Script generated by TTT

Title: groh: profile1 (23.05.2014)

Date: Fri May 23 09:12:52 CEST 2014

Duration: 93:32 min

Pages: 58

Geschachtelte Anfrage

- Unteranfrage in der **where**-Klausel
- Welche Prüfungen sind besser als durchschnittlich verlaufen?

```
select *
from prüfen p
where p.Note < ( select avg (Note)
                 from prüfen );
```


AICOS Technische Universität München

Geschachtelte Anfrage

- Unteranfrage in der **where**-Klausel
- Welche Prüfungen sind besser als durchschnittlich verlaufen?

```
select *
from prüfen p
where p.Note < ( select avg (Note)
                 from prüfen );
```

175

AICOS Technische Universität München

Geschachtelte Anfrage

- Unteranfrage in der **where**-Klausel
- Welche Prüfungen sind besser als durchschnittlich verlaufen?

```
select *
from prüfen p
where p.Note < ( select avg (Note)
                 from prüfen );
```

175

Geschachtelte Anfrage

- Unteranfrage in der **where**-Klausel
- Welche Prüfungen sind besser als durchschnittlich verlaufen?

```
select *
from prüfen p
where p.Note < ( select avg (Note)
                 from prüfen );
```


Geschachtelte Anfrage (Forts.)

- Unteranfrage in der **select**-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt
- Man beachte, dass die Unteranfrage korreliert ist (greift auf Attribute der umschließenden Anfrage zu)

```
select p.PersNr, p.Name, ( select sum (v.SWS) as
                           Lehrbelastung from Vorlesungen v
                           where v.gelesenVon=p.PersNr )
from Professoren p;
```


AICOS Technische Universität München

Geschachtelte Anfrage (Forts.)

- Unteranfrage in der select-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt
- Man beachte, dass die Unteranfrage korreliert ist (greift auf Attribute der umschließenden Anfrage zu)

select p.PersNr, p.Name, (select sum (v.SWS) as Lehrbelastung from Vorlesungen v where v.gelesenVon=p.PersNr) from Professoren p;

176

AICOS Technische Universität München

Geschachtelte Anfrage (Forts.)

- Unteranfrage in der select-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt
- Man beachte, dass die Unteranfrage korreliert ist (greift auf Attribute der umschließenden Anfrage zu)

select p.PersNr, p.Name, (select sum (v.SWS) as Lehrbelastung from Vorlesungen v where v.gelesenVon=p.PersNr) from Professoren p;

176

Geschachtelte Anfrage (Forts.)

- Unteranfrage in der **select**-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt
- Man beachte, dass die Unteranfrage korreliert ist (greift auf Attribute der umschließenden Anfrage zu)

```
select p.PersNr, p.Name, ( select sum (v.SWS) as
                                Lehrbelastung from Vorlesungen v
                           where v.gelesenVon=p.PersNr )
from Professoren p;
```


AICOS Technische Universität München

Geschachtelte Anfrage (Forts.)

- Unteranfrage in der **select**-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt
- Man beachte, dass die Unteranfrage korreliert ist (greift auf Attribute der umschließenden Anfrage zu)

```
select p.PersNr, p.Name, ( select sum (v.SWS) as
                                Lehrbelastung from Vorlesungen v
                           where v.gelesenVon=p.PersNr )
from Professoren p;
```


AICOS Technische Universität München

Geschachtelte Anfrage (Forts.)

- Unteranfrage in der **select**-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt
- Man beachte, dass die Unteranfrage korreliert ist (greift auf Attribute der umschließenden Anfrage zu)

```
select p.PersNr, p.Name, ( select sum (v.SWS) as
                                Lehrbelastung from Vorlesungen v
                           where v.gelesenVon=p.PersNr )
from Professoren p;
```


AICOS Technische Universität München

Entschachtelung korrelierter Unteranfragen durch Join

Welcher Assistent hat einen Boss der jünger ist als er selbst?

```
select a.*
from Assistenten a
where exists
    ( select p.*
      from Professoren p
      where a.Boss = p.PersNr and p.GebJahr>a.GebJahr)
```

- Entschachtelung durch Join

```
select a.*
from Assistenten a, Professoren p
where a.Boss=p.PersNr and p.GebJahr > a.GebJahr;
```

Verwertung der Ergebnismenge einer Unteranfrage

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
      from Studenten s, hören h
      where s.MatrNr=h.MatrNr
      group by s.MatrNr, s.Name)  tmp
where tmp.VorlAnzahl > 2;
```

Wer hört mehr als 2 Vorlesungen?

MatrNr	Name	VorlAnzahl
28106	Carnap	4
29120	Theophrastos	3

AICOS Technische Universität München

Verwertung der Ergebnismenge einer Unteranfrage

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
      from Studenten s, hören h
      where s.MatrNr=h.MatrNr
      group by s.MatrNr, s.Name)  tmp
where tmp.VorlAnzahl > 2;
```

Wer hört mehr als 2 Vorlesungen?

MatrNr	Name	VorlAnzahl
28106	Carnap	4
29120	Theophrastos	3

180

AICOS Technische Universität München

Weitere Anfragen mit Unteranfragen (WDH)

```
( select Name
  from Assistenten )
union
( select Name
  from Professoren );
```

```
select Name
from Studenten
where Semester > = all
( select Semester
from Studenten );
```

```
select Name
from Professoren
where PersNr not in ( select gelesenVon
                      from Vorlesungen );
```


AICOS Technische Universität München

finde die Studenten die alle vierstündigen Vorlesungen hören:

```
select s.*
        from Studenten s
        where not exists
                (select v.*
                from Vorlesungen v
                where v.SWS=4 and not exists
                        (select h.*
                        from hören h
                        where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr));
```

⟶

```
select h.MatrNr
        from hören h
        group by h.MatrNr
                having count (*) = (select count (*)
                        from Vorlesungen v where v.SWS=4);
```

finde die Studenten die alle vierstündigen Vorlesungen hören:

```
select s.*
        from Studenten s
        where not exists
                (select v.*
                from Vorlesungen v
                where v.SWS=4 and not exists
                        (select h.*
                        from hören h
                        where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr));
```

⟶

```
select h.MatrNr
        from hören h
        group by h.MatrNr
                having count (*) = (select count (*)
                        from Vorlesungen v where v.SWS=4);
```


AICOS Technische Universität München

finde die Studenten die alle vierstündigen Vorlesungen hören:

select s.*
from Studenten s
where not exists
(select v.*
from Vorlesungen v
where v.SWS=4 and not exists
(select h.*
from hören h
where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr));

⟶

select h.MatrNr
from hören h
group by h.MatrNr
having count (*) = (select count (*)
from Vorlesungen v where v.SWS=4);

185

AICOS Technische Universität München

finde die Studenten die alle vierstündigen Vorlesungen hören:

select s.*
from Studenten s
where not exists
(select v.*
from Vorlesungen v
where v.SWS=4 and not exists
(select h.*
from hören h
where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr));

⟶

select h.MatrNr
from hören h
group by h.MatrNr
having count (*) = (select count (*)
from Vorlesungen v where v.SWS=4);

185

AICOS Technische Universität München

finde die Studenten die alle vierstündigen Vorlesungen hören:

select s.*
from Studenten s
where not exists
(select v.*
from Vorlesungen v
where v.SWS=4 and not exists
(select h.*
from hören h
where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr));

⟶

select h.MatrNr
from hören h
group by h.MatrNr
having count (*) = (select count (*)
from Vorlesungen v where v.SWS=4);

185

finde die Studenten die alle vierstündigen Vorlesungen hören:

```
select s.*
        from Studenten s
        where not exists
                (select v.*
                from Vorlesungen v
                where v.SWS=4 and not exists
                        (select h.*
                        from hören h
                        where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr));
```

⟶

```
select h.MatrNr
        from hören h
        group by h.MatrNr
                having count (*) = (select count (*)
                        from Vorlesungen v where v.SWS=4);
```


AICOS Technische Universität München

finde die Studenten die alle vierstündigen Vorlesungen hören:

```
select s.*
        from Studenten s
        where not exists
                (select v.*
                from Vorlesungen v
                where v.SWS=4 and not exists
                        (select h.*
                        from hören h
                        where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr));
```

⟶

```
select h.MatrNr
        from hören h
        group by h.MatrNr
                having count (*) = (select count (*)
                        from Vorlesungen v where v.SWS=4);
```


AICOS Technische Universität München

finde die Studenten die alle vierstündigen Vorlesungen hören:

```
select s.*
        from Studenten s
        where not exists
                (select v.*
                from Vorlesungen v
                where v.SWS=4 and not exists
                        (select h.*
                        from hören h
                        where h.VorlNr = v.VorlNr and h.MatrNr = s.MatrNr));
```

⟶

```
select h.MatrNr
        from hören h
        group by h.MatrNr
                having count (*) = (select count (*)
                        from Vorlesungen v where v.SWS=4);
```


AICOS Technische Universität München

Nullwerte

- unbekannter Wert
- wird vielleicht später nachgereicht
- Nullwerte können auch im Zuge der Anfrageauswertung entstehen (Bsp. äußere Joins)
- manchmal sehr überraschende Anfrageergebnisse, wenn Nullwerte vorkommen

  select count (*)
  from Studenten
  where Semester < 13 **or** Semester > =13
- Wenn es Studenten gibt, deren *Semester*-Attribut den Wert **null** hat, werden diese nicht mitgezählt
- Der Grund liegt in folgenden Regeln für den Umgang mit **null**-Werten begründet:

Auswertung bei Null-Werten

1. In arithmetischen Ausdrücken werden Nullwerte propagiert, d.h. sobald ein Operand **null** ist, wird auch das Ergebnis **null**. Dementsprechend wird z.B. **null** + 1 zu **null** ausgewertet. Aber auch null * 0 wird zu **null** ausgewertet.
2. SQL hat eine dreiwertige Logik, die nicht nur **true** und **false** kennt, sondern auch einen dritten Wert **unknown**. Diesen Wert liefern Vergleichsoperationen zurück, wenn mindestens eines ihrer Argumente **null** ist. Beispielsweise wertet SQL das Prädikat *(PersNr=...)* immer zu **unknown** aus, wenn die *PersNr* des betreffenden Tupels den Wert **null** hat.
3. Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

AICOS Technische Universität München

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

not	
true	false
unknown	unknown
false	true

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

AICOS Technische Universität München

Auswertung bei Null-Werten

1. In arithmetischen Ausdrücken werden Nullwerte propagiert, d.h. sobald ein Operand **null** ist, wird auch das Ergebnis **null**. Dementsprechend wird z.B. **null** + 1 zu **null** ausgewertet. Aber auch null * 0 wird zu **null** ausgewertet.
2. SQL hat eine dreiwertige Logik, die nicht nur **true** und **false** kennt, sondern auch einen dritten Wert **unknown**. Diesen Wert liefern Vergleichsoperationen zurück, wenn mindestens eines ihrer Argumente **null** ist. Beispielsweise wertet SQL das Prädikat *(PersNr=...)* immer zu **unknown** aus, wenn die *PersNr* des betreffenden Tupels den Wert **null** hat.
3. Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

187

AICOS Technische Universität München

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

not	
true	false
unknown	unknown
false	true

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

188

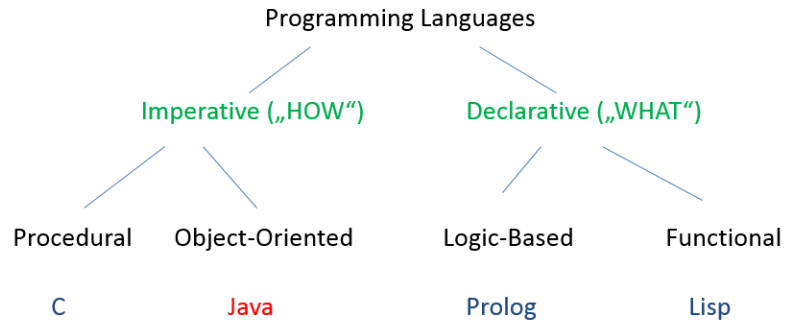
Vergleiche mit like

Platzhalter "%" ; "_"

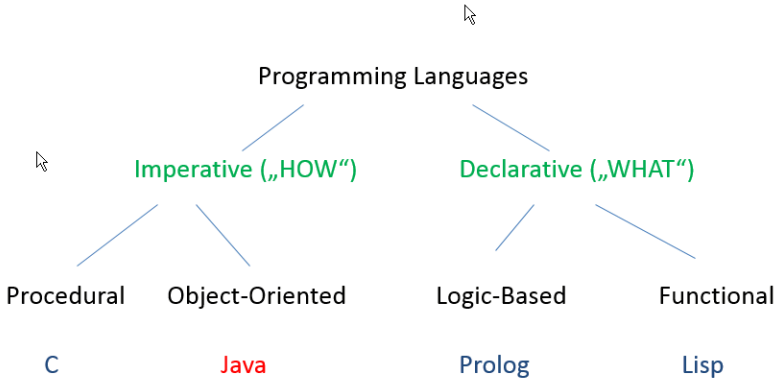
- "%" steht für beliebig viele (auch gar kein) Zeichen
- "_" steht für genau ein Zeichen

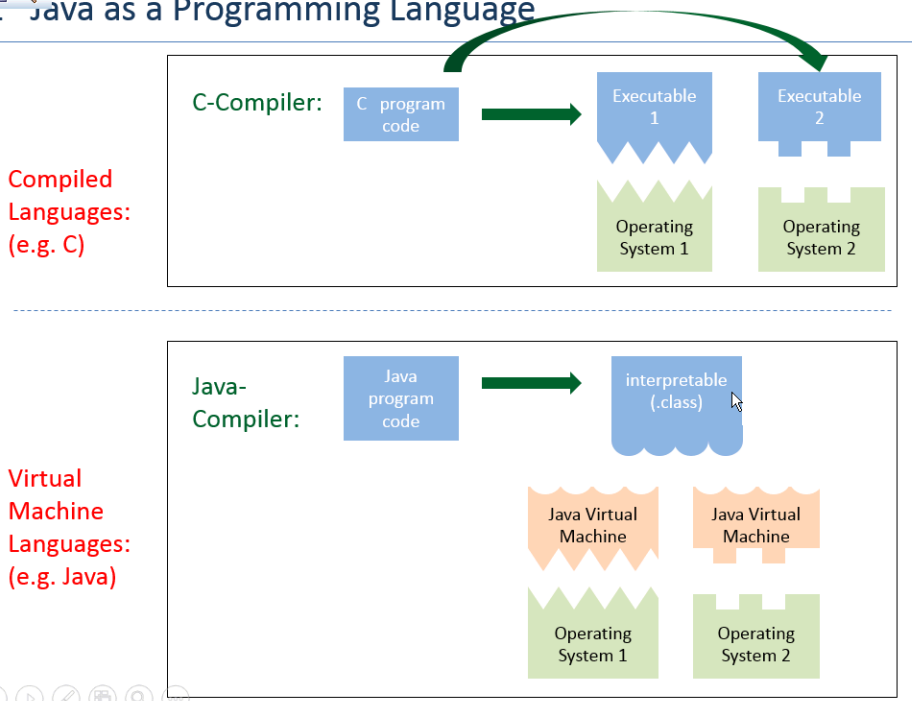
```
select * from Studenten
where Name like ´T%eophrastos´;

select distinct Name
from Vorlesungen v, hören h, Studenten s
where s.MatrNr = h.MatrNr and h.VorlNr = v.VorlNr and
      v.Titel = ´%thik%´;
```


Programming Languages

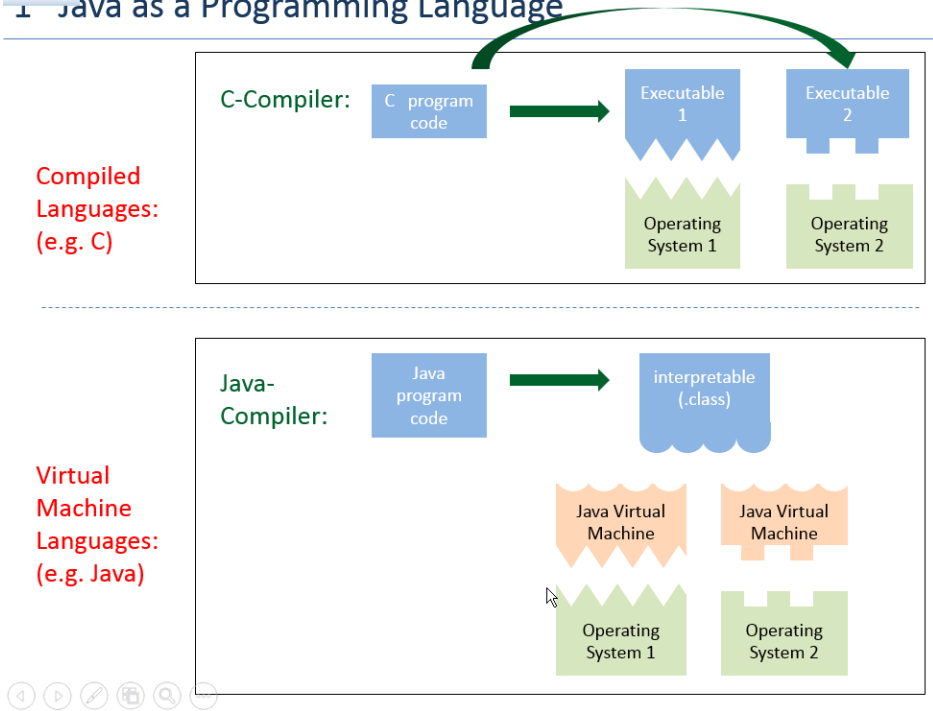


Programming Languages





Java as a Programming Language



Java as a Programming Language

C-Compiler:
C program code
Executable 1
Executable 2
Operating System 1
Operating System 2
Compiled Languages: (e.g. C)
Java-Compiler:
Java program code
interpretable (.class)
Java Virtual Machine
Java Virtual Machine
Operating System 1
Operating System 2
Virtual Machine Languages: (e.g. Java)

Java as a Programming Language

Imperative Programming

- Imperative program:
  Sequence of statements
- Instructions change state (especially memory) of computer system

control flow

```
boolean plato;
int horst;
int heiner;
int fritz;
plato = false;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
```

memory (simplified model)

cell nr	cell name	cell content	
1123	plato	false	data
1124			
1125	horst	101	
1126	heiner	0	
1127			
1128	fritz	0	
4027		boolean plato;	instructions
4028		int horst;	
4029		int heiner;	
4030		int fritz;	
4029		plato = false;	
4030		horst = 101;	

Java as a Programming Language

Imperative Programming

- Imperative program:
  Sequence of statements
- Instructions change state (especially memory) of computer system

control flow

```
boolean plato;
int horst;
int heiner;
int fritz;
plato = false;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
```

memory (simplified model)

cell nr	cell name	cell content	
1123	plato	false	data
1124			
1125	horst	101	
1126	heiner	0	
1127			
1128	fritz	0	
4027		boolean plato;	instructions
4028		int horst;	
4029		int heiner;	
4030		int fritz;	
4029		plato = false;	
4030		horst = 101;	

Imperative Programming

- Imperative program:
  Sequence of statements
- Instructions change state (especially memory) of computer system

```
boolean plato;
int horst;
int heiner;
int fritz;
plato = false;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
```

control flow

memory (simplified model)

cell nr	cell name	cell content	
1123	plato	false	data
1124			
1125	horst	101	
1126	heiner	0	
1127			
1128	fritz	0	
4027		boolean plato;	instructions
4028		int horst;	
4029		int heiner;	
4030		int fritz;	
4029		plato = false;	
4030		horst = 101;	

1 Java as a Programming Language

Imperative Programming

- Imperative program:
  Sequence of statements
- Instructions change state (especially memory) of computer system

```
boolean plato;
int horst;
int heiner;
int fritz;
plato = false;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
```

control flow

memory (simplified model)

cell nr	cell name	cell content	
1123	plato	false	data
1124			
1125	horst	101	
1126	heiner	0	
1127			
1128	fritz	0	
4027		boolean plato;	instructions
4028		int horst;	
4029		int heiner;	
4030		int fritz;	
4029		plato = false;	
4030		horst = 101;	

1 Java as a Programming Language

Imperative Programming

- Imperative program:
  Sequence of statements
- Instructions change state (especially memory) of computer system

```
boolean plato;
int horst;
int heiner;
int fritz;
plato = false;
horst = 2000;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
```

control flow

memory (simplified model)

cell nr	cell name	cell content	
1123	plato	false	data
1124			
1125	horst	2000	
1126	heiner	2	
1127			
1128	fritz	103	
4027		boolean plato;	instructions
4028		int horst;	
4029		int heiner;	
4030		int fritz;	
4029		plato = false;	
4030		horst = 101;	

1 Java as a Programming Language

Procedural Programming

- Group sequences of instructions into named „procedures“ („functions“, „methods“, „sub-routines“ etc.)

```
int doSelfSumSquare(int someNumber){
        int a;
        a = someNumber + someNumber;
        a = a * a;
        return a;
}
```

$f(x) = (x + x)^2$

- Advantages
  - no copying of instruction sequences
  - better testing
  - modularity (e.g. code change inside function doesn‘t affect caller)
  - code re-use
  - etc.

Java as a Programming Language

Procedural Programming

- Group sequences of instructions into named „procedures“ („functions“, „methods“, „sub-routines“ etc.)

```
int doSelfSumSquare(int someNumber){
        int a;
        a = someNumber + someNumber;
        a = a * a;
        return a;
}
```

$f(x) = (x + x)^2$

- Advantages
  - no copying of instruction sequences
  - better testing
  - modularity (e.g. code change inside function doesn‘t affect caller)
  - code re-use
  - etc.

Java as a Programming Language

```
int horst;
int heiner;
horst = 101;
heiner = 2;
heiner = doSelfSumSquare(horst);
heiner = doSelfSumSquare(117);
horst = horst + 2;
```

⋮

```
int doSelfSumSquare(int someNumber){
        int a;
        a = someNumber + someNumber;
        a = a * a;
        return a;
}
```

- In the example: Control flow is transferred to function, back to main program, back to function and back to main program

Java as a Programming Language

Imperative Programming

- Imperative program: Sequence of statements
- Instructions change state (especially memory) of computer system

control flow

```
boolean plato;
int horst;
int heiner;
int fritz;
plato = false;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
```

memory (simplified model)

cell nr	cell name	cell content	
		⋮	
1123	plato	false	data
1124			
1125	horst	2000	
1126	heiner	2	
1127			
1128	fritz	103	
		⋮	
4027		boolean plato;	instructions
4028		int horst;	
4029		int heiner;	
4030		int fritz;	
4029		plato = false;	
4030		horst = 101;	
		⋮	

Java as a Programming Language

```
int horst;
int heiner;
horst = 101;
heiner = 2;
heiner = doSelfSumSquare(horst);
heiner = doSelfSumSquare(117);
horst = horst + 2;
```

⋮

```
int doSelfSumSquare(int someNumber){
        int a;
        a = someNumber + someNumber;
        a = a * a;
        return a;
}
```

- In the example: Control flow is transferred to function, back to main program, back to function and back to main program

Java as a Programming Language

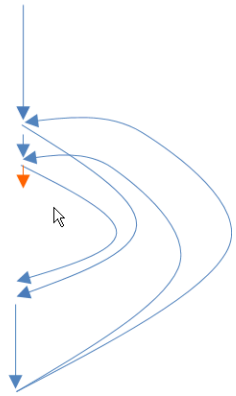
```
int horst;
int heiner;
horst = 101;
heiner = 2;
heiner = doSelfSumSquare(horst);
heiner = doSelfSumSquare(117);
horst = horst + 2;
```

```
int doSelfSumSquare(int someNumber){
        int a;
        a = someNumber + someNumber;
        a = a * a;
        return a;
}
```

- In the example: Control flow is transferred to function, back to main program, back to function and back to main program

Java as a Programming Language

```
int horst;
int heiner;
horst = 101;
heiner = 2;
heiner = doSelfSumSquare(horst);
heiner = doSelfSumSquare(117);
horst = horst + 2;
```



```
int doSelfSumSquare(int someNumber){
        int a;
        a = someNumber + someNumber;
        a = a * a;
        return a;
}
```

- In the example: Control flow is transferred to function, back to main program, back to function and back to main program

Java as a Programming Language

```
int horst;
int heiner;
horst = 101;
heiner = 2;
heiner = doSelfSumSquareHeiner(horst);
heiner = doSelfSumSquareHeiner(117);
horst = horst + 2;
```

```
int doSelfSumSquareHeiner(int someNumber){

        int a;
        a = someNumber + someNumber;
        a = a * a;
        a = a * heiner;
        return a;
}
```

- functions often access global variables (bad style!)
- Goal: „Keep things local“ (better testing, code re-use etc.)

Java as a Programming Language

```
int horst;
int heiner;
horst = 101;
heiner = 2;
heiner = doSelfSumSquareHeiner(horst);
heiner = doSelfSumSquareHeiner(117);
horst = horst + 2;
```

```
int doSelfSumSquareHeiner(int someNumber){

        int a;
        a = someNumber + someNumber;
        a = a * a;
        a = a * heiner;
        return a;
}
```

- functions often access global variables (bad style!)
- Goal: „Keep things local“ (better testing, code re-use etc.)

```
int horst;
int heiner;
horst = 101;
heiner = 2;
heiner = doSelfSumSquareHeiner(horst);
heiner = doSelfSumSquareHeiner(117);
horst = horst + 2;
```

⋮

```
int doSelfSumSquareHeiner(int someNumber){

        int a;
        a = someNumber + someNumber;
        a = a * a;
        a = a * heiner;
        return a;
}
```

- functions often access global variables (bad style!)
- Goal: „Keep things local“ (better testing, code re-use etc.)

Object-oriented Programming

- Object-oriented programming:

  Group data and functions into objects ←→
  Models of state and behaviour of **real world** objects
  state: „fields“ ; behaviour: „methods“

- Methods should mainly act on an object‘s fields
- Classes: Blueprints for objects → Objects: Instances of classes
- Advantages
  - Intuitive models
  - Information hiding
  - Increased modularity, locality etc.
  - Increased code re-use
  - etc.

Java as a Programming Language

Object-oriented Programming

- Object-oriented programming:

  Group data and functions into objects ←→
  Models of state and behaviour of **real world** objects
  state: „fields“ ; behaviour: „methods“

- Methods should mainly act on an object‘s fields
- Classes: Blueprints for objects → Objects: Instances of classes
- Advantages
  - Intuitive models
  - Information hiding
  - Increased modularity, locality etc.
  - Increased code re-use
  - etc.

Java as a Programming Language

```
class Bicycle {
        int cadence = 0;
        int speed = 0;
        int gear = 1;

        void changeCadence(int newValue) {
             cadence = newValue;
        }

        void changeGear(int newValue) {
             gear = newValue;
        }

        void speedUp(int increment) {
             speed = speed + increment;
        }

        void applyBrakes(int decrement) {
             speed = speed - decrement;
        }
}
```

fields (state)

methods (behaviour)

class

Source: [JTutorial]

Java as a Programming Language

```
class BicycleDemo {
     public static void main(String[] args) {
          // Create two different Bicycle objects
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle();

          // Invoke methods on these objects
          bike1.changeCadence(50);
          bike1.speedUp(10);
          bike1.changeGear(2);

          bike2.changeCadence(50);
          bike2.speedUp(10);
          bike2.changeGear(2);
          bike2.changeCadence(40);
          bike2.speedUp(10);
          bike2.changeGear(3);
       }
}
```

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```

Source: [JTutorial]

Java as a Programming Language

```
class BicycleDemo {
     public static void main(String[] args) {
          // Create two different Bicycle objects
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle();

          // Invoke methods on these objects
          bike1.changeCadence(50);
          bike1.speedUp(10);
          bike1.changeGear(2);

          bike2.changeCadence(50);
          bike2.speedUp(10);
          bike2.changeGear(2);
          bike2.changeCadence(40);
          bike2.speedUp(10);
          bike2.changeGear(3);
       }
}
```

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```

Source: [JTutorial]

Java as a Programming Language

```
class BicycleDemo {
     public static void main(String[] args) {
          // Create two different Bicycle objects
          Bicycle bike1 = new Bicycle();
          Bicycle bike2 = new Bicycle();

          // Invoke methods on these objects
          bike1.changeCadence(50);
          bike1.speedUp(10);
          bike1.changeGear(2);

          bike2.changeCadence(50);
          bike2.speedUp(10);
          bike2.changeGear(2);
          bike2.changeCadence(40);
          bike2.speedUp(10);
          bike2.changeGear(3);
       }
}
```

```
class Bicycle {
       int cadence = 0;
       int speed = 0;
       int gear = 1;

       void changeCadence(int newValue) {
            cadence = newValue;
       }

       void changeGear(int newValue) {
            gear = newValue;
       }

       void speedUp(int increment) {
            speed = speed + increment;
       }

       void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
}
```

Source: [JTutorial]

Datenbanken

Java

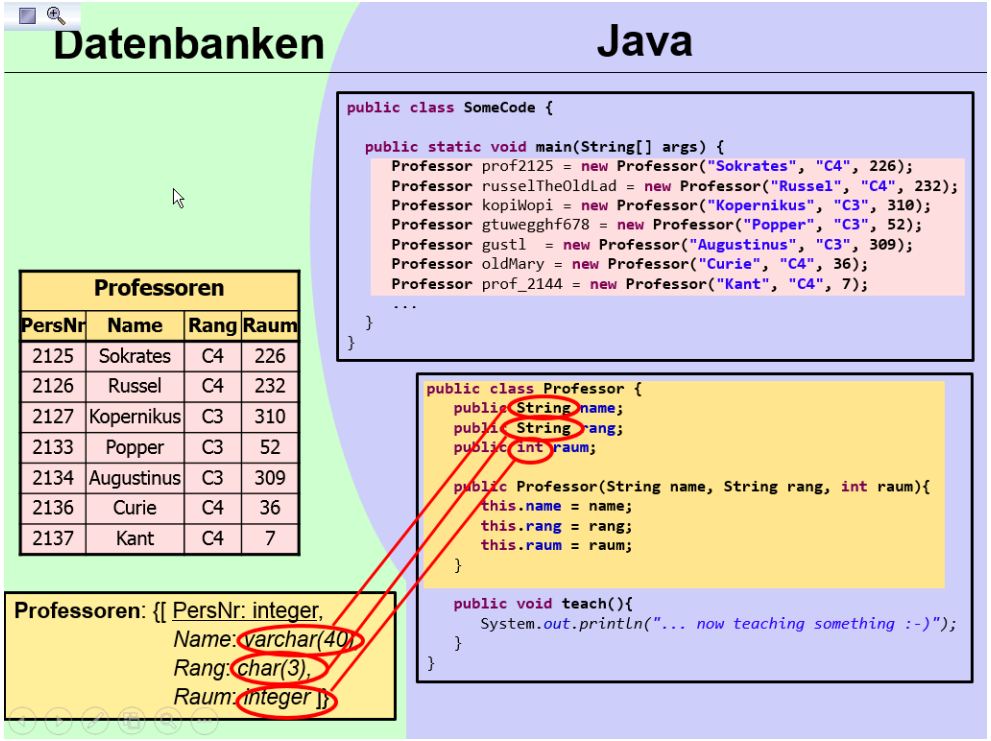
Professoren

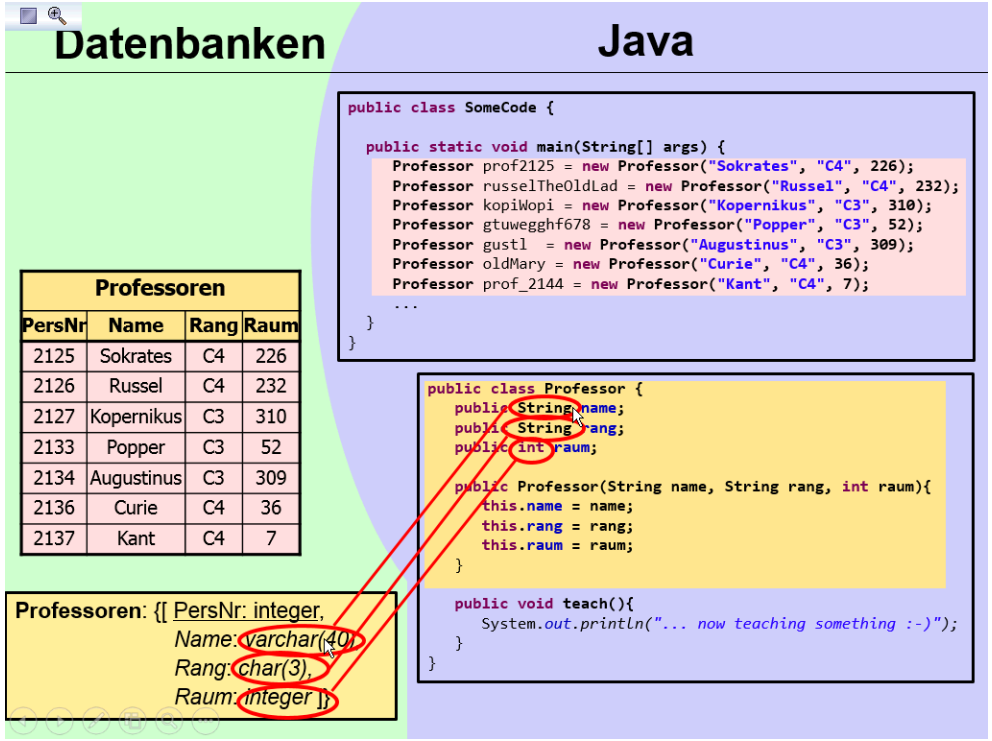
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

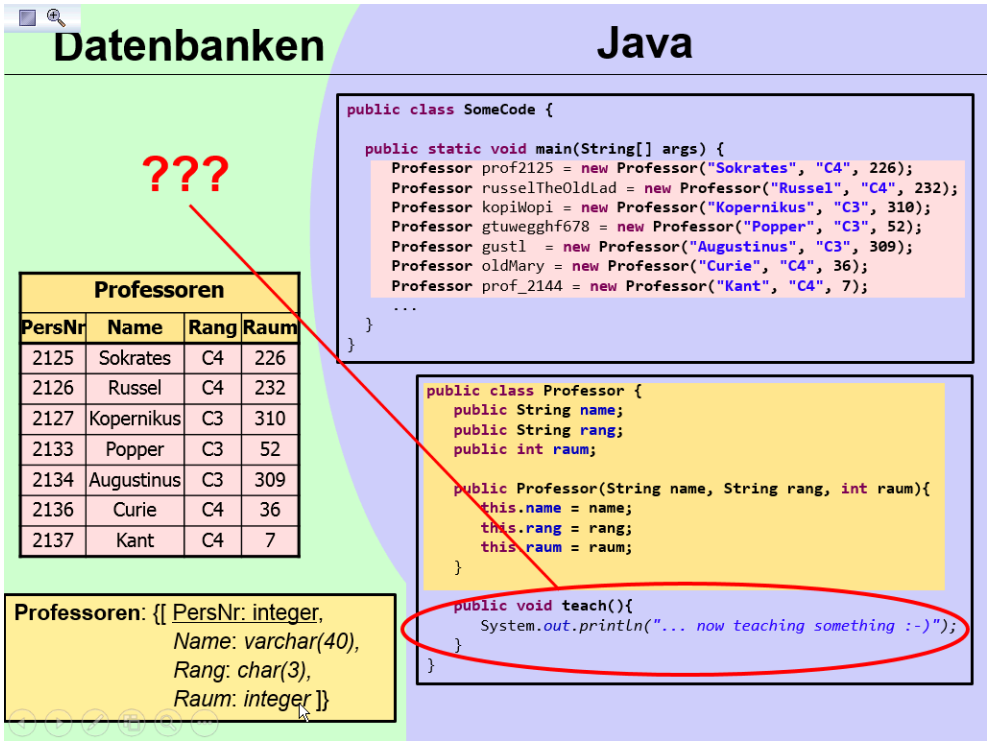
Professoren: {[~~PersNr: integer~~,
Name: varchar(40),
Rang: char(3),
Raum: integer]}

```
public class SomeCode {

  public static void main(String[] args) {
     Professor prof2125 = new Professor("Sokrates", "C4", 226);
     Professor russelTheOldLad = new Professor("Russel", "C4", 232);
     Professor kopiWopi = new Professor("Kopernikus", "C3", 310);
     Professor gtuwegghf678 = new Professor("Popper", "C3", 52);
     Professor gustl = new Professor("Augustinus", "C3", 309);
     Professor oldMary = new Professor("Curie", "C4", 36);
     Professor prof_2144 = new Professor("Kant", "C4", 7);
     ...
  }
}
```

```
public class Professor {
   public String name;
   public String rang;
   public int raum;

   public Professor(String name, String rang, int raum){
      this.name = name;
      this.rang = rang;
      this.raum = raum;
   }

   public void teach(){
      System.out.println("... now teaching something :-)");
   }
}
```


Datenbanken
Java
public class SomeCode {
public static void main(String[] args) {
Professor prof2125 = new Professor("Sokrates", "C4", 226);
Professor russelTheOldLad = new Professor("Russel", "C4", 232);
Professor kopiWopi = new Professor("Kopernikus", "C3", 310);
Professor gtuwegghf678 = new Professor("Popper", "C3", 52);
Professor gustl = new Professor("Augustinus", "C3", 309);
Professor oldMary = new Professor("Curie", "C4", 36);
Professor prof_2144 = new Professor("Kant", "C4", 7);
...
}
}
Professoren
PersNr Name Rang Raum
2125 Sokrates C4 226
2126 Russel C4 232
2127 Kopernikus C3 310
2133 Popper C3 52
2134 Augustinus C3 309
2136 Curie C4 36
2137 Kant C4 7
public class Professor {
public String name;
public String rang;
public int raum;
public Professor(String name, String rang, int raum){
this.name = name;
this.rang = rang;
this.raum = raum;
}
public void teach(){
System.out.println("... now teaching something :-)");
}
}
Professoren: {[PersNr: integer,
Name: varchar(40),
Rang: char(3),
Raum: integer]}


Datenbanken
Java
public class SomeCode {
public static void main(String[] args) {
Professor prof2125 = new Professor("Sokrates", "C4", 226);
Professor russelTheOldLad = new Professor("Russel", "C4", 232);
Professor kopiWopi = new Professor("Kopernikus", "C3", 310);
Professor gtuwegghf678 = new Professor("Popper", "C3", 52);
Professor gustl = new Professor("Augustinus", "C3", 309);
Professor oldMary = new Professor("Curie", "C4", 36);
Professor prof_2144 = new Professor("Kant", "C4", 7);
...
}
}
Professoren
PersNr Name Rang Raum
2125 Sokrates C4 226
2126 Russel C4 232
2127 Kopernikus C3 310
2133 Popper C3 52
2134 Augustinus C3 309
2136 Curie C4 36
2137 Kant C4 7
public class Professor {
public String name;
public String rang;
public int raum;
public Professor(String name, String rang, int raum){
this.name = name;
this.rang = rang;
this.raum = raum;
}
public void teach(){
System.out.println("... now teaching something :-)");
}
}
Professoren: {[PersNr: integer,
Name: varchar(40),
Rang: char(3),
Raum: integer]}


Datenbanken
Java
???
public class SomeCode {
public static void main(String[] args) {
Professor prof2125 = new Professor("Sokrates", "C4", 226);
Professor russelTheOldLad = new Professor("Russel", "C4", 232);
Professor kopiWopi = new Professor("Kopernikus", "C3", 310);
Professor gtuwegghf678 = new Professor("Popper", "C3", 52);
Professor gustl = new Professor("Augustinus", "C3", 309);
Professor oldMary = new Professor("Curie", "C4", 36);
Professor prof_2144 = new Professor("Kant", "C4", 7);
...
}
}
Professoren
PersNr Name Rang Raum
2125 Sokrates C4 226
2126 Russel C4 232
2127 Kopernikus C3 310
2133 Popper C3 52
2134 Augustinus C3 309
2136 Curie C4 36
2137 Kant C4 7
public class Professor {
public String name;
public String rang;
public int raum;
public Professor(String name, String rang, int raum){
this.name = name;
this.rang = rang;
this.raum = raum;
}
public void teach(){
System.out.println("... now teaching something :-)");
}
}
Professoren: {[PersNr: integer,
Name: varchar(40),
Rang: char(3),
Raum: integer]}