ResizingArrayQueue.java

Below is the syntax highlighted version of ResizingArrayQueue.java from §1.3 Stacks and Queues.

```
/*************************************************************************
 *  Compilation:  javac ResizingArrayQueue.java
 *  Execution:    java ResizingArrayQueue < input.txt
 *  Dependencies: StdIn.java StdOut.java
 *  Data files:   http://algs4.cs.princeton.edu/13stacks/tobe.txt
 *
 *  Queue implementation with a resizing array.
 *
 *  % java ResizingArrayQueue < tobe.txt
 *  to be or not to be (2 left on queue)
 *
 *************************************************************************/

import java.util.Iterator;
import java.util.NoSuchElementException;

/**
 *  The {@code ResizingArrayQueue} class represents a first-in-first-out (FIFO)
 *  queue of generic items.
 *  It supports the usual <em>enqueue</em> and <em>dequeue</em>
 *  operations, along with methods for peeking at the first item,
 *  testing if the queue is empty, and iterating through
 *  the items in FIFO order.
 *  <p>
 *  This implementation uses a resizing array, which double the underlying array
 *  when it is full and halves the underlying array when it is one-quarter full.
 *  The <em>enqueue</em> and <em>dequeue</em> operations take constant amortized time.
 *  The <em>size</em>, <em>peek</em>, and <em>is-empty</em> operations takes
 *  constant time in the worst case.
 *  <p>
 *  For additional documentation, see <a href="http://algs4.cs.princeton.edu/13stacks">Section 1.3</a> of
 *  <i>Algorithms, 4th Edition</i> by Robert Sedgewick and Kevin Wayne.
 *
 *  @author Robert Sedgewick
 *  @author Kevin Wayne
 */
public class ResizingArrayQueue<Item> implements Iterable<Item> {
    private Item[] q;       // queue elements
    private int n;          // number of elements on queue
    private int first;      // index of first element of queue
    private int last;       // index of next available slot


    /**
     * Initializes an empty queue.
     */
    public ResizingArrayQueue() {
        q = (Item[]) new Object[2];
        n = 0;
        first = 0;
        last = 0;
    }

    /**
     * Is this queue empty?
     * @return true if this queue is empty; false otherwise
     */
    public boolean isEmpty() {
        return n == 0;
    }

    /**
     * Returns the number of items in this queue.
     * @return the number of items in this queue
     */
    public int size() {
        return n;
    }

    // resize the underlying array
    private void resize(int capacity) {
        assert capacity >= n;
        Item[] temp = (Item[]) new Object[capacity];
        for (int i = 0; i < n; i++) {
            temp[i] = q[(first + i) % q.length];
        }
        q = temp;
        first = 0;
        last  = n;
```

```
    }

    /**
     * Adds the item to this queue.
     * @param item the item to add
     */
    public void enqueue(Item item) {
        // double size of array if necessary and recopy to front of array
        if (n == q.length) resize(2*q.length);   // double size of array if necessary
        q[last++] = item;                        // add item
        if (last == q.length) last = 0;          // wrap-around
        n++;
    }

    /**
     * Removes and returns the item on this queue that was least recently added.
     * @return the item on this queue that was least recently added
     * @throws java.util.NoSuchElementException if this queue is empty
     */
    public Item dequeue() {
        if (isEmpty()) throw new NoSuchElementException("Queue underflow");
        Item item = q[first];
        q[first] = null;                            // to avoid loitering
        n--;
        first++;
        if (first == q.length) first = 0;           // wrap-around
        // shrink size of array if necessary
        if (n > 0 && n == q.length/4) resize(q.length/2);
        return item;
    }

    /**
     * Returns the item least recently added to this queue.
     * @return the item least recently added to this queue
     * @throws java.util.NoSuchElementException if this queue is empty
     */
    public Item peek() {
        if (isEmpty()) throw new NoSuchElementException("Queue underflow");
        return q[first];
    }


    /**
     * Returns an iterator that iterates over the items in this queue in FIFO order.
     * @return an iterator that iterates over the items in this queue in FIFO order
     */
    public Iterator<Item> iterator() {
        return new ArrayIterator();
    }

    // an iterator, doesn't implement remove() since it's optional
    private class ArrayIterator implements Iterator<Item> {
        private int i = 0;
        public boolean hasNext()  { return i < n;                               }
        public void remove()      { throw new UnsupportedOperationException();  }

        public Item next() {
            if (!hasNext()) throw new NoSuchElementException();
            Item item = q[(i + first) % q.length];
            i++;
            return item;
        }
    }

   /**
     * Unit tests the {@code ResizingArrayQueue} data type.
     *
     * @param args the command-line arguments
     */
    public static void main(String[] args) {
        ResizingArrayQueue<String> queue = new ResizingArrayQueue<String>();
        while (!StdIn.isEmpty()) {
            String item = StdIn.readString();
            if (!item.equals("-")) queue.enqueue(item);
            else if (!queue.isEmpty()) StdOut.print(queue.dequeue() + " ");
        }
        StdOut.println("(" + queue.size() + " left on queue)");
    }

}
```


Last updated: Tue Aug 30 10:09:18 EDT 2016.