

## Script generated by TTT

Title: Seidl: Informatik\_1 (21.11.2012)

Date: Wed Nov 21 15:15:54 CET 2012

Duration: 90:46 min

Pages: 43

- 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` zugreifen.

*Xo info*  
~~*Count.count*~~

345

## 10.2 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
```

340

`count` 0

`Count a = new Count();`



341

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;
}
```

347

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.

348

## 11 Abstrakte Datentypen

- Spezifiziere nur die Operationen!
- Verberge Details
  - der Datenstruktur;
  - der Implementierung der Operationen.

⇒ Information Hiding

349

Sinn:

- Verhindern illegaler Zugriffe auf die Datenstruktur;
- Entkopplung von Teilproblemen für
  - Implementierung, aber auch
  - Fehlersuche und
  - Wartung;
- leichter Austausch von Implementierungen (↑`rapid prototyping`).

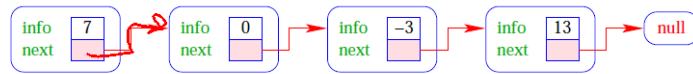
350

## 11.1 Ein konkreter Datentyp: Listen

Nachteil von Feldern:

- feste Größe;
- Einfügen neuer Elemente nicht möglich;
- Streichen ebenfalls nicht.

Idee: Listen



351

... 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.
```

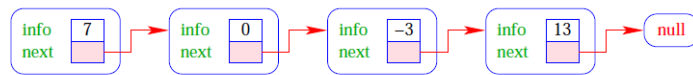
352

## 11.1 Ein konkreter Datentyp: Listen

Nachteil von Feldern:

- feste Größe;
- Einfügen neuer Elemente nicht möglich;
- Streichen ebenfalls nicht.

Idee: Listen



351

... das heißt:

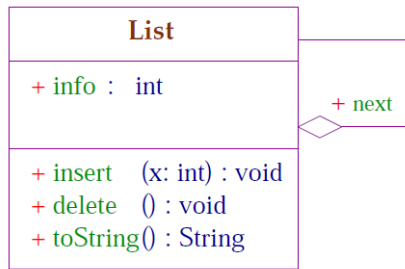
- info == Element der Liste;
- next == Verweis auf das nächste Element;
- null == leeres Objekt.

Operationen:

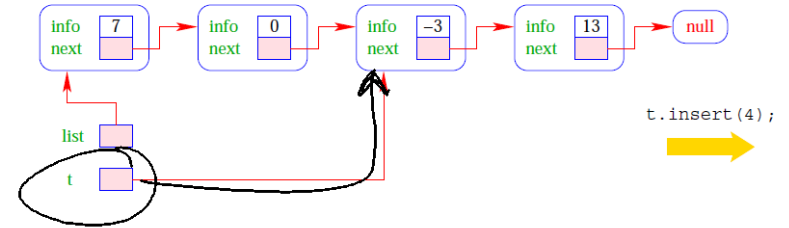
```
void insert(int x) : fügt neues  $x$  hinter dem aktuellen Element ein;  
void delete() : entfernt Knoten hinter dem aktuellen Element;  
String toString() : liefert eine String-Darstellung.
```

352

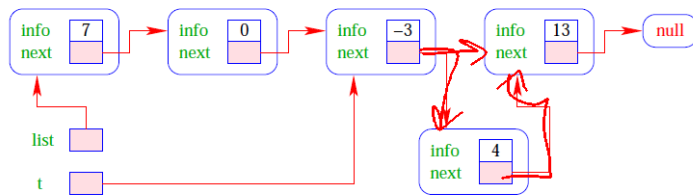
Modellierung:



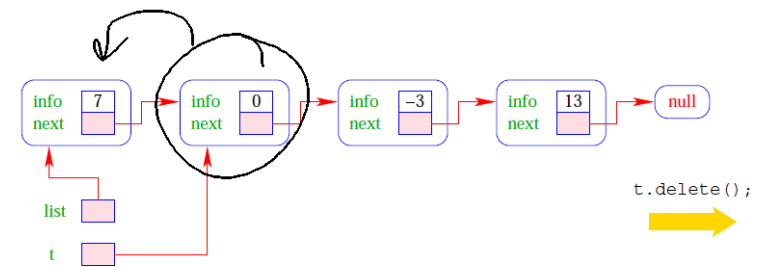
353



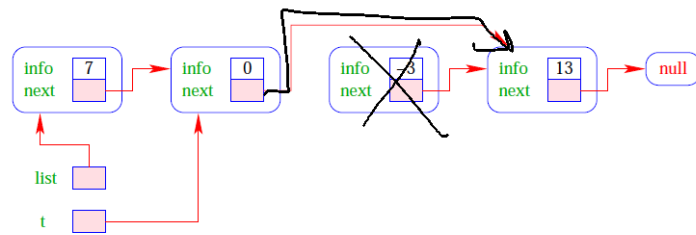
354



355



356



357

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.

358

Weiterhin sollte man

- ... eine Liste auf Leerheit testen können;

**Achtung:**

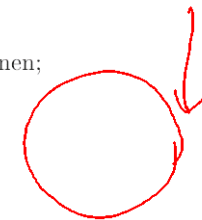
das null-Objekt versteht **keinerlei** Objekt-Methoden!!!

- ... 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.

359

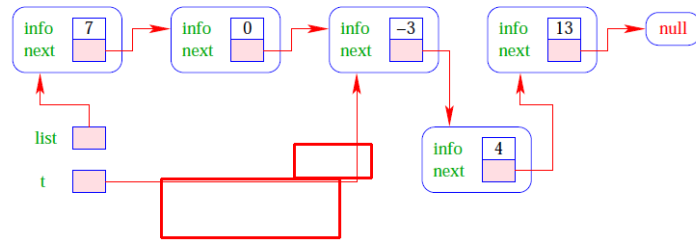
Weiterhin sollte man

- ... 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.

358



355

```
// Objekt-Methoden:
public void insert(int x) {
    next = new List(x,next);
}
public void delete() {
    if (next != null)
        next = next.next;
}
public String toString() {
    String result = "["+info;
    for(List t=next; t!=null; t=t.next)
        result = result+", "+t.info;
    return result+"]";
}
...

```

361

Weiterhin sollte man

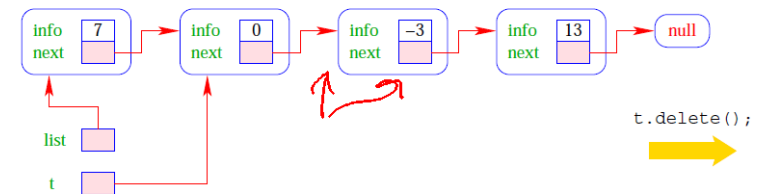
- ... eine Liste auf Leerheit testen können;

**Achtung:**

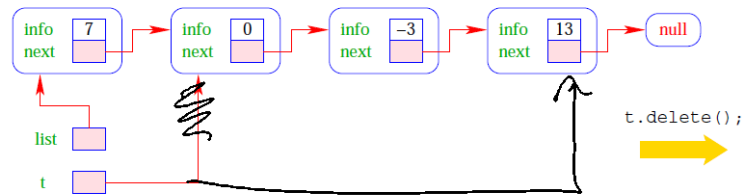
das null-Objekt versteht **keinerlei** Objekt-Methoden!!!

- ... 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.

359



356

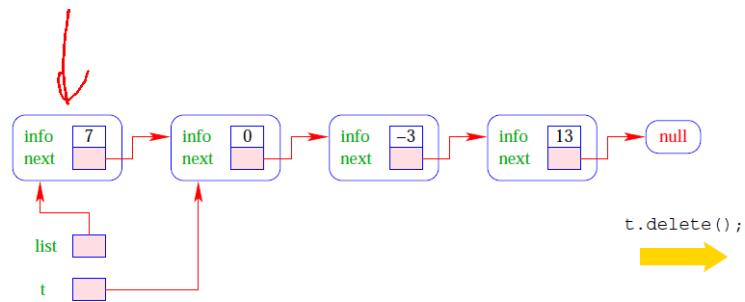


356

```
// Objekt-Methoden:
public void insert(int x) {
    next = new List(x,next);
}
public void delete() {
    if (next != null)
        next = next.next;
}
public String toString() {
    String result = "["+info;
    for(List t=next; t!=null; t=t.next)
        result = result+", "+t.info;
    return result+"]";
}
...

```

361



356

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.

358

- Die Attribute sind `public` und daher beliebig einsehbar und modifizierbar  $\implies$  sehr flexibel, sehr fehleranfällig.
- `insert()` legt einen neuen Listenknoten an fügt ihn hinter dem aktuellen Knoten ein.
- `delete()` setzt den aktuellen `next`-Verweis auf das übernächste Element um.

### Achtung:

Wenn `delete()` mit dem letzten Element der Liste aufgerufen wird, zeigt `next` auf `null`.

$\implies$  Wir tun dann nix.

362

```
// Objekt-Methoden:
public void insert(int x) {
    next = new List(x, next);
}

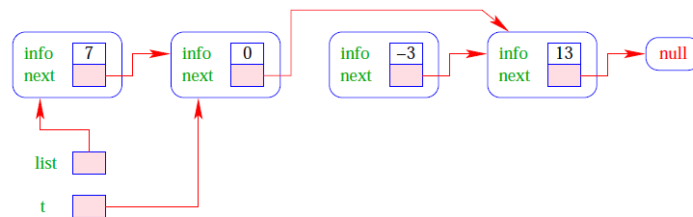
public void delete() {
    if (next != null)
        next = next.next;
}

public String toString() {
    String result = "["+info;
    for(List t=next; t!=null; t=t.next)
        result = result+", "+t.info;
    return result+"]";
}
...

```

*Handwritten red notes:*  
`next = new List(42, list);`  
`next = list.next;`

361



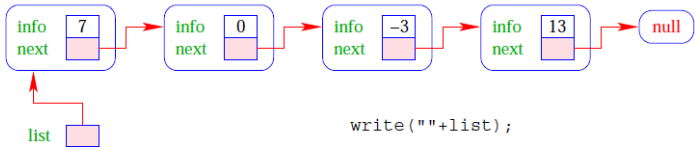
357

- Weil Objekt-Methoden nur für von `null` verschiedene Objekte aufgerufen werden können, kann die leere Liste nicht mittels `toString()` als `String` dargestellt werden.
- Der Konkatinations-Operator `+` ist so schlau, vor Aufruf von `toString()` zu überprüfen, ob ein `null`-Objekt vorliegt. Ist das der Fall, wird `"null"` ausgegeben.
- Wollen wir eine andere Darstellung, benötigen wir eine Klassen-Methode `String toString(List l)`.



365

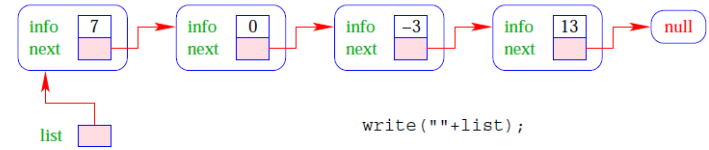




write(""+list);



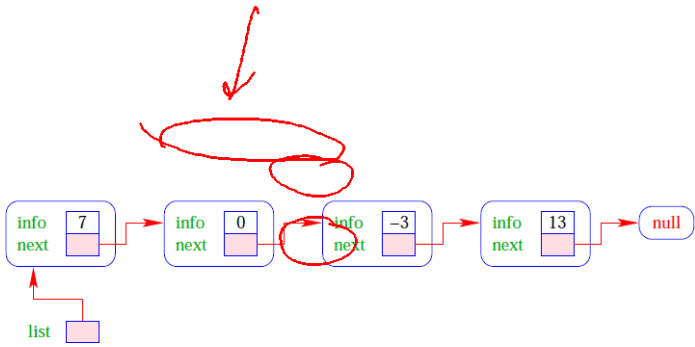
~~if (l == null) return "[]";~~  
 if (l == null) return "[]";  
 else l.toString();  
 }  
 366



write(""+list);



366



"[7, 0, -3, 13]"

367



write(""+list);



368



"null"

369

```
public static List arrayToList(int[] a) {
    List result = null;
    for(int i = a.length-1; i>=0; --i)
        result = new List(a[i],result);
    return result;
}
public int[] listToArray() {
    List t = this;
    int n = length();
    int[] a = new int[n];
    for(int i = 0; i < n; ++i) {
        a[i] = t.info;
        t = t.next;
    }
    return a;
}
...
371
```

- Damit das erste Element der Ergebnis-Liste a[0] enthält, beginnt die Iteration in `arrayToList()` beim **größten** Element.
- `listToArray()` ist als Objekt-Methode realisiert und funktioniert darum nur für **nicht-leere** Listen.
- Um eine Liste in ein Feld umzuwandeln, benötigen wir seine Länge.

```
private int length() {
    int result = 1;
    for(List t = next; t!=null; t=t.next)
        result++;
    return result;
}
} // end of class List
```

372

- Weil `length()` als `private` deklariert ist, kann es nur von den Methoden der Klasse `List` benutzt werden.
- Damit `length()` auch für `null` funktioniert, hätten wir analog zu `toString()` auch noch eine Klassen-Methode `int length(List l)` definieren können.
- Diese Klassen-Methode würde uns ermöglichen, auch eine Klassen-Methode `static int [] listToArray (List l)` zu definieren, die auch für leere Listen definiert ist.

373

- Weil `length()` als `private` deklariert ist, kann es nur von den Methoden der Klasse `List` benutzt werden.
- Damit `length()` auch für `null` funktioniert, hätten wir analog zu `toString()` auch noch eine Klassen-Methode `int length(List l)` definieren können.
- Diese Klassen-Methode würde uns ermöglichen, auch eine Klassen-Methode `static int [] listToArray (List l)` zu definieren, die auch für leere Listen definiert ist.

373

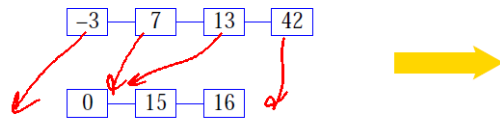
Anwendung: Mergesort – Sortieren durch Mischen

Mischen:

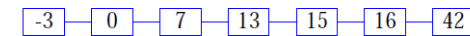
Eingabe: zwei sortierte Listen;

Ausgabe: eine gemeinsame sortierte Liste.

374



375



377