# CircularFifoQueue.java

```
1    /*
2     * Licensed to the Apache Software Foundation (ASF) under one or more
3     * contributor license agreements.  See the NOTICE file distributed with
4     * this work for additional information regarding copyright ownership.
5     * The ASF licenses this file to You under the Apache License, Version 2.0
6     * (the "License"); you may not use this file except in compliance with
7     * the License.  You may obtain a copy of the License at
8     *
9     *      http://www.apache.org/licenses/LICENSE-2.0
10    *
11    * Unless required by applicable law or agreed to in writing, software
12    * distributed under the License is distributed on an "AS IS" BASIS,
13    * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14    * See the License for the specific language governing permissions and
15    * limitations under the License.
16    */
17   package org.usi.sq.util;
18
19   import java.io.Serializable;
20   import java.util.*;
21
22   /**
23    * CircularFifoQueue is a first-in first-out queue with a fixed size that
24    * replaces its oldest element if full.
25    * <p>
26    * The removal order of a {@link CircularFifoQueue} is based on the
27    * insertion order; elements are removed in the same order in which they
28    * were added.  The iteration order is the same as the removal order.
29    * </p>
30    * <p>
31    * This queue prevents null objects from being added.
32    * </p>
33    *
34    * @param <E> the type of elements in this collection
35    * @since 4.0
36    */
37   public class CircularFifoQueue<E> {
38
39       /** Serialization version. */
40       private static final long serialVersionUID = -8423413834657610406L;
41
42       /** Underlying storage array. */
43       private transient E[] elements;
44
45       /** Array index of first (oldest) queue element. */
46       private transient int start = 0;
47
48       /**
49        * Index mod maxElements of the array position following the last queue
50        * element.  Queue elements start at elements[start] and "wrap around"
51        * elements[maxElements-1], ending at elements[decrement(end)].
52        * For example, elements = {c,a,b}, start=1, end=1 corresponds to
53        * the queue [a,b,c].
54        */
55       private transient int end = 0;
56
57       /** Flag to indicate if the queue is currently full. */
58       private transient boolean full = false;
59
60       /** Capacity of the queue. */
```

```
61       private final int maxElements;
62
63       /**
64        * Constructor that creates a queue with the default size of 32.
65        */
66       public CircularFifoQueue() {
67           this(32);
68       }
69
70       /**
71        * Constructor that creates a queue with the specified size.
72        *
73        * @param size  the size of the queue (cannot be changed)
74        * @throws IllegalArgumentException  if the size is &lt; 1
75        */
76       @SuppressWarnings("unchecked")
77       public CircularFifoQueue(final int size) {
78 2         if (size <= 0) {
79               throw new IllegalArgumentException("The size must be greater than 0");
80           }
81           elements = (E[]) new Object[size];
82           maxElements = elements.length;
83       }
84
85       /**
86        * Returns {@code true} if the capacity limit of this queue has been reached,
87        * i.e. the number of elements stored in the queue equals its maximum size.
88        *
89        * @return {@code true} if the capacity limit has been reached, {@code false} otherwise
90        * @since 4.1
91        */
92       public boolean isAtFullCapacity() {
93 2         return size() == maxElements;
94       }
95
96       /**
97        * Adds the given element to this queue. If the queue is full, the least recently added
98        * element is discarded so that a new element can be inserted.
99        *
100       * @param element  the element to add
101       * @return true, always
102       * @throws NullPointerException  if the given element is null
103       */
104      public boolean add(final E element) {
105 1         if (null == element) {
106               throw new NullPointerException("Attempted to add null object to queue");
107           }
108
109 1         if (isAtFullCapacity()) {
110               remove();
111           }
112
113 1         elements[end++] = element;
114
115 2         if (end >= maxElements) {
116               end = 0;
117           }
118
119 1         if (end == start) {
120               full = true;
121           }
122
123 1         return true;
124      }
```

```
125
126       /**
127        * Adds the given element to this queue. If the queue is full, the least recently added
128        * element is discarded so that a new element can be inserted.
129        *
130        * @param element  the element to add
131        * @return true, always
132        * @throws NullPointerException  if the given element is null
133        */
134       public boolean offer(final E element) {
135 2         return add(element);
136       }
137
138       /**
139        * Increments the internal index.
140        *
141        * @param index  the index to increment
142        * @return the updated index
143        */
144       private int increment(int index) {
145 1         index++;
146 2         if (index >= maxElements) {
147               index = 0;
148           }
149 1         return index;
150       }
151
152       /**
153        * Decrements the internal index.
154        *
155        * @param index  the index to decrement
156        * @return the updated index
157        */
158       private int decrement(int index) {
159 1         index--;
160 2         if (index < 0) {
161 1             index = maxElements - 1;
162           }
163 1         return index;
164       }
165
166       /**
167        * Returns true if this queue is empty; false otherwise.
168        *
169        * @return true if this queue is empty
170        */
171       public boolean isEmpty() {
172 2         return size() == 0;
173       }
174
175       public E remove() {
176 1         if (isEmpty()) {
177               throw new NoSuchElementException("queue is empty");
178           }
179
180           final E element = elements[start];
181 1         if (null != element) {
182 1             elements[start++] = null;
183
184 2             if (start >= maxElements) {
185                   start = 0;
186               }
187               full = false;
188           }
```

```
189 1         return element;
190       }
191
192       /**
193        * Returns the number of elements stored in the queue.
194        *
195        * @return this queue's size
196        */
197       public int size() {
198           int size = 0;
199
200 2         if (end < start) {
201 2             size = maxElements - start + end;
202 1         } else if (end == start) {
203 1             size = full ? maxElements : 0;
204           } else {
205 1             size = end - start;
206           }
207
208 1         return size;
209       }
210
211
212       /**
213        * Returns an iterator over this queue's elements.
214        *
215        * @return an iterator over this queue's elements
216        */
217       public Iterator<E> iterator() {
218 1         return new Iterator<E>() {
219
220               private int index = start;
221               private int lastReturnedIndex = -1;
222               private boolean isFirst = full;
223
224               public boolean hasNext() {
225 3                 return isFirst || index != end;
226               }
227
228               public E next() {
229 1                 if (!hasNext()) {
230                       throw new NoSuchElementException();
231                   }
232                   isFirst = false;
233                   lastReturnedIndex = index;
234                   index = increment(index);
235 1                 return elements[lastReturnedIndex];
236               }
237
238               public void remove() {
239 1                 if (lastReturnedIndex == -1) {
240                       throw new IllegalStateException();
241                   }
242
243                   // First element can be removed quickly
244 1                 if (lastReturnedIndex == start) {
245                       CircularFifoQueue.this.remove();
246                       lastReturnedIndex = -1;
247                       return;
248                   }
249
250 1                 int pos = lastReturnedIndex + 1;
251 4                 if (start < lastReturnedIndex && pos < end) {
252                       // shift in one part
```

```
253 2                   System.arraycopy(elements, pos, elements, lastReturnedIndex, end - pos);
254                 } else {
255                     // Other elements require us to shift the subsequent elements
256 1                   while (pos != end) {
257 2                       if (pos >= maxElements) {
258 1                           elements[pos - 1] = elements[0];
259                             pos = 0;
260                         } else {
261                             elements[decrement(pos)] = elements[pos];
262                             pos = increment(pos);
263                         }
264                     }
265                 }
266
267                 lastReturnedIndex = -1;
268                 end = decrement(end);
269                 elements[end] = null;
270                 full = false;
271                 index = decrement(index);
272             }
273
274         };
275     }
276 }
```

## Mutations

| | |
|---|---|
| 78 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED |
| 93 | 1. replaced boolean return with true for org/usi/sq/util/CircularFifoQueue::isAtFullCapacity → KILLED<br>2. negated conditional → KILLED |
| 105 | 1. negated conditional → KILLED |
| 109 | 1. negated conditional → KILLED |
| 113 | 1. Replaced integer addition with subtraction → KILLED |
| 115 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED |
| 119 | 1. negated conditional → KILLED |
| 123 | 1. replaced boolean return with false for org/usi/sq/util/CircularFifoQueue::add → KILLED |
| 135 | 1. replaced boolean return with false for org/usi/sq/util/CircularFifoQueue::offer → KILLED<br>2. replaced boolean return with true for org/usi/sq/util/CircularFifoQueue::offer → SURVIVED |
| 145 | 1. Changed increment from 1 to -1 → KILLED |
| 146 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED |
| 149 | 1. replaced int return with 0 for org/usi/sq/util/CircularFifoQueue::increment → KILLED |
| 159 | 1. Changed increment from -1 to 1 → KILLED |
| 160 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED |
| 161 | 1. Replaced integer subtraction with addition → KILLED |
| 163 | 1. replaced int return with 0 for org/usi/sq/util/CircularFifoQueue::decrement → KILLED |
| 172 | 1. replaced boolean return with true for org/usi/sq/util/CircularFifoQueue::isEmpty → KILLED<br>2. negated conditional → KILLED |
| 176 | 1. negated conditional → KILLED |
| 181 | 1. negated conditional → KILLED |
| 182 | 1. Replaced integer addition with subtraction → KILLED |
| 184 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED |
| 189 | 1. replaced return value with null for org/usi/sq/util/CircularFifoQueue::remove → KILLED |
| 200 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED |
| 201 | 1. Replaced integer subtraction with addition → KILLED<br>2. Replaced integer addition with subtraction → KILLED |
| 202 | 1. negated conditional → KILLED |
| 203 | 1. negated conditional → KILLED |
| 205 | 1. Replaced integer subtraction with addition → KILLED |
| 208 | 1. replaced int return with 0 for org/usi/sq/util/CircularFifoQueue::size → KILLED |

| | |
|---|---|
| 218 | 1. replaced return value with null for org/usi/sq/util/CircularFifoQueue::iterator → KILLED |
| 225 | 1. replaced boolean return with true for org/usi/sq/util/CircularFifoQueue$1::hasNext → KILLED<br>2. negated conditional → KILLED<br>3. negated conditional → KILLED |
| 229 | 1. negated conditional → KILLED |
| 235 | 1. replaced return value with null for org/usi/sq/util/CircularFifoQueue$1::next → KILLED |
| 239 | 1. negated conditional → KILLED |
| 244 | 1. negated conditional → KILLED |
| 250 | 1. Replaced integer addition with subtraction → KILLED |
| 251 | 1. changed conditional boundary → SURVIVED<br>2. changed conditional boundary → SURVIVED<br>3. negated conditional → SURVIVED<br>4. negated conditional → KILLED |
| 253 | 1. Replaced integer subtraction with addition → KILLED<br>2. removed call to java/lang/System::arraycopy → KILLED |
| 256 | 1. negated conditional → KILLED |
| 257 | 1. changed conditional boundary → KILLED<br>2. negated conditional → KILLED |
| 258 | 1. Replaced integer subtraction with addition → KILLED |

## Active mutators

- BOOLEAN_FALSE_RETURN
- BOOLEAN_TRUE_RETURN
- CONDITIONALS_BOUNDARY_MUTATOR
- EMPTY_RETURN_VALUES
- INCREMENTS_MUTATOR
- INVERT_NEGS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- NULL_RETURN_VALUES
- PRIMITIVE_RETURN_VALS_MUTATOR
- VOID_METHOD_CALL_MUTATOR

## Tests examined

- org.usi.sq.util.CircularFifoQueueTest.[engine:junit-jupiter]/[class:org.usi.sq.util.CircularFifoQueueTest]/[method:testIterator()] (11 ms)

Report generated by PIT 1.5.2