5. Globalübung (zu Übungsblatt 7)

- Wiederholung: Rekursion
- Rekursive dynamische Datenstrukturen
  - lineare Liste
  - weitere Beispiele
  - ausführliches Beispiel (Vielwegbaum)

Rekursion (Wdh.)

- Aufruf oder Definition einer Funktion durch sich selbst

- Beispiele:
  - Fakultätsfunktion
  - Fibonacci-Funktion
  - Türme von Hanoi

Verschränkte Rekursion

- Funktionen rufen sich gegenseitig auf:

```
public void f (...){
    ...
    g(...);
    ...
}
public void g(...){
    ...
    f(...);
    ...
}
```

Beispiel:

- Bestimme, ob eine Zahl gerade oder ungerade ist:

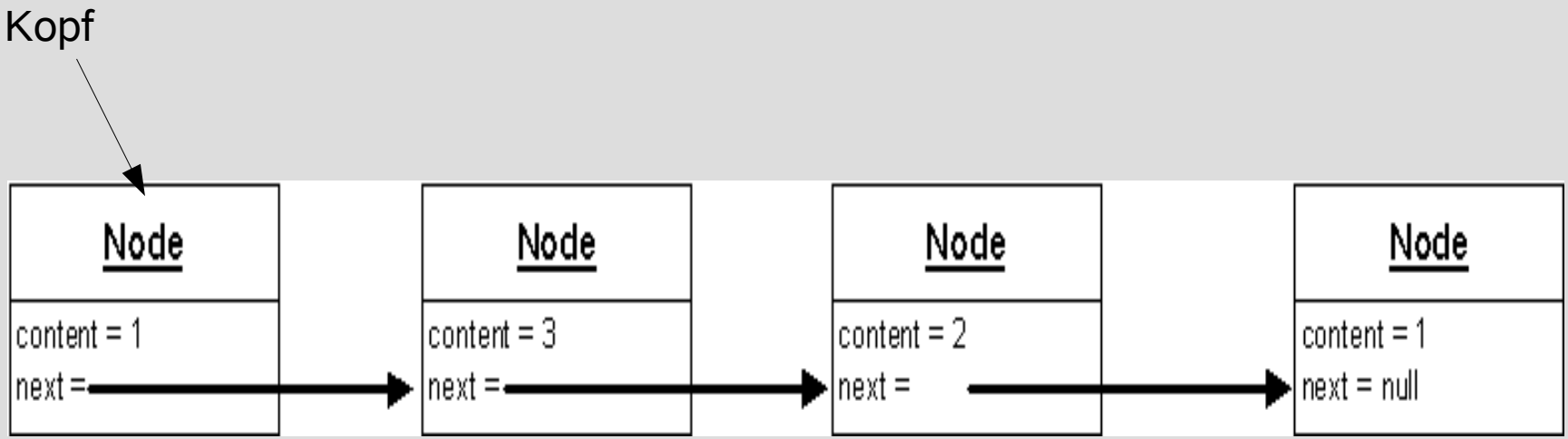
```
public static boolean even (int x) {
  if (x == 0)       return true;
  else if (x > 0)   return odd(x-1);
  else              return odd(x+1);
}

public static boolean odd (int x) {
  if       (x == 0) return false;
  else if (x > 0)   return even(x-1);
  else              return even(x+1);
}
```

Rekursive dynamische Datenstrukturen

- Vorteil:
  - passen ihre Größe dynamisch an (im Gegensatz zu z.B. Arrays)
  - häufig einfach zu implementierende rekursive Methoden (dank rekursivem Aufbau der Datenstruktur)

Lineare Liste

- Jeder Knoten hat einen Inhalt
  (z.B. int, boolean, aber auch Listen)
- und einen Verweis auf das nächste Element

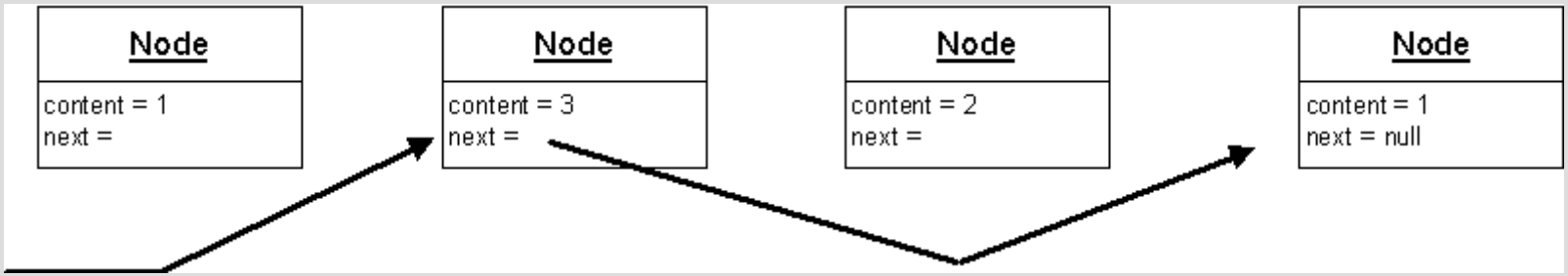


Einfügen in eine sortierte Liste

```
public void fuegeSortiertEin (int wert) {
  kopf = fuegeSortiertEin (wert, kopf);
}


private Element fuegeSortiertEin (int wert, Element element) {
  if (element == null){          // Ende der Liste ist erreicht
    return new Element(wert);
  }
  else if (wert < element.wert()) { // Einfügestelle gefunden
    return new Element (wert, element);
  }
  else {              // wenn wert>element.wert dann suche weiter
    element.next = fuegeSortiertEin (wert, element.next);
    return element;
  }
}
```

Löschen aus einer Liste:

- entspricht dem Umsetzen von Verweisen
  - d.h. das next Element eines Knotens wird auf ein anderes Element gesetzt (auf den Nachfolger des zu löschenden Knotens)



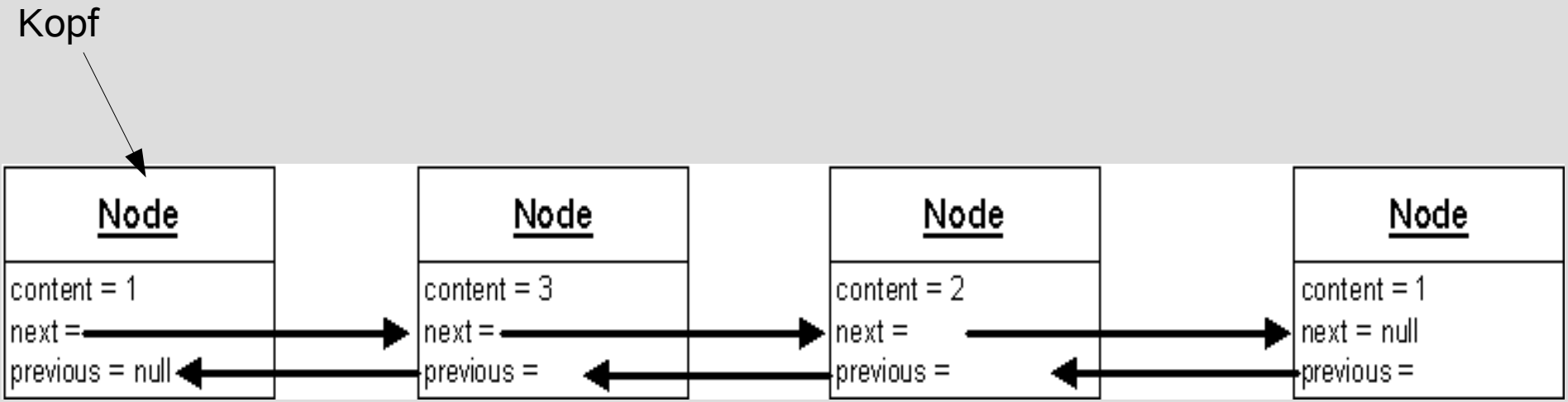
Hinweis

- in Listen können natürlich auch beliebige andere Werte wie z.B. Strings oder komplexere Datentypen gespeichert werden

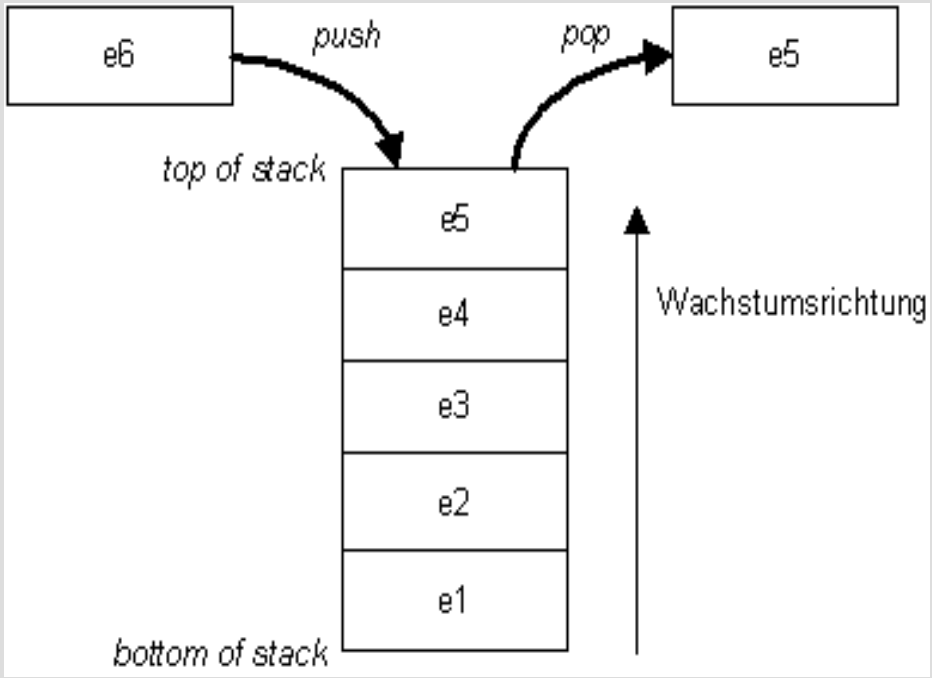
(auch z.B. Listen (siehe Übungsblatt 7))

Doppelt verkettete Liste

- Jeder Knoten besitzt einen Zeiger auf seinen Vorgänger und einen auf seinen Nachfolger
- dadurch wird das Einfügen und Löschen vereinfacht



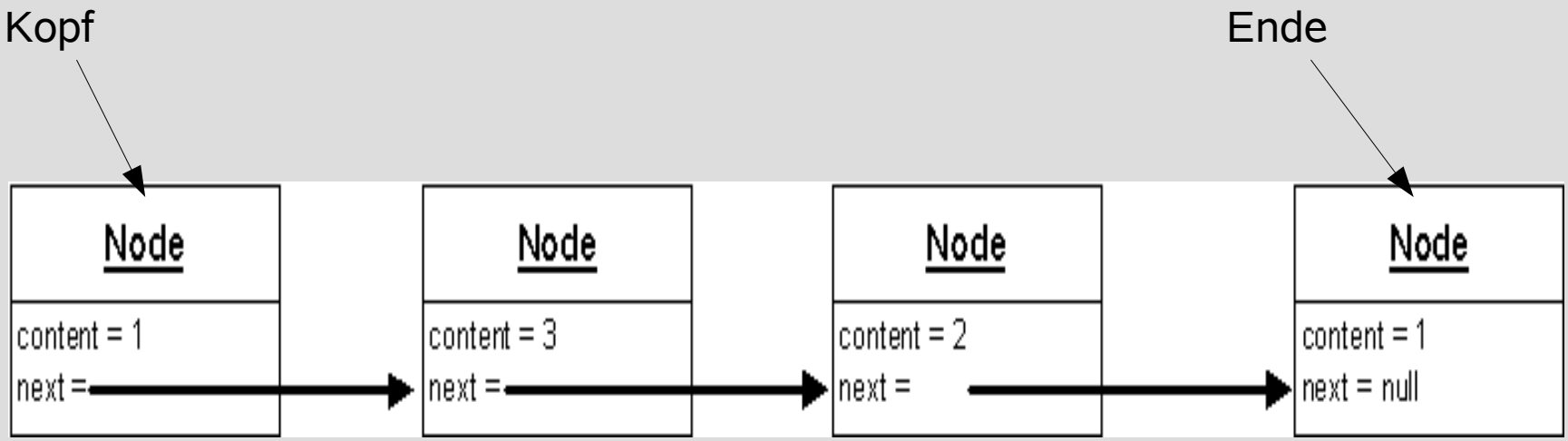
Stack (Stapel)



- Zeiger auf oberstes Element
- Methoden
  - push (hinzufügen)
  - pop (löschen)
  - top (anschauen)
- hinzufügen & löschen immer an der Spitze

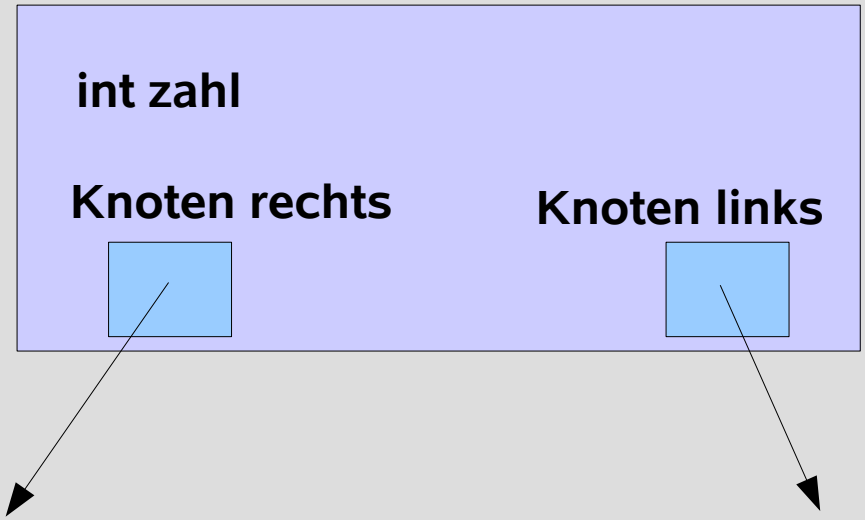
Queue (Schlange)

- Lineare Liste
- Methoden:
  - einfügen nur am Kopf
  - löschen/entnehmen nur am Ende

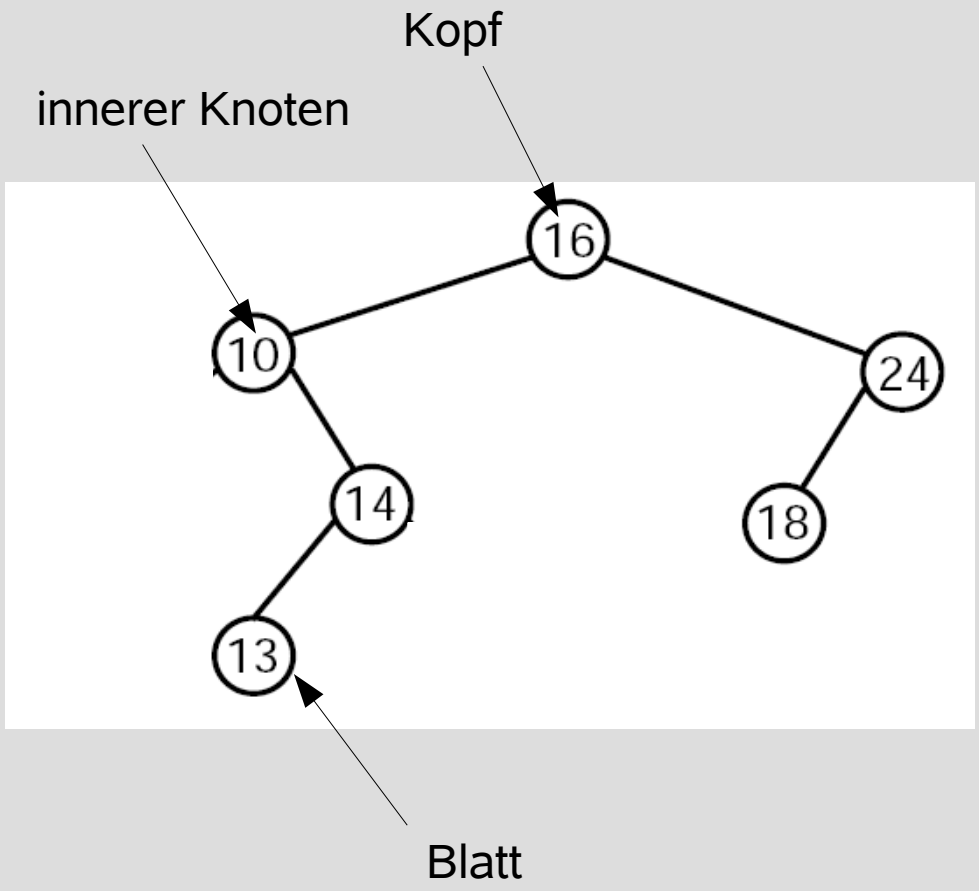


Binärer Baum

- Besteht aus:
  - Verweis auf die Wurzel des Baums
- hat Knoten als Elemente:



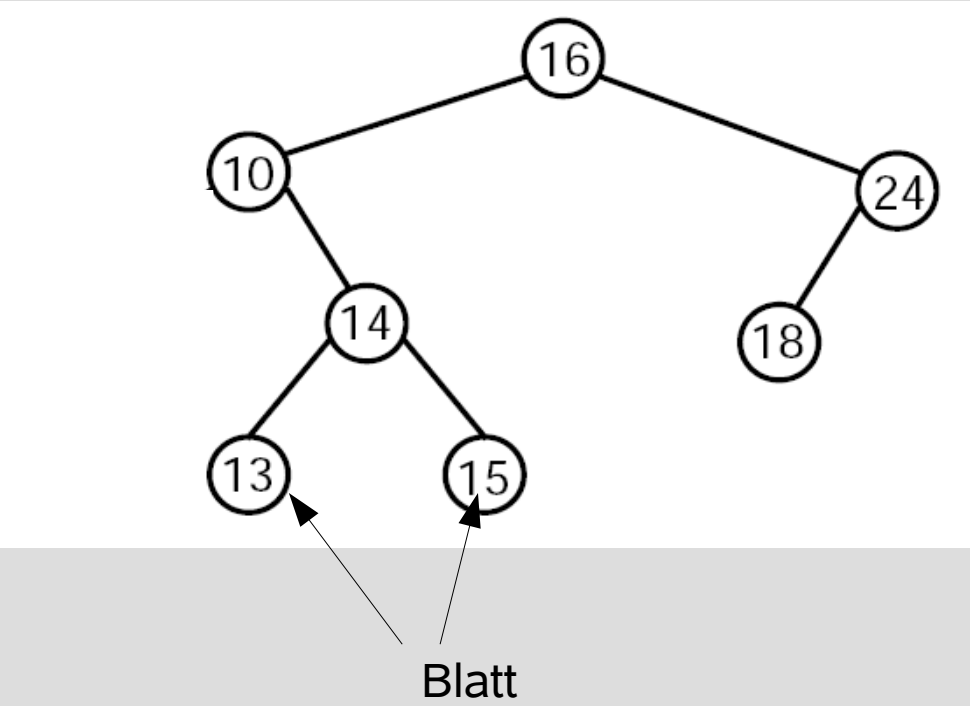
Binärer Baum



- Methoden
  - geordnet einfügen
  - löschen (gezielt im Baum absteigen)
  - ...

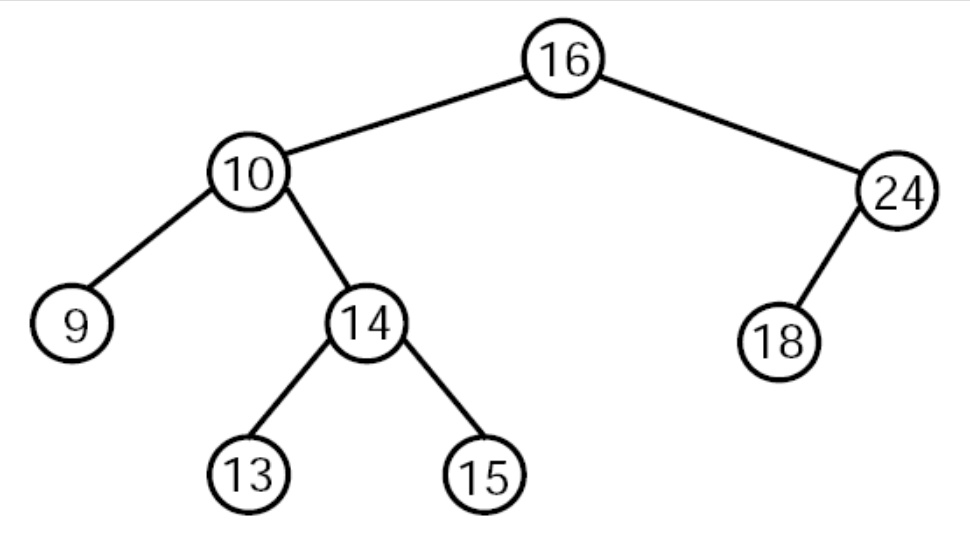
Füge 15 geordnet ein ...

Binärer Baum

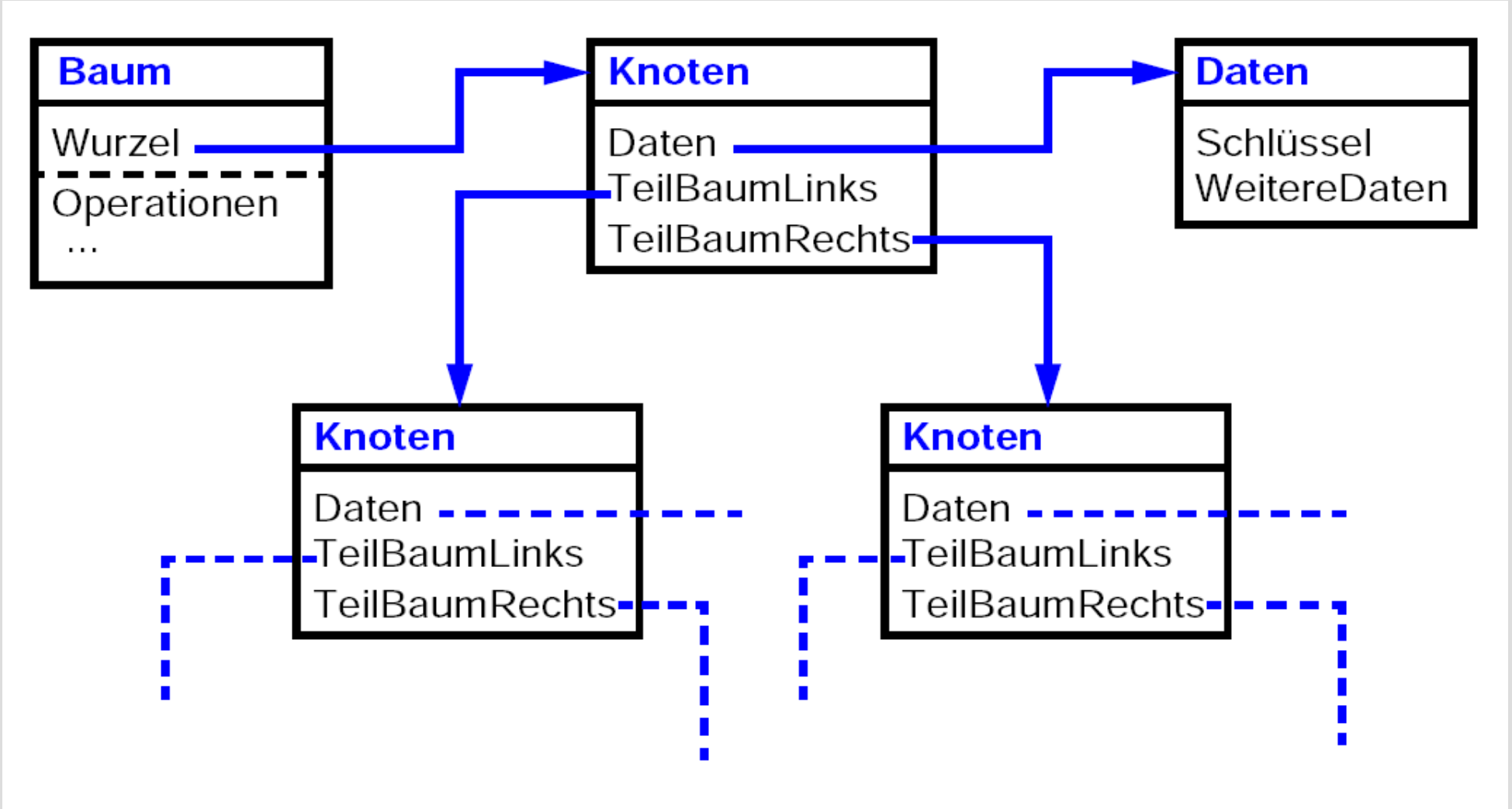


Füge 9 geordnet ein ...

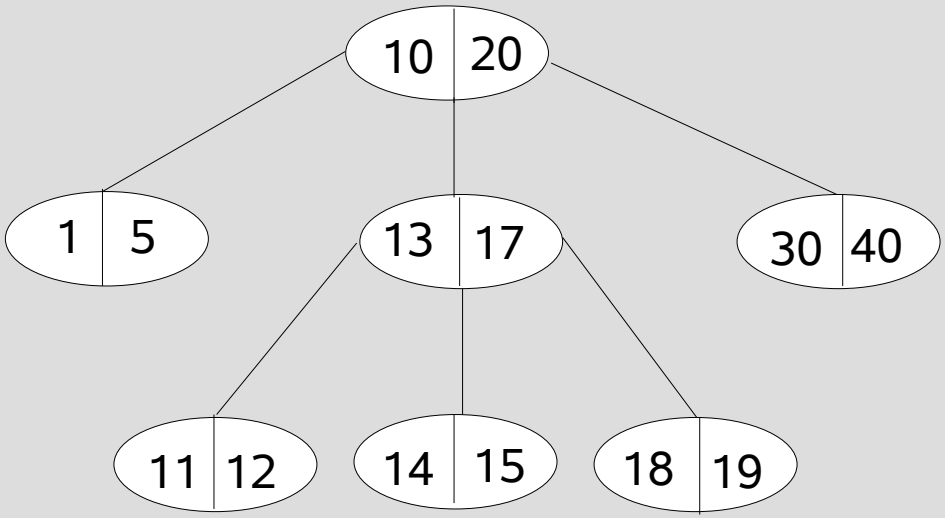
Binärer Baum



Aufbau Binärbaum



Analog: Ternärbaum



- Jeder Knoten:
  - beinhaltet zwischen 1 und 2 Zahlen (lower und upper)
  - hat 3 Nachfolger
- Methoden wie üblich

Vielwegbaum

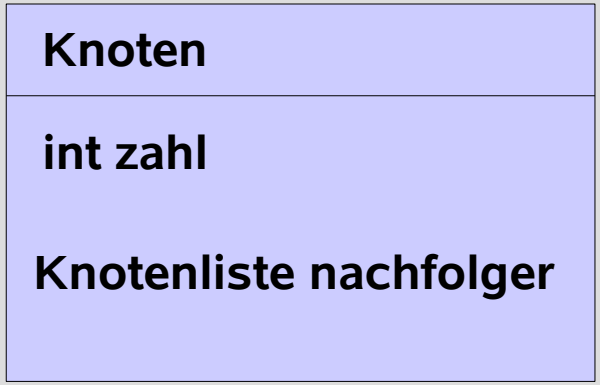
- vorher: Baum hat feste Anzahl von Nachfolgern (Binärbaum, Ternärbaum, ...)

- jetzt: ein Knoten hat beliebig viele Nachfolger

  - d.h.: Jeder Knoten hat eine Liste von Knoten als Nachfolger

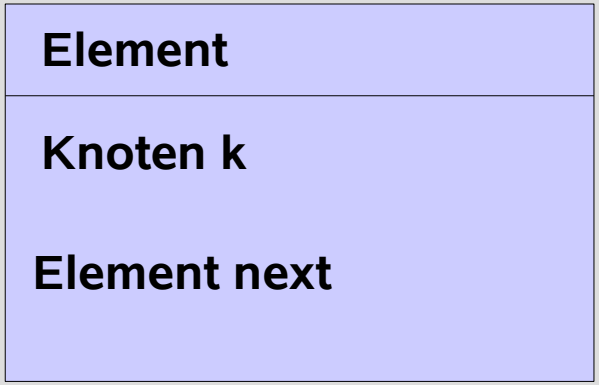
Beispiel:

- Klasse Knoten mit Attributen:
  - int zahl
  - Knotenliste nachfolger



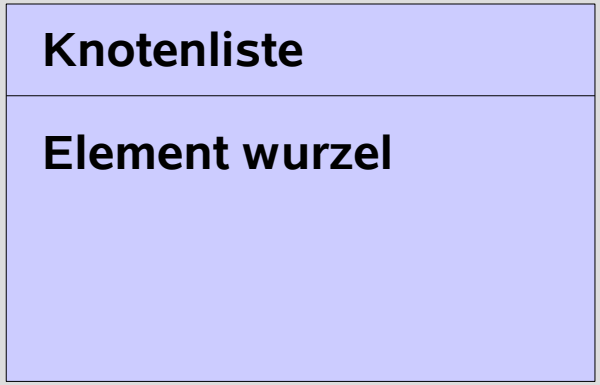
Beispiel:

- Klasse Element mit Attributen:
  - Knoten k
  - Element next

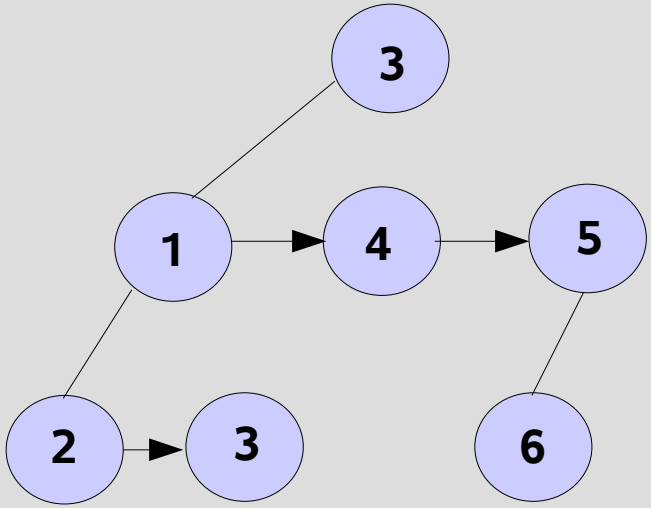


Beispiel:

- Klasse Knotenliste mit Attributen:
  - Element wurzel (Referenz auf Listenanfang)



Beispiel: Vielwegbaum



Beispiel: Vielwegbaum

- Addition aller Elemente im Baum
  - mittels verschränkter Rekursion (also Funktionen, die sich gegenseitig aufrufen)
  - dazu: 2 Methoden:
    - addiereAlle() in Klasse `Knoten`
    - addiereAlle() in Klasse `Knotenliste`

```
private int addiereAlle(Element current){
  if(current != null){
    Knoten k = current.getKnoten();
    int result = k.addiereAlle();
    return result + addiereAlle(current.getNext());
  }else{ return 0;}
}
public int addiereAlle(){
  return this.addiereAlle(this.wurzel);
}
```

Klasse Knotenliste

```
private int addiereAlle(Knoten x){
  if (x != null){
    int result = x.zahl;
    result += this.nachfolger.addiereAlle();
    return result;
  }else{ return 0; }
}

public int addiereAlle(){
  return addiereAlle(this);
}
```

Klasse Knoten