Exercise 4a. (Automatically expanding simple queue)

The simplest way to implement a queue is to use an array as a storage for queued elements. In the enqueue() operation a new item is always put as the last element and the number of elements is incremented. In the dequeue() operation an item is always taken from the beginning of the array (index 0) and all other items are moved forward in the array. This is not the most efficient way, because data needs to be moved in each dequeue() operation. But anyway it is very easy to understand, because it behaves like the real queue of persons in the bank for example (all people take one step forward, when one customer moves to get the service). We saw the data definition and basic principles of how operation functions of this kind of queue can be implemented. You can find this kind of working queue (simpleQueue.java) and a test program from the Tube-portal. This queue uses fixed sized array, so that the program gives an error message, if you try to enqueue data when the queue is full.

It is possible to modify the queue to automatically expand its size when needed. Dynamic memory (e.g. new–operation) is needed here. Start for example with the queue size of 5 (when the queue is initialized allocate space for five elements) and every time when more space is needed automatically increase the size of the array by for example 5. In order to increase the size you need to do the following tasks

1. Create a new array with larger size
2. Copy all elements from the old (smaller) array to the new (larger) array
3. Switch to use the new array (copy the reference of the new array to the place of old array reference[1])
4. Update the new size information of the new array

To make testing easy, put an output like "size is increased and is now xx items" to the enqueue() operation, when size is increased to make it easier to follow the working of your program. Do not also use global variables in your solution, and don't never decrease the size of the array in enqueue() operation.

The most important principle here is that expanding is really automatic. It is automatic for the user of the queue. The use of the queue is as easy as before. The only difference for the user is that enqueue() function never returns false, because enqueue() operation always succeeds (assuming that the operating system has infinite amount of memory available).

Download the program simpleQueue.java from the Tube portal and compile and run them so that you understand how they works. After that do all modifications needed to make the queue automatically expandable.

Remark 1. Do not modify the main function, because you still can use it to test your more advanced queue. This also means that you must not modify the prototypes of the operation functions of the queue. The implementation of operation functions of course need modifications because now we need to take care of the current array size.

[1] Java automatic Garbage collection frees the old array memory because there is no reference to it anymore

Extra Exercise 4b. (Automatically expanding circular buffer queue, 0.25p)

In the simple queue implementation data needs to be moved in each dequeue() operation. Therefore the implementation is not very efficient. In order to improve the operation of the queue, implement the queue using the circular buffer technique presented in the lectures. Notice that this implementation must also be automatically expanding, i.e. when the space for the queue fills, more space will be allocated (this is not so easy task in this case).

Use the following test program to check that your implementation works in a right way.

```
    private static final int N = 100000;       // number of elements inserted/retrieved
                                               // from the queue, must be even

    public static void main(String[] args) {
        circularQueue<Integer> queue = new circularQueue();
        Integer                item;

       // first we play a little with the queue to test that it really works
        System.out.println("Fill the queue with " + N + " items");
       item = 0;
        for (int i = 0; i < N; i++) {
            if (!queue.enqueue(item++)) {
                System.err.println("Enqueue failed");
                return;
            }
       }
        System.out.println("Remove " + 3*N/4 + " of them");
        for (int i = 0; i < 3*N/4; i++) {
            if ((item = queue.dequeue()) == null) {
                System.err.println("Dequeue failed");
                return;
            }
       }
        System.out.println("Add then new " + N/2 + " items to the queue");
       item++;
        for (int i = 0; i < N/2; i++) {
            if (!queue.enqueue(item++)) {
                System.err.println("Enqueue failed");
                return;
            }
       }

       // then we deque elements and measure the execution time
       System.out.println("Then dequeue " +3*N/4 + " of them");
       for (int i = 0; i < 3*N/4; i++)
            item = queue.dequeue();
       System.out.println("Last item value is " + item +
                          " (should be " + (5*N/4-1) + ")");  // 3N/4 + N/2 = 5N/4
    }
```