CanvasTA Assignment Submit
CS1131 Fall 2020 - Combined Section
Program Buffer Queue

Caleb Jacobs
cajacobs@mtu.edu

Submit Summary

- Compilation succeeded!
- You passed all the instructor tests.

Compilation Results

Compilation: Succeeded with 0 Errors and 0 Warnings

Instructor Unit Tests

Success! All tests passed.

- **Number of tests: 19**
- **Number of passed: 19**
- **Number of failed: 0**
- **Number of warnings: 0**

Style Critique

Style: CanvasTA found no problems with your coding style. Good work!

File:"Week11Program.java" has 0 style issues.

```
 1  import java.io.File;
 2  import java.io.FileNotFoundException;
 3  import java.util.Scanner;
 4
 5  /**
 6   * Read files into buffer queue and display the file.
 7   *
 8   * Date Last Modified: November 28, 2020
 9   * @author Caleb Jacobs
10   *
11   * CS1131 Fall 2020
12   * Lab Section: L03
13   */
14  public class Week11Program {
15      /**
16       * Display and empty queue
17       *
18       * @param queue - queue to be emptied
19       */
20      public static void emptyQueue(Queue<Character> queue) throws
    QueueEmptyException {
21          while (!queue.isEmpty()) {
22              System.out.print(queue.dequeue());
23          }
24          System.out.print("\n");
25      }
26
27      /**
28       * Driver for printing files using buffer queue
```

```
29        *
30        * @param args - file to print
31        */
32       public static void main(String[] args) throws
QueueFullException, QueueEmptyException {
33           File inputFile = new File(args[0]);
34           try (Scanner input = new Scanner(inputFile)) {
35               input.useDelimiter("");
36
37               // Get queue size
38               String tmp = "" + input.next() + input.next();
39               int sizeLimit = Integer.parseInt(tmp);
40
41               // Initialize character queue
42               Queue<Character> queue = new Queue<>(sizeLimit);
43
44               // Read and print characters in file
45               while (input.hasNext()) {
46                   if (queue.size() < sizeLimit) {
47                       queue.enqueue(input.next().charAt(0));
48                   } else {
49                       // Print and empty queue
50                       emptyQueue(queue);
51
52                       // Check if queue is still full
53                       if (queue.isFull()) {
54                           throw new QueueFullException();
55                       }
56                   }
57               }
58
59               // Print remaining characters in file
60               emptyQueue(queue);
61
62           } catch (FileNotFoundException e) {
```

```
63              System.out.println("File, " + inputFile.toString() +
   ", not found\n");
64              e.printStackTrace();
65          }
66      }
67   }
```

File:"Queue.java" has 0 style issues.

```
 1   import java.util.LinkedList;
 2
 3   /**
 4    * Personal buffer queue data structure.
 5    *
 6    * Date Last Modified: November 28, 2020
 7    * @author Caleb Jacobs
 8    *
 9    * CS1131 Fall 2020
10    * Lab Section: L03
11    */
12   public class Queue<E> implements QueueInterface<E> {
13       private int sizeLimit = 0;                              // Max
   size of queue
14       private LinkedList<E> queue = new LinkedList<E>();      //
   Buffer queue
15
16       /**
17        * Set max size of buffer queue
18        *
19        * @param sizeLimit Limit on the number of elements in queue
20        */
21       Queue(int sizeLimit) {
22           this.sizeLimit = sizeLimit;
23       }
24
25       /**
```

```
26          * Adds one element to the rear of this queue.
27          *
28          * @param element the element to be added to the rear of the
    queue
29          */
30         @Override
31         public void enqueue(E element) throws QueueFullException {
32             if (queue.size() < sizeLimit) {
33                 queue.addFirst(element);
34             } else {
35                 throw new QueueFullException();
36             }
37         }
38
39         /**
40          * Removes and returns the element at the front of this
    queue.
41          *
42          * @return the element at the front of the queue
43          */
44         @Override
45         public E dequeue() throws QueueEmptyException {
46             if (!queue.isEmpty()) {
47                 return queue.removeLast();
48             } else {
49                 throw new QueueEmptyException();
50             }
51         }
52
53         /**
54          * Returns without removing the element at the front of this
    queue.
55          *
56          * @return the first element in the queue
57          */
58         @Override
```

```
59      public E peek() throws QueueEmptyException {
60          if (!queue.isEmpty()) {
61              return queue.getLast();
62          } else {
63              throw new QueueEmptyException();
64          }
65      }
66
67      /**
68       * Returns true if this queue contains no elements.
69       *
70       * @return true if this queue is empty
71       */
72      @Override
73      public boolean isEmpty() {
74          return queue.isEmpty();
75      }
76
77      /**
78       * Returns true if this queue contains the maximum number of
    elements.
79       *
80       * @return true if this queue is full
81       */
82      @Override
83      public boolean isFull() {
84          return (!(queue.size() < sizeLimit));
85      }
86
87      /**
88       * Returns the number of elements in this queue.
89       *
90       * @return the integer representation of the size of the
    queue
91       */
92      @Override
```

```
 93       public int size() {
 94           return queue.size();
 95       }
 96
 97       /**
 98        * Get string of buffer queue
 99        *
100        * @return string of buffer queue
101        */
102       public String toString() {
103           return queue.toString();
104       }
105   }
```