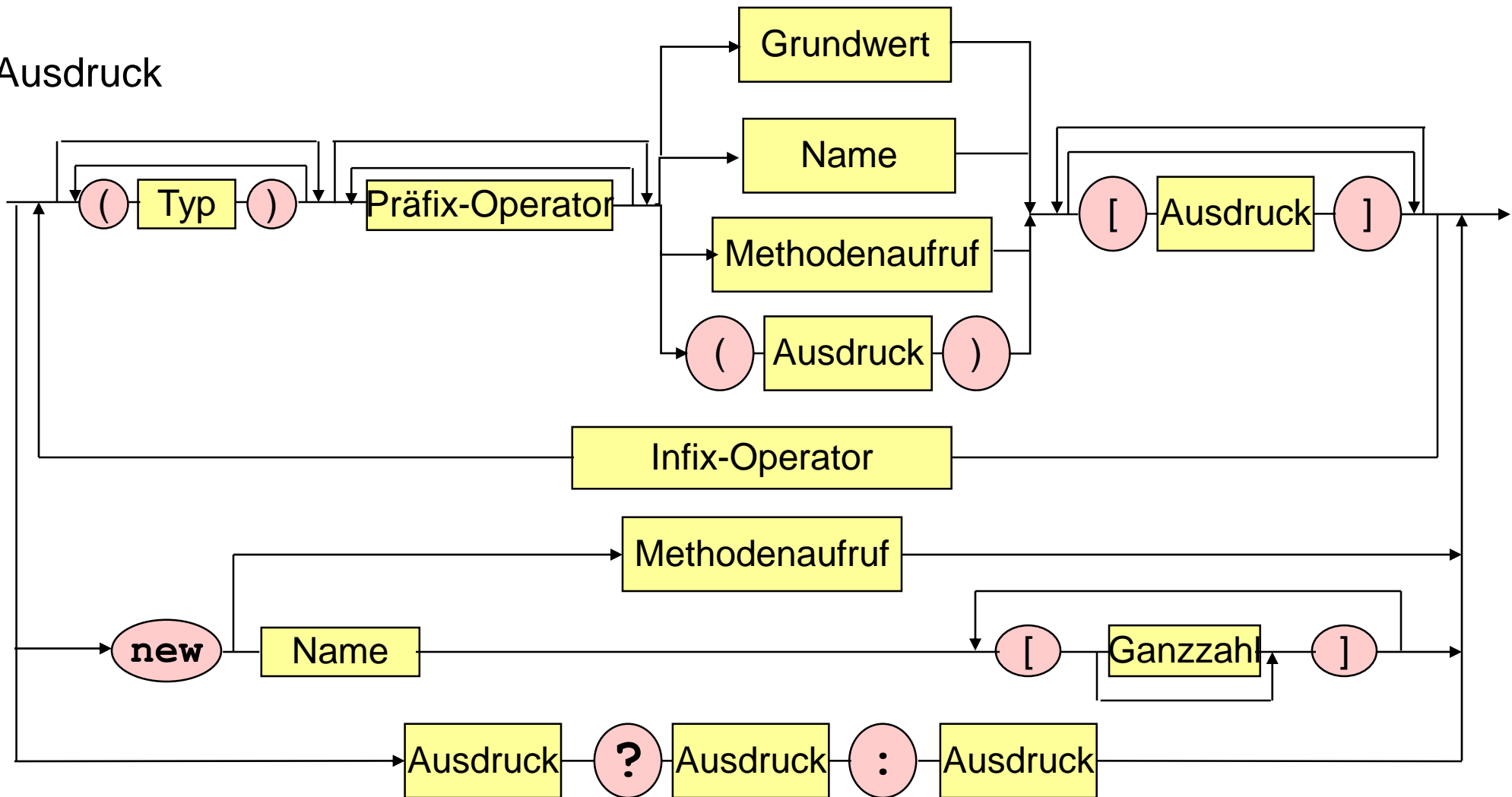

II.3. Rekursion und dynamische Datenstrukturen

- 1. Rekursive Algorithmen
- 2. Rekursive (dynamische) Datenstrukturen

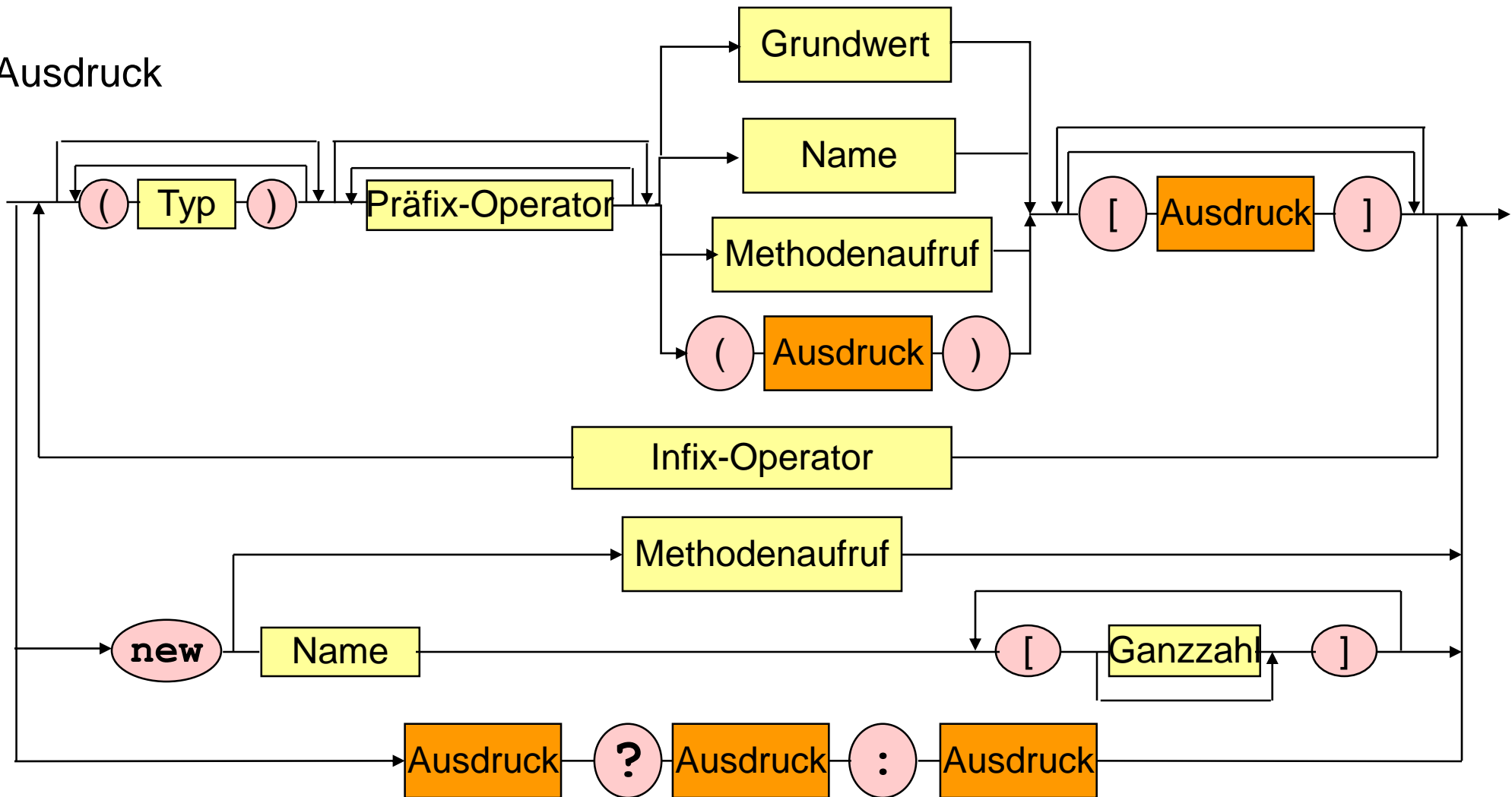
Ausdruck

Ausdruck

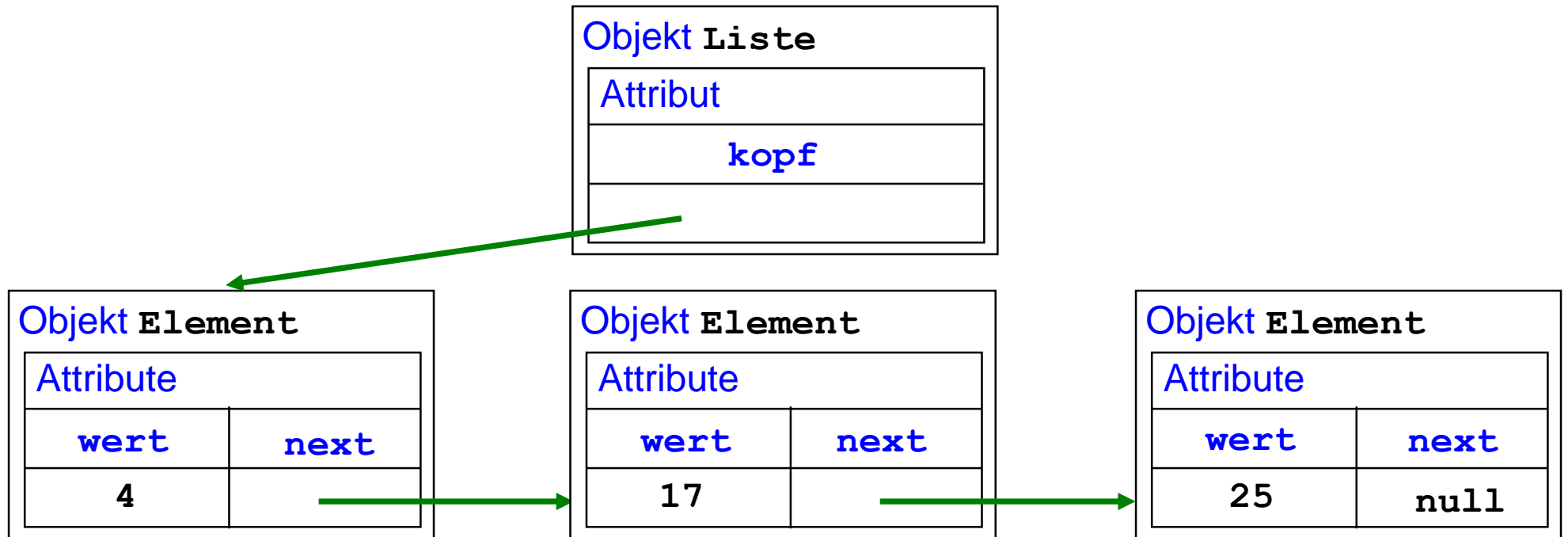


Ausdruck

Ausdruck



Realisierung von Listen



```
class Element {  
  
    int wert;  
    Element next;  
    ...  
}
```

```
public class Liste {  
  
    private Element kopf;  
    ...  
}
```

Schnittstellendokumentation

Klasse Element

konstruiert Element
- ohne Vorgänger

- Element (int wert)
- Element (int wert, Element next)
- int getWert ()
- void setWert (int wert)
- Element getNext()
- void setNext (Element next)
- String toString ()

druckt den Wert des
Elements

löscht erstes
Element mit vorgegebenem Wert

Selektoren

Klasse Liste

erzeugt neue
leere Liste liefert erstes Listenelement
mit gesuchtem Wert

- Liste ()
- Element suche (int wert)
- String toString ()
- void drucke ()
- void druckeRueckwaerts ()
- void fuegeVorneEin (int wert)
- void fuegeSortiertEin (int wert)
- void loesche (int wert)
- void loesche ()

- string mit allen
Listenelementen

gibt Listenelemente
- in anderer Reihenfolge aus

↑ fügt wert
vor dem

ersten größeren
Listenelement
ein

Verwendung von Listen

```
Liste l = new Liste ();  
l.fuegeVorneEin (30); l.fuegeVorneEin (25);  
l.fuegeVorneEin (17); l.fuegeVorneEin (4);  
l.drucke (); l.druckeRueckwaerts ();  
  
l.fuegeSortiertEin (28); l.fuegeSortiertEin (12);  
l.fuegeSortiertEin (45); l.fuegeSortiertEin (2); l.drucke ();  
if (l.suche (17) != null) System.out.println (l.suche(17));  
  
l.loesche (28); l.loesche (10); l.loesche (17); l.drucke ();  
l.loesche (); l.drucke ();
```

```
( 4 17 25 30 )  
( 30 25 17 4 )  
( 2 4 12 17 25 28 30 45 )  
17  
( 2 4 12 25 30 45 )  
( )
```

Element-Klasse

```
class Element {
    int wert;
    Element next;

    Element (int wert) { this.wert = wert; next = null; }

    Element (int wert, Element next) {
        this.wert = wert; this.next = next; }

    int getWert () { return wert; }
    void setWert (int wert) { this.wert = wert; }

    Element getNext () { return next; }
    void setNext (Element next) { this.next = next; }

    public String toString () {
        return new Integer(wert).toString(); }
}
```

Liste-Klasse: Erzeugung und Suche

```
public class Liste {  
  
    private Element kopf;
```

```
    public Liste () {  
        kopf = null;  
    }
```

```
    public Element suche (int wert) {  
        return suche (wert, kopf);  
    }
```

```
    private static Element suche (int wert, Element kopf) {  
        if (kopf == null) return null;  
        else if (kopf.wert == wert) return kopf;  
        else return suche (wert, kopf.next);  
    }
```

Statische Hilfsmethode

suche (x, e)

sucht nach dem ersten
Vorkommen von x in
der Liste, die mit
Element e beginnt

Liste-Klasse: Ausgabe

```
public String toString () {  
    return "( " + durchlaufe(kopf) + ")"; }  
}
```

```
private static String durchlaufe (Element kopf) {  
    if (kopf != null)  
        return kopf.wert + " " + durchlaufe(kopf.next);  
    else return "";  
}
```

```
public void drucke() { System.out.println (this); }
```

```
public String toStringRueckwaerts () {  
    return "(" + durchlaufeRueckwaerts(kopf) + " )"; }  
}
```

```
private static String durchlaufeRueckwaerts (Element kopf) {  
    if (kopf != null)  
        return durchlaufeRueckwaerts(kopf.next) + " " + kopf.wert;  
    else return "";  
}
```

```
public void druckeRueckwaerts () {  
    System.out.println (this.toStringRueckwaerts ()); }  
}
```

Liste-Klasse: Einfügen

```
public void fuegeVorneEin (int wert) {  
  
    if (kopf == null)        kopf = new Element (wert);  
    else                     kopf = new Element (wert, kopf);  
  
}
```

Liste-Klasse: Einfügen

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

```
private static Element fuegeSortiertEin (int wert, Element e) {  
    if (e == null)  
        return new Element (wert);  
  
    else if (wert < e.wert)  
        return new Element (wert, e);  
  
    else  
        {  
            e.next = fuegeSortiertEin (wert, e.next);  
            return e;  
        }  
}
```

```
public void fuegeSortiertEin (int wert) {  
    Element element = kopf;  
    if (kopf == null || wert < kopf.wert) fuegeVorneEin(wert);  
    else {while (element.next != null && wert >= element.next.wert)  
        element = element.next;  
        element.next = new Element (wert, element.next); }  
}
```

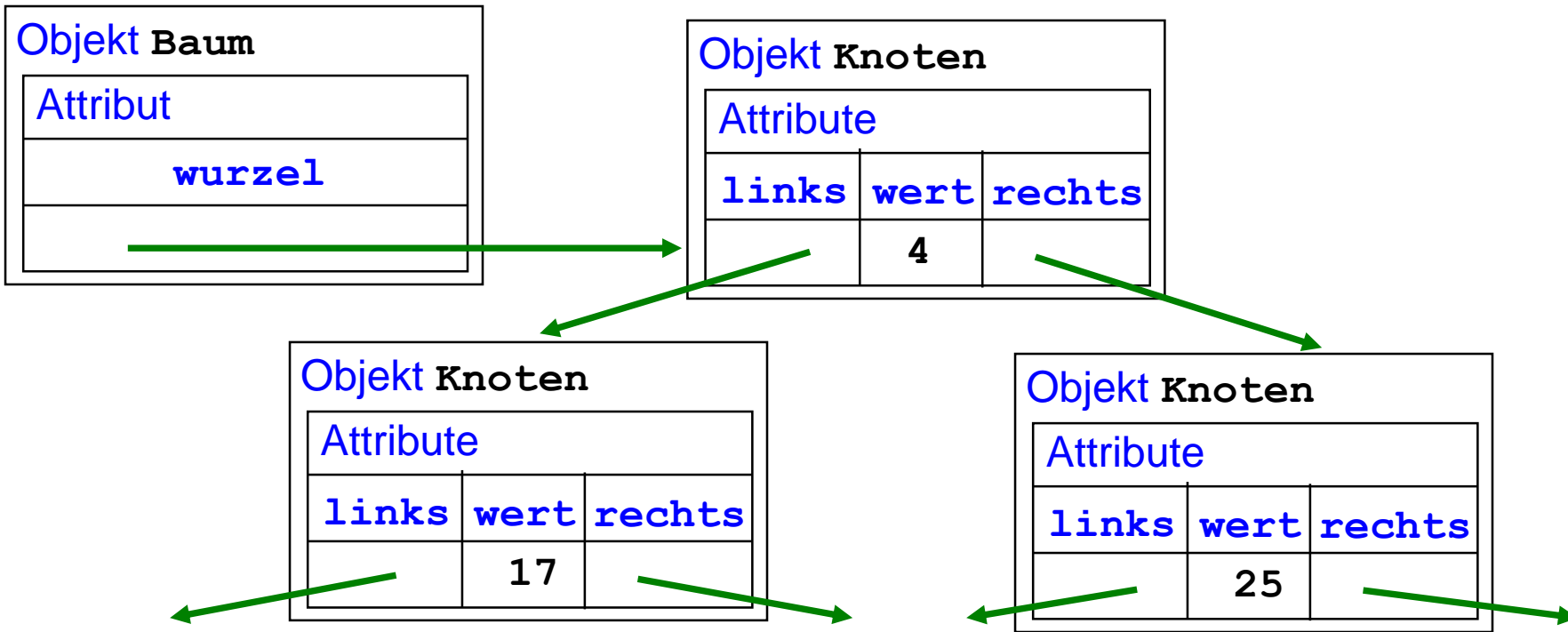
Liste-Klasse: Löschen

```
public void loesche () {  
    kopf = null;  
}
```

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

```
private static Element loesche (int wert, Element element) {  
  
    if      (element == null)      return null;  
    else if (wert == element.wert) return element.next;  
    else                               {  
        element.next = loesche (wert, element.next);  
        return element;  
    }  
  
}}
```

Realisierung von binären Bäumen



```
class Knoten {  
    int wert;  
    Knoten links, rechts;  
    ...  
}
```

```
public class Baum {  
    private Knoten wurzel;  
    ...  
}
```