



Zentralübung zu  
Einführung in die Informatik I

Dr. Christian Herzog  
Technische Universität München

Wintersemester 2000/2001  
29. Januar 2001

# Übersicht

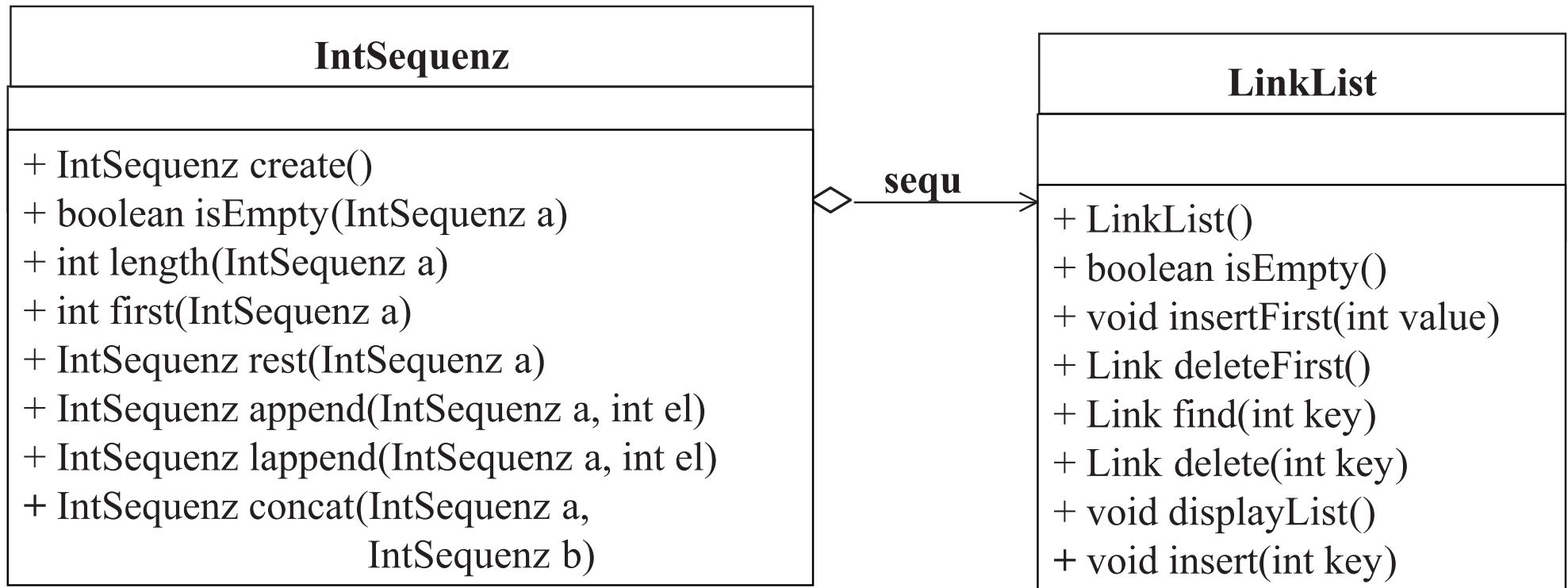
Programmieren mit Java:

- ❖ Realisierung von IntSequenz mit linearen Listen
- ❖ Wo bleibt das funktionale Prinzip?

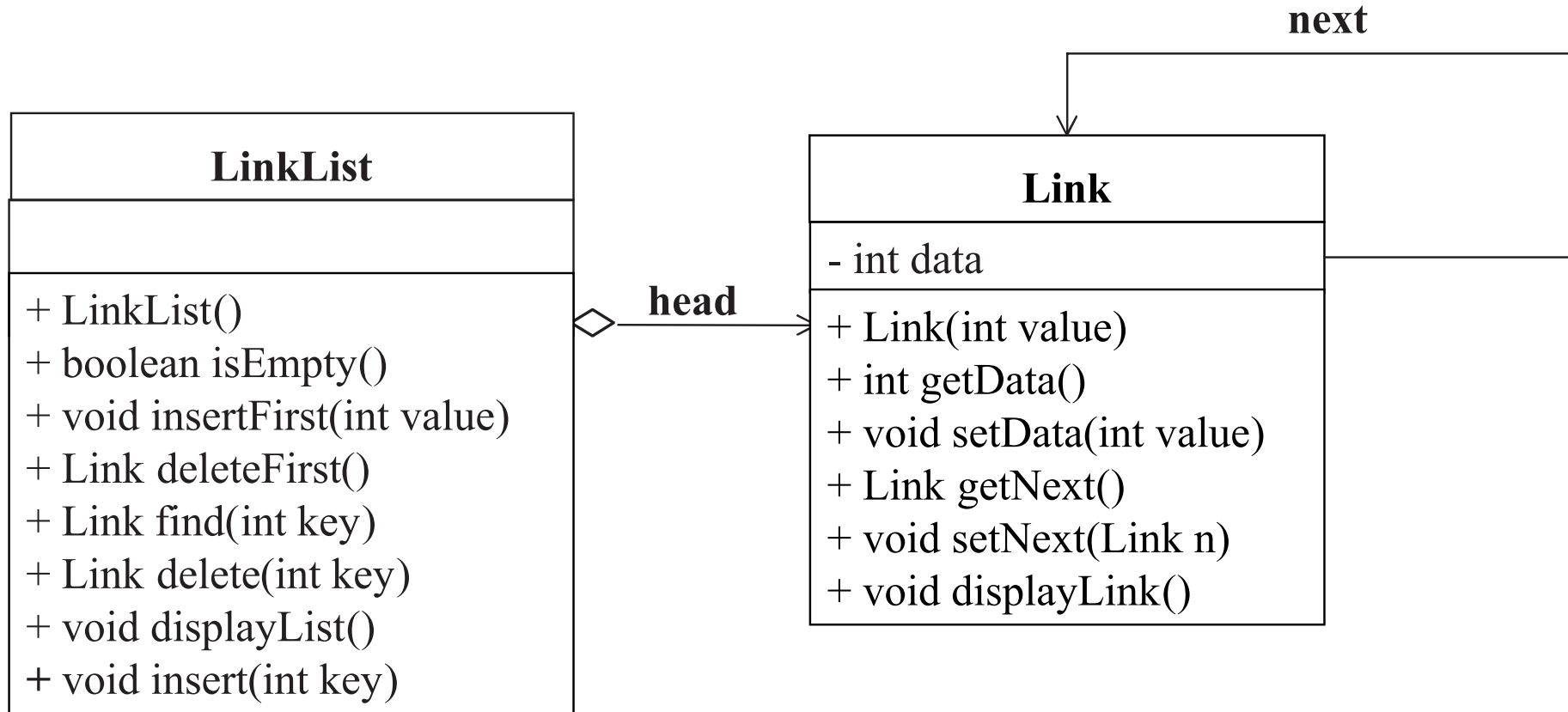
# Realisierung von IntSequenz

- ❖ Geforderte Schnittstelle (z.B. aus Aufgabe 43)
  - IntSequenz create()
  - boolean isEmpty(IntSequenz a)
  - int length(IntSequenz a)
  - int first(IntSequenz a)
  - IntSequenz rest(IntSequenz a)
  - IntSequenz append(IntSequenz a, int el)
  - IntSequenz lappend(IntSequenz a, int el)
  - IntSequenz concat(IntSequenz a, IntSequenz b)

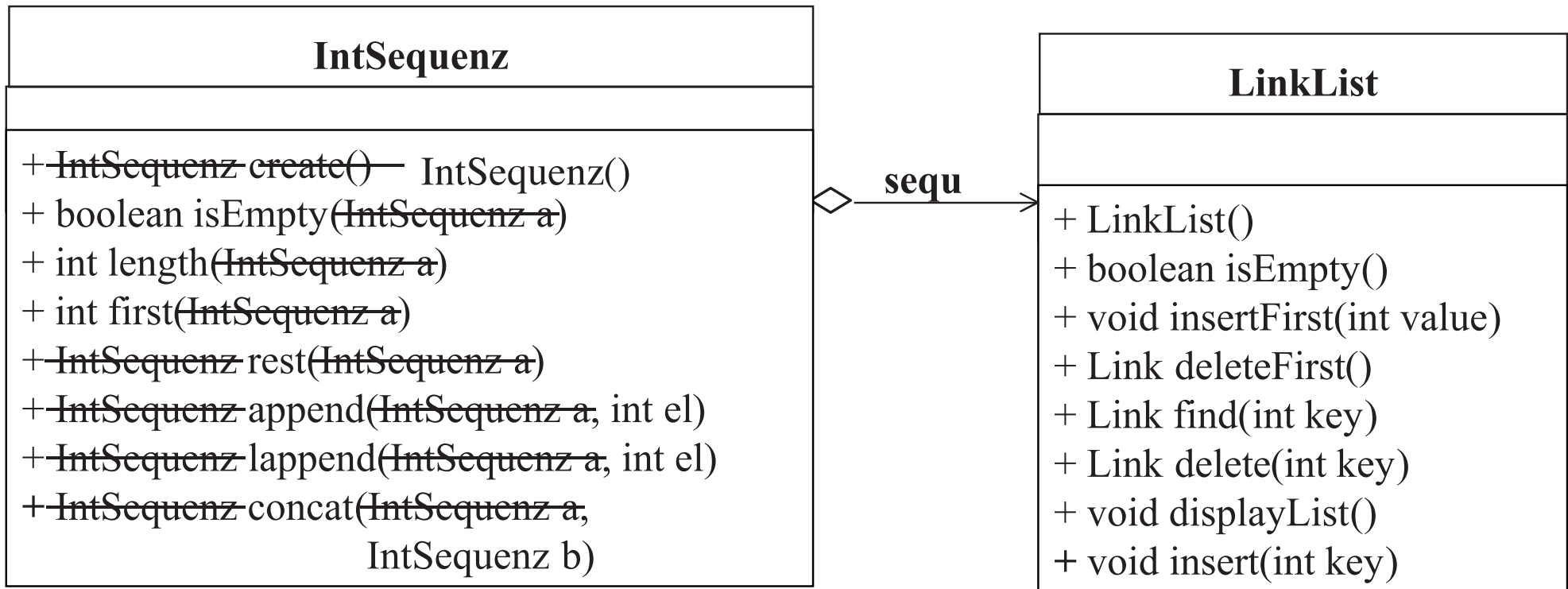
# Idee: Abstützen auf LinkList



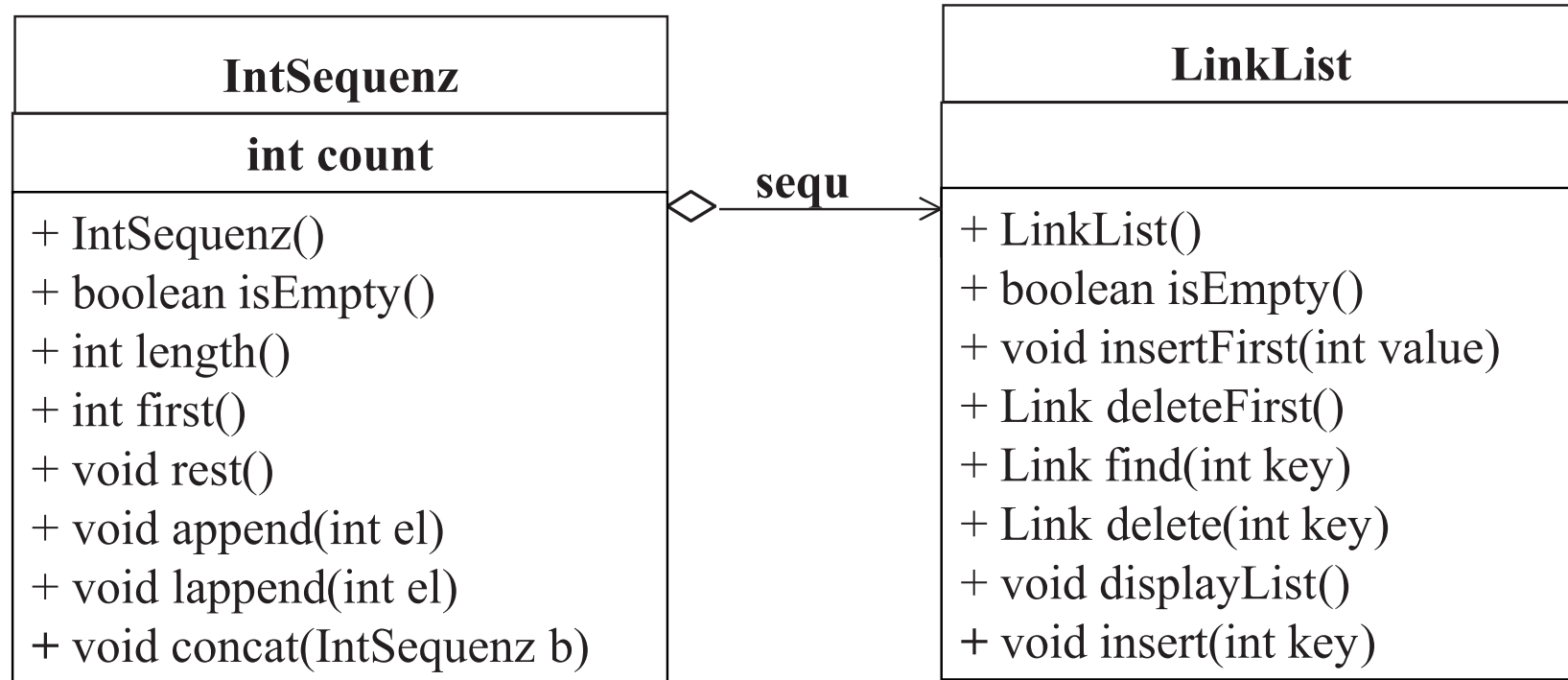
# Erinnerung: LinkList aus der Vorlesung ist abgestützt auf Link



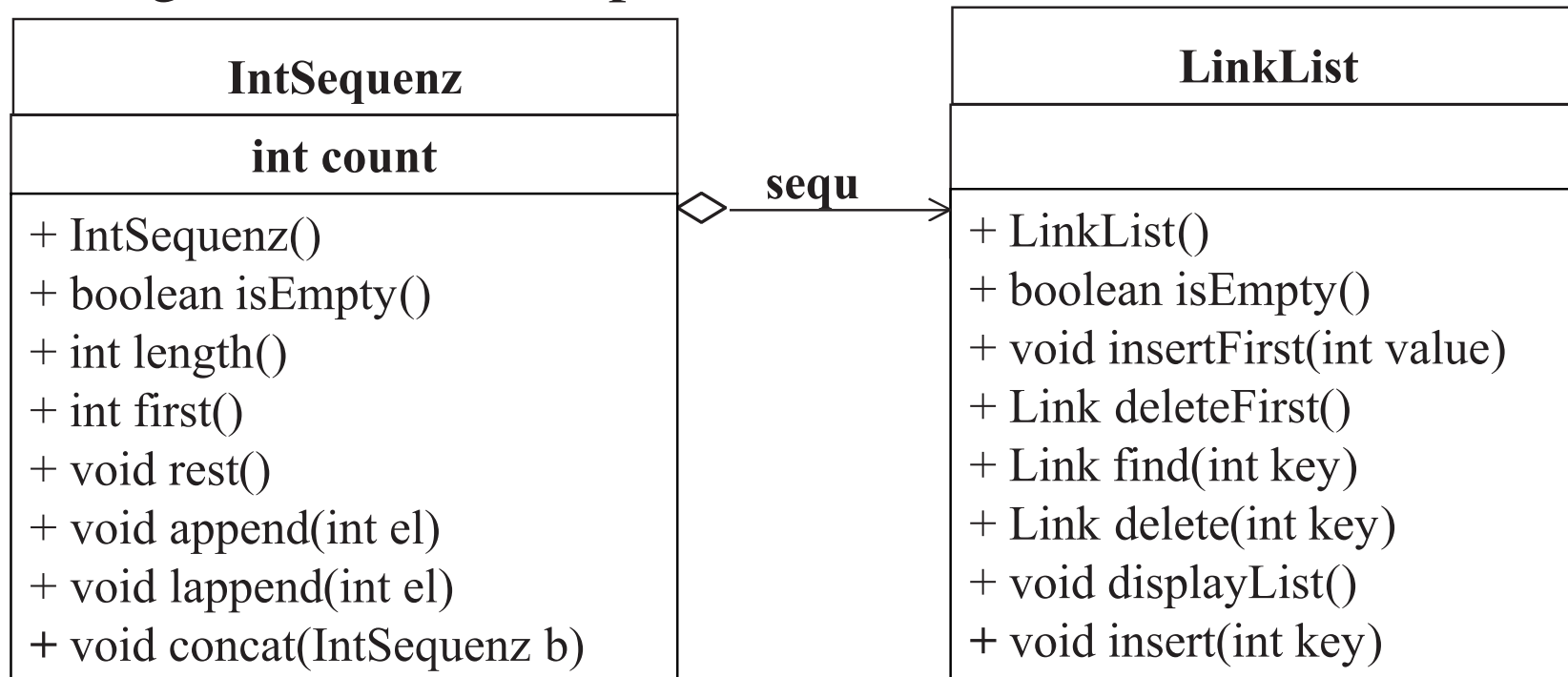
# Zunächst: Vernachlässigen der ungewöhnlichen Signatur



# Zusätzliches Attribut für die Länge



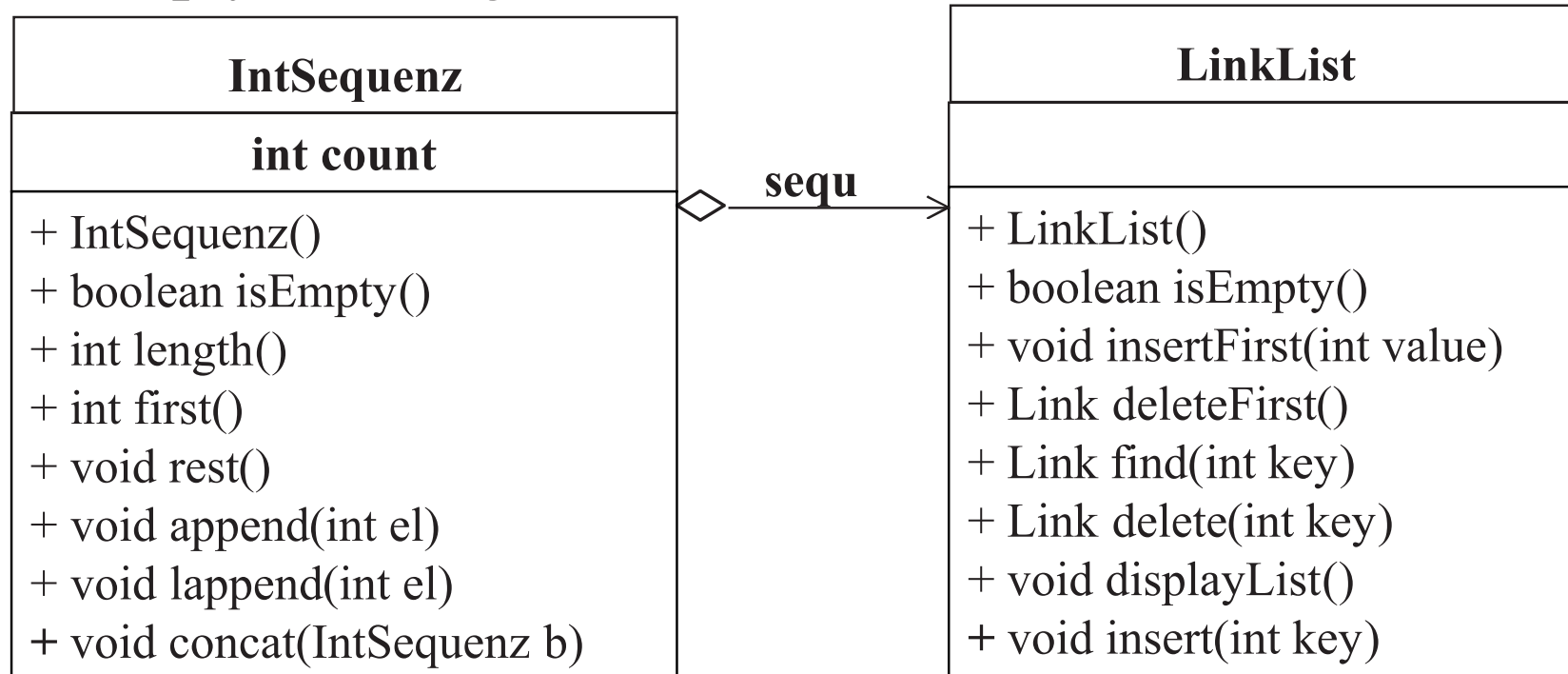
# Fangen wir an zu implementieren



```
class IntSequenz {
    private LinkList sequ;
    private int count;

    public IntSequenz () {
        sequ = new LinkList();
        count = 0;
    }
}
```

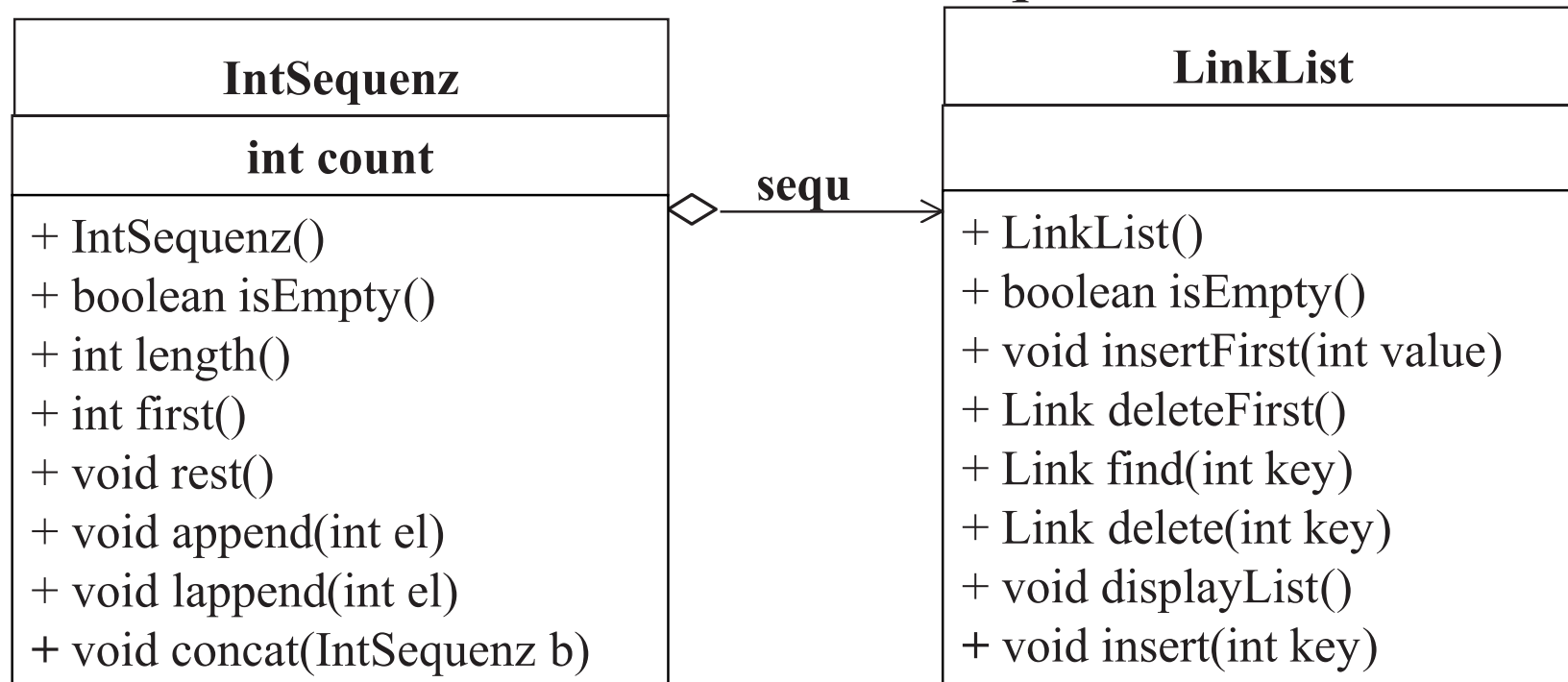
isEmpty und length sind einfach:



```
public boolean isEmpty () {
    return sequ.isEmpty();
}

public int length () {
    return count;
}
```

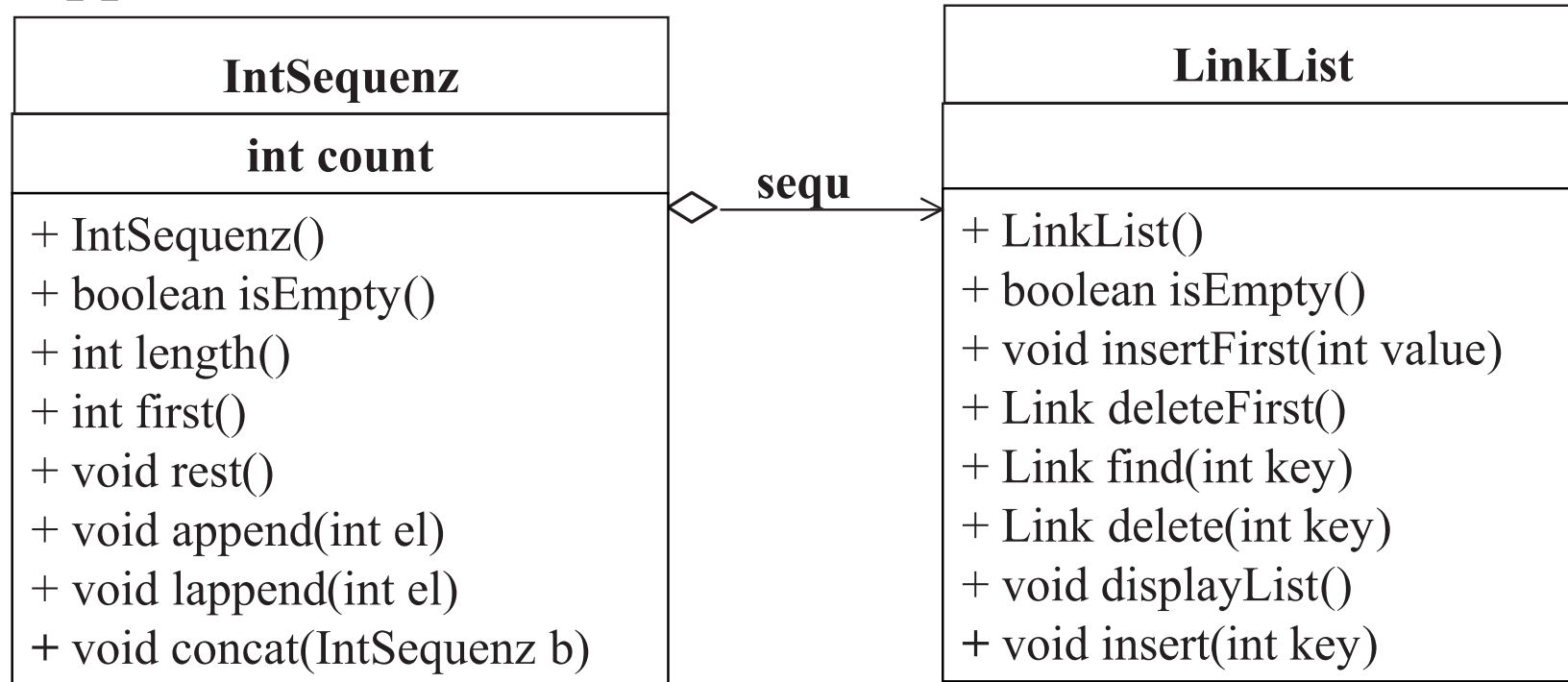
first und rest setzen voraus, dass Sequenz nicht leer ist!



```
public int first() {
    return sequ.getHead().getData();
}

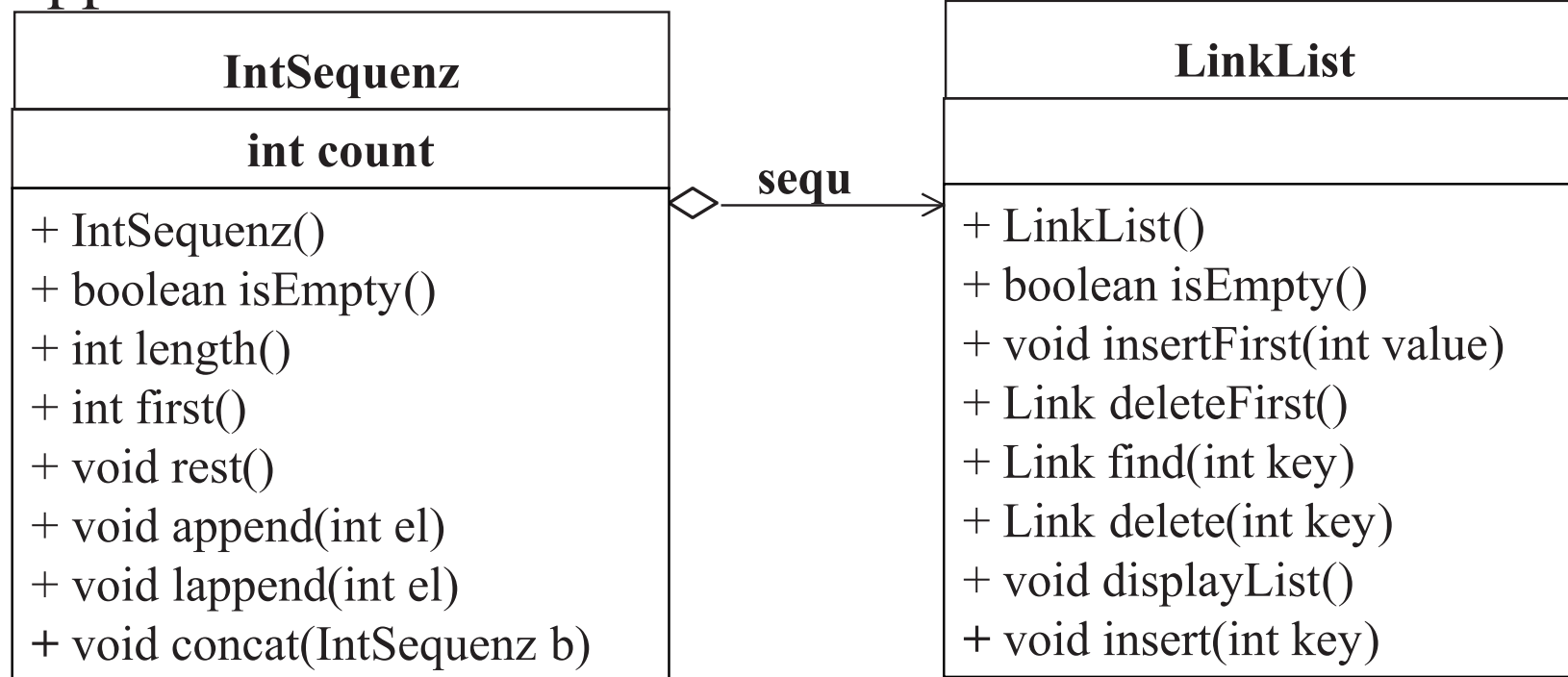
public void rest () {
    Link dummy = sequ.deleteFirst();
    count--;
}
```

# lappend ist wieder sehr einfach!



```
public void lappend(int el) {
    sequ.insertFirst(el);
    count++;
}
```

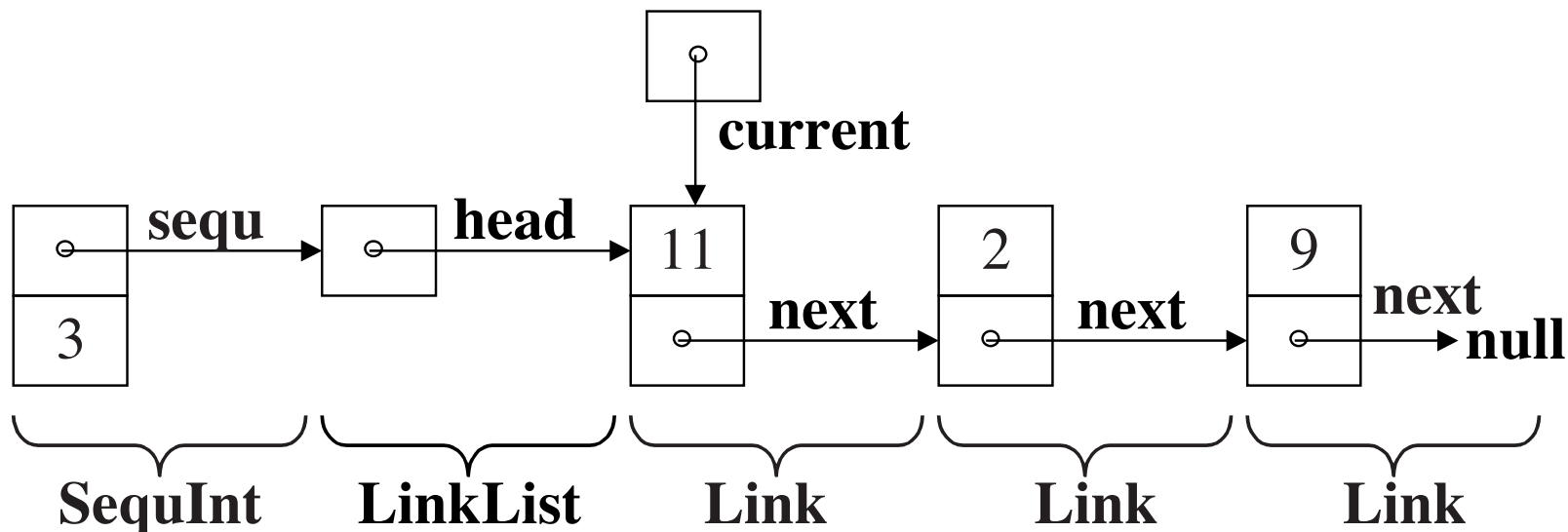
append ist umständlicher!



```
public void append(int el) {
    if (sequ.isEmpty()) sequ.insertFirst(el);
    else { Link current = sequ.getHead();
        while (current.getNext() != null)
            current = current.getNext();
        current.setNext(new Link(el));
        current.getNext().setNext(null); }    count++;
}
```

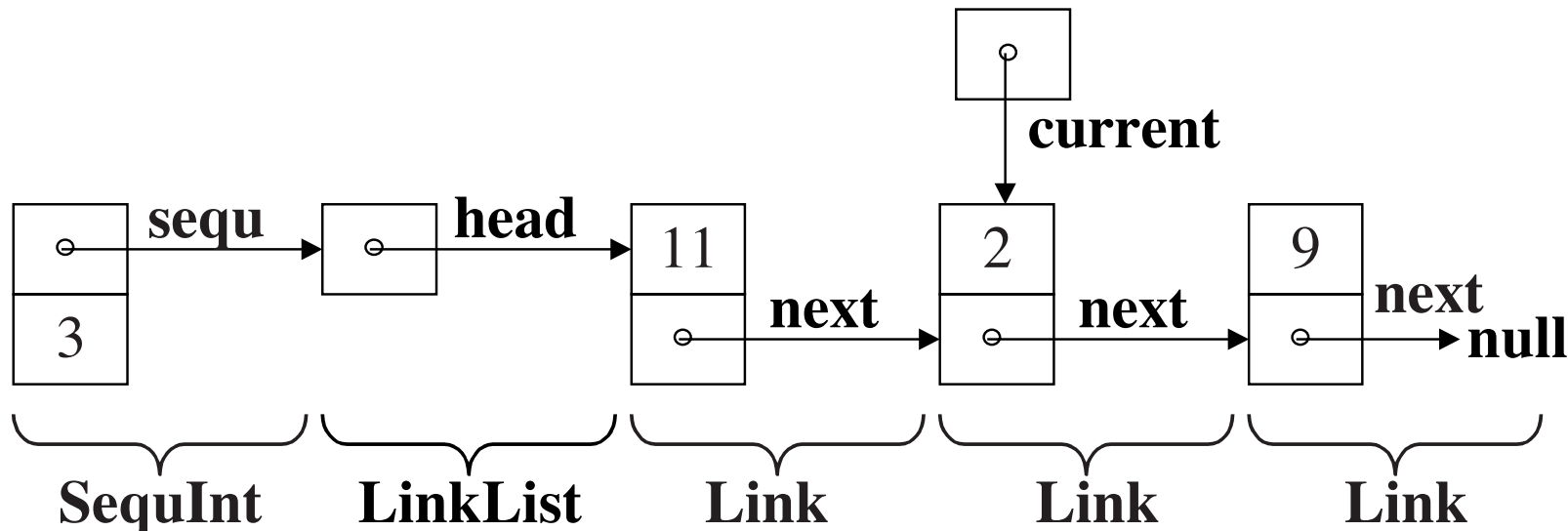
## Ein Beispiel: append(7)

```
public void append(int el) {  
    if (sequ.isEmpty()) sequ.insertFirst(el);  
    else { Link current = sequ.getHead();  
        while (current.getNext() != null)  
            current = current.getNext();  
        current.setNext(new Link(el));  
        current.getNext().setNext(null); } count++;  
}
```



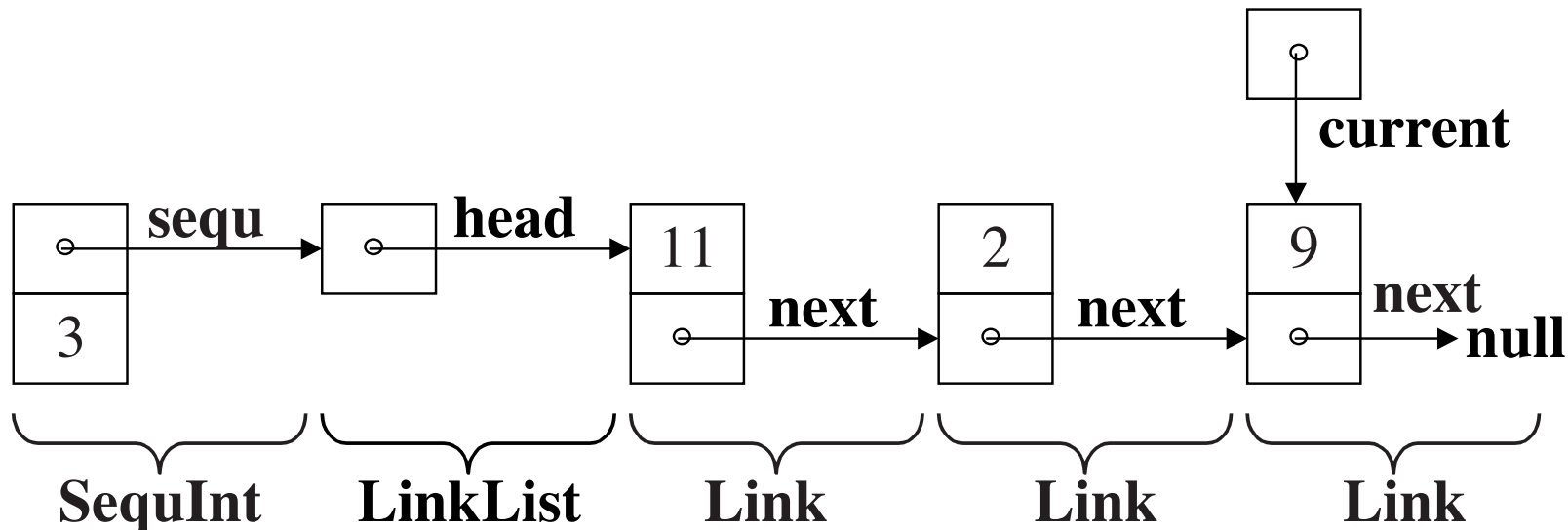
## Ein Beispiel: append(7)

```
public void append(int el) {  
    if (sequ.isEmpty()) sequ.insertFirst(el);  
    else { Link current = sequ.getHead();  
        while (current.getNext() != null)  
            current = current.getNext();  
        current.setNext(new Link(el));  
        current.getNext().setNext(null); } count++;  
}
```



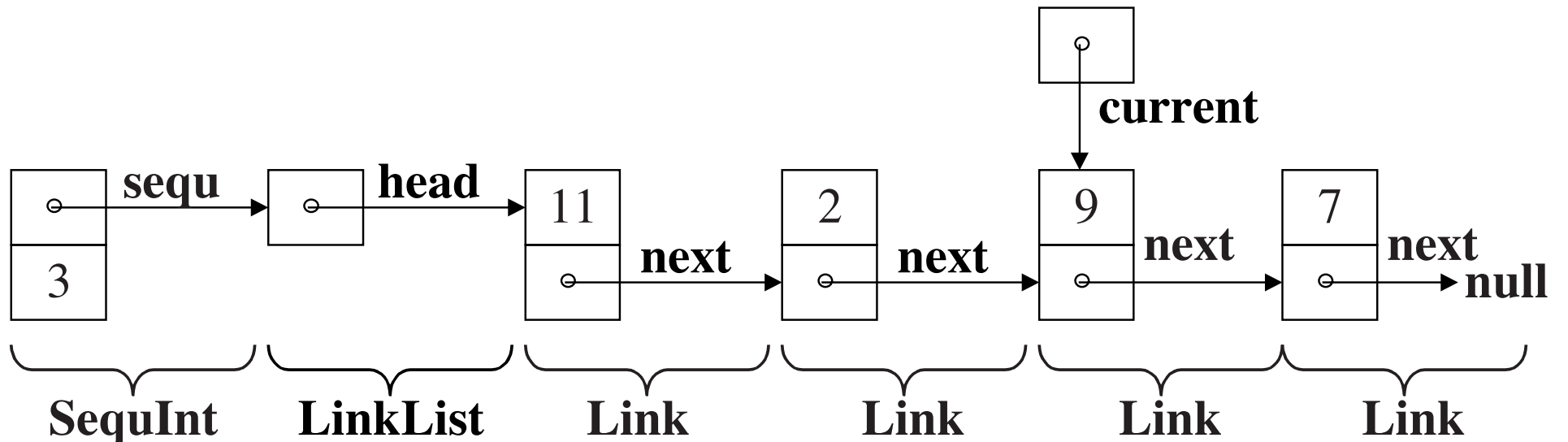
# Ein Beispiel: append(7)

```
public void append(int el) {  
    if (sequ.isEmpty()) sequ.insertFirst(el);  
    else { Link current = sequ.getHead();  
        while (current.getNext() != null)  
            current = current.getNext();  
        current.setNext(new Link(el));  
        current.getNext().setNext(null); } count++;  
}
```



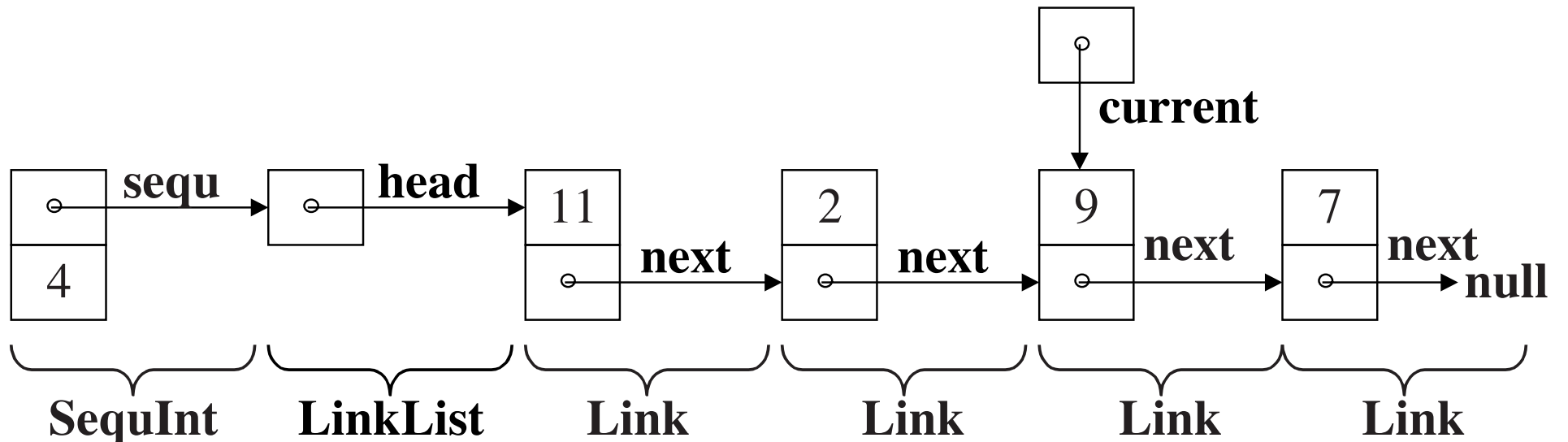
# Ein Beispiel: append(7)

```
public void append(int el) {  
    if (sequ.isEmpty()) sequ.insertFirst(el);  
    else { Link current = sequ.getHead();  
        while (current.getNext() != null)  
            current = current.getNext();  
        current.setNext(new Link(el));  
        current.getNext().setNext(null); } count++;  
}
```

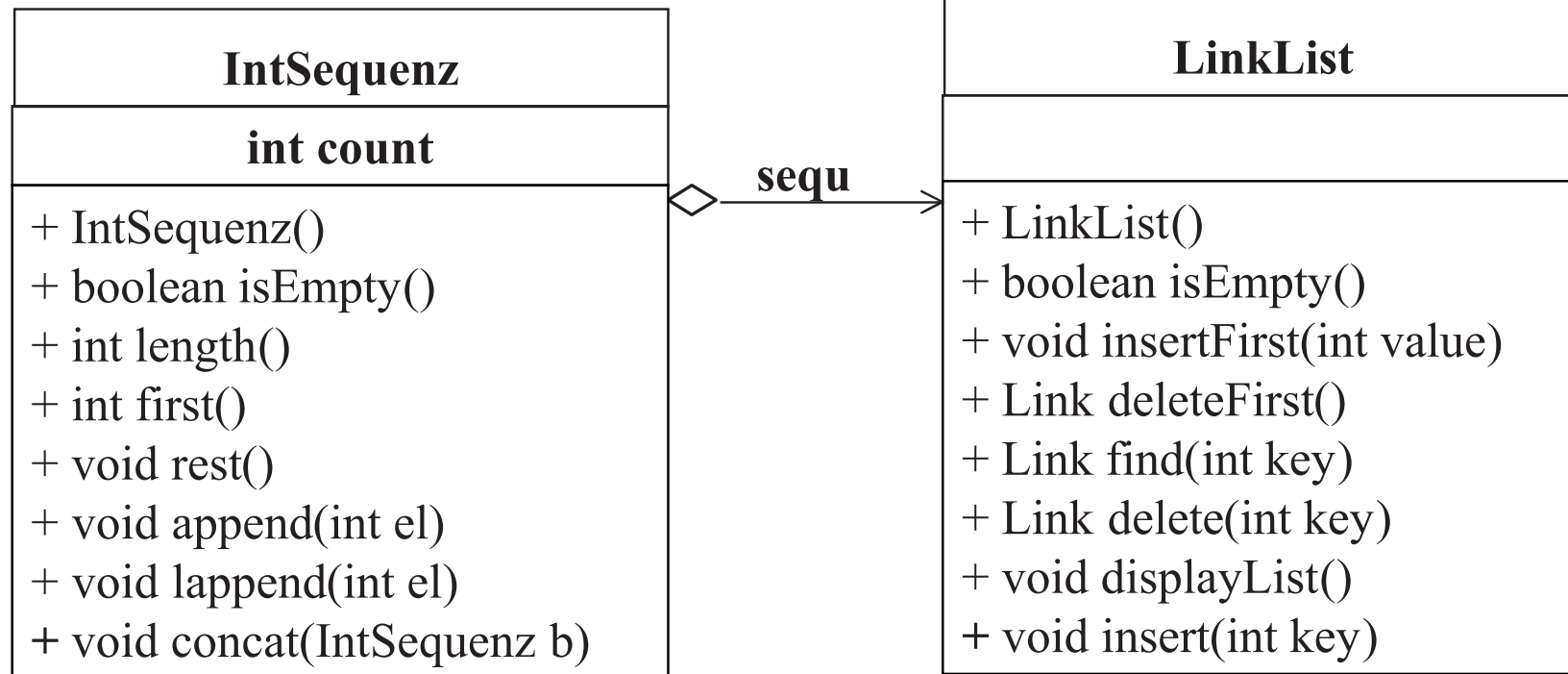


# Ein Beispiel: append(7)

```
public void append(int el) {  
    if (sequ.isEmpty()) sequ.insertFirst(el);  
    else { Link current = sequ.getHead();  
        while (current.getNext() != null)  
            current = current.getNext();  
        current.setNext(new Link(el));  
        current.getNext().setNext(null); } count++;  
}
```



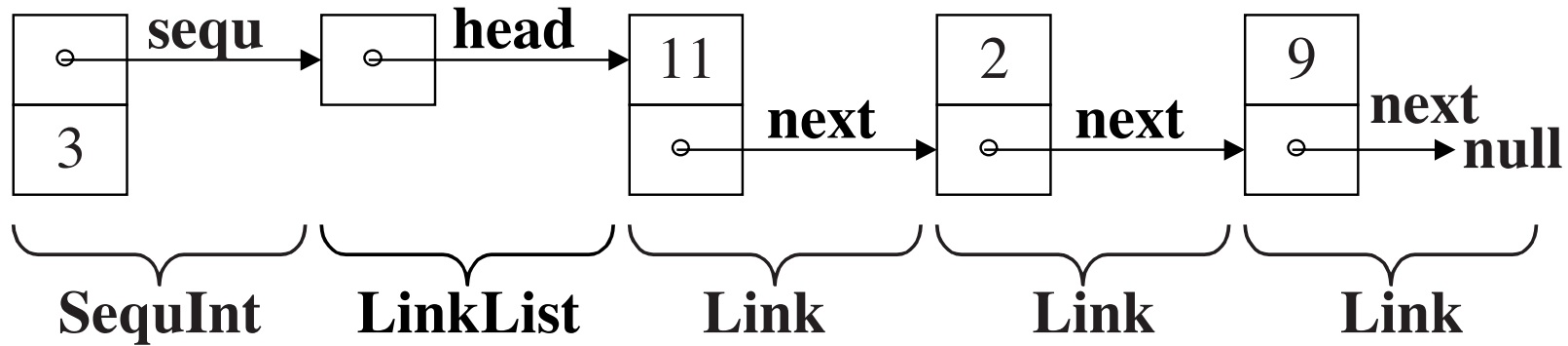
## Zum Schluss noch concat



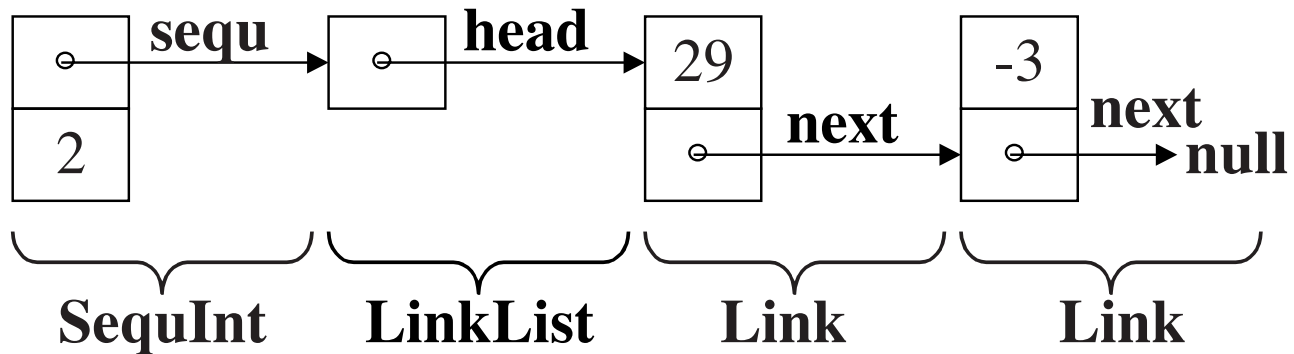
```
public void concat(IntSequenz b) {
    if (sequ.isEmpty())
        sequ = b.sequ;
    else {
        // siehe Programmieraufgabe 45 c)
    }
}
```

# Beispiel für Wirkungsweise von concat

Ausgangssequenz:

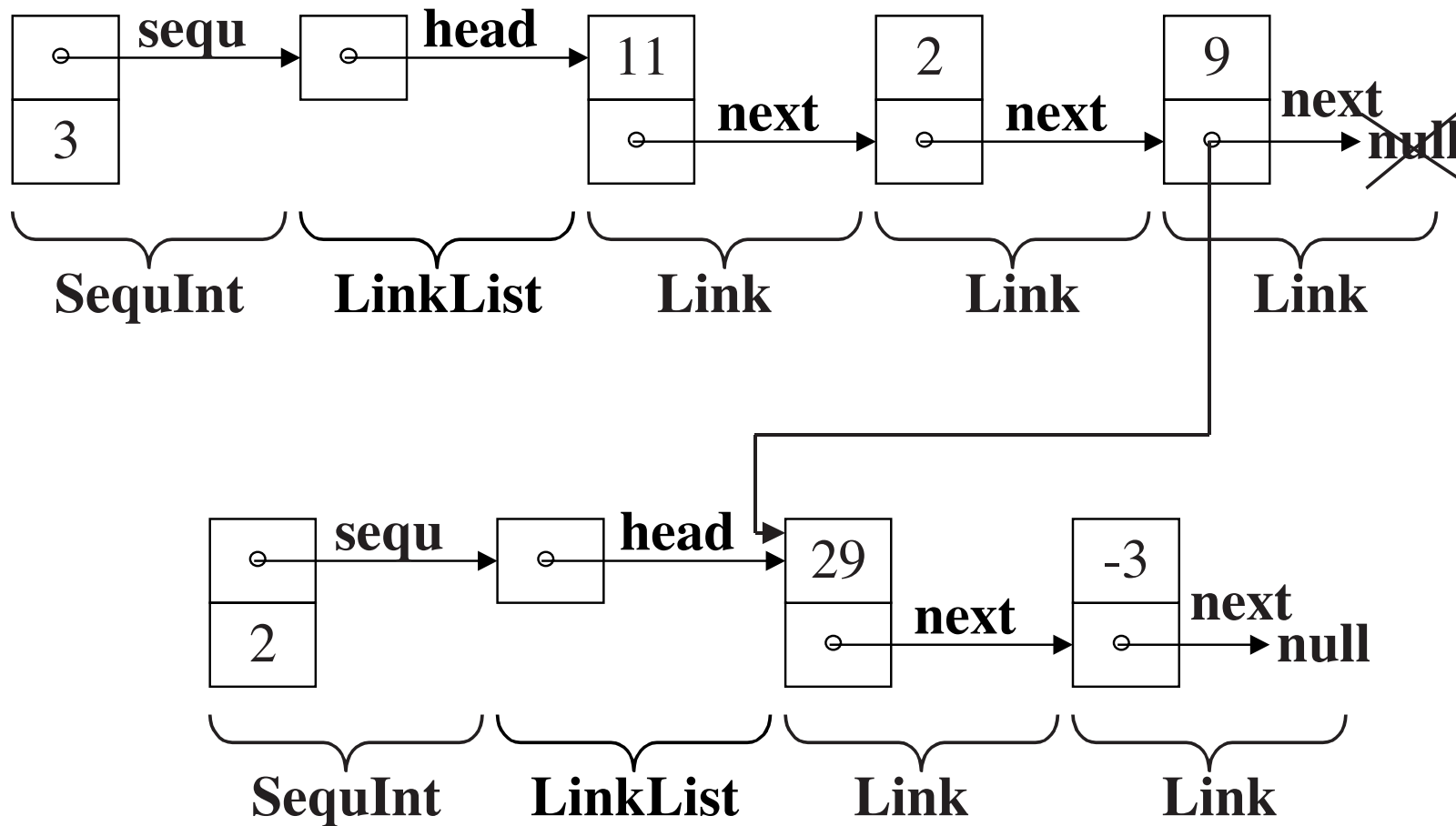


Eine zweite Sequenz b:



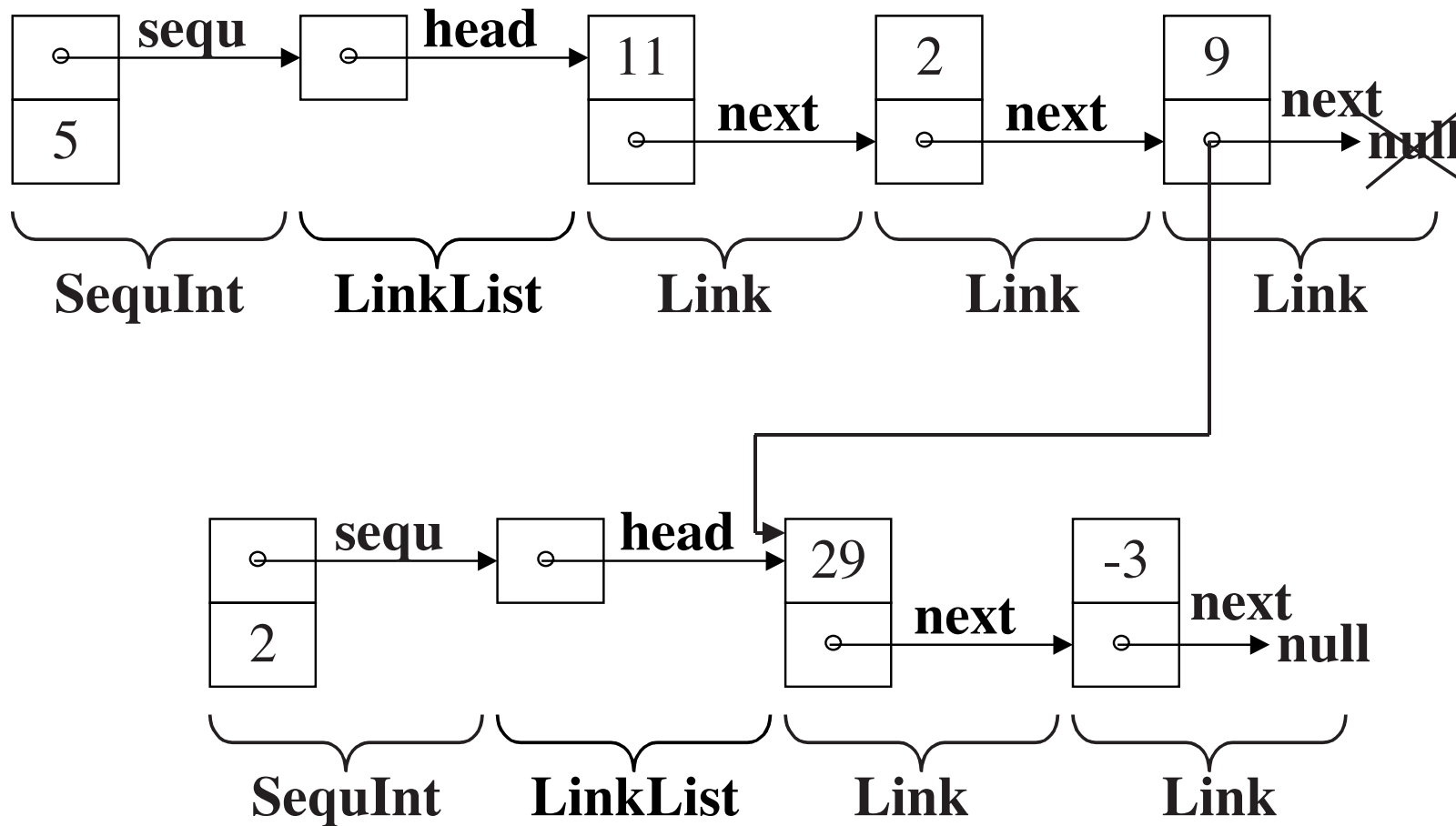
# Beispiel für Wirkungsweise von concat

Konkatenation:



# Beispiel für Wirkungsweise von concat

Konkatenation:



# Was wollten wir eigentlich?

Was haben wir?

<b>IntSequenz</b>
<b>int count</b>
+ IntSequenz() + boolean isEmpty() + int length() + int first() + void rest() + void append(int el) + void lappend(int el) + void concat(IntSequenz b)

Was wollten wir ursprünglich?

<b>IntSequenz</b>
+ IntSequenz create() + boolean isEmpty(IntSequenz a) + int length(IntSequenz a) + int first(IntSequenz a) + IntSequenz rest(IntSequenz a) + IntSequenz append(IntSequenz a, int el) + IntSequenz lappend(IntSequenz a, int el) + IntSequenz concat(IntSequenz a, IntSequenz b)

**Lösung:** zusätzliche Klassenmethoden

## Zusätzliche Klassenmethoden

```
class IntSequenz {
    // Instanzvariablen und -methoden wie bisher

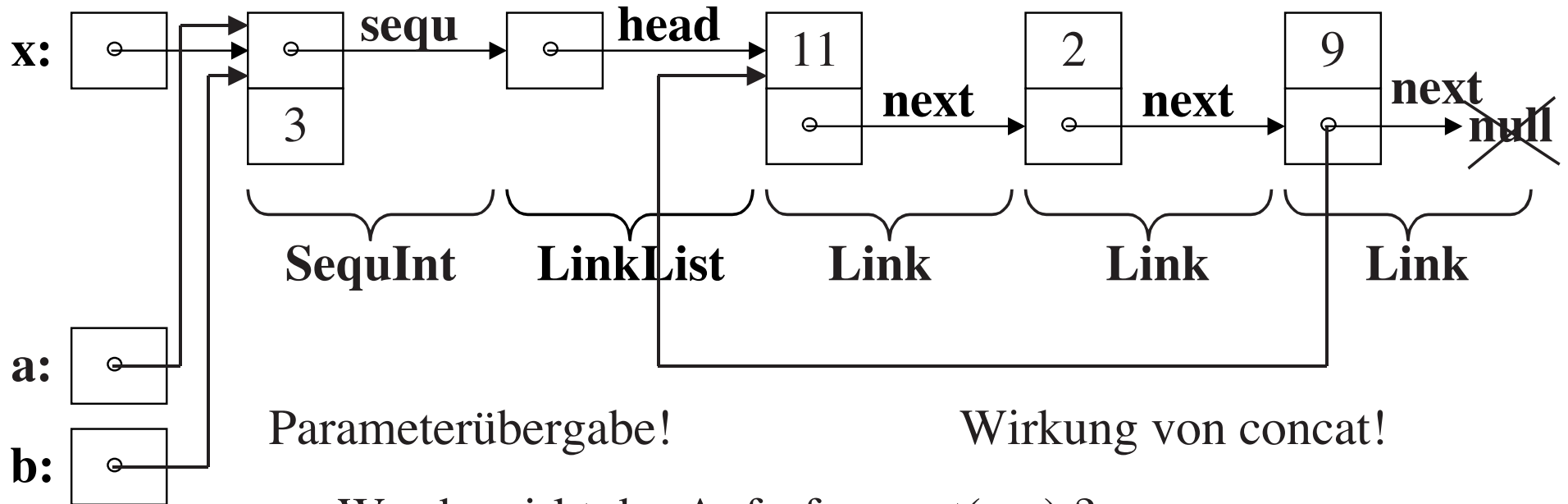
    public static IntSequenz create () {
        return new IntSequenz();
    }

    public static boolean isEmpty(IntSequ a) {
        return a.isEmpty();
    }
    ...
    public static IntSequenz concat (IntSequenz a,
                                     IntSequenz b) {
        return a.concat(b);
    }
}
```

# Können wir damit funktional programmieren?

NEIN!

Beispiel:

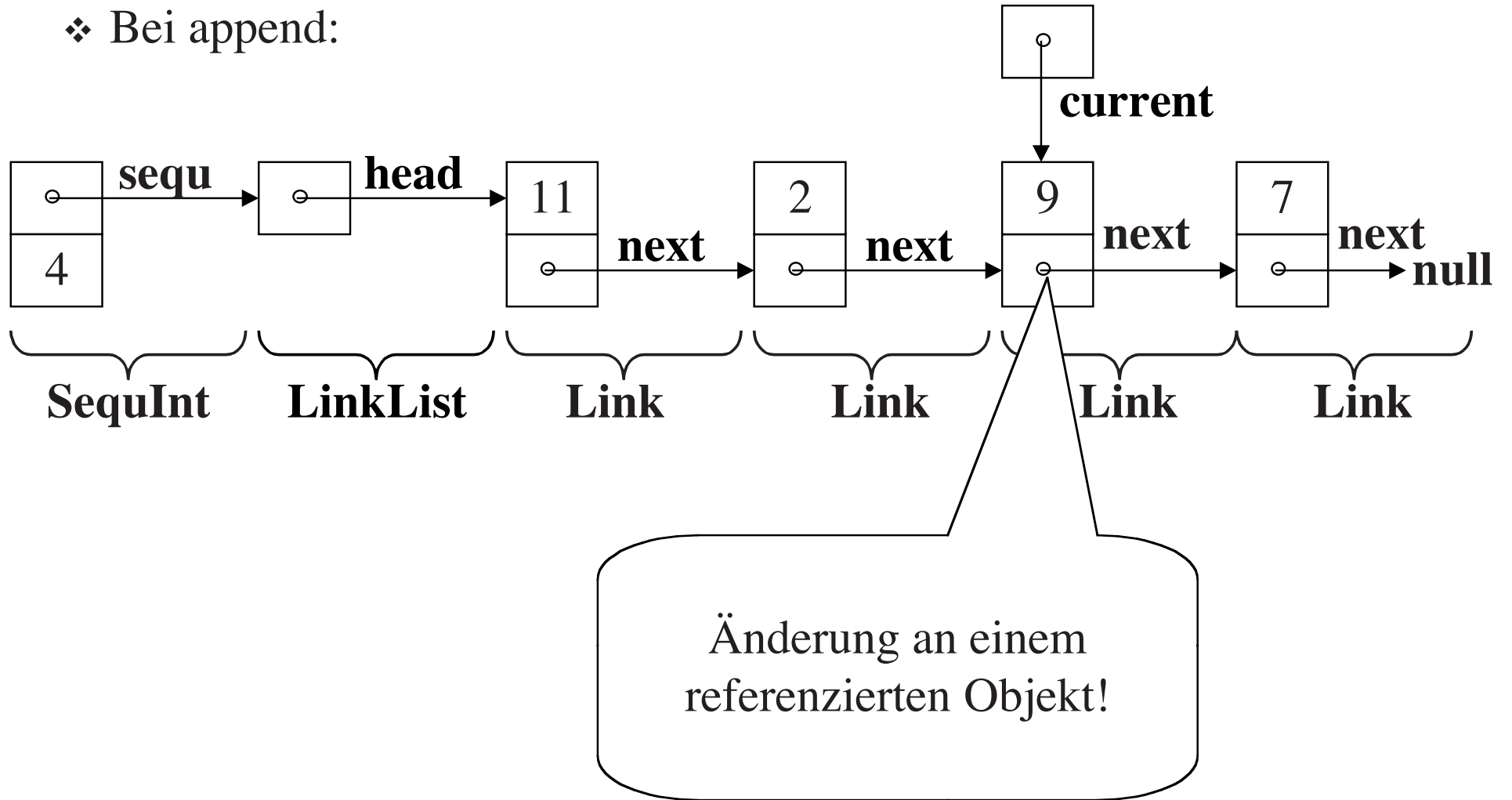


Was bewirkt der Aufruf `concat(x,x)` ?

**Ist das die unendliche Sequenz??**

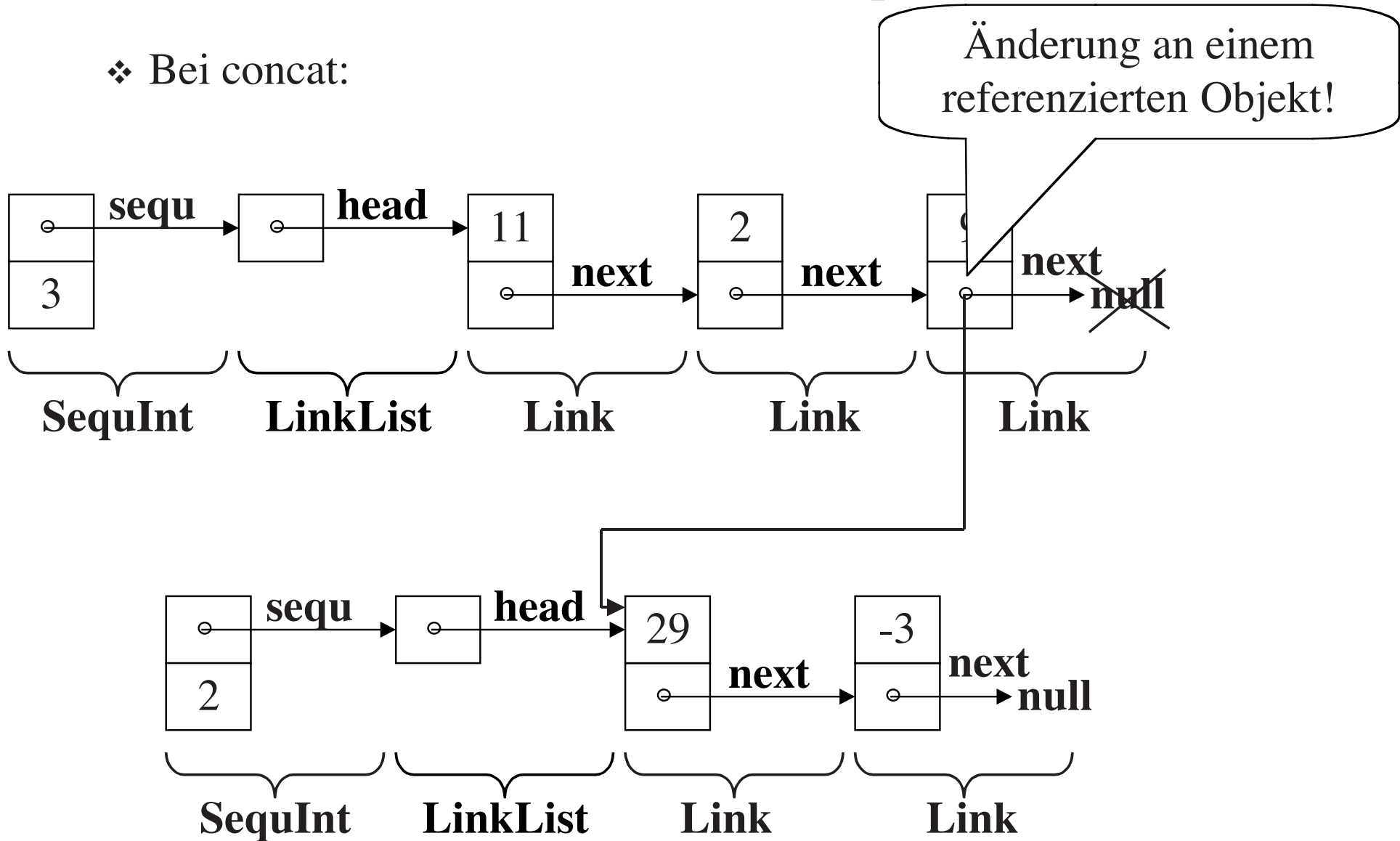
# Wo haben wir das funktionale Prinzip verletzt?

❖ Bei append:



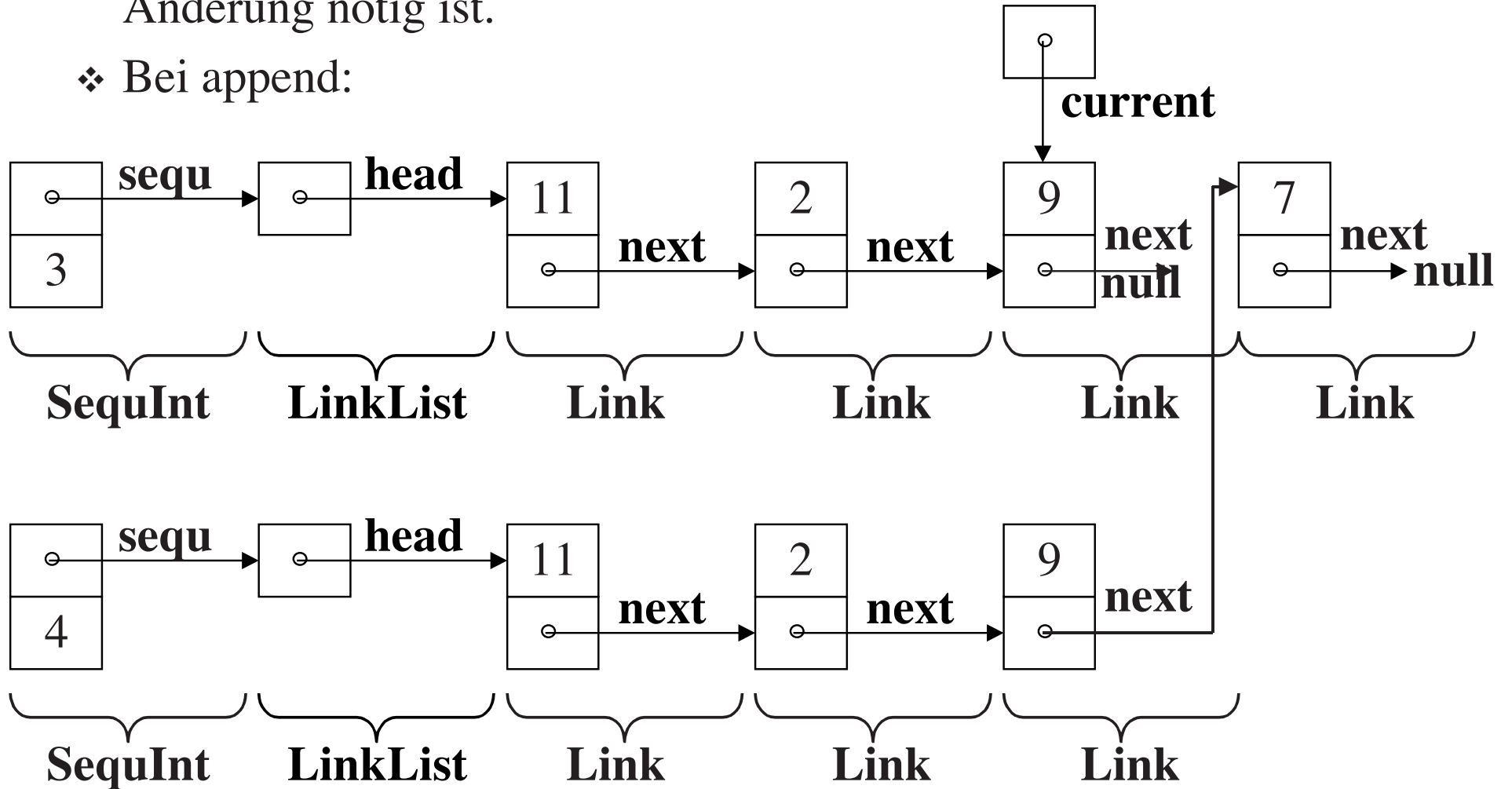
# Wo haben wir das funktionale Prinzip verletzt?

❖ Bei concat:



# Wie können wir das funktionale Prinzip retten?

- ❖ Durch Kopie bis zu der Stelle im referenzierten Objekt, wo die Änderung nötig ist.
- ❖ Bei append:



# Fazit

- ❖ Bei der Einhaltung des funktionalen Prinzips entstehen baumartige Geflechte, bei denen die Einstiegspunkte nicht die Wurzeln sondern die Blätter sind.
- ❖ In jedem Fall sind die Geflechtstrukturen zyklennfrei.
- ❖ In Gofer werden Sequenzen in ähnlicher Weise realisiert
- ❖ In unserem speziellen Beispiel verlieren wir durch den Kopier-Aufwand den Vorteil von Listen
  - (d.h. deren Flexibilität bei Einfügen und Löschen)
- ❖ Deshalb wurde IntSequenz von der Übungsleitung
  - nicht mittels Listen realisiert
  - sondern mit Reihungen
  - bzw. Java-Vektoren

**Vielen Dank für die  
Aufmerksamkeit  
und viel Erfolg  
weiterhin!**