Dynamische Datenstrukturen III : Die Warteschlange

Lernziele:

- Sie kennen das Prinzip der Warteschlange und können sie anwenden

1 Statt LIFO ist es FIFO

Beim Stack haben wir gesehen, dass das zuletzt-hinzugefügte Element das zuerst-behandelte Element ist (last-in, first-out). Diese Datenstruktur bildet aber keine gute Dienstleistungsstrategie ab, wenn wir Elemente in einer „fairen" Weise behandeln wollen. Das kann die Warteschlange viel besser, wie wir es auch aus dem richtigen Leben kennen: es wird der zuerst bedient, der auch am längsten gewartet hat.

Die Warteschlange (oder Queue) arbeitet nach dem FIFO-Prinzip (first-in, first-out).

Warteschlangen werden immer dann eingesetzt, wenn Sie Elemente unter Wahrung ihrer relativen Reihenfolge in einer Collection speichern wollen. Sie werden in der gleichen Reihenfolge entnommen, in der sie eingefügt wurden.

Die Grundoperationen für eine Warteschlange sind:

enqueue -> ein Element ans Ende hinzufügen
dequeue -> ein Element am Anfang entfernen

2 Aufgabe: Anwendung des Prinzips FIFO

Überlegen Sie sich, wie Sie Ihre verkettete Liste umbauen können, um eine Warteschlange zu implementieren.
Hinweis: Sie benötigen zwei Knoten (Nodes) in Ihrer Warteschlange: ein Knoten für den Anfang (first) und einen Knoten als Referenz für das Ende (last).

Beim Hinzufügen ist jeweils der neueste Knoten der Letzte.

Hinweis: Natürlich bietet Java eine Collection-Klasse *Queue* an, um dieses Prinzip anzuwenden. Probieren Sie aber zuerst Ihre eigene Version mit der bestehenden *Node* Klasse.

Lösungsansatz: (aus Sedgwick)

```
public class Queue<Item> implements Iterable<Item>
{
   private Node first; // link to least recently added node
   private Node last;  // link to most recently added node
   private int N;      // number of items on the queue

   private class Node
   {  // nested class to define nodes
      Item item;
      Node next;
   }

   public boolean isEmpty() {  return first == null;  }  // Or: N == 0.
   public int size()        {  return N;  }

   public void enqueue(Item item)
   {  // Add item to the end of the list.
      Node oldlast = last;
      last = new Node();
      last.item = item;
      last.next = null;
      if (isEmpty()) first = last;
      else           oldlast.next = last;
      N++;
   }

   public Item dequeue()
   {  // Remove item from the beginning of the list.
      Item item = first.item;
      first = first.next;
      if (isEmpty()) last = null;
      N--;
      return item;
   }
```