5 Listen

5.1 Fibonacci-Reihe

Zum Aufwärmen eine simple Aufgabe: Schreiben Sie ein Programm, dass die ersten 25 **Fibonacci-Zahlen** ausgibt. Die Fibonacci-Reihe ist folgendermassen definiert:

$$f_1 = f_2 = 1 \tag{1}$$

$$f_n = f_{n-1} + f_{n-2} \tag{2}$$

Sie können z. B. folgendes Codegerüst verwenden:

```
1 package fibonacci;
2
3 public class PrintFibonacci
4 {   public static void main(String[] args)
5     {   // TODO:
6         // Calculate and print the 25 first Fibonacci-numbers
7     }
8 }
```

5.2 Verkettete Liste

Mit `Integer.MAX_VALUE` können wir in Java den **maximal möglichen** Wert einer `int`-Variablen abfragen. Wollten wir alle Fibonacci-Zahlen die kleiner als `Integer.MAX_VALUE` in eine Liste speichern hätten wir wieder das Problem der **zu Beginn unbekannten** Array-Länge. Ihre Lösung von letzter Woche bietet hier Hilfe, weil das Array bei Bedarf automatisch in vergrösserter Form neu erstellt wird. Diese Woche wollen wir eine alternative Implementation verwenden, die **verkettete Liste** (Linked List).

In der Nachbesprechung der ersten Übungsserie (Jumbling Strings `MixMiddle.java`) erwähnten wir nebenbei, dass eine verkettete Liste den Vorteil hat, dass sich Elemente beliebig aus der Liste entfernen lassen. In einem Array entstehen beim **Entfernen Löcher**, die nur vermieden werden können, indem nach jedem Entfernen alle nachfolgenden Elemente ein Element nach vorne verschoben werden. Gleiches gilt für das **Einfügen**. Das Verschieben der nachfolgenden Elemente für eine Einfügeoperation war Teil der zweiten Übungsserie (`Highscore`). War mühsam, oder?

Um etwas hinter die Kulissen zu sehen, implementieren wir eine solche **verkettete Liste**. Der Kern der Sache ist eine Hilfsklasse `ListElement`, die einerseits die Nutzdaten, andereseits eine Referenz auf ein Objekt der gleichen Klasse **kapselt**.

```
1 public class ListElement
2 {   public int data;
3     public ListElement next;
4     ... // Constructors + toString
```

Jedes Element hat so den **Link** zum nächsten Element. Dadurch entsteht eine Kette rsp. eine Liste. Das letzte Element referenziert "`null`" *(Nichts, Null)*. Die Klasse `LinkedList` verwendet Objekte von `ListElement`, um eine Liste von `int`-Zahlen zu speichern.

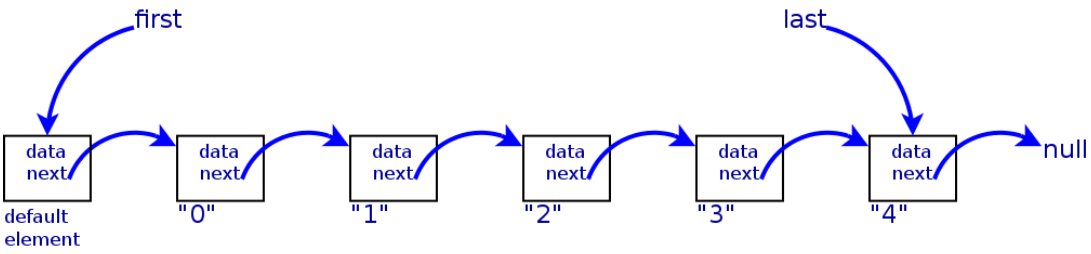

*Abbildung 1: Das Bild zeigt die Objekte der Klasse `ListElement`, wie sie eine Liste formen. Die **Liste** im Bild enthält fünf Elemente (Indices 0 bis 4). Das "Default Element" wird von der Klasse `LinkedList` im Konstruktor erstellt und bleibt bestehen. Es vereinfacht die Implementation wesentlich, ist aber nicht Teil der nach aussen sichtbaren Liste.*

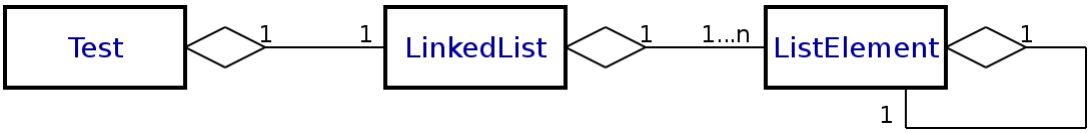

Abbildung 2: Einfaches Klassendiagramm (ohne Methoden) zum Programmaufbau. Der "Pfeil" mit dem Rhombus bezeichnet eine Aggregation (has-a-Relation). Die Pfeile lassen sich an den Feldern in den Klassen ablesen und umgekehrt. Der Rhombus symbolisiert die "Variable" (Feld), deswegen liegt es auf der Seite der benutzenden Klasse.

Folgende Methoden (gesamthaft **Interface**) könnte `LinkedList` z. B. anbieten:

```
1  package lists;
2
3  public class LinkedList
4  {    private ListElement first; // reference of the first element
5       private ListElement last; // reference of the last element (aids fast adding)
6
7       public LinkedList() // constructor
8       public boolean isEmpty() // test if empty
9       public void purge() // erase all
10      public int length() // return length
11      public String toString() // human readable form
12      public int get(int pos) // get the data from index pos
13      public void insert(int pos, int dat) // insert at index
14      public void add(int n) // inserts at the end
15      public void delete(int pos) // delete element at index
16 }
```

Wenn Sie nicht *"auf der grünen Wiese"* anfangen möchten, finden Sie ein **Template** mit einem grossen Teil des Codes als Zip-File auf der Kurswebsite.

5.3 Sortieren

Hier noch eine **freiwillige Zusatzaufgabe** für die Fortgeschrittenen. Spendieren Sie Ihrer `LinkedList` noch eine **Sortierfunktion**, also eine Methode `sort` ohne Parameter, die die ganze Liste **absteigend** sortiert.

Es gibt viele Möglichkeiten das Sortieren zu implementieren. Ein Variante ist es, Element für Element, immer den Wert mit dem Nachfolgenden zu vergleichen und falls die Reihenfolge **nicht** stimmt, die Elemente zu **vertauschen**. Dieser Algorithmus nennt sich **BubbleSort**. BubbleSort ist abgeschlossen, wenn ein voller Durchgang durch die Liste **keine** Elemente mehr vertauscht hat.

Das Arbeiten mit Referenzen hat **Vorteile**, wenn grössere Mengen an Daten (z. B. Dossier, Bild, Film, ...) in den Elementen vorhanden sind. So müssen zum Sortieren nur Referenzen geändert werden, die Daten selber bleiben an Ort und Stelle. Das Austauschen zweier Referenzen dauert nur wenige Nanosekunden.