Zusammenfassung:

Eine Klassen-Deklaration besteht folglich aus Deklarationen von:

- Attributen für die verschiedenen Wert-Komponenten der Objekte;
- Konstruktoren zur Initialisierung der Objekte;
- Methoden, d.h. Operationen auf Objekten.

Diese Elemente heißen auch Members der Klasse.

```
public class Rational {
   // Attribute:
   private int zaehler, nenner;
   // Konstruktoren:
   public Rational (int x, int y) {
      zaehler = x;
      nenner = y;
      }
   public Rational (int x) {
      zaehler = x;
      nenner = 1;
      }
       ...
```

```
// Objekt-Methoden:
public Rational add (Rational r) {
   int x = zaehler * r.nenner +r.zaehler * nenner;
   int y = nenner * r.nenner;
   return new Rational (x,y);
   }
public boolean equals (Rational r) {
   return (zaehler * r.nenner == r.zaehler * nenner);
   }
public String toString() {
   if (nenner == 1) return "" + zaehler;
   if (nenner > 0) return zaehler +"/"+ nenner;
   return (-zaehler) +"/"+ (-nenner);
   }
}  // end of class Rational
```

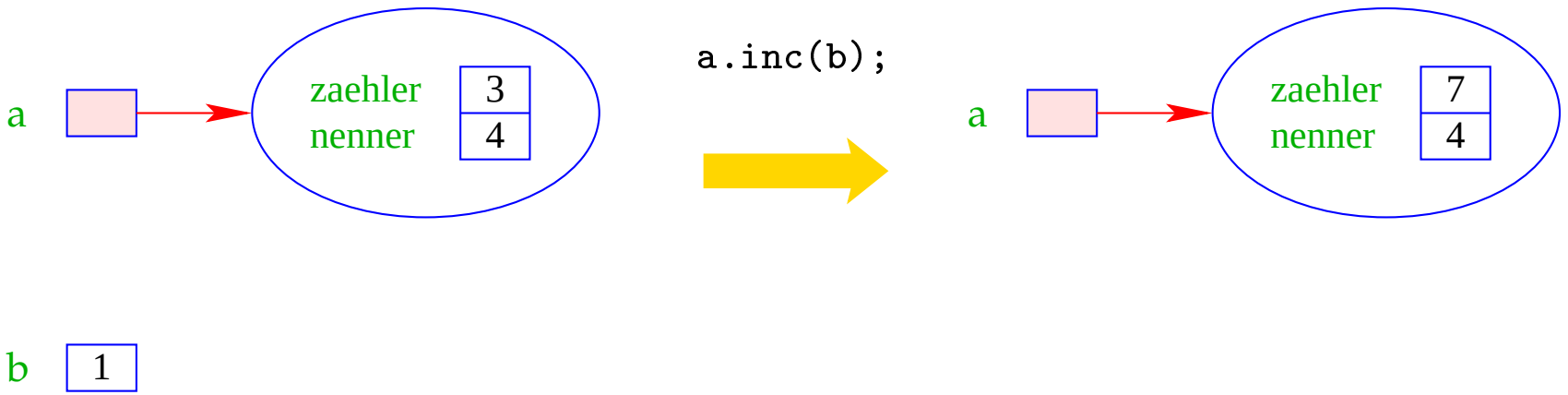
Bemerkungen:

- Jede Klasse sollte in einer separaten Datei des entsprechenden Namens stehen.
- Die Schlüsselworte `private` bzw. `public` klassifizieren, für wen die entsprechenden Members sichtbar, d.h. zugänglich sind.
- `private` heißt: nur für Members der gleichen Klasse sichtbar.
- `public` heißt: innerhalb des gesamten Programms sichtbar.
- Nicht klassifizierte Members sind nur innerhalb des aktuellen ↑Package sichtbar.

- Konstruktoren haben den gleichen Namen wie die Klasse.
- Es kann mehrere geben, sofern sie sich im Typ ihrer Argumente unterscheiden.
- Konstruktoren haben keine Rückgabewerte und darum auch keinen Rückgabetyp.
- Methoden haben dagegen stets einen Rückgabe-Typ, evt. `void`.

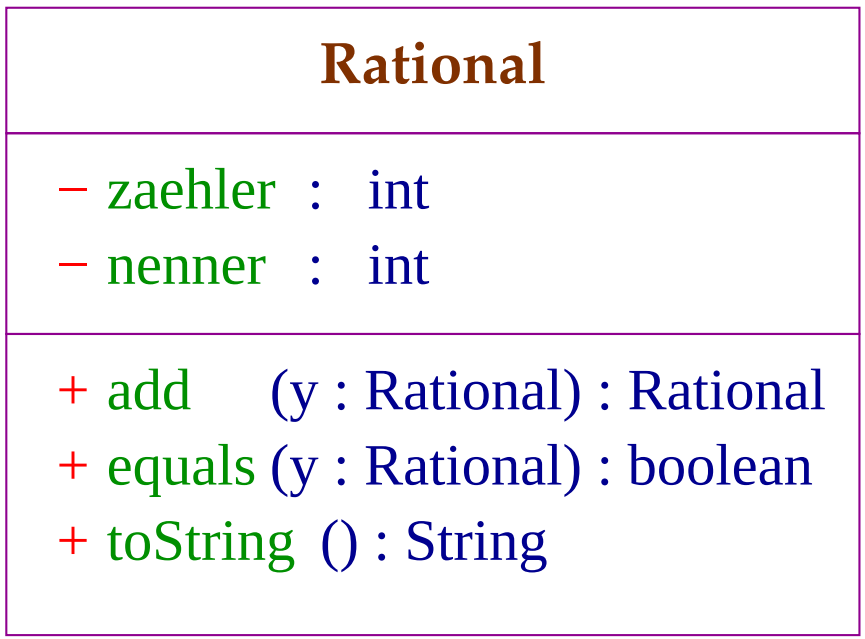
```
public void inc (int b) {
   zaehler = zaehler + b * nenner;
   }
```

- Die Objekt-Methode `inc()` modifiziert das Objekt, für das sie aufgerufen wurde.



- Die Objekt-Methode `equals()` ist nötig, da der Operator "==" bei Objekten die Identität der Objekte testet, d.h. die Gleichheit der Referenz !!!
- Die Objekt-Methode `toString()` liefert eine `String`-Darstellung des Objekts.
- Sie wird implizit aufgerufen, wenn das Objekt als Argument für die Konkantenation "+" auftaucht.
- Innerhalb einer Objekt-Methode/eines Konstruktors kann auf die Attribute des Objekts direkt zugegriffen werden.
- `private`-Klassifizierung bezieht sich auf die Klasse nicht das Objekt: die Attribute aller `Rational`-Objekte sind für `add` sichtbar !!

Eine graphische Visualisierung der Klasse Rational, die nur die wesentliche Funktionalität berücksichtigt, könnte so aussehen:



Diskussion und Ausblick:

- Solche Diagramme werden von der UML, d.h. der Unified Modelling Language bereitgestellt, um Software-Systeme zu entwerfen (↑Software Engineering)
- Für eine einzelne Klasse lohnt sich ein solches Diagramm nicht wirklich :-)
- Besteht ein System aber aus sehr vielen Klassen, kann man damit die Beziehungen zwischen verschiedenen Klassen verdeutlichen :-))

Diskussion und Ausblick:

- Solche Diagramme werden von der UML, d.h. der Unified Modelling Language bereitgestellt, um Software-Systeme zu entwerfen (↑Software Engineering)
- Für eine einzelne Klasse lohnt sich ein solches Diagramm nicht wirklich :-)
- Besteht ein System aber aus sehr vielen Klassen, kann man damit die Beziehungen zwischen verschiedenen Klassen verdeutlichen :-))

Achtung:

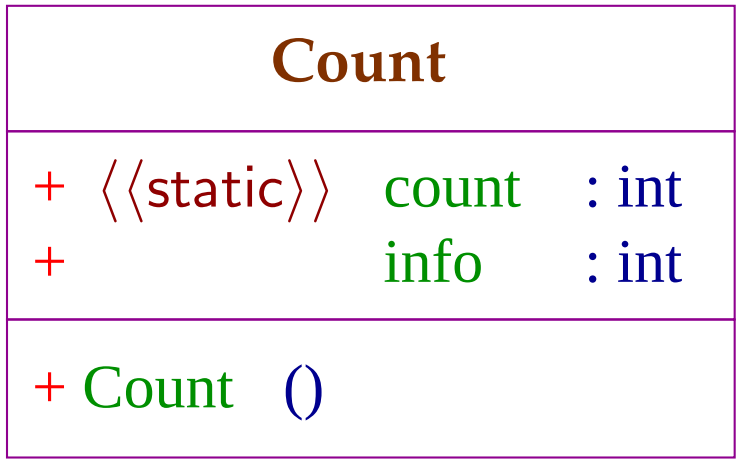
UML wurde nicht speziell für Java entwickelt. Darum werden Typen abweichend notiert. Auch lassen sich manche Ideen nicht eindeutig oder nur schlecht modellieren :-(

10.1 Klassen-Attribute

- Objekt-Attribute werden für jedes Objekt neu angelegt,
- Klassen-Attribute einmal für die gesamte Klasse :-)
- Klassen-Attribute erhalten die Qualifizierung `static`.

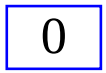
```
public class Count {
   private static int count = 0;
   private int info;
   // Konstruktor
   public Count() {
      info = count; count++;
      } ...
   } // end of class Count
```

Modellierung:

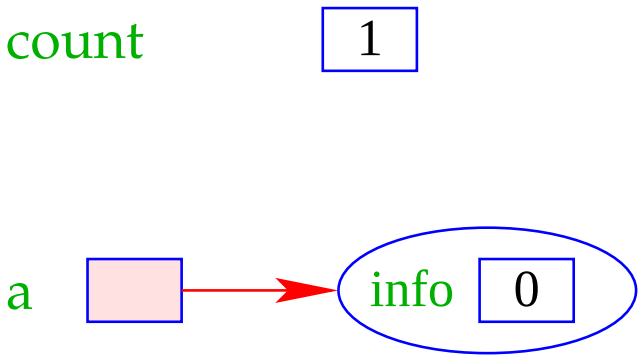


- Die Zusatzinformation, dass das Attribut `count` statisch ist, wird in in spitzen Klammern im Diagramm vermerkt.
- Solche Annotationen heißen Stereotype.

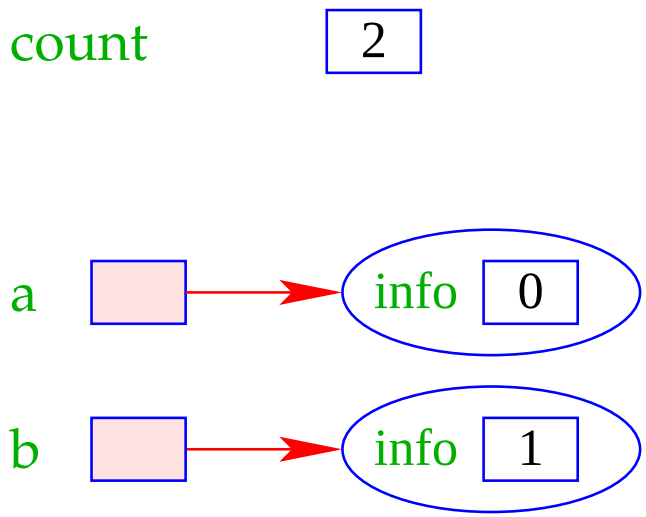
count



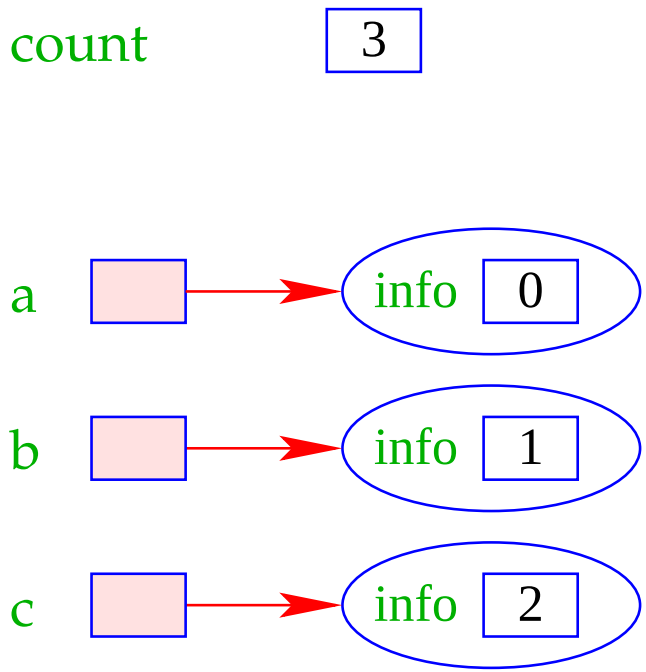

```
Count a = new Count();
```




```
Count b = new Count();
```




```
Count c = new Count();
```


count
3
a
info
0
b
info
1
c
info
2

- Das Klassen-Attribut `count` zählt hier die Anzahl der bereits erzeugten Objekte.
- Das Objekt-Attribut `info` enthält für jedes Objekt eine eindeutige Nummer.
- Außerhalb der Klasse Class kann man auf eine öffentliche Klassen-Variable name mithilfe von Class.name zugegreifen.

- Das Klassen-Attribut `count` zählt hier die Anzahl der bereits erzeugten Objekte.
- Das Objekt-Attribut `info` enthält für jedes Objekt eine eindeutige Nummer.
- Außerhalb der Klasse Class kann man auf eine öffentliche Klassen-Variable name mithilfe von Class.name zugegreifen.

- Objekt-Methoden werden stets mit einem Objekt aufgerufen ...
- dieses Objekt fungiert wie ein weiteres Argument :-)
- Funktionen und Prozeduren der Klasse ohne dieses implizite Argument heißen Klassen-Methoden und werden durch das Schlüsselwort `static` kenntlich gemacht.

In `Rational` könnten wir definieren:

```
public static Rational[] intToRationalArray(int[] a) {
   Rational[] b = new Rational[a.length];
   for(int i=0; i < a.length; ++i)
      b[i] = new Rational (a[i]);
   return b;
   }
```

In `Rational` könnten wir definieren:

```
public static Rational[] intToRationalArray(int[] a) {
   Rational[] b = new Rational[a.length];
   for(int i=0; i < a.length; ++i)
      b[i] = new Rational (a[i]);
   return b;
   }
```

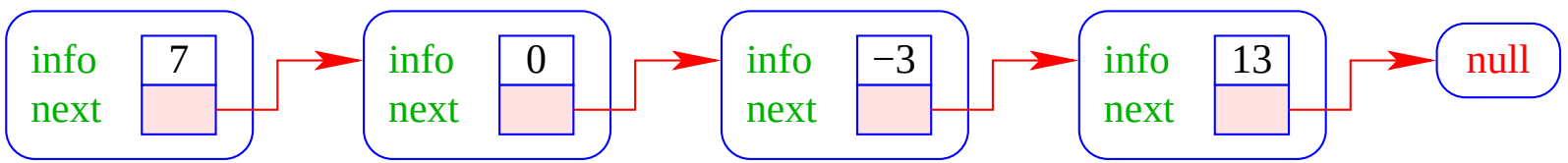
- Die Funktion erzeugt für ein Feld von `int`'s ein entsprechendes Feld von `Rational`-Objekten.
- Außerhalb der Klasse Class kann die öffentliche Klassen-Methode meth() mithilfe von `Class.meth(...)` aufgerufen werden.

11 Ein konkreter Datentyp: Listen

Nachteil von Feldern:

- feste Größe;
- Einfügen neuer Elemente nicht möglich;
- Streichen ebenfalls nicht :-(

Idee: Listen



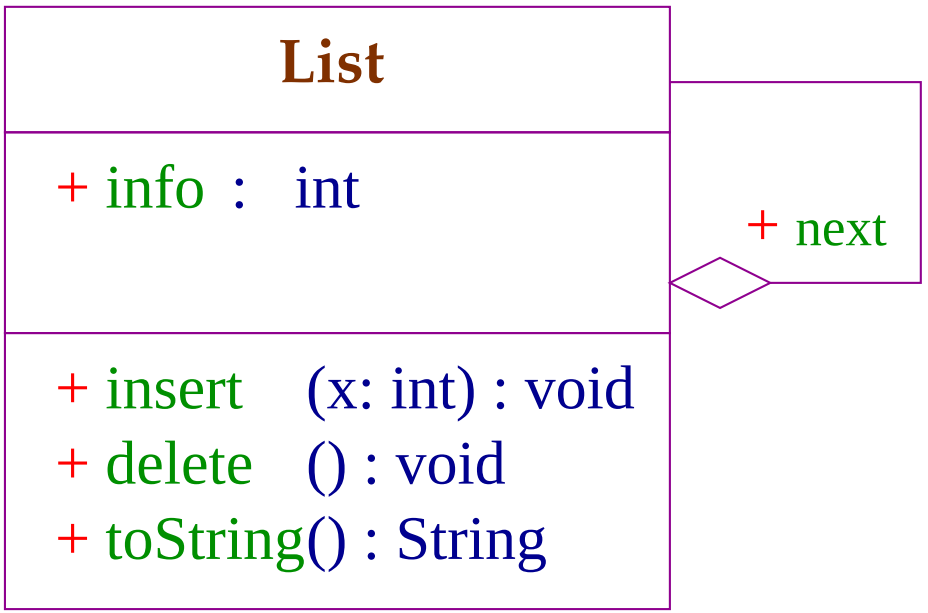
... das heißt:

- `info` $=$ Element der Liste;
- `next` $=$ Verweis auf das nächste Element;
- `null` $=$ leeres Objekt.

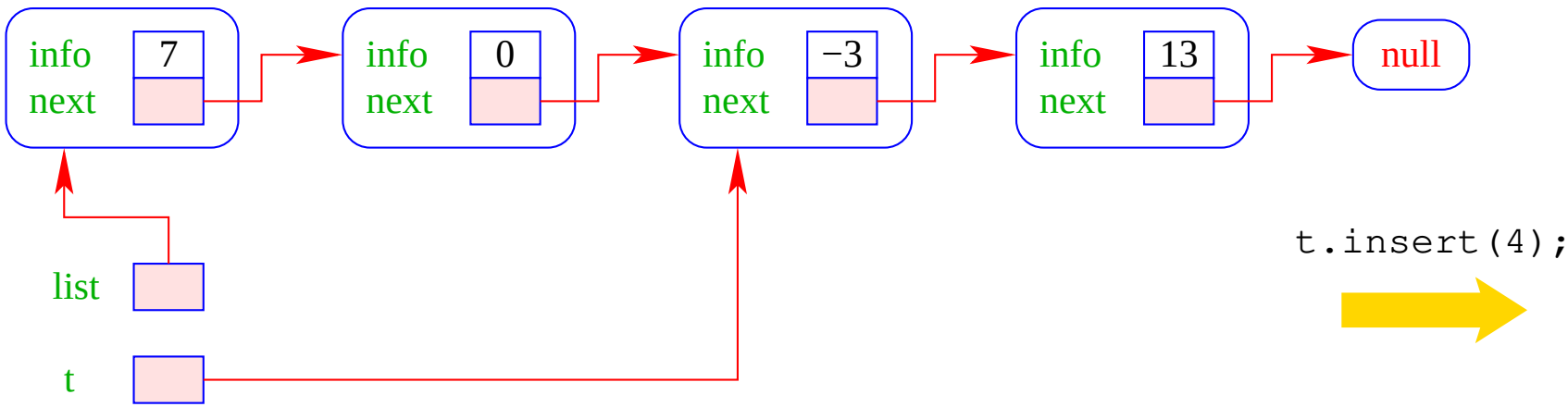
Operationen:

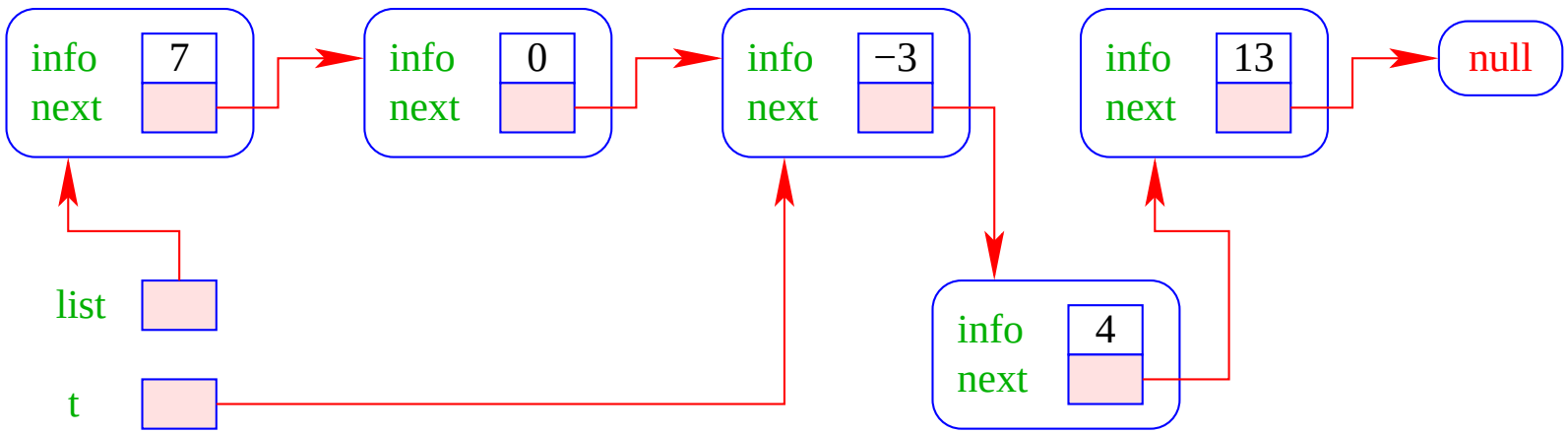
`void insert(int x)`	:	fügt neues x hinter dem aktuellen Element ein;
`void delete()`	:	entfernt Knoten hinter dem aktuellen Element;
`String toString()`	:	liefert eine String-Darstellung.

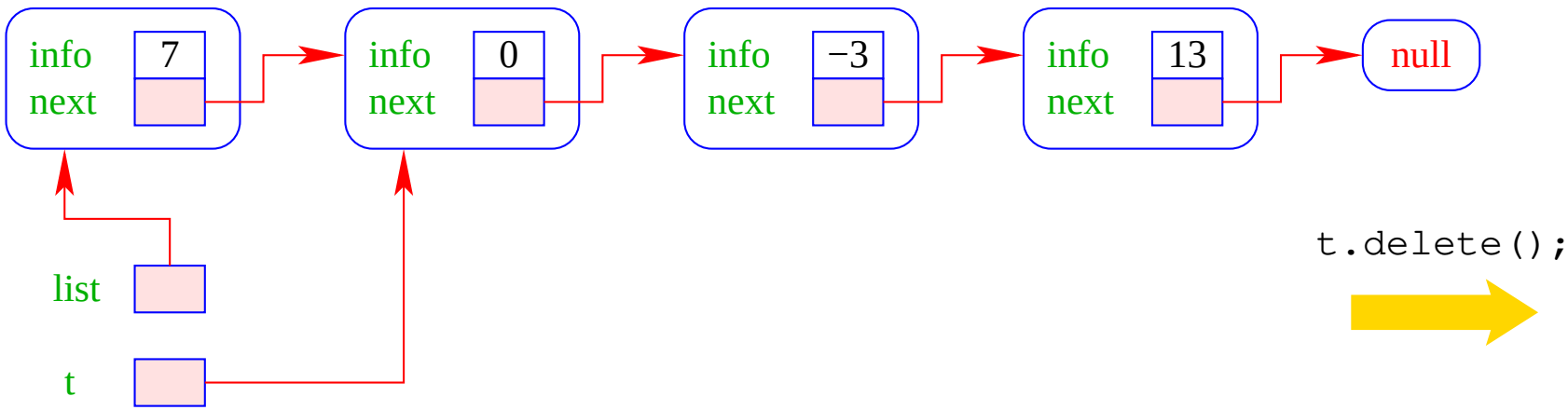
Modellierung:

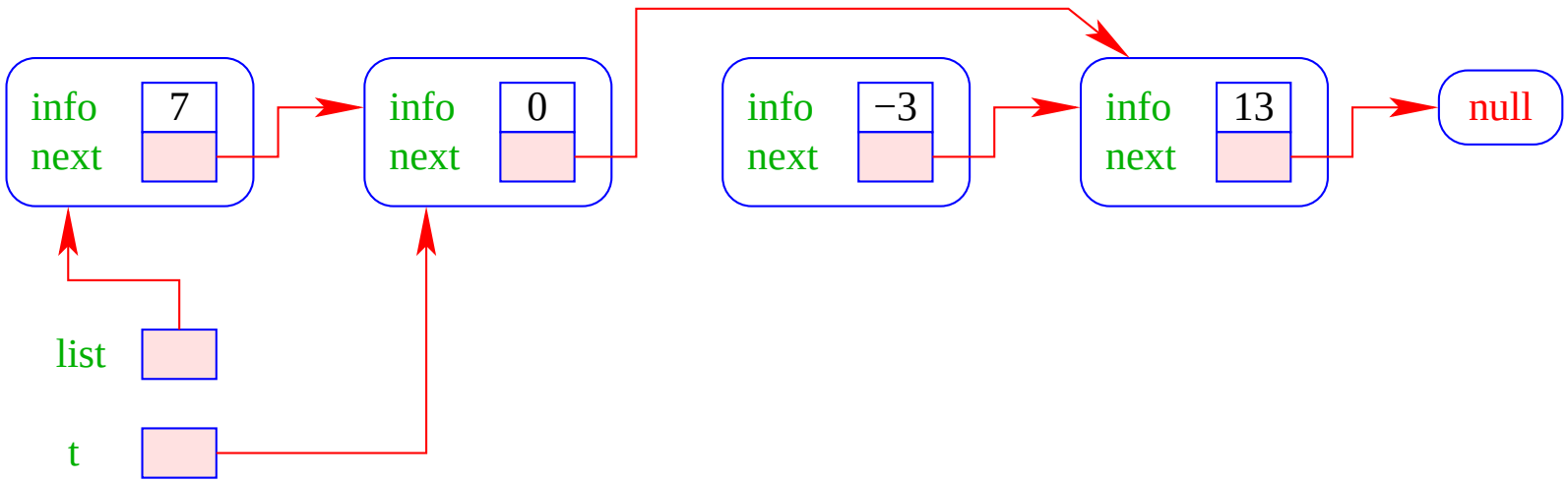


Die Rauten-Verbindung heißt auch Aggregation.


info
next
7
info
next
0
info
next
−3
info
next
13
null
list
t
t.insert(4);


info
next
7
info
next
0
info
next
−3
info
next
13
null
info
next
4
list
t


info
next
7
info
next
0
info
next
−3
info
next
13
null
list
t
t.delete();


info
next
7
info
next
0
info
next
−3
info
next
13
null
list
t

Weiterhin sollte man

- ... eine Liste auf Leerheit testen können;

- ... neue Listen bauen können, d.h. etwa:
  - ... eine ein-elementige Liste anlegen können;
  - ... eine Liste um ein Element verlängern können;
- ... Listen in Felder und Felder in Listen umwandeln können.

Weiterhin sollte man

- ... eine Liste auf Leerheit testen können;

Achtung:

das `null`-Objekt versteht keinerlei Objekt-Methoden!!!

- ... neue Listen bauen können, d.h. etwa:
  - ... eine ein-elementige Liste anlegen können;
  - ... eine Liste um ein Element verlängern können;
- ... Listen in Felder und Felder in Listen umwandeln können.

```
public class List {
   public int info;
   public List next;
// Konstruktoren:
   public List (int x, List l) {
      info = x;
      next = l;
      }
   public List (int x) {
      info = x;
      next = null;
      }
   ...
```