ETH Zürich
Chair of Software Engineering

# Einführung in die Programmierung

Prof. Dr. Bertrand Meyer

## Lektion 7: Referenzen und Zuweisungen

# Das heutige Programm

Wie werden Objekte "gemacht" ?
(Mehr Details)

Wie modifizieren wir ihre Inhalte?

- Zuweisungen
- Speicherverwaltung, GC
- Mehr über Entitäten, Typen, Variablen...

# Referenzen sind schwierig…

```
require
        -- "Meine Schwester ist ledig"
        -- "Meine Schwester lebt in Dietlikon"
do
        -- "Der Enkel meines Nachbarn heiratete
        -- gestern eine Ärztin in Sydney"
ensure
        -- "Meine Schwester ist ledig"
        -- "Meine Schwester lebt in Dietlikon"
end
```

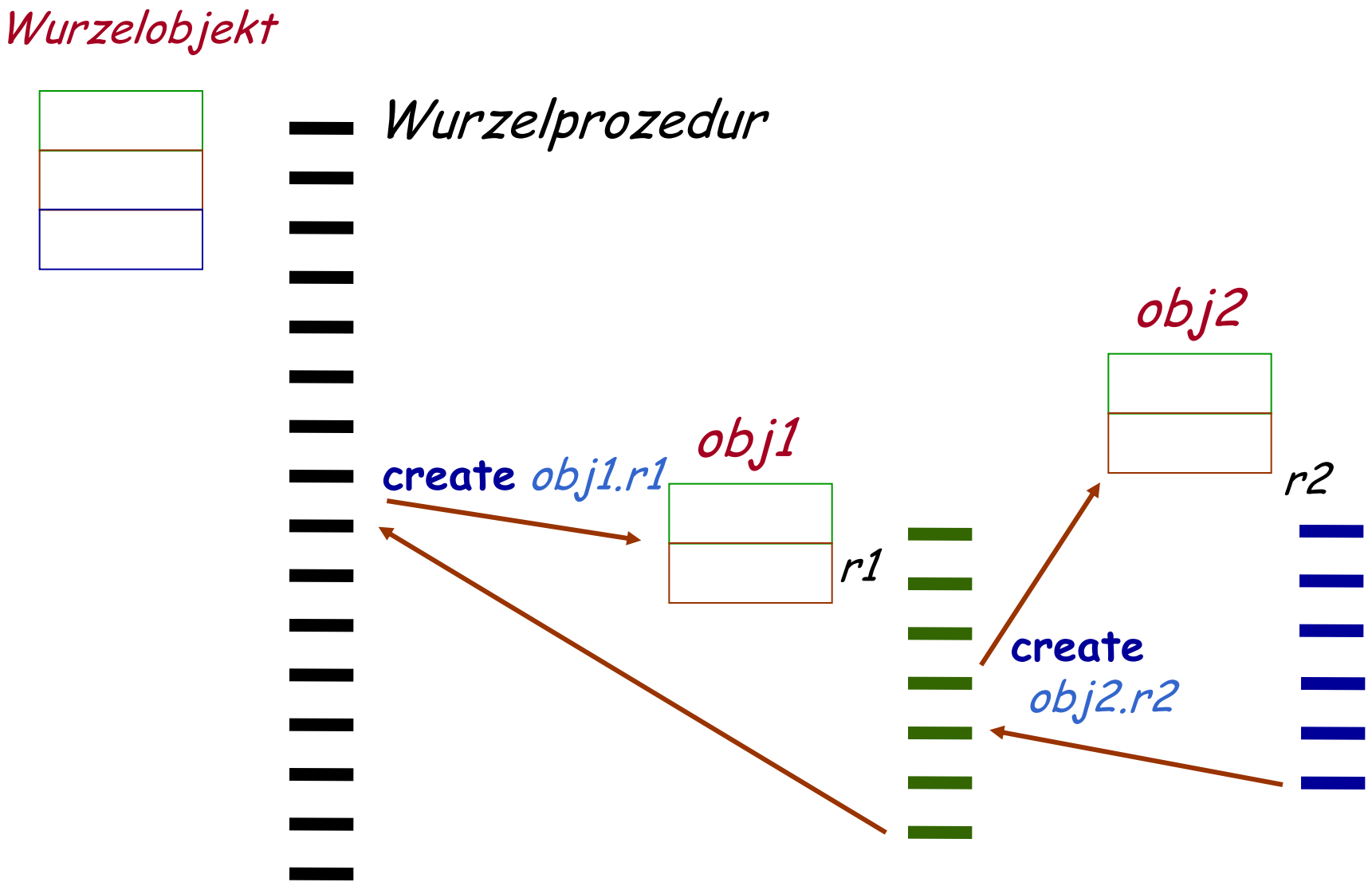
# Wie alles beginnt...

Die Ausführung eines Systems beginnt mit der Erzeugung eines **Wurzelobjektes**, das eine Instanz einer vom System vorgesehenen Klasse (die **Wurzelklasse**) ist, mittels einer vorgesehenen Erzeugungsprozedur dieser Klasse (die **Wurzelprozedur**).

Eine Wurzel-Erzeugungsprozedur kann:

- Neue Objekte erzeugen
- Auf diese Features aufrufen, die wiederum neue Objekte erzeugen können

# Ausführung eines Systems



# Klassen und Objekte

- Zur Laufzeit: Objekte (Softwaremaschinen)
- Im Programmtext: Klassen

Jede Klasse beschreibt eine Menge von möglichen Laufzeitobjekten.

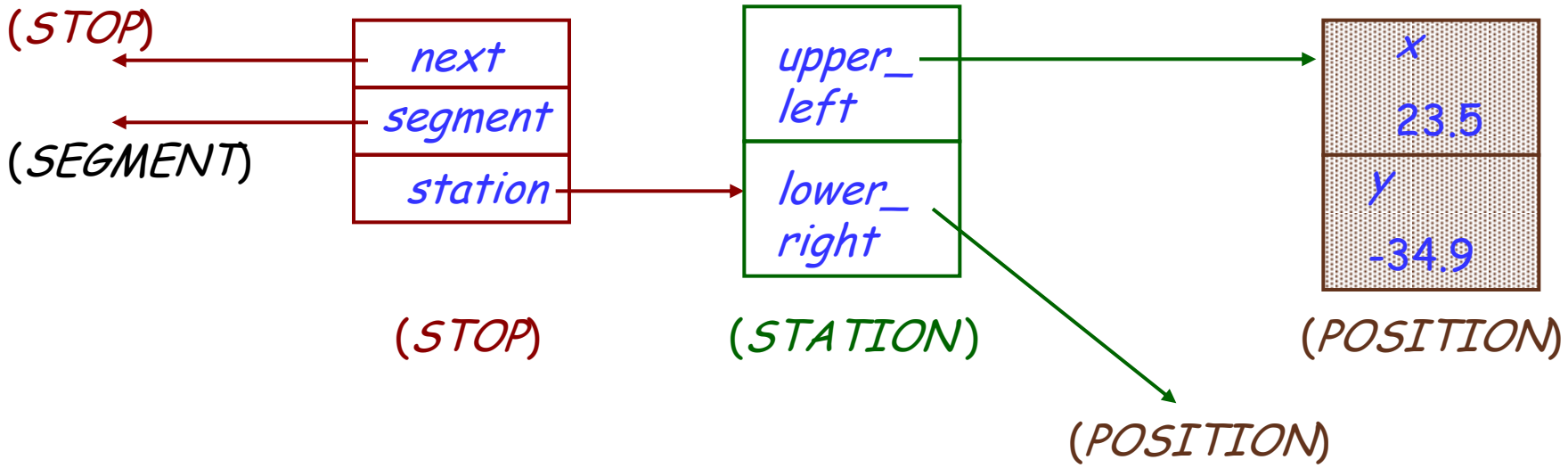
# Objektstruktur

Ein Objekt besteht aus Feldern

Jedes Feld ist ein Wert, entweder:

- Ein Basiswert: Ganze Zahl (*integer*), Zeichen (*character*), "reelle" Zahl, …

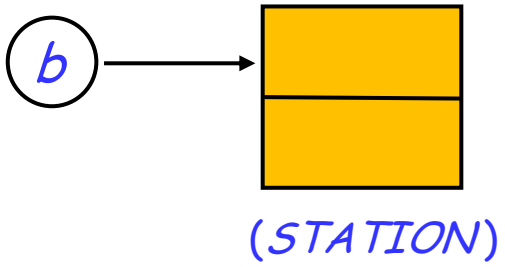
  (genannt „expandierte Werte")
- Eine Referenz zu einem anderen Objekt.



# Zwei Arten von Typen

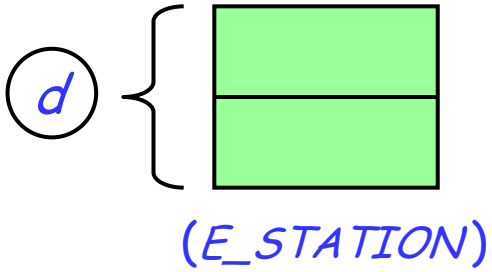
Referenztypen: Der Wert jeder Entität ist eine Referenz

*b*: *STATION*




Expandierte Typen: Der Wert jeder Entität ist ein Objekt

*d*: *E_STATION*



# Expandierte und Referenztypen

Was ist der Unterschied zwischen:

- „Jeder Wagen hat einen Motor“
- „Jeder Wagen hat eine Marke und einen Hersteller“

?

# Expandierte Klassen

Eine Klasse kann als expandiert deklariert werden:

**expanded class** *E_STATION*
... Der Rest wie in *STATION* ...

Dann hat jede Entität deklariert als

*d*: *E_STATION*

die eben beschriebene expandierte Semantik.

# Basistypen als expandierte Klassen

**expanded class** *INTEGER* ...
(intern: *INTEGER_32*, *INTEGER_64* etc.)
**expanded class** *BOOLEAN* ...
**expanded class** *CHARACTER* ...
**expanded class** *REAL* ...
(intern: *REAL_32*, *REAL_64* etc.)

*n* : *INTEGER*

# Initialisierung

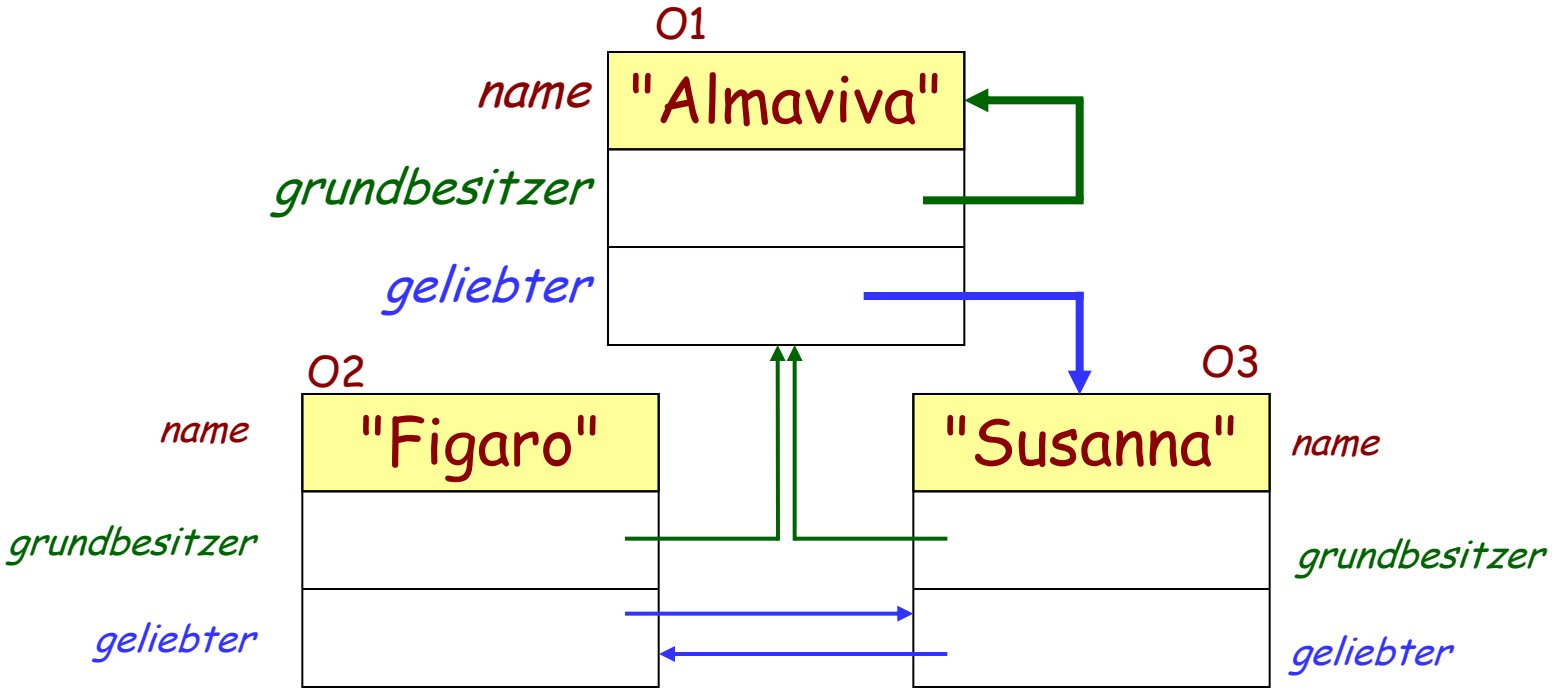
Automatische Initialisierungsregeln:

- 0 für Zahlen (ganze Zahlen, reelle Zahlen)
- "Null"-Zeichen für Zeichen
- **False** für Boole'sche Werte
- **Void** für Referenzen

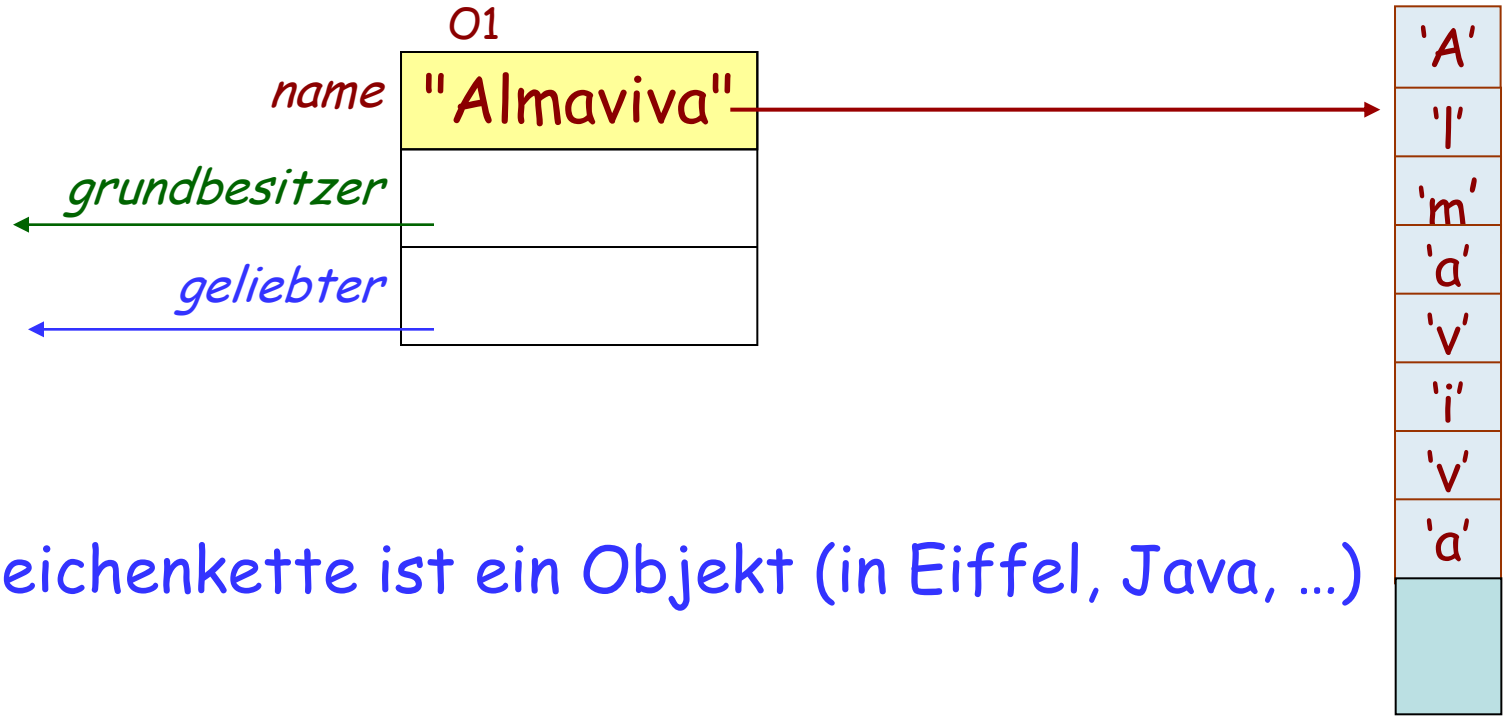
Diese Regeln gelten für:

- Felder (von Klassenattributen), bei der Objekterzeugung.
- Lokale Variablen, beim Start der Routinenausführung. (gilt auch für **Result**)

# Referenzen können Zyklen bilden.



# Und was ist mit Zeichenketten (strings)?




Eine Zeichenkette ist ein Objekt (in Eiffel, Java, ...)

Das Feld *name* ist ein Referenzfeld.

# Felder wiederspiegeln Attribute der Klasse

```
class
        POSITION
feature -- Zugriff
        x: REAL
                -- Horizontale Position
        y: REAL
                -- Vertikale Position

end
```

Ein Attribut

Noch ein Attribut

Attribute sind Features der Klasse.

# Feldern einen Wert zuweisen (*assign*)

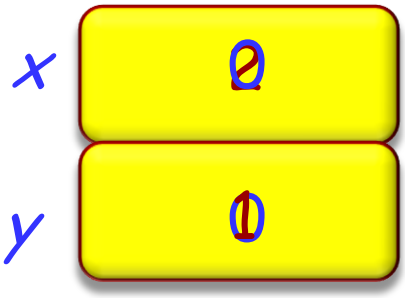
```
class
      POSITION
feature -- Zugriff
       x: REAL
          -- Horizontale Position
       y: REAL
          -- Vertikale Position
feature --  Element-Veränderungen
      setze (xwert, ywert: REAL)
                    -- Setzt Koordinaten auf [xwert, ywert].
          require
                    x_nicht_negativ : xwert >= 0
                    y_nicht_negativ : ywert >= 0
          do
                    ...
          ensure
                    x_gesetzt : x = xwert
                    y_gesetzt : y = ywert
          end
end
```

# Feldern einen Wert zuweisen

```
class
        POSITION
feature -- Zugriff
        x: REAL
                -- Horizontale Position
        y: REAL
                -- Vertikale Position
feature – Element-Veränderungen
        setze (xwert, ywert: REAL)
                        -- Setzt Koordinaten auf [xwert, ywert].
                require
                        x_nicht_negativ : xwert >= 0
                        y_nicht_negativ : ywert >= 0
                do
                        x := xwert
                        y := ywert
                ensure
                        x_gesetzt : x = xwert
                        y_gesetzt : y = ywert
                end
end
```

Eine Instanz von *POSITION*

Ausführung von *setze (2, 1)* auf diese Instanz



# Was Sie tun können

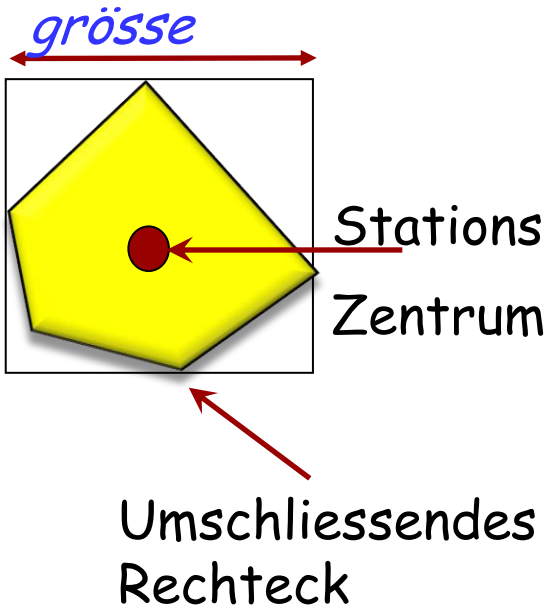
**class** *STATION* **feature**

```
x, y: REAL
      -- Koordinaten des Stationsmittelpunkts

grösse: REAL
      -- Grösse des umschliessenden
      -- Rechtecks.

obere_linke: POSITION
      -- Obere linke Position des
      -- umschliessenden Rechtecks.

setze_positionen
      -- Position des umschliessenden
      -- Rechtecks setzen.
      do
          obere_linke.setze (x - grösse/2, y + grösse/2)
          ...
      end
```



# Qualifizierte und unqualifizierte Featureaufrufe

In Klasse *POSITION*:

```
setze (xwert, ywert: REAL)
        ...
    do
        ...
    end
```

In einer anderen Klasse, z.B. *STATION*

```
obere_linke: POSITION

setze_positionen
    do
        obere_linke.setze (x - grösse/2, y + grösse/2)
        ...
    end
```

Qualifizierter Aufruf

In Klasse *POSITION* :

```
verschiebe (dx, dy: REAL)
            -- Um dx horizontal und dy Vertikale verschieben.
        require
            ... [Bitte ausfüllen!] ...
        do
            setze (x + dx, y + dy)
        ensure
            ... [Bitte ausfüllen!] ...
        end
```

Unqualifizierter Aufruf

# Das aktuelle Objekt (current object)

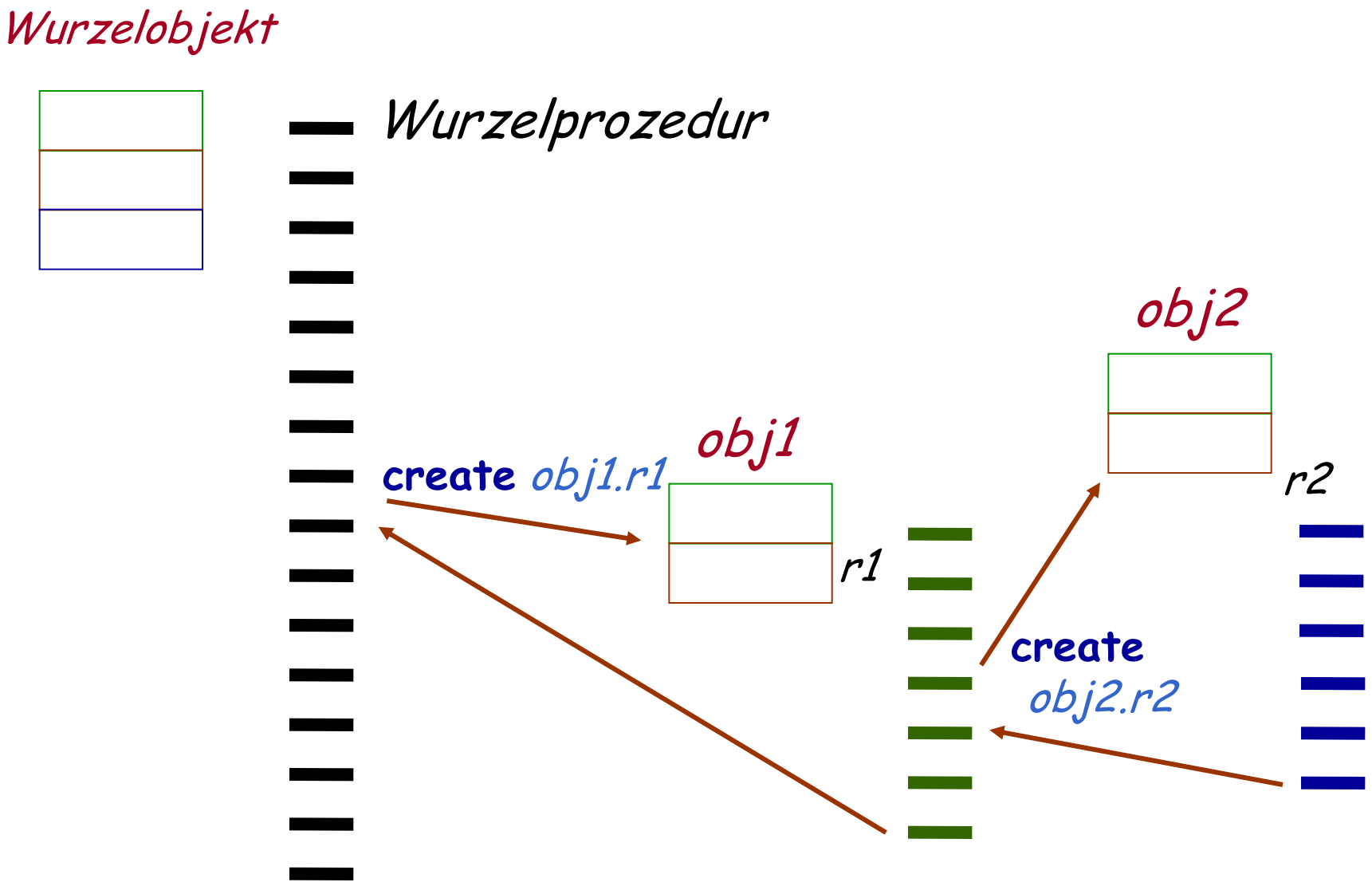
Zu jeder Zeit während einer Ausführung gibt es ein aktuelles Objekt, auf welches das aktuelle Feature aufgerufen wird.

Zu Beginn: Das Wurzelobjekt. Danach:

- Ein unqualifizierter Aufruf wie z.B. *setze (u, v)* wird auf das aktuelle Objekt angewendet.
- Ein qualifizierter Aufruf wie z.B. *x.setze (u, v)* bedingt, dass das an *x* gebundene Objekt zum aktuellen Objekt wird. Nach dem Aufruf wird das vorherige Objekt wieder zum aktuellen Objekt.

Um auf das aktuelle Objekt zuzugreifen: Benutzen Sie **Current**

# Ausführung eines Systems



# Das aktuelle Objekt

Zu jeder Zeit während einer Ausführung gibt es ein aktuelles Objekt, auf welches das aktuelle Feature aufgerufen wird.

Zu Beginn ist dies das Wurzelobjekt.

Während eines „qualifizierten" Aufrufs $x.f(a)$ ist das neue aktuelle Objekt dasjenige, das an $x$ gebunden ist.

Nach einem solchen Aufruf übernimmt das vorherige aktuelle Objekt wieder seine Rolle.

"Allgemeine Relativität"

# Featureaufrufe

```
In STATION:

        setze_positionen
                do
                        obere_linke.setze (x - grösse/2, y + grösse/2)
                        ...
                end

        obere_linke: POSITION
```

# Die Kundenbeziehung (*client relation*)

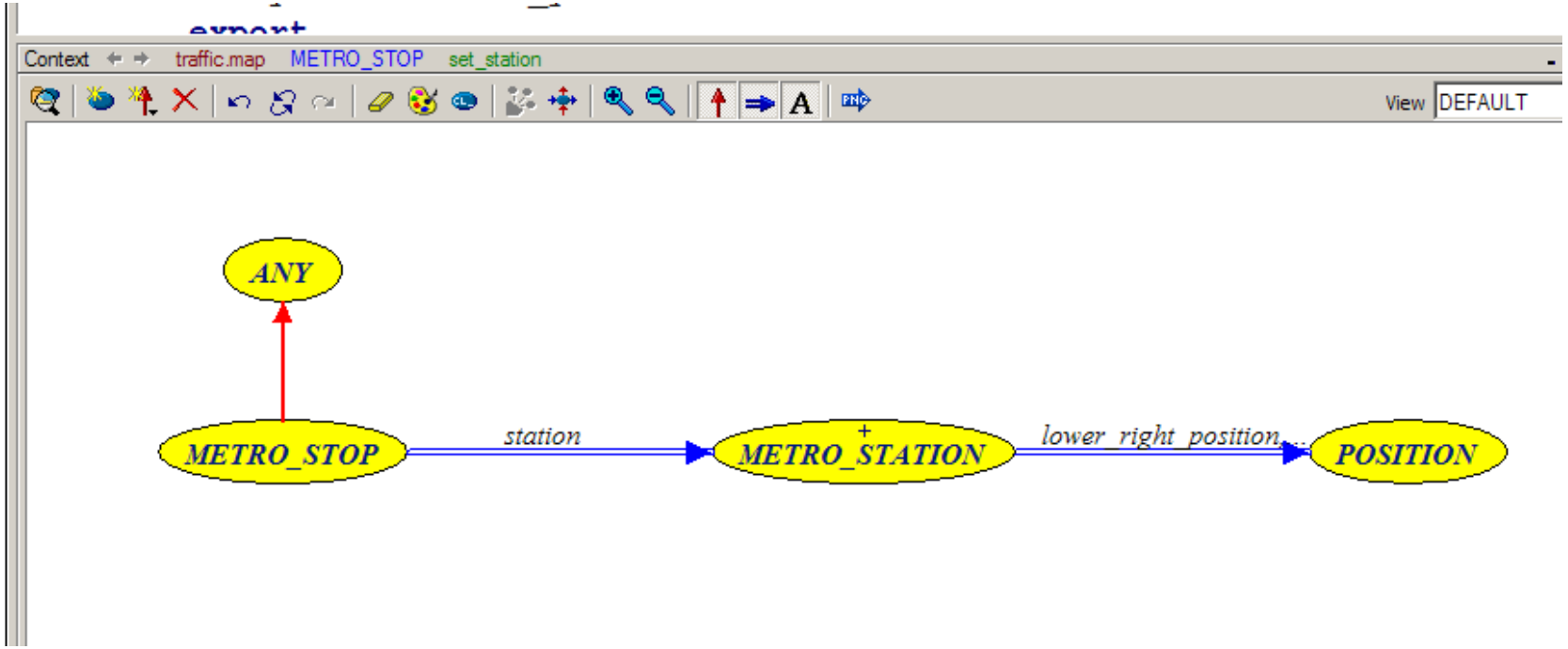
Da die Klasse *STATION* ein Feature

*obere_linke* : *POSITION*

hat (und Aufrufe der Form *obere_linke.setze (...)* ), ist

*STATION* ein Kunde der Klasse *POSITION*

# Kunden und Vererbung - graphisch




# Entitäten

Eine Entität ist ein Name im Programm, das mögliche Laufzeitwerte bezeichnet.

Manche Entitäten sind **konstant**

Andere sind **variabel**:

- Attribute
- Lokale Variablen

# Werte von Variablen ändern: Zuweisungen

*ziel* := *quelle*

*quelle* ist ein **Ausdruck**:

- Das Ergebnis einer Abfrage :
  - *position*
  - *obere_linke*.*position*
- Arithmetische oder Boole'sche Ausdrucke:
  - *a* + (*b* * *c*)
  - (*a* < *b*) **and** (*c* = *d*)

*ziel* ist eine **Variable**. Die Möglichkeiten sind:

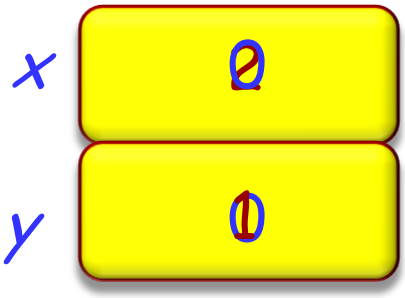
- Ein **Attribut**
- **Result** in einer Funktion (noch nicht gesehen bisher)
- Eine "**lokale Variable**" einer Routine (noch nicht gesehen)

# Zuweisungen (Erinnerung)

```
class
        POSITION
feature -- Zugriff
        x: REAL
                -- Horizontale Position
        y: REAL
                -- Vertikale Position
feature -- Element-Veränderungen
        setze (xwert, ywert: REAL)
                        -- Setzt Koordinaten auf [xwert, ywert].
                require
                        x_nicht_negativ : xwert >= 0
                        y_nicht_negativ : ywert >= 0
                do
                        x := xwert
                        y := ywert
                ensure
                        x_gesetzt : x = xwert
                        y_gesetzt : y = ywert
                end
end
```

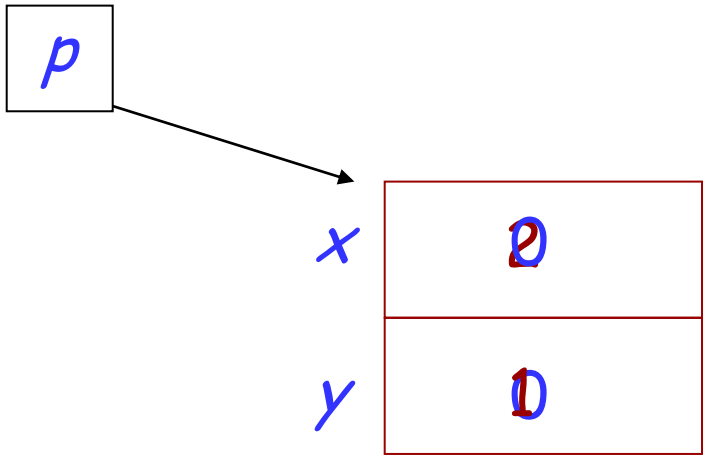
Eine Instanz von *POSITION*

Ausführung von *setze (2, 1)* auf diese Instanz



# Zuweisung

Eine Zuweisung ist eine Instruktion, die einen Wert durch einen anderen ersetzt




*p.setze (2, 1)*

```
setze (xwert, ywert : REAL)
        do
                x := xwert
                y := ywert
        end
```

# Zuweisung nicht mit Gleichheit verwechseln!

x := y

Instruction

if x = y then...

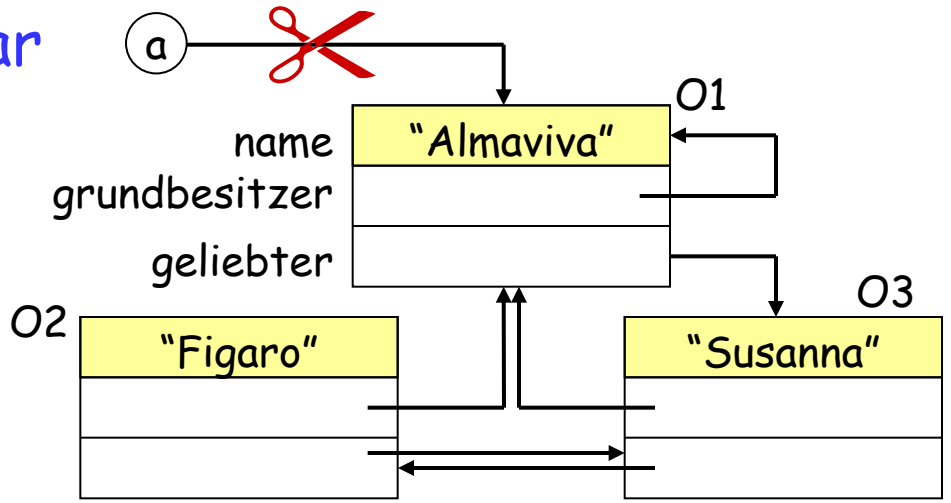
Ausdruck

e

if x = Current then...

# Was mit unerreichbaren Objekten tun?

Referenzzuweisungen können manche Objekte unerreichbar machen




Zwei mögliche Ansätze

- Manuelles "free" (C++, Pascal)
- Automatische Speicherbereinigung (garbage collection, d.h. Müllabfuhr) (Eiffel, Java, C#, .NET)

# Die Ansicht eines C-Programmierers

Newsgroup-Eintrag von Ian Stephenson, 1993 (Zitat aus *Object-Oriented Software Construction*):

> *I say a big NO! Leaving an unreferenced object around is BAD PROGRAMMING. Object pointers ARE like ordinary pointers — if you allocate an object you should be responsible for it, and free it when its finished with. (Didn't your mother always tell you to put your toys away when you'd finished with them?)*

# Argumente für automatische Bereinigung

Manuelle Bereinigung ist eine Gefahr für die Zuverlässigkeit:

- Falsche "frees" sind Bugs, die nur sehr schwierig zu finden und zu beheben sind.

Manuelle Bereinigung ist mühsam.

Moderne (automatische) Speicherbereiniger haben eine akzeptable Performance.

Speicherbereinigung ist einstellbar: an/abschalten, parametrisieren…

# Eigenschaften einer Speicherbereinigung

Konsistenz :

**_Nur_ erreichbare Objekte werden bereinigt**

Vollständigkeit :

**_Alle_ unerreichbare Objekte werden bereinigt**

Konsistenz (auch *Soundness* genannt) ist eine absolute Anforderung. Lieber keine Speicherbereinigung als eine unsichere Speicherbereinigung.

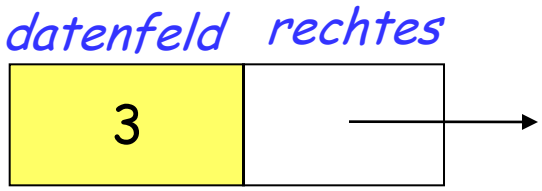
Aber: sichere automatische Speicherbereinigung ist schwierig für C-basierte Sprachen.

# Effekt einer Zuweisung

Referenztypen: Referenzzuweisung

Expandierte Typen: Kopie des Wertes

```
class WERTE_PAAR
feature
    datenfeld : INTEGER
    rechtes: WERTE_PAAR
    setze (n : INTEGER ; r : WERTE_PAAR)
            -- Beide Felder zurücksetzen.
        do
            item := n
            rechtes := r
        end
end
```



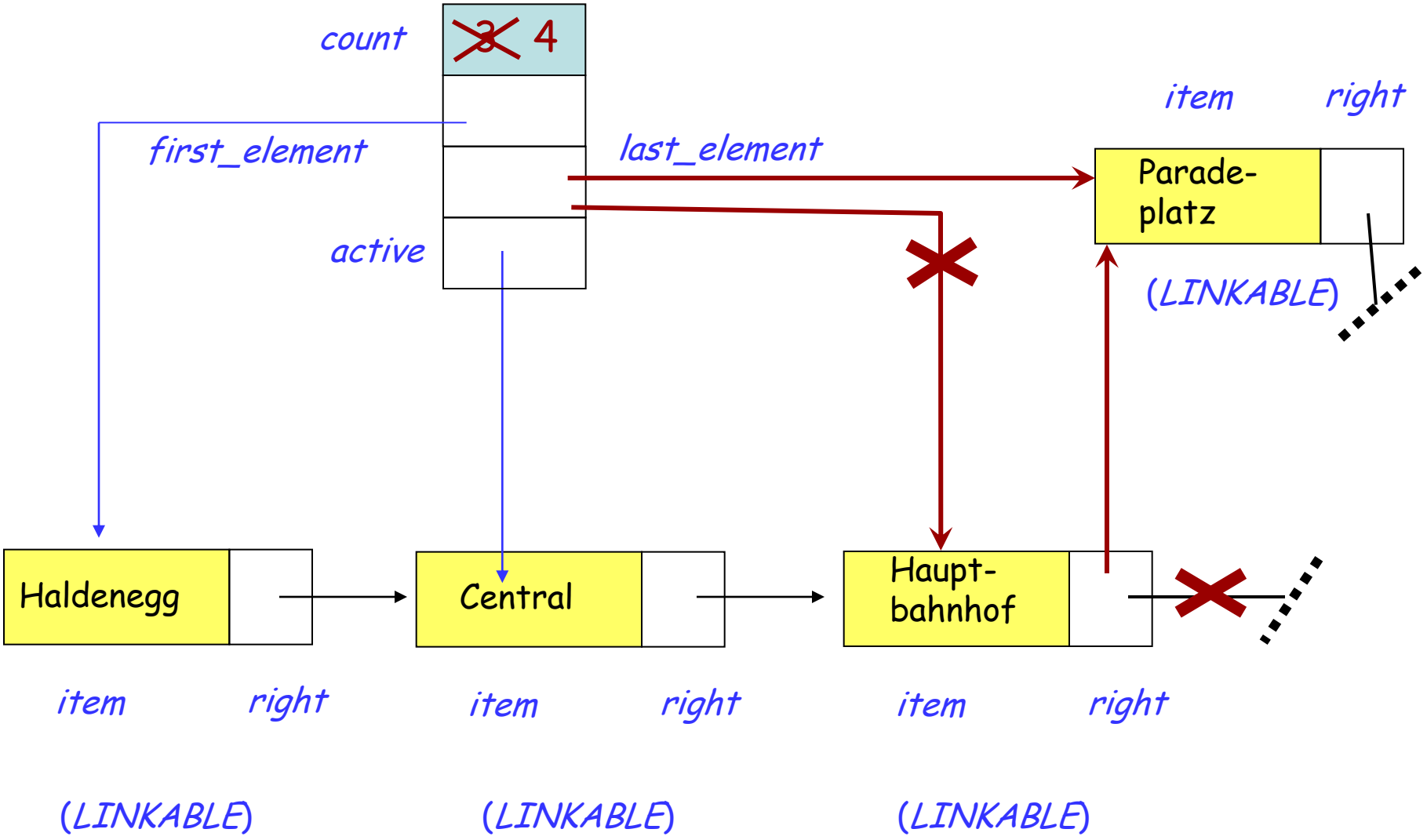

```
t: WERTE_PAAR
...
create t
...
t.setze (25, Void)
```

# Zuweisung

```
class STATION  feature
        ort : POSITION
        name : STRING
        länge : REAL

        setze_alle (o : POSITION; l : REAL; n : STRING)
            do
                ort := o
                länge := l
                name := n
            end
end
```

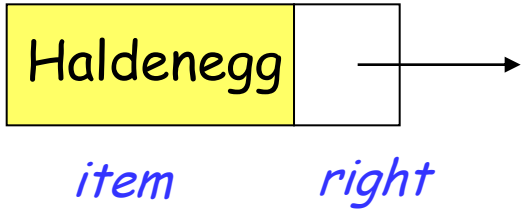
# Eine verkettete Liste von Strings: Einfügen am Ende



# Ein Element am Ende einfügen

```
extend (v: STRING)
                -- v am Ende hinzufügen.
                -- Cursor nicht verschieben.
        local
                p: LINKABLE [STRING]
        do
                create p.make (v)
                if is_empty then
                        first_element := p
                        active := p
                else
                        last_element.put_right (p)
                        if after then active := p end
                end
                last_element := p
                count := count + 1
        end
```

# *LINKABLE* - Zellen




```
class LINKABLE feature
        item: STRING
                        -- Wert dieser Zelle

        right: LINKABLE
                        -- Zelle, zu welcher diese Zelle angehängt ist
                        -- (falls vorhanden)

        put_right (other: like Current)
                        -- Setzt other rechts neben die aktuelle Zelle.
                do
                        right := other
                ensure
                        chained : right = other
                end
end
```

# Lokale Variablen (in Routinen)

Eine Art von Entität (Sie heissen auch „lokale Entitäten“).

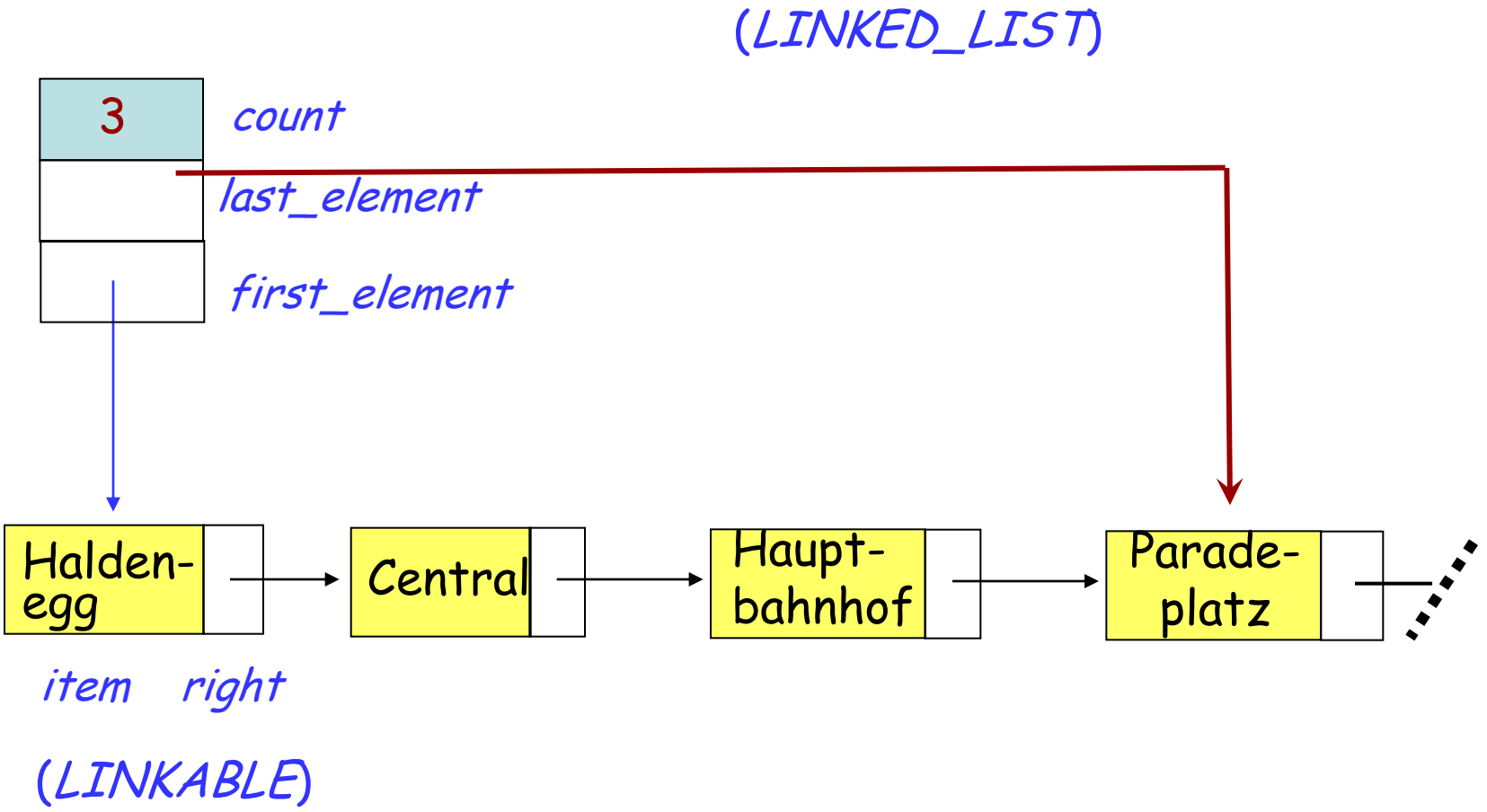
Deklarieren Sie sie einfach zu Beginn einer Routine (in der **local** Klausel)

```
r (...)
                    -- Kopfkommentar
        require
                    ...
        local
                    x : REAL
                    m : STATION
        do
                    ... x und m sind hier benutzbar ...
        ensure
                    ...
        end
```

*Result* ist (für Funktionen) auch eine lokale Variable.

# Übung (beinhaltet Schleifen)

Kehren Sie eine Liste um!



# Leseaufgaben für die nächsten 2 Wochen

Control structures: Kapitel 7

# Was wir gesehen haben

- Das aktuelle Objekt
- Expandierte Typen und Referenztypen
- Zuweisung:
  - Für Referenzen
  - Für expandierte Werte
- Verkettete Datenstrukturen
- Einen kurzen Blick auf bedingte Anweisungen.