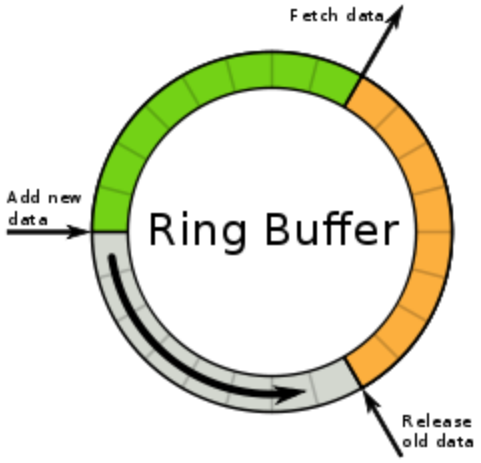
Programmieren 2 (INF)
Übungsblatt 7



Diesmal geht es um die Erstellung flexibler Container mit Hilfe generischer Klassen. Es soll eine spezielle Implementierung einer zweiendigen Warteschlange (englisch double ended queue, kurz deque) entwickelt werden. Sie folgt dem Prinzip eines Ring-Puffers. Ringpuffer haben im Einfachsten Fall eine feste Kapazität und verfügen über zwei typischerweise als Index-Attribute realisierte Referenzen. Eine zeigt auf den Anfang des gerade genutzten Bereiches, eine auf das Ende. An beiden Stellen sind Einfüge- und Löschoperationen mit konstantem Aufwand möglich, weil die Elemente dazwischen nicht verschoben werden müssen. Ringspeicher lassen sich mit Arrays problemlos realisieren. Hier werden Referenzen, wenn sie über den letzten Index hinaus erhöht werden, einfach an den Anfang gesetzt.

Aufgabe 1

Entwickeln Sie eine generische Klasse `MyDeque`, die eine Deque für beliebige Elementtypen implementiert. Im Konstruktor wird die Kapazität fest vorgegeben. Die Verwaltung der Elemente erfolgt in einem Array. Denken Sie daran, dass `new T[xy]` nicht funktioniert. Nutzen Sie stattdessen `(T[]) new Object[xy]` oder verwenden Sie einfach ein `Object`-Array-Attribut und machen sie die Casts in den Zugriffsmethoden.

Folgende Methoden sollen bereitgestellt werden:

- `boolean isEmpty()`
- `boolean isFull()`
- `int size()`
  liefert die Anzahl der aktuell enthaltenen Elemente.
- `int capacity()`
- `T get(int i)`
  `get(0)` liefert immer das erste Element, unabhängig davon, wo genau es im Array abgelegt ist!
- `void addFirst(T elem)`
  fügt vor dem ersten Element ein und macht das neue zum ersten Element.
- `T getFirst()`
- `T removeFirst()`
- `void addLast(T elem)`
  fügt nach dem letzten Element ein und macht das neue zum letzten Element.
- `T getLast()`
- `T removeLast()`

- `String toString()`
  erzeugt eine String-Repräsentation des aktuellen Zustands.

Aufgabe 2

Schreiben Sie eine kleine Klasse `MyDequeUtil`, die nützliche *statische* Methoden zum Arbeiten mit `Deques` aufnehmen soll. Im Rahmen dieses Übungsblattes werden nur zwei kleine Methoden realisiert:

- `merge`
  erhält zwei Deques und liefert eine neue Deque, welche die Elemente der beiden als Argument übergebenen Deques enthält.
- `selectLess`
  erhält eine Deque und ein Vergleichselement und gibt eine neue Deque zurück, die genau die Elemente aus der übergebenen Deque enthält, die kleiner sind als das Vergleichselement. Beachten Sie, dass diese Methode nur Sinn ergibt, wenn das Vergleichselement vom Typ der Container-Elemente ist und auch mit diesen vergleichbar ist.

Arbeiten Sie mit *generischen Methoden* und versuchen Sie die Parameter so allgemein wie möglich zu halten.

Aufgabe 3

Statten Sie Ihre Deque mit einem Iterator aus.

Aufgabe 4

Schreiben Sie eine Testklasse und Testen Sie darin Ihre Implementierung mit folgendem Code:

```
MyDeque<Integer> deque = new MyDeque<Integer>(10);
for (int i = 0; i < 3; i++)
      deque.addFirst(i);
System.out.println(deque);
for (int i = 3; i < 6; i++)
      deque.addLast(i);
System.out.println(deque);
for (int i = 0; i < 2; i++)
      deque.removeFirst();
System.out.println(deque);
while (!deque.isEmpty())
      deque.removeLast();
System.out.println(deque);
for (int i = 0; i < 8; i++)
      deque.addLast(i);
System.out.println(deque);
System.out.println(MyDequeUtil.merge(deque, deque));
System.out.println(MyDequeUtil.selectLess(deque, 5));
for (int elem : deque)
      System.out.print(elem + ", ");
```

```
[ 2 1 0 ]
[ 2 1 0 3 4 5 ]
[ 0 3 4 5 ]
[ ]
[ 0 1 2 3 4 5 6 7 ]
[ 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 ]
[ 0 1 2 3 4 ]
0, 1, 2, 3, 4, 5, 6, 7,
```

Die Ausgabe sollte etwas wie im Kästchen gezeigt aussehen!