Brendan Whitaker

CSE 2221 Homework 14

Professor Bucci

29 March 2017

1. We implement the given static method:

```
/**
 * Reports the smallest integer in the given {@code Queue<Integer>}.
 *
 * @param q
 *            the queue of integer
 * @return the smallest integer in the given queue
 * @requires q /= empty_string
 * @ensures <pre>
 * min is in entries(q) and
 *  for all x: integer
 *      where (x is in entries(q))
 *    (min <= x)
 * </pre>
 */
private static int min(Queue<Integer> q) {
    int small = Integer.MAX_VALUE;
    if (q.length() > 0) {
        int frontPiece = q.dequeue();
        small = min(q);
        if (small >= frontPiece) {
            small = frontPiece;
        }
        q.enqueue(frontPiece);
    }
    return small;
}
```

i. We need the requires clause since it is impossible to find the minimum value if there are no values in the queue.

ii. The first line of the ensures clause is essential since if it was not there, we could just set the minimum to be the minimum value of integer which would always hold. This line ensures that we actually return the minimum value in the queue.

2. We implement the given static method:

```
/**
 * Reports the largest integer in the given {@code Queue<Integer>}.
 *
 * @param q
 *            the queue of integer
 * @return the largest integer in the given queue
 * @requires q /= empty_string
 * @ensures <pre>
 * max is in entries(q) and
 *  for all x: integer
 *      where (x is in entries(q))
 *    (max >= x)
 * </pre>
 */
private static int max(Queue<Integer> q) {
    int large = Integer.MIN_VALUE;
    if (q.length() > 0) {
        int frontPiece = q.dequeue();
        large = max(q);
        if (large <= frontPiece) {
            large = frontPiece;
        }
        q.enqueue(frontPiece);
    }
    return large;
}

/**
 * Reports an array of two {@code int}s with the smallest and the largest
 * integer in the given {@code Queue<Integer>}.
 *
 * @param q
 *            the queue of integer
 * @return an array of two {@code int}s with the smallest and the largest
 *         integer in the given queue
 * @requires q /= empty_string
 * @ensures <pre>
 * { minAndMax[0], minAndMax[1] } is subset of entries(q) and
```

```
 *  for all x: integer
 *       where (x in in entries(q))
 *     (minAndMax[0] <= x <= minAndMax[1])
 * </pre>
 */
private static int[] minAndMax(Queue<Integer> q) {
    int[] minMaxArray = { min(q), max(q) };
    return minMaxArray;
}
```

3. We implement the given static method:

```
/**
 * Reports an array of two {@code int}s with the smallest and the largest
 * integer in the given {@code Queue<Integer>}.
 *
 * @param q
 *            the queue of integer
 * @return an array of two {@code int}s with the smallest and the largest
 *         integer in the given queue
 * @requires q /= empty_string
 * @ensures <pre>
 * { minAndMax[0], minAndMax[1] } is subset of entries(q) and
 *  for all x: integer
 *       where (x in in entries(q))
 *     (minAndMax[0] <= x <= minAndMax[1])
 * </pre>
 */
private static int[] minAndMax(Queue<Integer> q) {
    int[] array = { Integer.MAX_VALUE, Integer.MIN_VALUE };
    if (q.length() > 0) {
        int elem1 = q.dequeue();
        if (q.length() > 0) {
            int elem2 = q.dequeue();
            if (elem1 < elem2) {
                array = minAndMax(q);
                if (elem1 < array[0]) {
                    array[0] = elem1;
                }
                if (elem2 > array[1]) {
                    array[1] = elem2;
                }
            } else {
                if (elem2 < array[0]) {
                    array[0] = elem2;
```

```
                }
                if (elem1 > array[1]) {
                    array[1] = elem1;
                }
            }
            q.enqueue(elem2);
        }
        else {
            if (elem1 < array[0]) {
                array[0] = elem1;
            }
            if (elem1 > array[1]) {
                array[1] = elem1;
            }
        }
        q.enqueue(elem1);
    }

    return array;
}
```