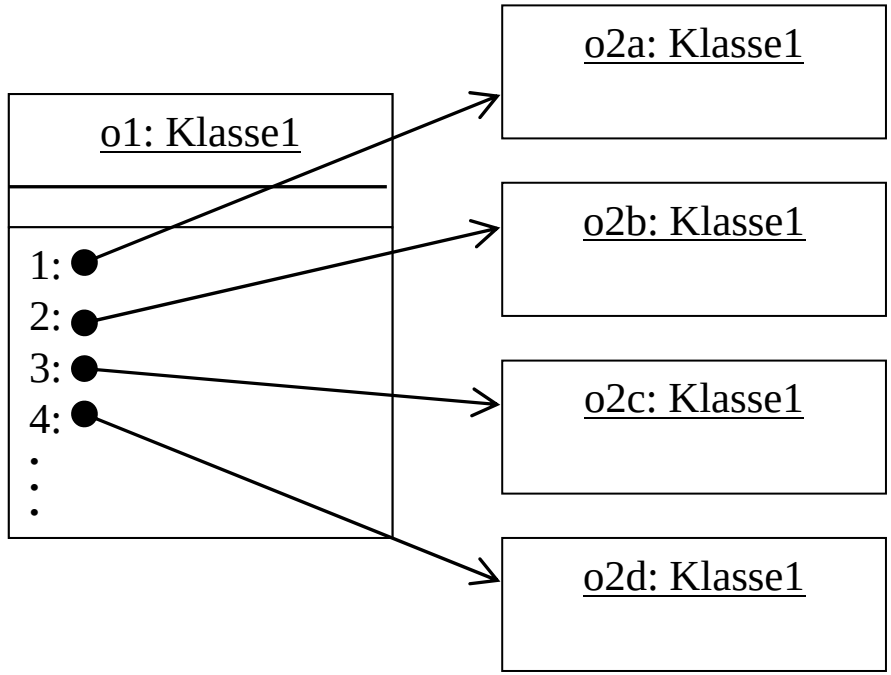
Gängige Umsetzung von Zu-*n*-Beziehungen





Umsetzung mit einer Collection als Zwischenobjekt

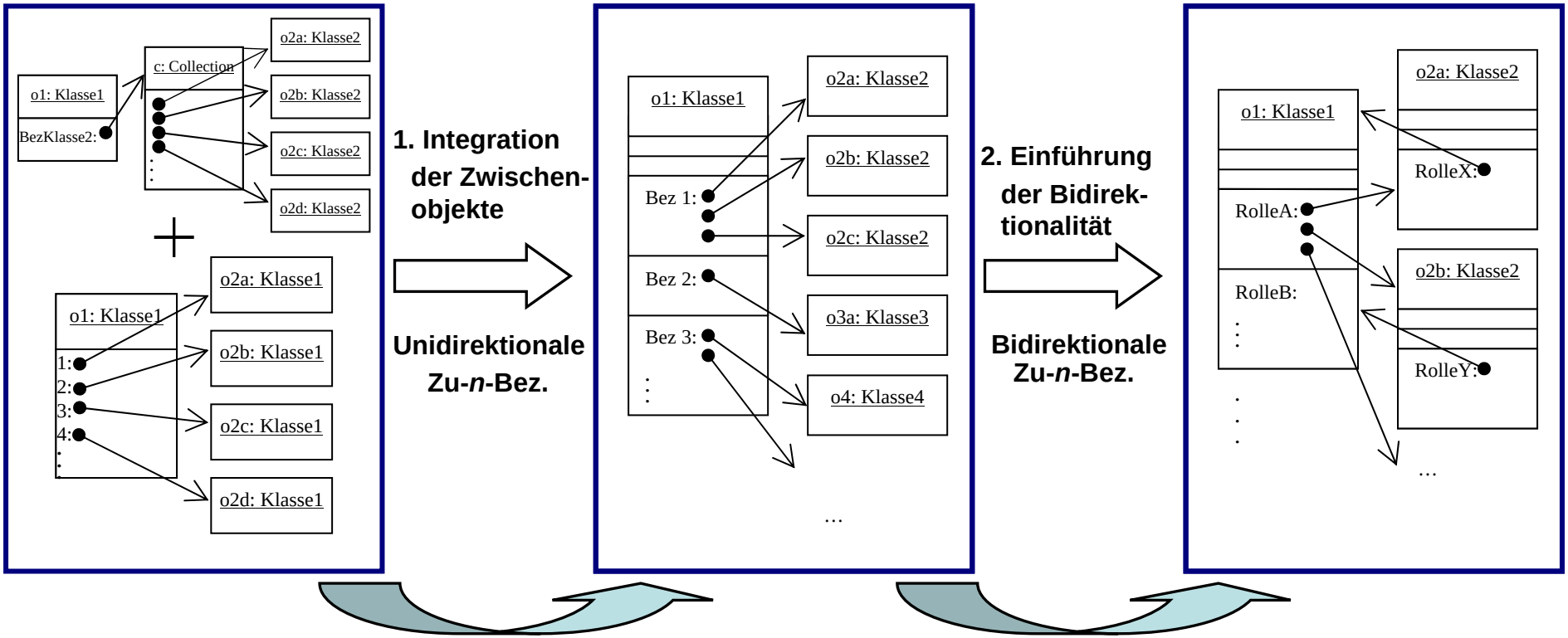
Gängige Umsetzung von Zu-*n*-Beziehungen



Umsetzung anhand von indizierten (anstatt benannten) Instanzvariablen (Smalltalk)

Nachteil: Nur eine Beziehung pro Objekt

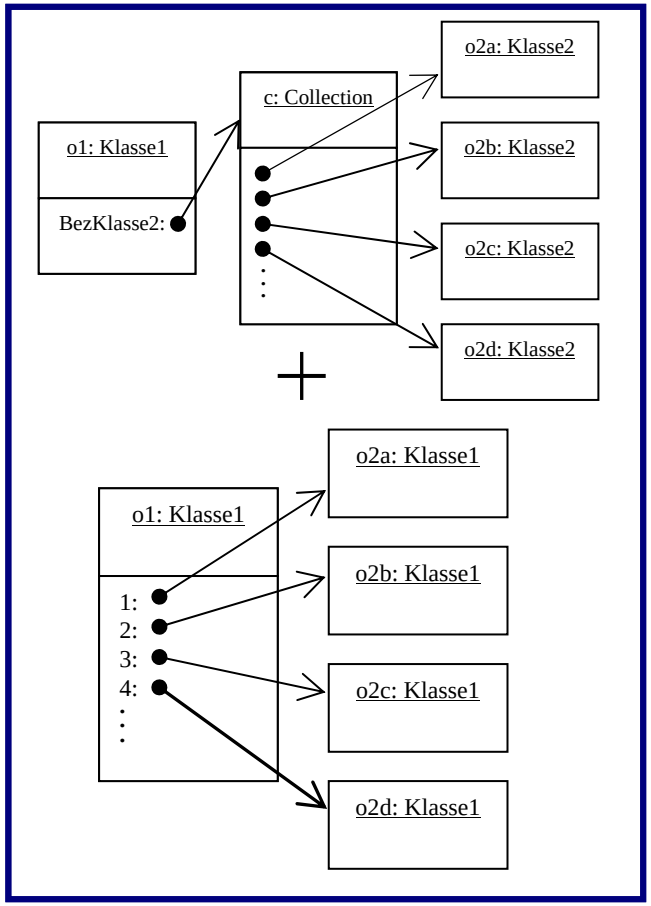
Zu-*n*-Beziehungen in Objektrelationalem Programmiermodell



⇨ **Vereinheitlichung von Zu-1- und Zu-*n*-Beziehungen**

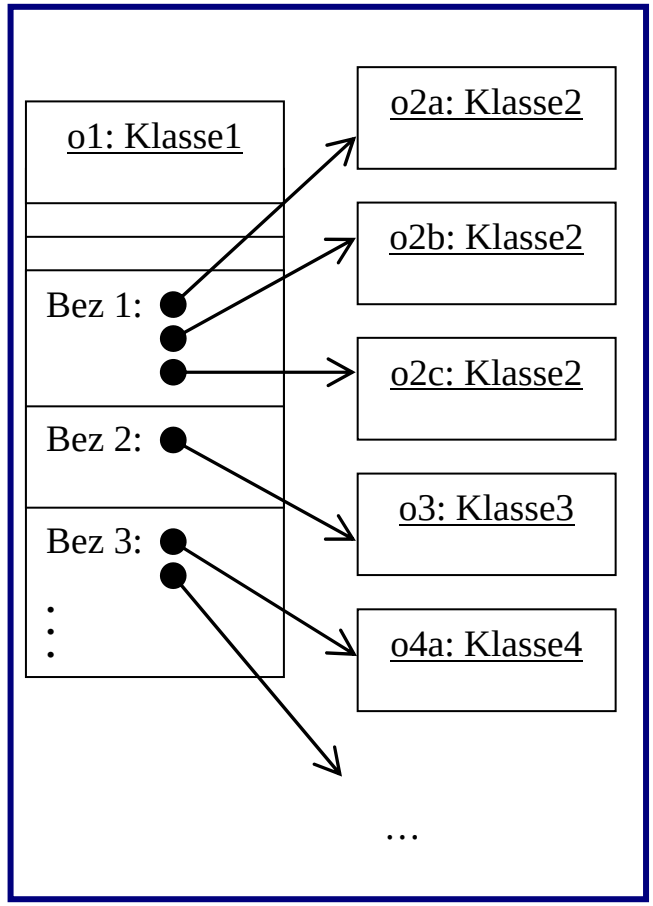
⇨ **Bidirektionale Zu-*n*-Beziehungen**

1. Schritt: Integration der Zwischenobjekte in die Objekte selbst

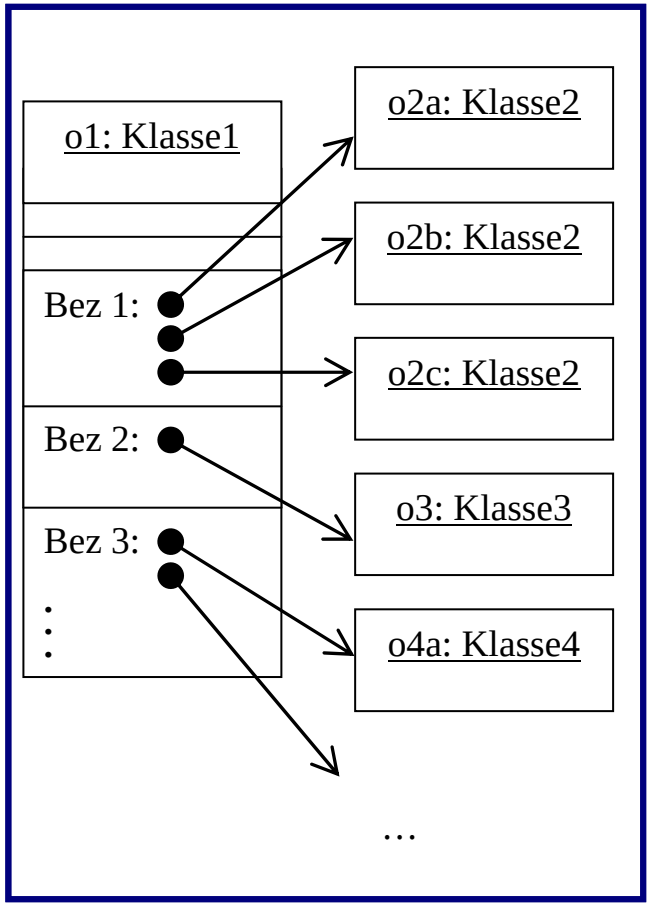


Unidirektionale
Zu-*n*-Beziehungen

Vereinheitlichung von Zu-1- und Zu-*n*-Beziehungen

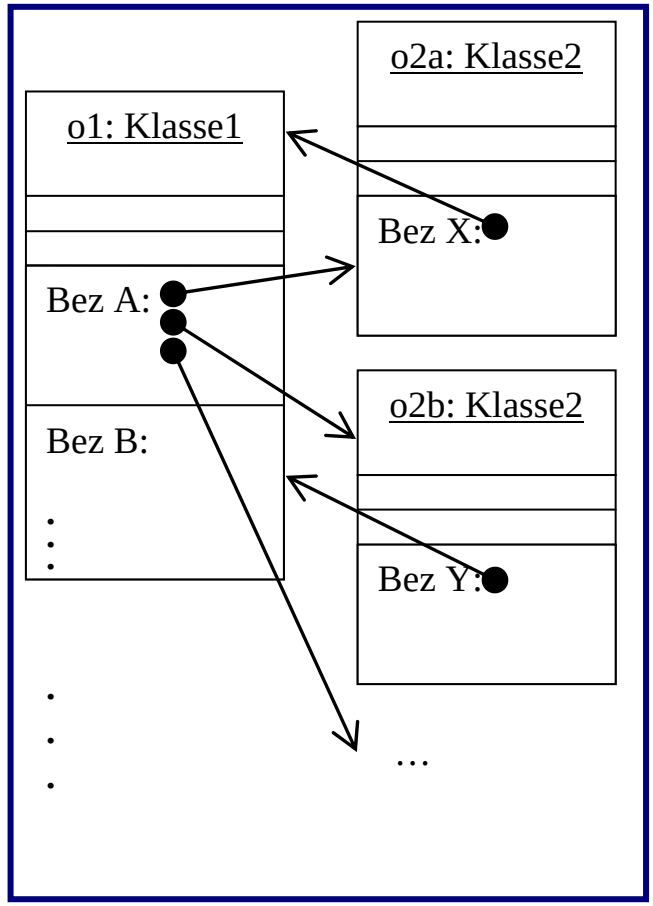


2. Schritt: Einführung der Bidirektionalität

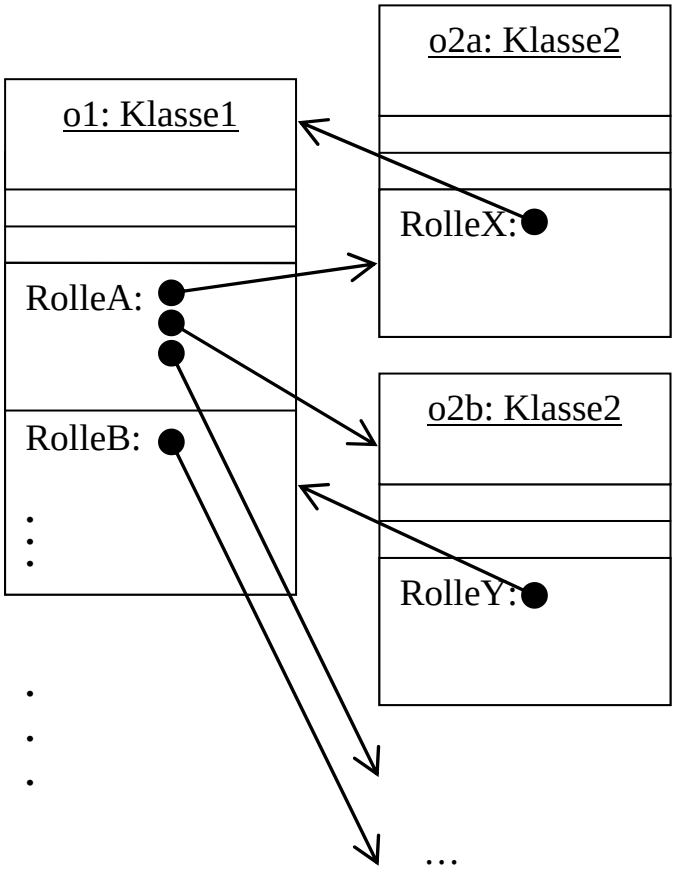


Bidirektionale
Zu-*n*-Beziehungen

Einführung der Bidirektionalität



2. Schritt: Einführung der Bidirektionalität



Syntax

Rollendeklaration der Form:

Bez1 = (*RolleA*, *RolleX*)

Operationen für eine Rolle *R*:

$+=_R$
$-=_R$
} entsprechen :=

$\$_R$ entspricht Feld-Dereferenzierung mit *R* als Feldname und ist Iterable

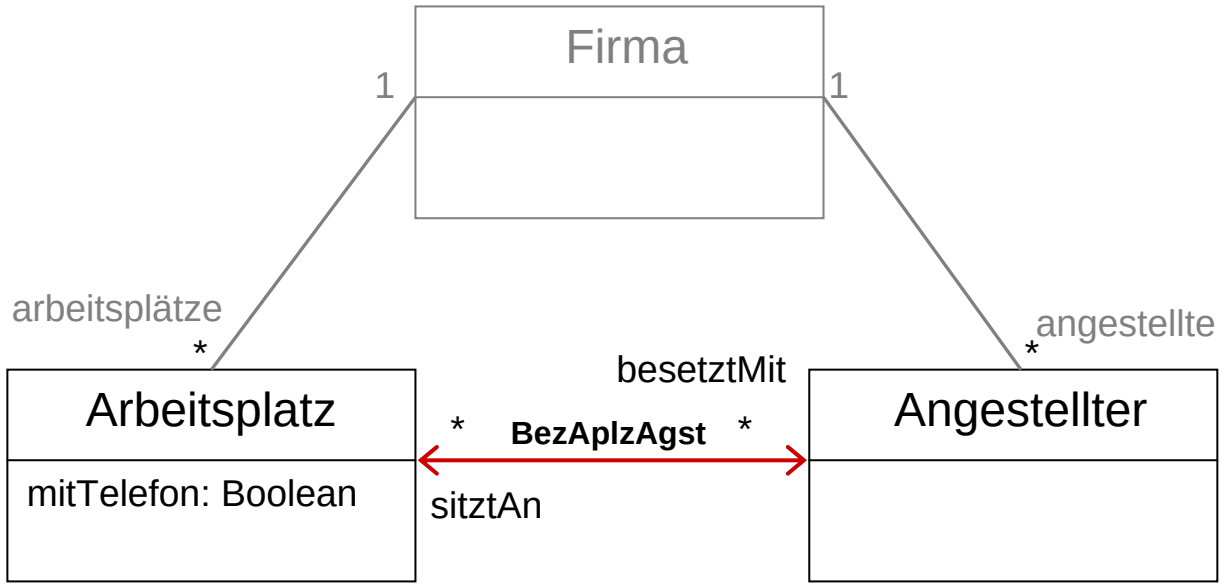
Semantik

wenn *S* = counterrole(*R*):

$a +=_R b \leftrightarrow b +=_S a$

$a +=_R b \rightarrow b$ **in** $a \$_R \wedge a$ **in** $b \$_S$

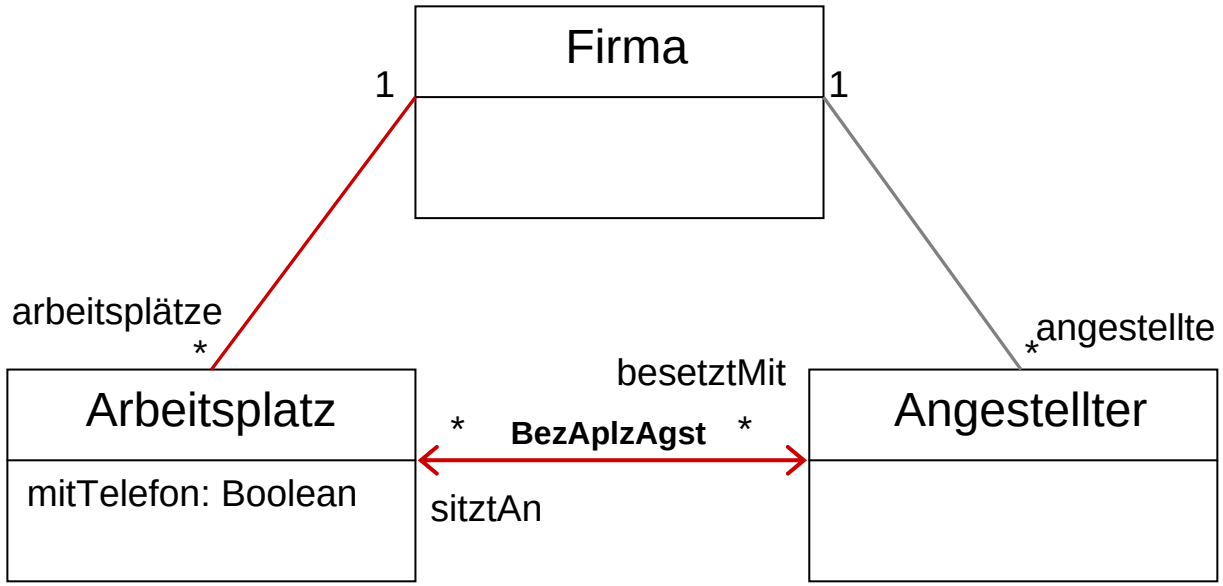
Noch mal das Beispiel



BezAplzAgst = (*besetztMit*, *sitztAn*)

platz1 $+=_{besetztMit}$ herrM $\Leftrightarrow$ herrM $+=_{sitztAn}$ platz1

Noch mal das Beispiel



Welche *freie* Arbeitsplätze mit Telefon hat eine Firma?

```
var freiAplzMitTel =  from a in $arbeitsplätze
                      where a.mitTelefon && a.besetztMit.Count==0
                      select a;
```

Inhalt

- ✓ **Motivation und Problembeschreibung**
  - ✓ **Das Problem der Unidirektionalität**
  - ✓ **Das Problem der Unterscheidung von Zu-1- und Zu-n-Beziehungen**
- ✓ **Die Idee der Objektrelationalen Programmierung**
- **Implementierungsansätze**
- **Zusammenfassung**

Implementierung I

- **Problem: Allokierung von Speicher zur Aufnahme der „bidirektionalen Zu-*n*-Zeiger“ für jedes Objekt**
  - dasselbe Problem wie Arrays variabler Größe
- **Umsetzung in Smalltalk**
  - Relationsdeklarationen als Abbildungen von Rollennamen auf die Namen ihrer Gegenrollen (Dictionary Relationships)
  - indizierte Instanzvariablen nehmen die Zu-*n*-Pointer auf
  - Erweiterung der Klasse Object um Methoden für $+=_R$, $-=_R$, $\$_R$
  - Erweiterung der Klasse Class um Methoden zur Erzeugung von Klassen mit Relationen

Implementierung II

- **Umsetzung in Scala**
  - Verlagerung der Beziehungen in Rollen als Traits
  - Rollentraits stellen Speicher und Operationen für Umsetzung der Relationen bereit
  - Rollentraits als Typen bieten Typsicherheit

- **Future Work**
  - Umsetzung in C#: noch nicht untersucht
  - vermutlich Erweiterung der Virtuellen Maschine notwendig
  - Integration mit LINQ to SQL, LINQ to Entities, Mapping noch nicht untersucht

Zusammenfassung

- **Ziel:**

  Minimal-invasive Erweiterung der objektorientierten Programmierung

  -> native Integration von den relationalen Elementen, die noch fehlen

- **Umsetzung:**

  1. Vereinheitlichung von Zu-1- und Zu-*n*-Beziehungen durch Integration der Zwischenobjekte in die Objekte selbst

  2. Einführung von bidirektionalen Beziehungen (Relationen): Beziehung als Paar von zwei zusammengehörigen unidirektionalen Beziehungen

⇨Weiterhin Query-Sprache (z.B. LINQ) notwendig (fassen wir nicht an)

Vielen Dank
Fragen????

Dilek Stadtler
Friedrich Steimann