## 1.1. Stack und Warteschlangen

### 1.1.1. Stack

```
public class Stack<E>
extends Vector<E>
```

Die Klasse Stack repräsentiert einen Stapelspeicher. Dieser funktioniert anch dem LIFO-Prinzip (Last-In-First-Out). Beim Hinzufügen von Elementen wächst die Datenstruktur dynamisch. Die Klasse Stack ist als Erweiterung der Klasse Vector designed, was jedoch im Gegensatz zu den charakteristischen Eigenschaften eines Stapels stehen. Dazu zählen unter anderem die Methoden elementAt(), indexOf(), insertElementAt(), removeElementAt(), setElementAt(),… - diese Methoden sind für einen Stapel eigentlich nicht vorgesehen.

### 1.1.2. Queues und Deques

```
public interface Queue<E>
extends Collection<E>
```

Während das Interface *Queue* nur das FIFO-Prinzip unterstützt, kann mit Hilfe des Interfaces *Deque* auch in umgekehrter Richtung durch die Schlange gewandert werden.

```
public interface Deque<E>
extends Queue<E>
```

Queue erweitert direkt das Interface Collection, wobei jeweils zwei Möglichkeiten der Abfrage, Entfernen und Hinzufügen in die Queue möglich ist:

| | mit Ausnahme | ohne Ausnahme |
|---|---|---|
| Abfragen | element() | peek() |
| Entfernen | remove() | poll() |
| Hinzufügen | add() | offer() |

Tabelle 1: Es gibt zwei Möglichkeiten auf Queues zuzugreifen

Entsprechend liefert eine Deque mehrere Möglichkeiten:

| | erstes Element | | letztes Element | |
|---|---|---|---|---|
| | mit Ausnahme | ohne Ausnahme | mit Ausnahme | ohne Ausnahme |
| Abfragen | getFirst() | peekFirst() | getLast() | peekLast() |
| Hinzufügen | addFirst(e) | offerFirst(e) | addLast(e) | offerLast(e) |
| Entfernen | removeFirst() | pollFirst() | removeLast() | pollLast() |

Tabelle 2: Zugriff auf eine DoubleEndedQueUE

Wichtigste Implementierung ist die LinkedList. Weitere Implementierungen sind von Deque sind ArrayDeque und LinkedBlockingDeque.
Die ArrayDeque ist intern durch ein Array realisiert.
Die LinkedBlockingDeque realisiert BlockingDeque als blockierende Deque und ist intern wie eine LinkedList realisiert.

Beispiel: Erzeuger

#### 1.1.2.1. Blockierende Queues und Prioritätswarteschlangen

Interessante Queue-Klassen sind:

- *ConcurrentLinkedQueue:* Thread-sichere Queue durch verkettete Listen implementiert. Null-Elemente sind nicht erlaubt!
- *DelayQueue:* Queue, aus der die Elemente erst verzögert entnommen werden können.
- *ArrayBlockingQueue:* Queue mit einer maximalen Kapazität, abgebildet auf ein Feld.
- *LinkedBlockingQueue:* Queue beschränkt oder mit maximaler Kapazität, abgebildet durch eine verkettete Liste.
- *PriorityQueue:* Hält in einem Heap-Speicher Elemente sortiert und liefert bei Anfragen das jeweils kleinste Element. Wie beim TreeSet müssen die Elemente entweder Comparable implementieren, oder es muss ein Comparator angegeben werden. Unbeschränkt.
- *PriorityBlockingQueue:* Wie PriorityQueue, nur blockierend.
- *SynchronousQueue:* Eine blockierende Queue zum Austausch von genau einem Element. Durch diese Funktionsweise benötigt die SynchronousQueue keine Kapazität, denn Elemente werden, falls platziert, direkt konsumiert und müssen nicht zwischengelagert werden.

Die Priority-Klassen implementieren im Gegensatz zu den übrigen kein FIFO-Verhalten.