1. Die rekursive Datenstruktur Liste

1.4 Methoden der Datenstruktur Liste

Die **Warteschlange (Queue)** ist ein Sonderfall der Datenstruktur **Liste**.

Bei der Warteschlange werden Einfügen und Entfernen nach dem Prinzip **FIFO (First In, First Out)** umgesetzt. Dafür gibt es die Methoden HintenEinfuegen und AnfangEntfernen.

Für **Liste** gibt es weitere Methoden, wie zum Beispiel VorneEinfuegen, EndeEntfernen, EinfuegenVor, SortiertEinfuegen, usw.
Viele dieser Methoden kann man rekursiv implementieren.

Beispiel (Länge der Liste bestimmen)

In der Klasse Liste:

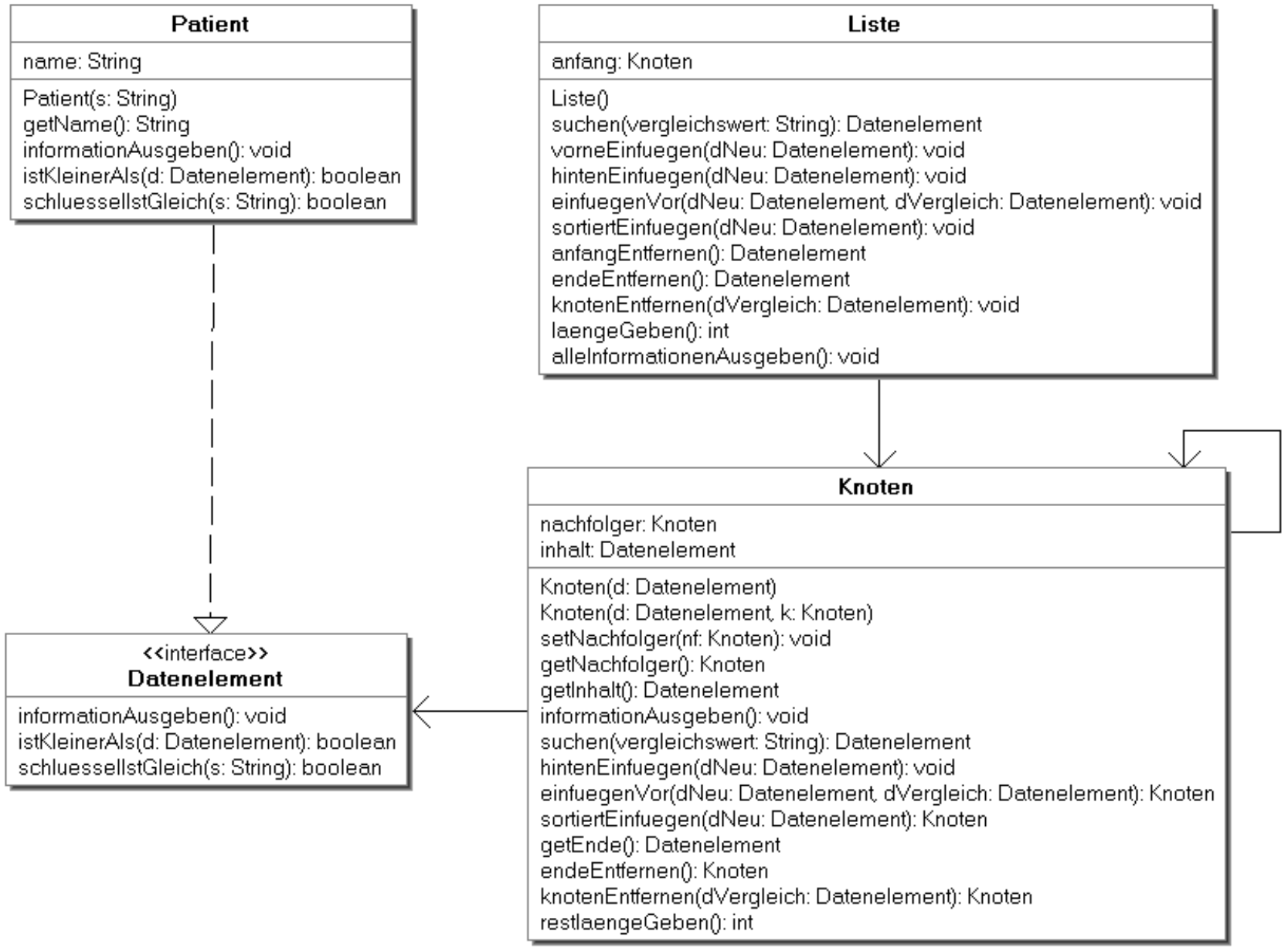
```
public int laengeGeben(){
  if (anfang==null){
    return 0;
  }
  else{
    return anfang.restlaengeGeben();
  }
}
```

In der Klasse Knoten:

```
public int restlaengeGeben(){
  if (nachfolger==null){
    return 1;
  }
  else{
    return nachfolger.restlaengeGeben()+1;
  }
}
```

Modellierung:

Das Ende der Liste erfasst man, indem man durch die Liste läuft und prüft, ob der Nachfolger null ist. Das Ende als eigenes Attribut zu verwalten ist also nicht zwingend nötig.



Übung

Im Buch sind auf den Seiten 22 - 32 einige Methoden von Liste ausführlich beschrieben.

Arbeite sie der Reihe nach sorgfältig durch und implementiere sie.

Als Gerüst kannst du das BlueJ-Projekt liste_2 verwenden.

Bei einigen Methoden benötigt man beim rekursiven Durchlaufen der Liste den Zugriff auf den *Vorgänger* des aktuellen Knotens.

Man könnte dazu ein eigenes Attribut verwalten. Dies erfordert aber eine deutliche Änderung der Modellierung.

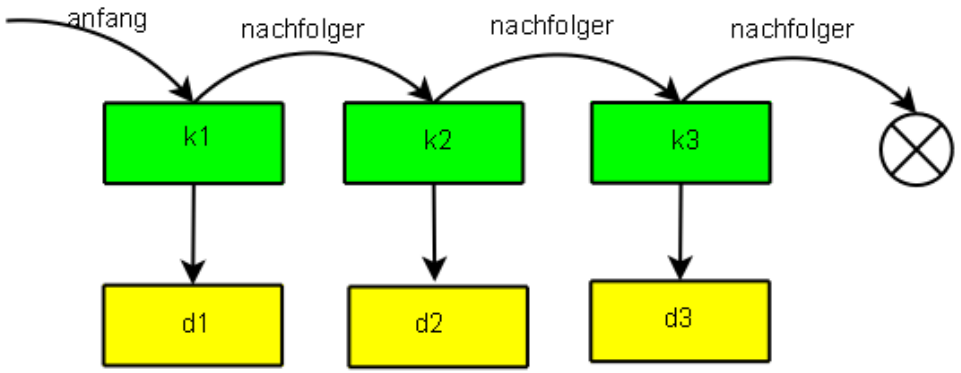
Ein kleiner Trick bei der Implementierung vermiedet dies.

Am Beispiel der Methode

Knoten einfuegenVor (Datenelement dNeu, Datenelement dVergleich)

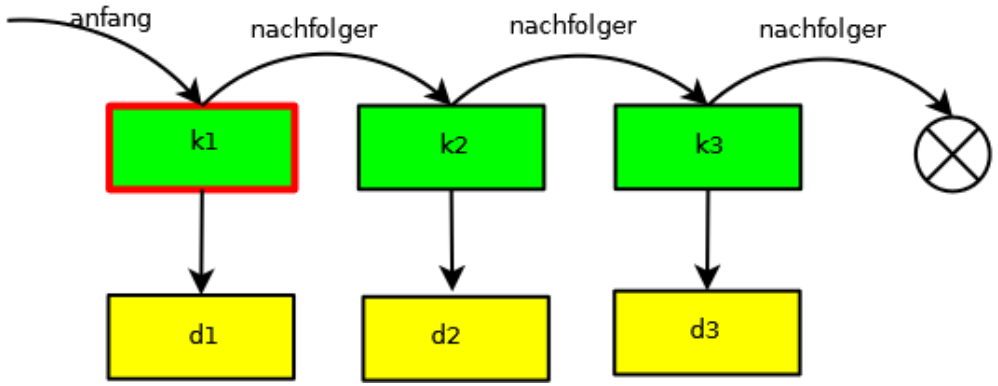
wird dies auf den folgenden Seiten verdeutlicht:

Die Methode public Knoten einfuegenVor (Datenelement dNeu, Datenelement dVergleich)




```
public Knoten einfuegenVor(Datenelement dNeu, Datenelement dVergleich){
    if(inhalt != dVergleich){
       if(nachfolger !=null){
          nachfolger=nachfolger.einfuegenVor(dNeu, dVergleich);
        }
        else  {  hintenEinfuegen(dNeu);   }
        return this;
    }
    else{
       Knoten kNeu;
       kNeu = new Knoten(dNeu, this);
       return kNeu;
     }
}
```

Die Methode public Knoten einfuegenVor (Datenelement dNeu, Datenelement dVergleich)

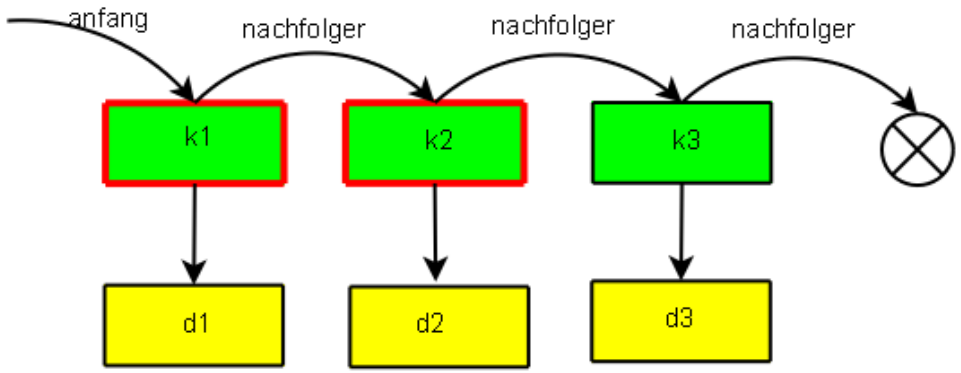


dNeu soll vor d3 eingefügt werden.

In k1 erfolgt der rekursive Aufruf
nachfolger = nachfolger.einfuegenVor(dNeu, dVergleich)

(Wird noch nicht ausgewertet, da das Rekursionsende noch nicht erreicht ist.)

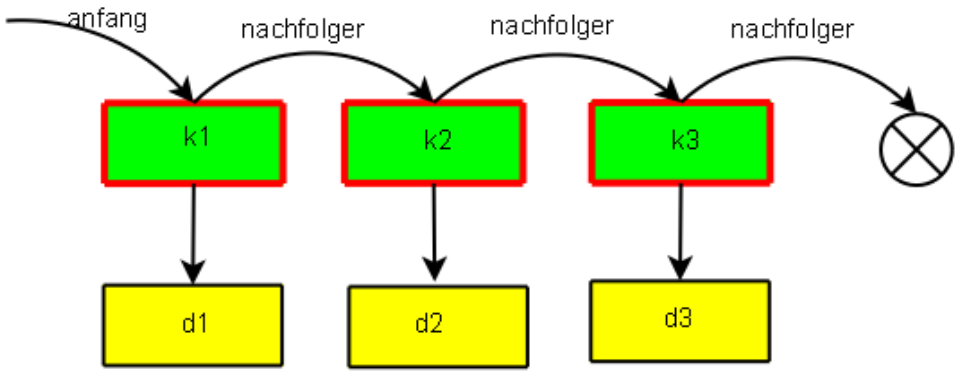
Die Methode public Knoten einfuegenVor (Datenelement dNeu, Datenelement dVergleich)



In k2 erfolgt der rekursive Aufruf
nachfolger = nachfolger.einfuegenVor(dNeu, dVergleich)

(Wird noch nicht ausgewertet, da das Rekursionsende noch nicht erreicht ist.)

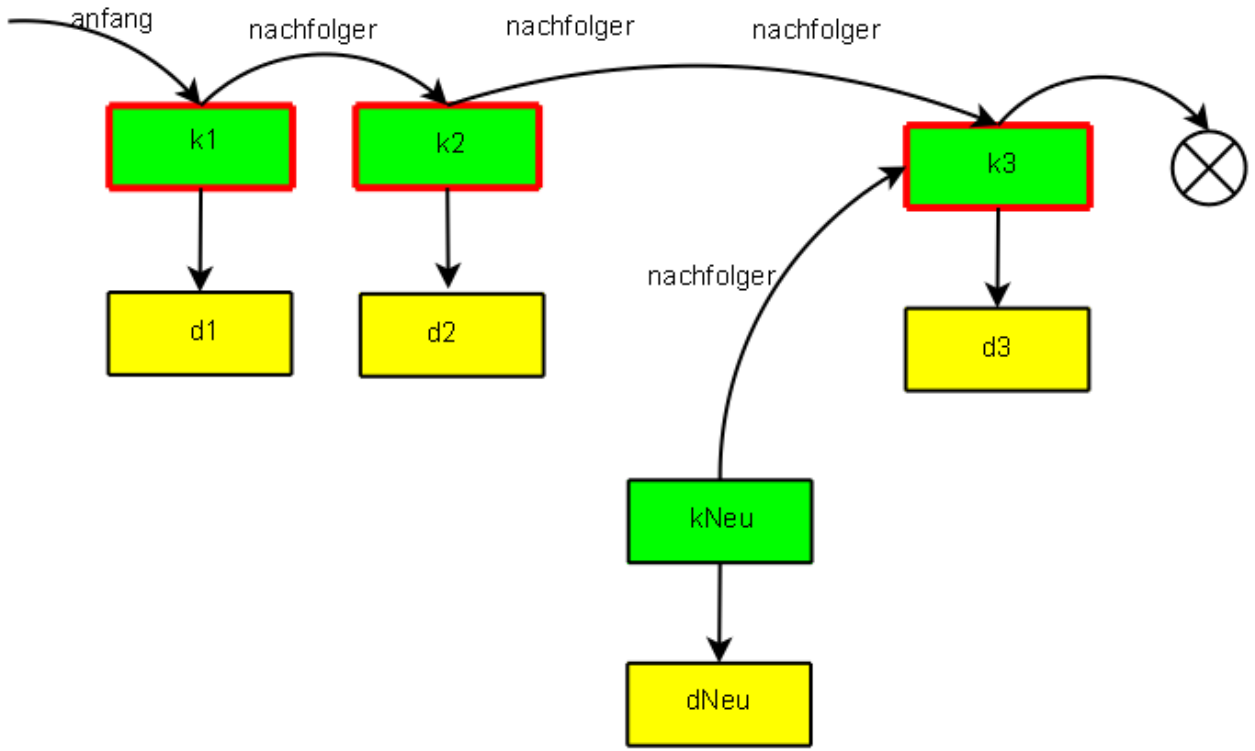
Die Methode public Knoten einfuegenVor (Datenelement dNeu, Datenelement dVergleich)



d3 ist das Datenelement, vor das dNeu eingefügt werden soll.

Der Aufruf
nachfolger = nachfolger.einfuegenVor(dNeu, dVergleich)
wird übersprungen, da inhalt != dVergleich falsch ist.

Die Methode public Knoten einfuegenVor (Datenelement dNeu, Datenelement dVergleich)



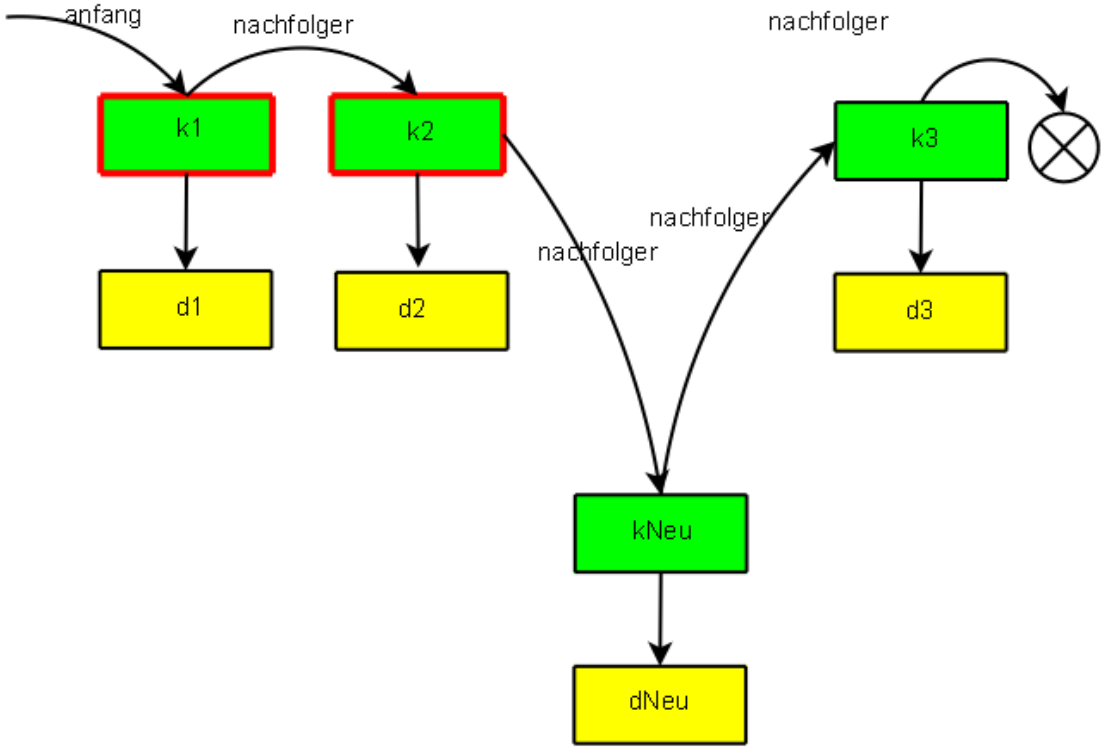
Knoten kNeu = new Knoten(dNeu, this)

erzeugt den neuen Knoten mit k3 (this) als Nachfolger.

return kNeu;

Das Rekursionsende ist erreicht.

Die Methode public Knoten einfuegenVor (Datenelement dNeu, Datenelement dVergleich)

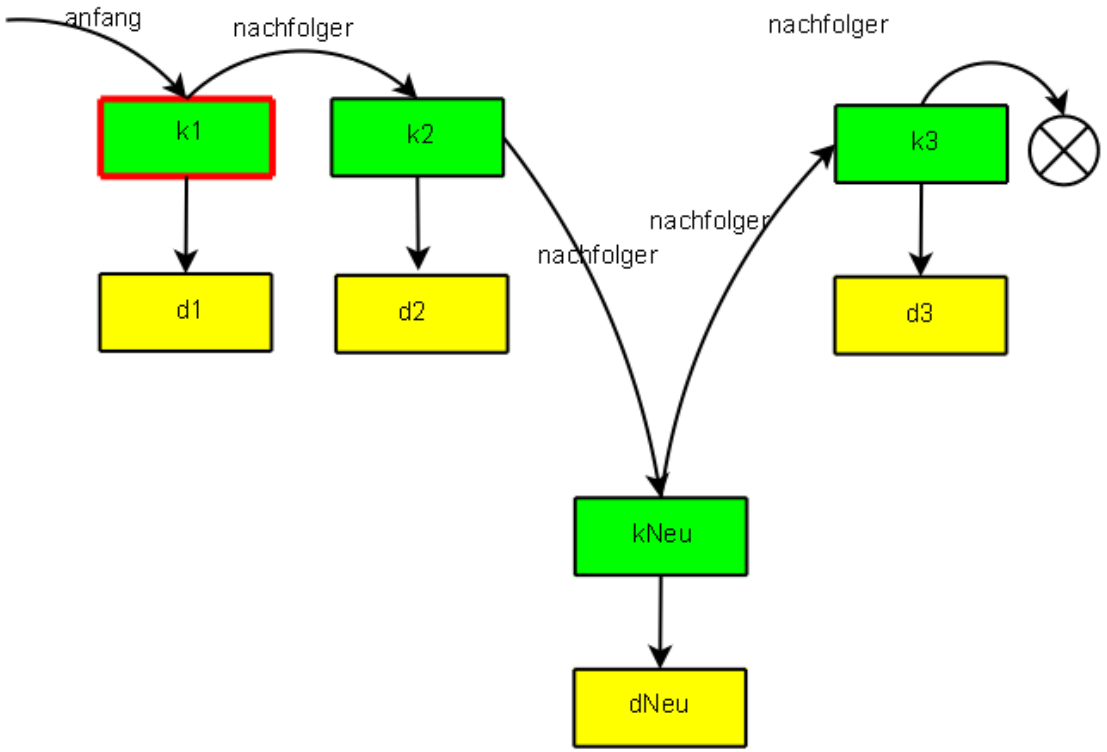


Darauf hat k2 gewartet!

nachfolger = kNeu;

return this; (also k2)

Die Methode public Knoten einfuegenVor (Datenelement dNeu, Datenelement dVergleich)

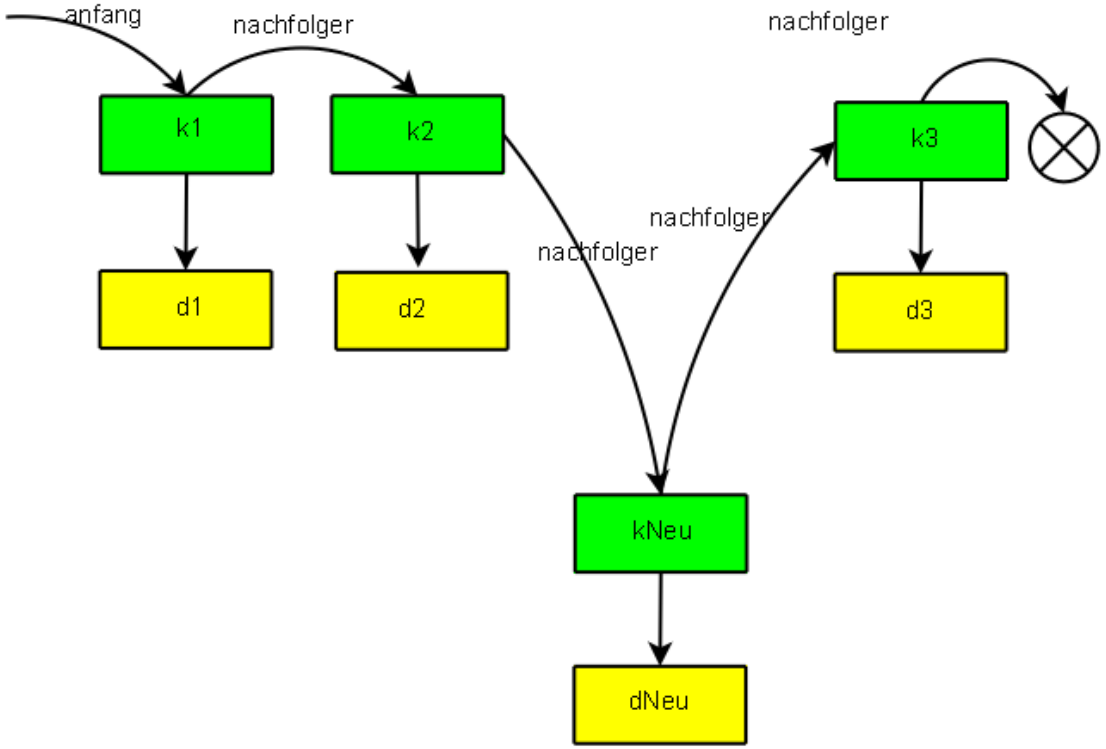


Darauf hat k1 gewartet!

nachfolger = k2;

return this; (also k1)

Die Methode public Knoten einfuegenVor (Datenelement dNeu, Datenelement dVergleich)



In Liste muss die Methode so aufgerufen werden:
anfang=anfang.einfuegenVor(dNeu,dVergleich);

anfang.einfuegenVor(dNeu,dVergleich) liefert k1 zurück,
also anfang = k1;

Dieses Verfahren kommt auch bei den folgenden Methoden zum Einsatz:

Knoten sortiertEinfuegen(Datenelement dNeu)

Knoten endeEntfernen()

Knoten knotenEntfernen(Datenelement dVergleich)