



Übung 03: Elementare Datenstrukturen

Aufgabe 1: Was macht das Programm?

- a) Was ist die Ausgabe des rechten Codes, wenn n gleich 50 ist? Erklären Sie, was der Code für positive Integer n berechnet.

```
Stack<Integer> stack =
    new Stack<Integer>();
while (n > 0) {
    stack.push(n % 2);
    n = n / 2;
}
for (int d : stack) {
    System.out.print(d);
}
System.out.println();
```

- b) Angenommen, die **Queue** Q enthält bereits einige Strings. Nun wird der rechte Code ausgeführt. Wie sieht die **Queue** Q danach aus? Begründen Sie Ihre Antwort.

```
Stack<String> stack =
    new Stack<String>();
while (!q.isEmpty()) {
    stack.push(q.dequeue());
}
while (!stack.isEmpty()) {
    q.enqueue(stack.pop());
}
```

Aufgabe 2: Queue mit 2 Stacks

Beschreiben Sie wie man mit Hilfe von 2 Stacks eine Queue implementieren kann. Implementieren Sie (auf dem Papier) die Operationen `empty`, `enqueue` und `dequeue` und gehen Sie davon aus, dass der Stack die Operationen `empty()`, `pop()` und `push()` bereitstellt. Die begonnene Java-Implementierung ist rechts abgebildet.

Wie ist die (Worst-Case) Laufzeit dieser Implementierung für die 3 Operationen in Abhängigkeit der Anzahl der Elemente n in der Queue?

```
class Queue {
    Stack a, b;

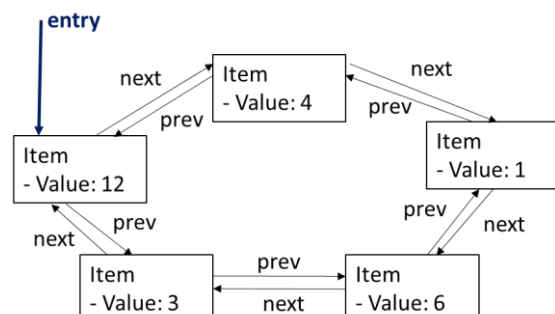
    Queue () {
        a = new Stack();
        b = new Stack ();
    }
    ...
}
```

Aufgabe 3: Doppelt verkettete Ringliste

Eine **doppelt verkettete** Ringliste ist eine spezielle doppelt verkettete Liste, bei dem kein `next`- und `prev`-Zeiger den Wert `null` hat. Alle Elemente der Datenstruktur sind in einem Zyklus/Kreis miteinander verbunden.

Im Gegensatz zu einer normalen doppelt verketteten Liste gibt es kein erstes oder letztes Element. Folglich genügt 1 Einstiegspunkt `entry`, während man bei einer (doppelt) verketteten Liste in der Regel als Einstiegspunkt sowohl `first/head` als auch `last/tail` speichert.

(umblättern)



Ihre Aufgabe: Implementieren Sie den *Abstrakten Datentyp (ADT) Queue* auf Basis einer doppelt verketteten Ringliste in Java. Sie können das bereitgestellte Codegerüst in der Datei `Ring.java` verwenden und müssen nur `enqueue` und `dequeue` implementieren. Der ADT Queue soll mindestens die folgenden Operationen bereitstellen.

- `public boolean isEmpty()`: bereits implementiert, siehe Vorgabe
- `public int size()`: bereits implementiert, siehe Vorgabe
- `public void enqueue(Item item)`: Fügt ein Element in die Queue ein. Wichtig dabei ist, dass das Element **NACH** dem Element eingefügt wird, auf das bislang der `entry`-Zeiger zeigt. Nach dem Einfügen soll der `entry`-Zeiger auf den neuen Knoten zeigen.
- `public Item dequeue()`: Entfernt das Element aus der Schlange, das als erstes eingefügt wurde. Überlegen Sie sich, an welcher Stelle im Kreis dieses Element steht.

Weitere Hinweise:

- Halten Sie sich an die Vorgabe. Sie dürfen der Klasse `Ring` keine weiteren Attribute hinzufügen!
- Fertigen Sie Skizzen an, um zu verstehen, was bei `dequeue` und `enqueue` gemacht werden muss.
- Sie erhalten eine JUnit Testklasse. Prüfen Sie, ob die bereitgestellten Tests erfolgreich sind.
- Achten Sie auf Spezialfälle, z.B.: Ein Element wird aus der leeren Queue entfernt.
- Der Übersichtlichkeit wird im Folgenden der existierende Code der Datei `Ring.java` aufgelistet.

```
public class Ring<Item> {
    private class Node<Item> {
        Item item;
        Node prev;    // pointer to next node
        Node next;    // pointer to previous node
    }

    private Node entry; // entry point into ring
    private int n;      // number of items in ring

    public Ring() {
        entry = null;
        n = 0;
    }

    public boolean isEmpty() {
        return entry == null;
    }

    public int size() {
        return n;
    }

    /* enqueue, adding element into queue IMPORTANT:
     * - add new element after current "entry" element.
     * - "entry" points afterwards to new node (== tail of queue)
     */
    public void enqueue(Item item) {
        // TODO
    }

    // dequeue, remove element from queue.
    public Item dequeue() {
        // TODO
    }

    public String toString() {
        // not shown here
    }
}
```