

nächste Woche

VL Di, Do, Fr

UE Mi

schriftliche Rücksprache letzte VL-Woche

vor Weihnachten, vermutlich 14.12 oder 15.12

Datenstrukturen

kennen bereits arrays, Strings

heute: Records, Listen

5.5. Records

- Merkmale:
- feste Komponentenanzahl ← wie arrays
 - direkter Zugriff auf Komponenten
über Namen von Komponenten ← bei Arrays über Indices
 - heterogene Komponententypen
↑ allgemeiner als arrays

Komponenten heißen (Daten-)Felder eines Records

Pseudocode:

```
type StudentRec = record
```

```
  String name;
```

```
  String adresse;
```

```
  int matrikelnr;
```

```
  String fach;
```

```
end record
```

} Komponententypen
und Namen

Zugriff auf Komponenten erfolgt mit •

```
StudentRec student;  
student.matrikelnr = 12758;
```

In Java gibt es keine Records, aber Klassen

allgemeiner als Records
da sie neben Datenfeldern
auch Methoden umfassen

Bsp: Java Klasse für Studenten

```
public class Student {
```

```
    public String name;  
    public String adresse;  
    public int matrikelnr;  
    public String fach;  
    public int fachsemester;
```

⋮

// Methoden:

// Konstruktor für Neuanzeige

```
public Student (String aktName, String aktAdr,  
                int aktMatNr, String aktFach) {
```

```
    name = aktName;  
    adresse = aktAdresse;
```

} Datenfelder

← Konstruktor heißen
wie die Klasse,
haben keinen Rückgabe-
^{typ}

```
matrikelnr = aktMatriNr;  
fach = aktFach;  
fachsemester = 1;
```

```
}
```

```
public void changeAddress ( String newAdr ) {  
    adresse = newAdr;  
}
```

```
// viele weitere Methoden
```

```
}
```

Verwendung der Klasse Student in einer Datenbank:

Array von Student

```
Student [] comTeilnehmer = new Student [160];
```

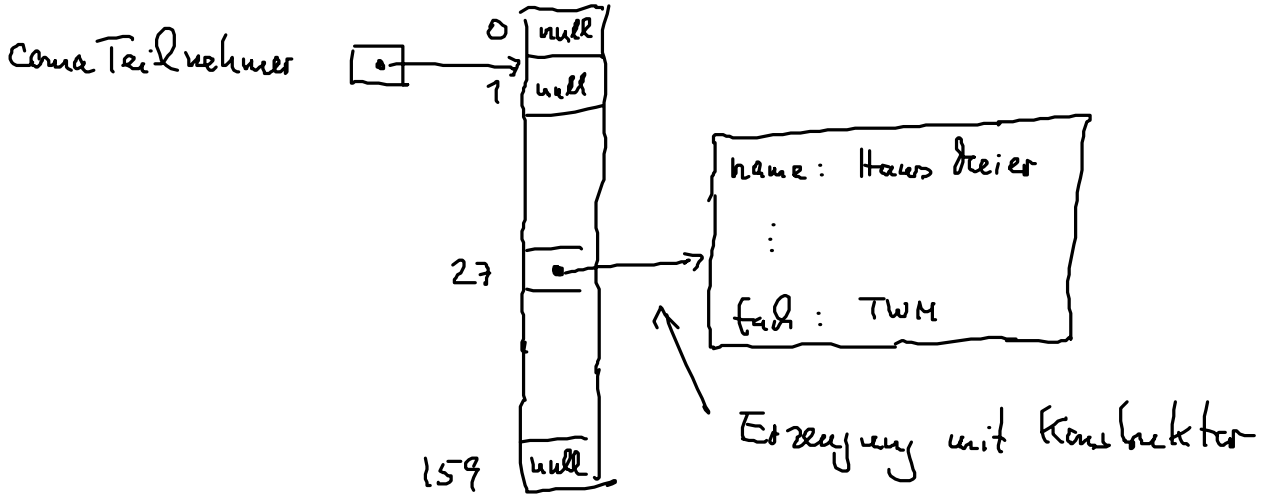
...

```
comTeilnehmer [27] = new Student ( "Hans Meier", "unbekannt",  
    ↗ 20307, "TUM" );
```

aufruf Konstruktor aus Klasse Student

```
comTeilnehmer [27].changeAddress ( "Unter den Linden 27" );
```

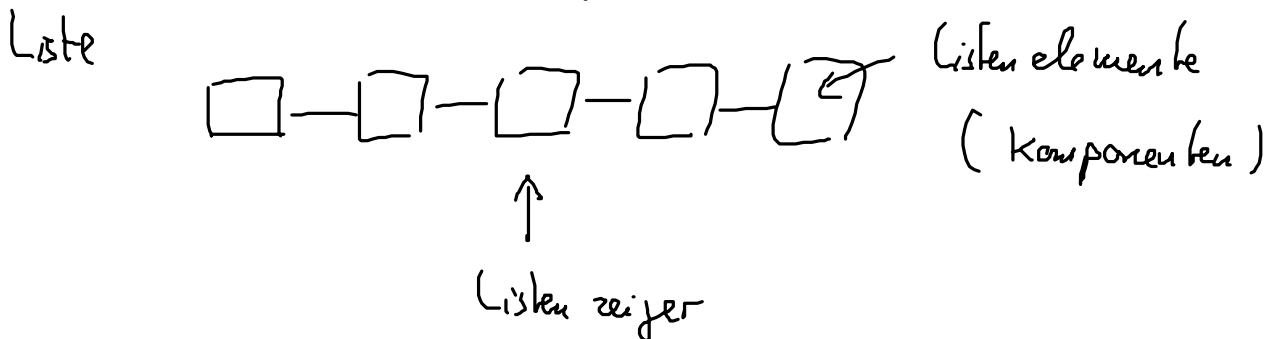
Im Speicher:



5.6 Listen

- Kennzeichen:
- veränderliche Länge (wachsen, schrumpfen)
 - homogene Komponenten (elementar oder strukturiert)
 - sequenzieller Zugriff

Vorstellung: Zugriff über Listenzeiger, der immer nur um eins weitergesetzt werden kann



"Güterwagen an Verladestation"

Listenoperationen:

- Einfügen eines neuen Elementes
- Löschen eines Elementes
- Ändern der Daten eines Elementes
-

Listen sehr wichtig wenn zusammen gesetzte Daten sich im Umfang ändern können

Bsp: Textverarbeitung

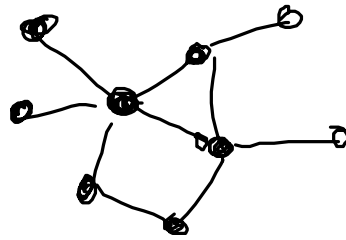
Zeile = Liste von Wörtern

Paragraf = Liste von Zeilen

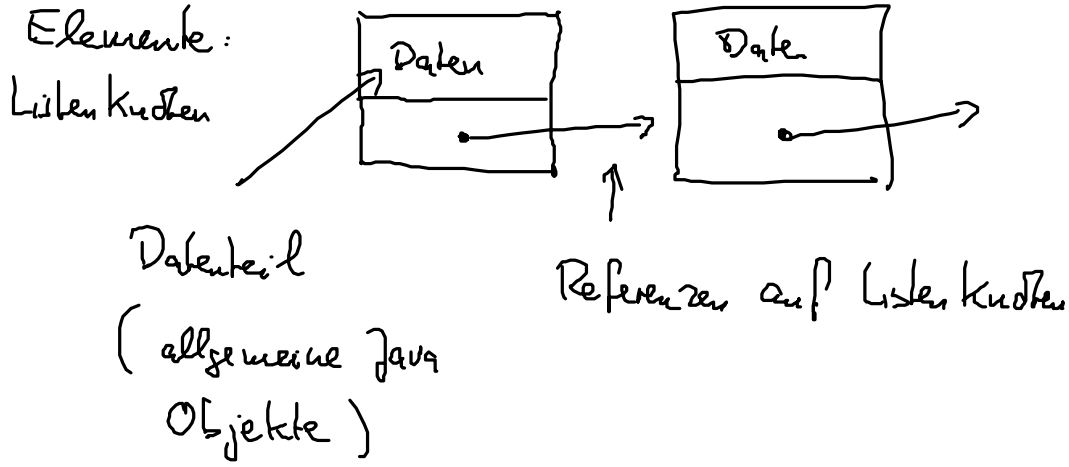
.....

Mathematik: Netzwerke

Nachbarn eines Knoten in Liste verwaltet



Wollen eigene Klasse für Listen entwerfen und in einer Testdemo verwenden



Klasse für Listenknoten: Datenfelder: data (allg. Objekt)
next (Referenz auf Listenknoten)

Methoden u. Konstruktoren

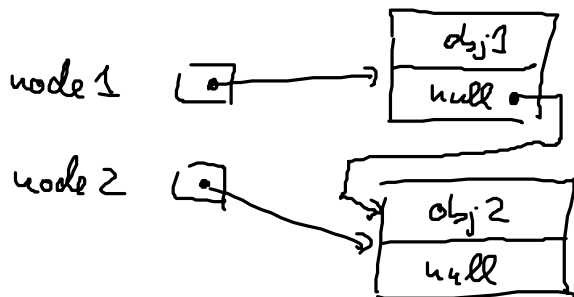
Zugriff (lesend und schreibend) auf Datenfelder
mit set, get Methoden

Java Code ansehen

Verwendung:

```
ListNode node1 = new ListNode(obj1);
```

```
ListNode node2 = new ListNode(obj2);
```



will node 1 auf node 2 zeigen lassen
next von

```
node1.setNext(node2); //
```

5.6.1 Einsdub javadoc (übung nachste Mittwoch mehr)

Kommentare in /** ... */
↑

werden von Programm javadoc verarbeitet,
erzeugt hieraus HTML Seiten zur Dokumentieren

Kommentare können Tags verwenden

@ see

@ author

@ param ← info über Parameter

@ return ← info über Rückgabewert

@ exception

Außerdem HTML code verwendbar

```
<code> ... </code>
```

↓
für diesen Text wird Schreibmaschinentext erzeugt.

Aufruf erfolgt mit

javadoc ListNode.java

javadoc *.java

Info über javadoc z.B. über man pages

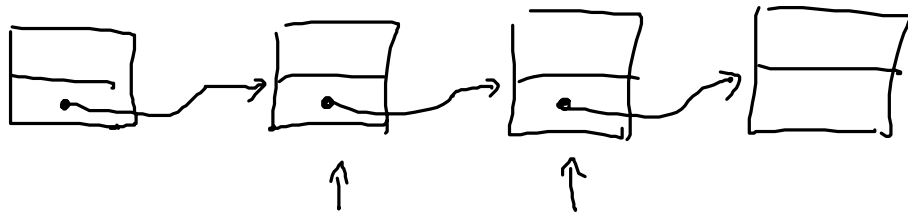
man javadoc

26.2. Eine Klasse für Listen

Objekte der Klasse sollen Listen sein (nicht einzelne Knoten wie bei ListNode)

Vorgabe:

intern,
mit
private
variablen



ListNode Objekte

die verkettet sind wie im Bild

Operationen:

neue Liste erzeugen (leere Liste)

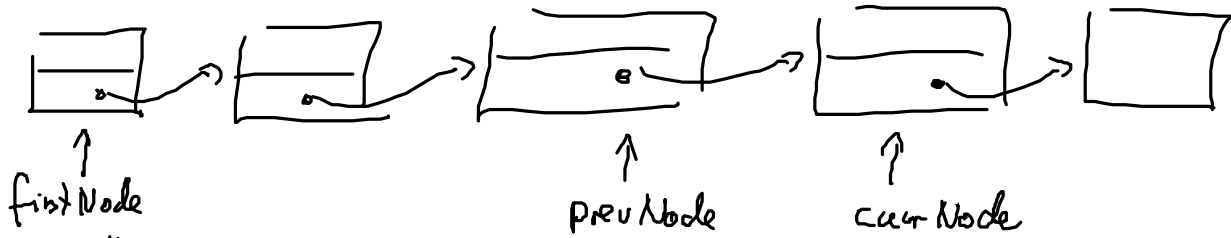
Elemente einfügen

Elemente löschen

set, get Methoden

Testen ob Liste leer,
Reset auf Anfang
...

Implementieren dieser Methoden (nach außen verborgen)



verwende 3 "Listenziger" (Referenzen auf ListNode)

firstNode zeigt auf erstes

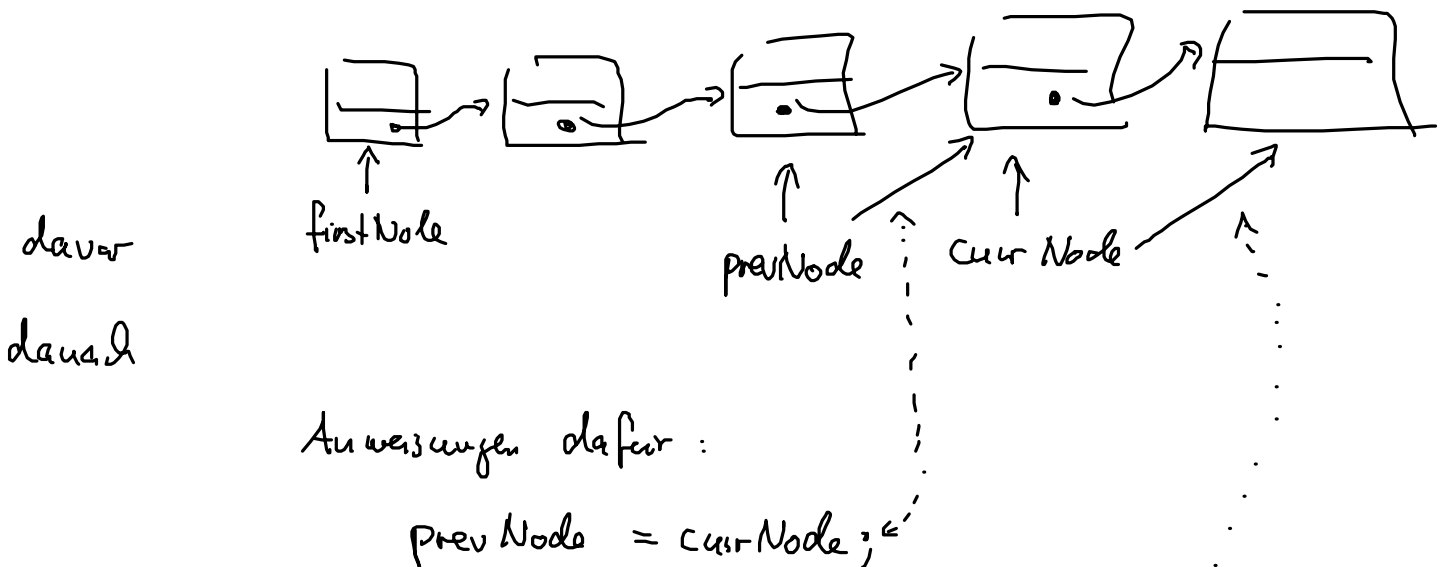
currNode zeigt auf momentanes Listenelement

prevNode zeigt auf Vorgänger des currNode

nach außen nicht sichtbar, da Implementations-abhängig

Java Code der Klasse

advance() im Regelfall



. curNode = curNode.getNext();

Exception werfen, wenn curNode == null