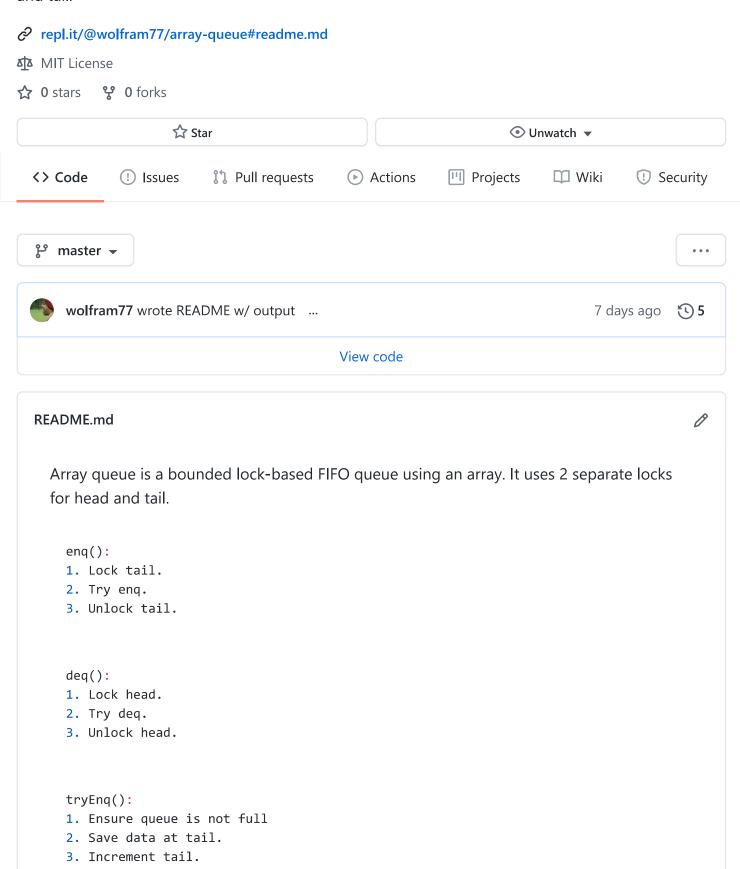
☐ javaf / array-queue

Array queue is a bounded lock-based FIFO queue using an array. It uses 2 separate locks for head and tail.



tryDeq():

```
1. Ensure queue is not empty.
2. Return data at head.
3. Increment head.
## OUTPUT
Starting 10 threads with sequential queue
2: failed eng
5: failed deq
6: failed deq
7: failed deq
8: failed deq
1: failed deq
4: failed deq
9: failed deq
1: dequeued 0/1000 values
2: dequeued 0/1000 values
4: dequeued 0/1000 values
5: dequeued 698/1000 values
6: dequeued 0/1000 values
7: dequeued 0/1000 values
8: dequeued 0/1000 values
9: dequeued 0/1000 values
Was LIFO? false
Starting 10 threads with array queue
Was LIFO? true
```

See ArrayQueue.java for code, Main.java for test, and repl.it for output.

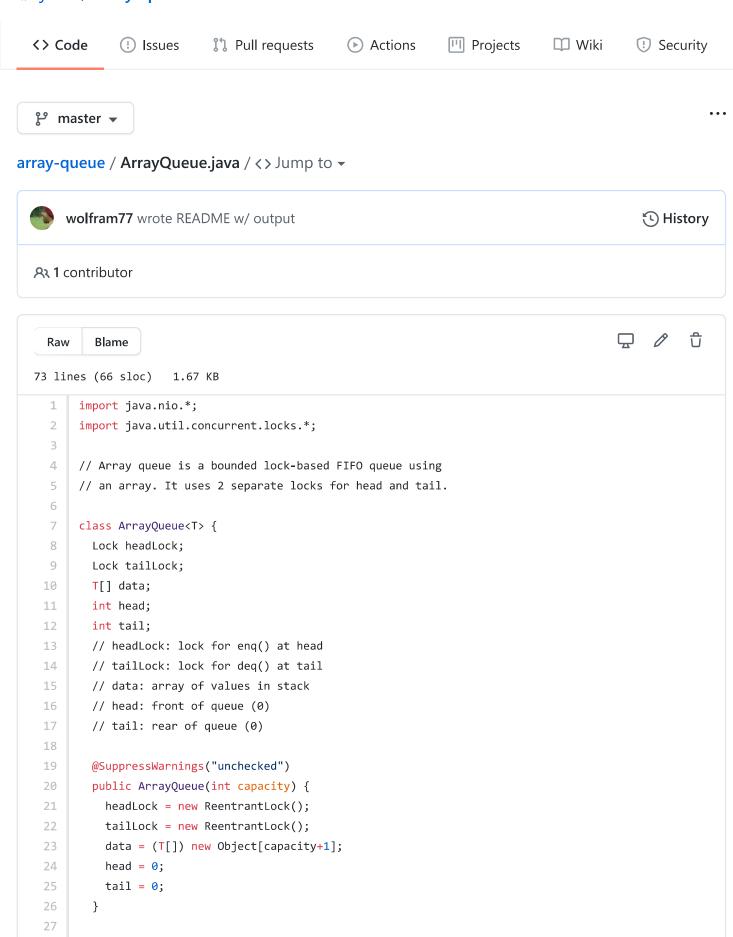
references

• The Art of Multiprocessor Programming :: Maurice Herlihy, Nir Shavit

Languages

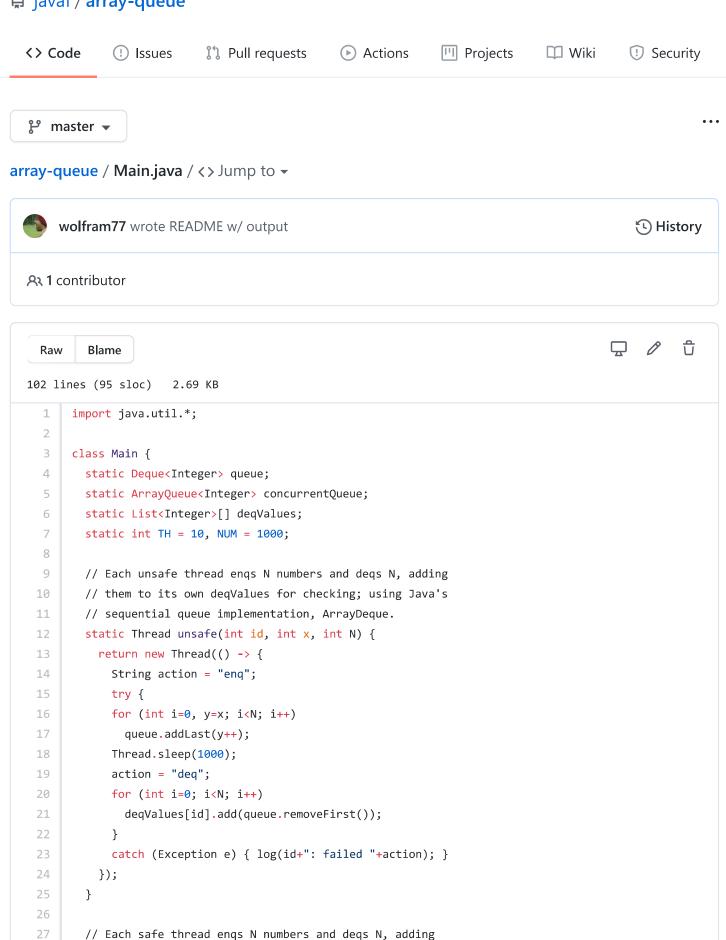
Java 100.0%

☐ javaf / array-queue



```
28
       // 1. Lock tail.
29
       // 2. Try enq.
       // 3. Unlock tail.
30
       public void enq(T x) {
31
         try {
32
         tailLock.lock(); // 1
33
                            // 2
34
         tryEnq(x);
         } finally {
36
         tailLock.unlock(); // 3
37
         }
38
       }
39
40
       // 1. Lock head.
       // 2. Try deq.
41
       // 3. Unlock head.
42
43
       public T deq() {
44
         try {
45
         headLock.lock(); // 1
         return tryDeq(); // 2
46
47
         } finally {
48
         headLock.unlock(); // 3
49
         }
50
       }
51
52
       // 1. Ensure queue is not full
       // 2. Save data at tail.
53
54
       // 3. Increment tail.
55
       protected void tryEnq(T x) {
56
         int tail2 = (tail + 1) % data.length; // 1, 3
57
         if (tail2 == head)
                                                 // 1
           throw new BufferOverflowException(); // 1
58
59
         data[tail] = x; // 2
         tail = tail2; // 3
60
61
       }
62
63
       // 1. Ensure queue is not empty.
64
       // 2. Return data at head.
65
       // 3. Increment head.
66
       protected T tryDeq() {
                                                  // 1
67
         if (head == tail)
68
           throw new BufferUnderflowException(); // 1
69
         T x = data[head];
                                           // 2
         head = (head + 1) % data.length; // 3
70
71
         return x;
                                           // 2
72
       }
73
```

☐ javaf / array-queue



```
28
       // them to its own deqValues for checking; using
29
       // ArrayQueue.
       static Thread safe(int id, int x, int N) {
30
31
         return new Thread(() -> {
32
           String action = "enq";
33
           try {
34
           for (int i=0, y=x; i<N; i++)</pre>
             concurrentQueue.enq(y++);
           Thread.sleep(1000);
37
           action = "deq";
38
           for (int i=0; i<N; i++)</pre>
39
             deqValues[id].add(concurrentQueue.deq());
           }
41
           catch (Exception e) { log(id+": failed "+action);
42
           e.printStackTrace(); }
43
         });
44
       }
45
46
       // Checks if each thread dequeued N values, and they are
47
       // globally unique.
       static boolean wasLIFO(int N) {
48
49
         Set<Integer> set = new HashSet<>();
50
         boolean passed = true;
         for (int i=0; i<TH; i++) {
51
52
           int n = deqValues[i].size();
53
           if (n != N) {
54
             log(i+": dequeued "+n+"/"+N+" values");
             passed = false;
56
           }
57
           for (Integer x : deqValues[i])
58
             if (set.contains(x)) {
59
               log(i+": has duplicate value "+x);
60
               passed = false;
             }
61
           set.addAll(deqValues[i]);
62
         }
63
64
         return passed;
65
       }
       @SuppressWarnings("unchecked")
67
68
       static void testThreads(boolean safe) {
         queue = new ArrayDeque<>();
         concurrentQueue = new ArrayQueue<>(TH*NUM);
70
         deqValues = new List[TH];
71
72
         for (int i=0; i<TH; i++)</pre>
73
           deqValues[i] = new ArrayList<>();
74
         Thread[] threads = new Thread[TH];
75
         for (int i=0; i<TH; i++) {</pre>
```

```
76
            threads[i] = safe?
 77
              safe(i, i*NUM, NUM) :
 78
              unsafe(i, i*NUM, NUM);
 79
            threads[i].start();
          }
 80
          try {
 81
          for (int i=0; i<TH; i++)</pre>
 82
 83
            threads[i].join();
          }
 84
 85
          catch (Exception e) {}
 86
        }
 87
 88
        public static void main(String[] args) {
          log("Starting "+TH+" threads with sequential queue");
 89
 90
          testThreads(false);
 91
          log("Was LIFO? "+wasLIFO(NUM));
 92
          log("");
 93
          log("Starting "+TH+" threads with array queue");
 94
          testThreads(true);
 95
          log("Was LIFO? "+wasLIFO(NUM));
          log("");
 96
97
        }
98
99
        static void log(String x) {
100
          System.out.println(x);
101
        }
102
      }
```