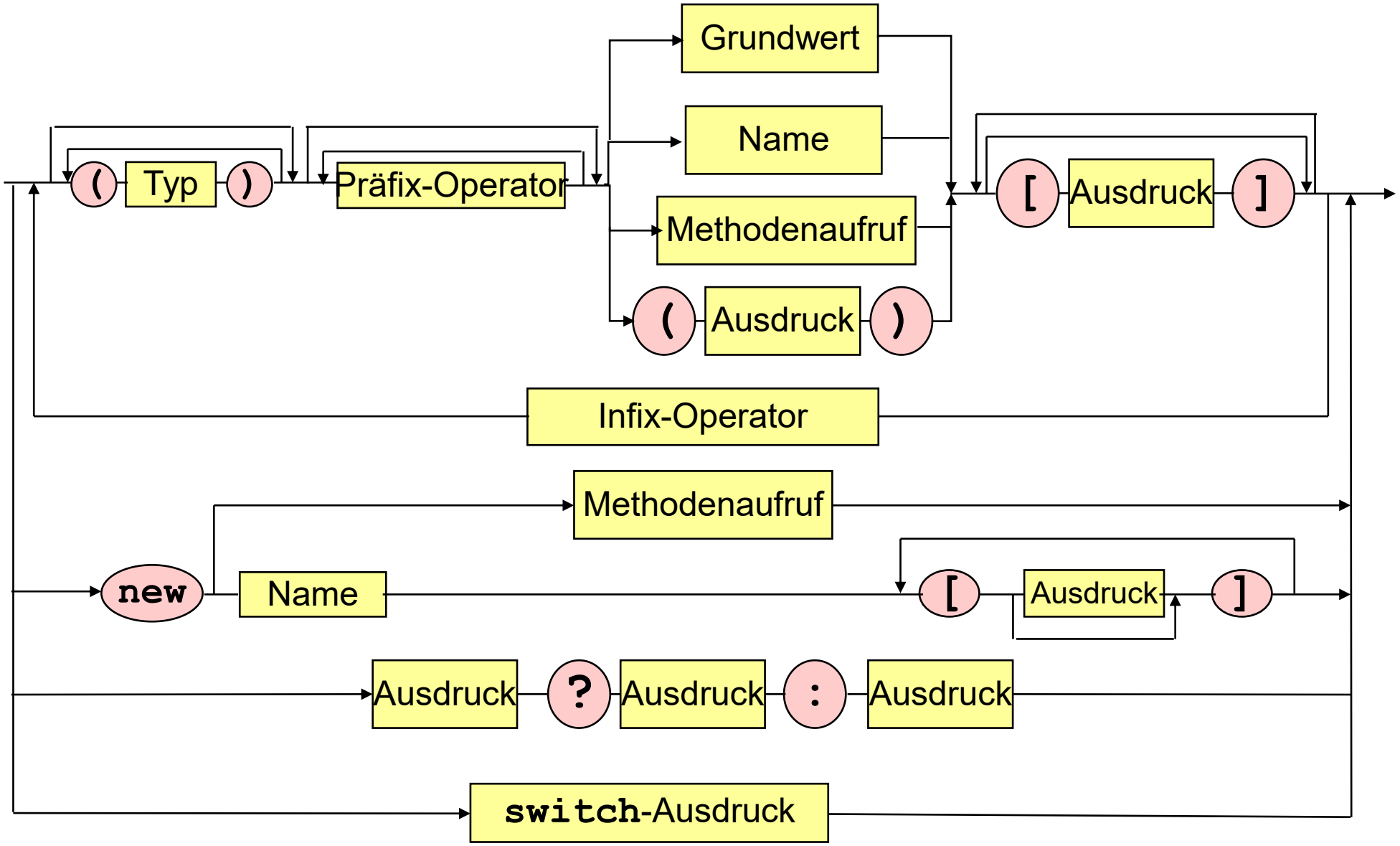
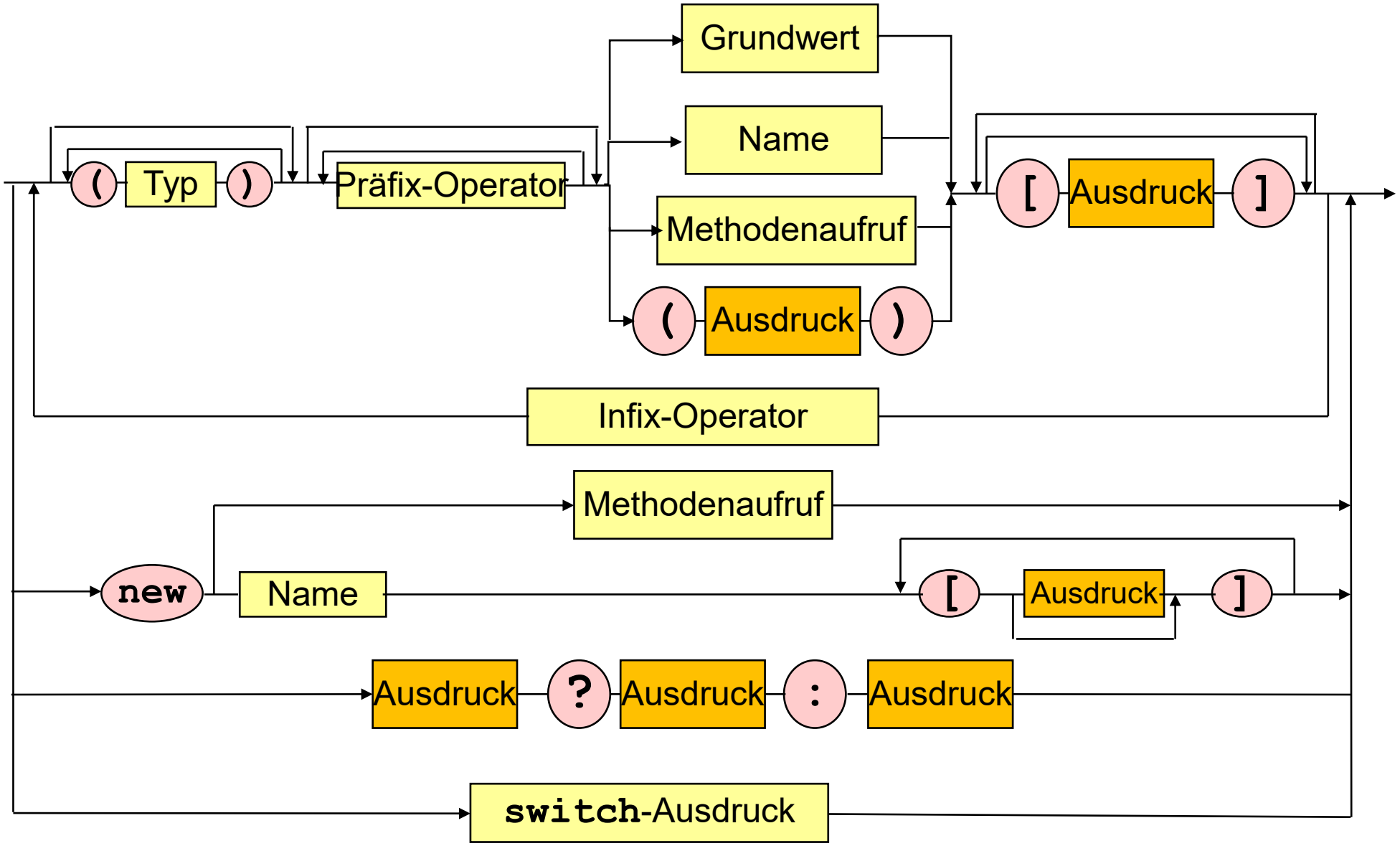
II.3. Rekursion und dynamische Datenstrukturen

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

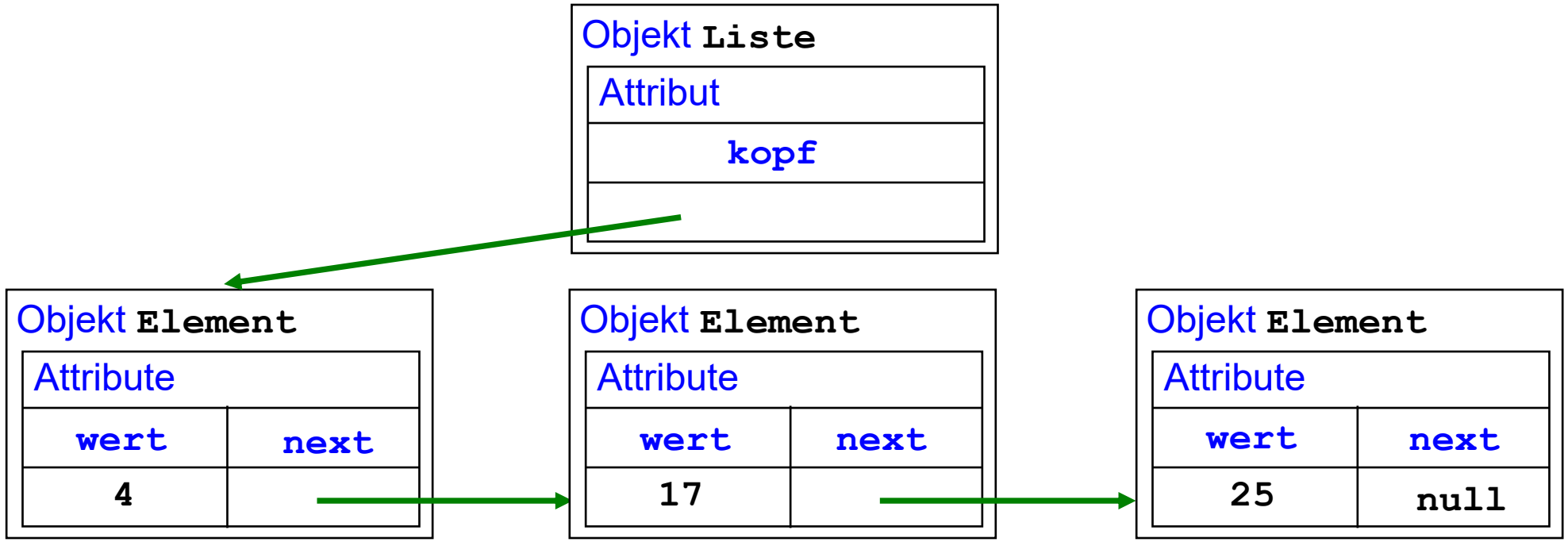
Ausdruck


Grundwert
Name
Methodenaufruf
(Ausdruck)
(Typ)
Präfix-Operator
[Ausdruck]
Infix-Operator
Methodenaufruf
new
Name
[Ausdruck]
Ausdruck ? Ausdruck : Ausdruck
switch-Ausdruck

Ausdruck


Grundwert
Name
Methodenaufruf
(Ausdruck)
(Typ)
Präfix-Operator
[Ausdruck]
Infix-Operator
Methodenaufruf
new
Name
[Ausdruck]
Ausdruck ? Ausdruck : Ausdruck
switch-Ausdruck

Realisierung von Listen




```
class Element {

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

```
public class Liste {

 private Element kopf;

 ...
}
```

Schnittstellendokumentation

Klasse `Element`

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

Klasse `Liste`

- Liste ()
- boolean enthaelt (int wert)
- String toString ()
- void drucke ()
- void druckeRueckwaerts ()
- void fuegeVorneEin (int wert)
- void fuegeSortiertEin (int wert)
- void loesche (int wert)
- void loesche ()

Verwendung von Listen

```
Liste xs = new Liste ();

xs.fuegeVorneEin (30); xs.fuegeVorneEin  (25);
xs.fuegeVorneEin (17); xs.fuegeVorneEin  (4);
xs.drucke (); xs.druckeRueckwaerts ();

xs.fuegeSortiertEin (28); xs.fuegeSortiertEin (12);
xs.fuegeSortiertEin (45); xs.fuegeSortiertEin (2); xs.drucke ();

if   (xs.enthaelt (17)) IO.println ("17 ist in Liste");

xs.loesche (28); xs.loesche (10); xs.loesche (17); xs.drucke ();
xs.loesche (); xs.drucke ();
```

```
( 4 17 25 30 )
( 30 25 17 4 )
( 2 4 12 17 25 28 30 45 )
17 ist in Liste
( 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 Integer.toString(wert);  }
}
```

Liste-Klasse: Erzeugung und Suche

```
public class Liste {

 private Element kopf;


 public Liste () {
  kopf = null;
 }

 public boolean enthaelt (int wert) {
  return enthaelt (wert, kopf);
 }

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

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() { IO.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()                          {
          IO.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

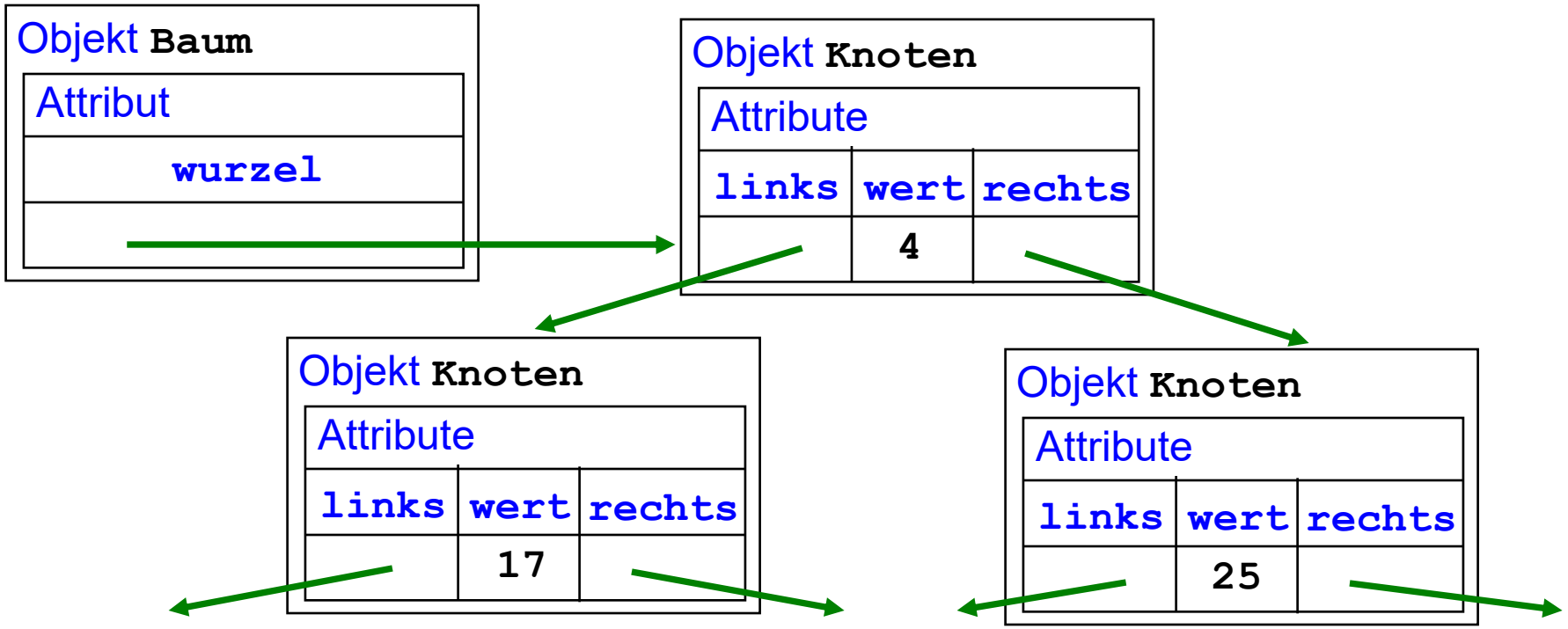
```
public void fuegeSortiertEin (int wert) {
 kopf = fuegeSortiertEin (wert, kopf);  }

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;
 ...
}
```