Westfälische Hochschule

Prof. Dr. Manfred Meyer
Fachbereich 6 / Bocholt
Professur BWL / Wirtschaftsinformatik

Algorithmen und Datenstrukturen
(Studiengang Wirtschaftsinformatik 3. Semester)

Aufgabenstellung für das Praktikum am 30.11.2023

Die heutigen Praktikumsaufgaben beziehen sich auf die in der Vorlesung besprochenen reduzierten Versionen der Interfaces

- `SimpleCollection.java`
- `SimpleIterator.java` und
- `SimpleList.java`

sowie die Listenimplementierung

- `MyLinkedList.java` und das „Hauptprogramm“
- `Aufgabe21und22.java`.

Diese Dateien stehen in der Archiv-Datei „WH-ALDS-WS23-P07-Vorgabe.jar“ zum Download bereit.

Aufgabe 21:
Schreiben Sie ein Java-Programm, in welchem Sie eine verkettete Liste vom Typ `MyLinkedList` erzeugen und der Reihe nach einige String-Objekte einfügen, zum Beispiel die Zeichenketten „Dies“, „ist“, „eine“, „schöne“, „Aufgabe“, „zum“, „Warmwerden“.

Greifen Sie nun über die Operation

```
public E get (int index)
```

der Reihe nach auf die einzelnen Elemente zu und lassen sich die an der entsprechenden Position gespeicherten String-Objekte der Reihe nach ausgeben. Falls Ihnen diese Aufgabe nicht gefällt, ersetzen Sie den dritten Eintrag („eine“) durch „keine“…

Aufgabe 22:
Programmieren Sie in der Klasse `MyLinkedList<E>` die Operation

```
public E remove(int index)
```

zum Entfernen eines Elements an einer bestimmten Stelle (Parameter `index`). Die Operation liefert als Wert das entfernte Objekt zurück bzw. `null`, falls das Objekt in der Liste nicht enthalten ist.

Testen Sie diese Operation, indem Sie auf einer Liste der Zahlen von 0 bis 19, die Sie zuvor mit der bereits in der Klasse `Aufgabe21` und `Aufgabe22` vorhandenen Operation

```
static void fillList (SimpleList<String> list)
```

erzeugen, zunächst das letzte Element und dann das erste Element aus der Liste entfernen.

Fügen Sie dann wieder die beiden zuvor entfernten Elemente in die Liste ein und führen Sie der Reihe nach die Operationen `remove(1)`, `remove(2)`, `remove(3)`, `remove(4)` und `remove(5)` aus. Lassen Sie sich jeweils im Anschluss an die Operation die Liste nochmal mit Hilfe der Funktion

```
static void printList (SimpleList<String> list)
```

ausgeben.

Was fällt Ihnen auf? Welches Element steht jetzt an der Stelle `5`?

Aufgabe 23:
Programmieren Sie analog zur Aufgabe 22 in der Klasse `MyArrayList` die Operation

```
public E remove(int index)
```

zum Entfernen eines Elements an einer bestimmten Stelle (Parameter `index`). Testen Sie auch diese Operation an einem Beispiel!

Aufgabe 24:
Programmieren Sie in der Klasse `MyLinkedList` die Operation

```
public int indexOf(E o)
```

zur Ermittlung des Index eines Elements in einer Liste. Wenn das Objekt in der Liste nicht enthalten ist, soll der Wert `-1` zurückgegeben werden.

Hinweis: Betrachten Sie die Arbeitsweise der Operation

```
public void remove(E o).
```

Testen Sie auch diese Operation an einigen Beispielen!

Aufgabe 25:
Realisieren Sie eine eigene Datenstruktur `StackQueue<E>`, die das Verhalten einer Queue ausschließlich mit Hilfe zweier Stapel (Stacks) als interne Attribute realisiert. Es dürfen also keine weiteren Funktionen implementiert werden.

Eine Queue zeichnet sich ja dadurch aus, dass sie das sogenannte FIFO-Verfahren widerspiegeln, also dass das Element, das sich am längsten in der Schlange befindet, sie als nächstes verlassen darf, demnach „an der Reihe“ ist.

Implementieren Sie die folgenden typischen Funktionen einer Queue:

- `public boolean offer (E o)`

  Diese Funktion fügt der Queue ein Element hinzu und gibt `true` zurück, wenn das Element erfolgreich eingefügt wurde und `false`, sofern dies nicht geklappt hat.

- `public E poll()`

  Diese Funktion liefert und entfernt das Element, das die Schlange verlassen darf, sich demnach am längsten in der Queue befindet. Ist die Schlange leer, soll stattdessen der Wert `null` zurückgegeben werden.

- `public E peek()`

  Diese Funktion liefert das Element, das die Schlange verlassen darf, sich also am längsten in der Queue befindet. Das Element wird allerdings im Gegensatz zur vorherigen Funktion nicht entfernt, sondern verbleibt in der Schlange. Ist die Schlange leer, soll stattdessen der Wert `null` zurückgegeben werden.

Challenge:
Ein Ringpuffer (engl. Buffer) ist eine Warteschlange mit einer vorgegebenen Größe. Er kann Daten auf einer Seite anfügen und auf der anderen Seite wieder auslesen.

Implementieren Sie dafür eine eigene Datenstruktur `RingBuffer<E>`, welche die folgenden Methoden zur Verfügung stellt:

- `public void add (E o)`
  Diese Funktion fügt ein Objekt in den Puffer ein.

- `public E get ()`
  Diese Funktion liest das Element aus, das schon am längsten im Puffer ist, und macht seinen Platz frei.

- `public int getSize ()`
  Diese Funktion liefert die Anzahl der sich aktuell in der Schlange befindlichen Elemente

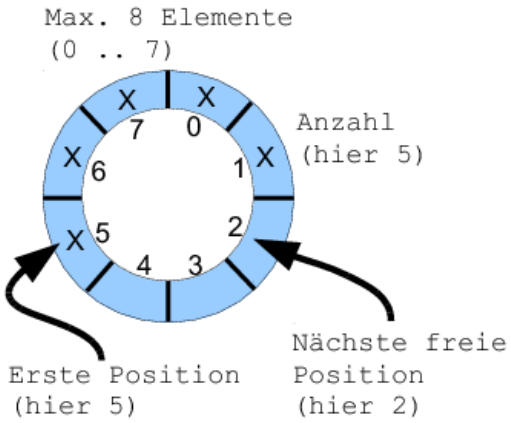
Die Datenstruktur soll die folgenden zwei Konstruktoren zur Verfügung stellen:

- `public RingBuffer ()`
  liefert einen RingBuffer mit Default-Größe n=42.

- `public RingBuffer (int n)`
  liefert einen RingBuffer mit Platz für n Elemente.

Implementieren Sie diese Datenstruktur mit einem Array, wobei sichergestellt sein soll, dass solange Elemente in den Ringpuffer eingefügt werden können, wie es freie Plätze in ihm gibt.

Da ein Ringpuffer eine Queue darstellt, wird immer das Element entfernt, das sich am längsten in dem Puffer befindet, somit kann der nächste freie Platz auch vor dem aktuell ersten Element der Schlange zu liegen kommen. Daher der Name **Ring**puffer (s. nachfolgende Grafik).



Viel Spaß und Erfolg bei den Aufgaben!