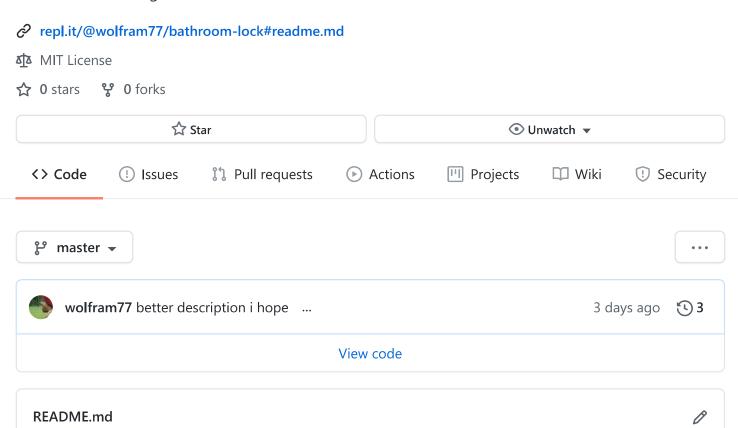
A Bathroom Lock allows N genders (thread types) to access a common bathroom (critical section) such that different genders do not clash.



A Bathroom Lock allows N genders (thread types) to access a common bathroom (critical

section) such that different genders do not clash.

Bathroom Lock allows threads of the same type (gender) to enter at the same time, but disallows different types of threads to occupy the critical section (bathroom) at the same time.

lock():

README.md

- 1. Acquire common lock.
- 2. Wait until there is no other gender.
- 3. Increment my gender count.
- 4. Release common lock.

unlock():

- 1. Acquire common lock.
- 2. Decrement my gender count.
- 3. If my gender cleared, signal others.
- 4. Release common lock.

```
## OUTPUT
Starting 100 unsafe males ...
Starting 100 unsafe females ...
F101: saw 5 males
F108: saw 2 males
F109: saw 2 males
F107: saw 2 males
F108: saw 2 males
F110: saw 2 males
F109: saw 2 males
F101: saw 2 males
F110: saw 2 males
F103: saw 2 males
F107: saw 2 males
F106: saw 2 males
F103: saw 2 males
F104: saw 2 males
F106: saw 2 males
F100: saw 5 males
F104: saw 2 males
F100: saw 2 males
F105: saw 2 males
F102: saw 2 males
M99: saw 2 females
F105: saw 2 males
F102: saw 2 males
F112: saw 1 males
Clashes occurred: 24
Starting 100 safe males ...
Starting 100 safe females ...
Clashes occurred: 0
```

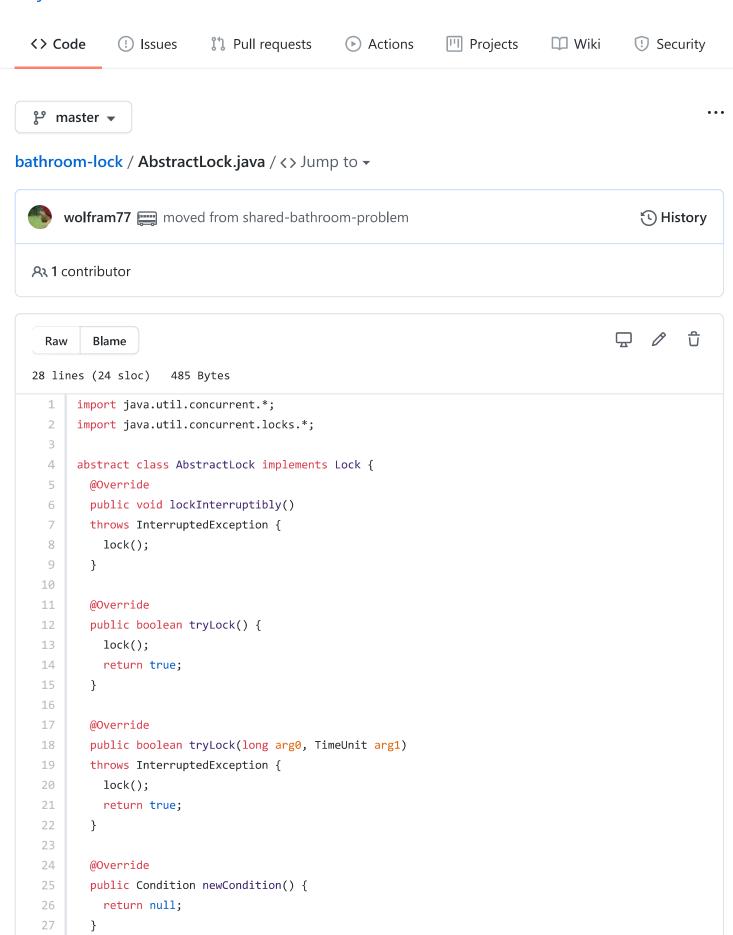
See BathroomLock.java for code, Main.java for test, and repl.it for output.

references

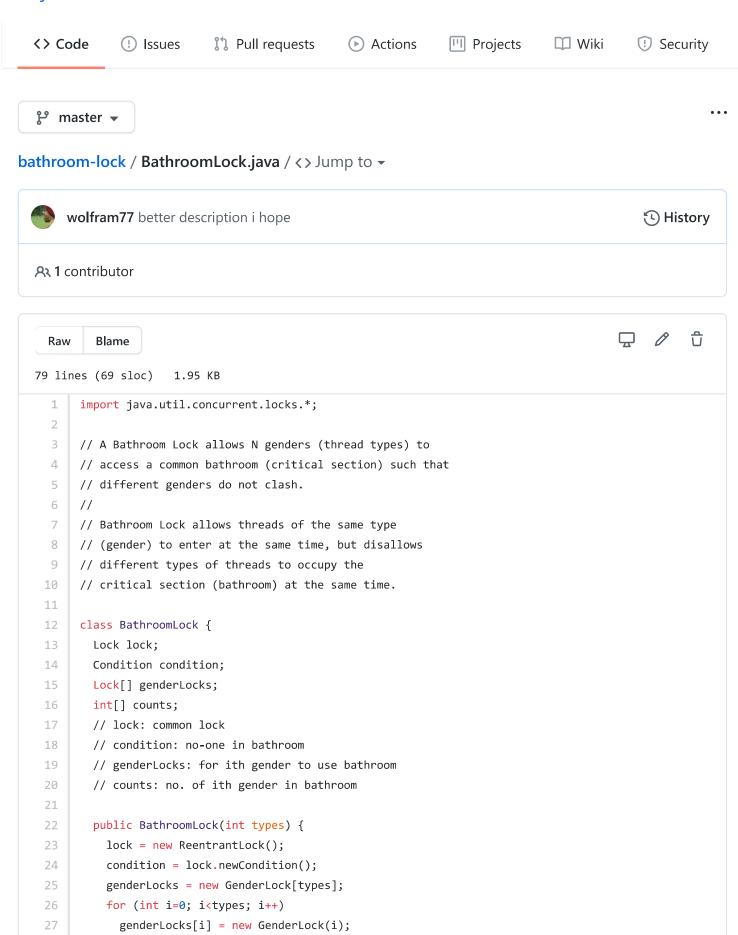
• The Art of Multiprocessor Programming :: Maurice Herlihy, Nir Shavit

Languages

Java 100.0%

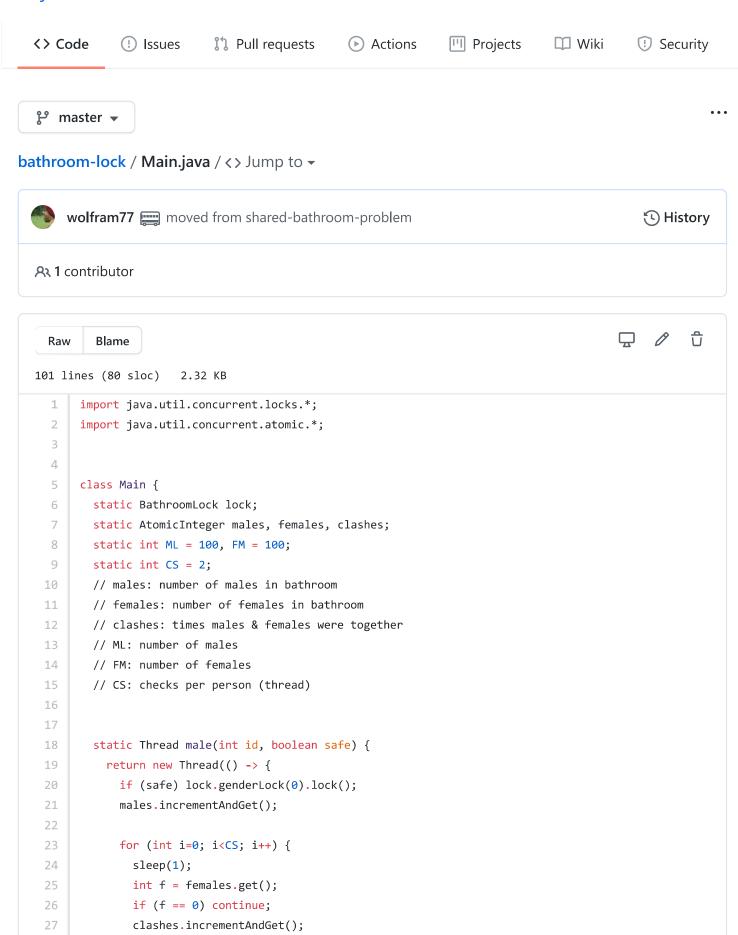


28



```
28
         counts = new int[types];
       }
29
30
31
       public Lock genderLock(int i) {
32
         return genderLocks[i];
33
       }
34
       class GenderLock extends AbstractLock {
37
         int type;
38
         // type: gender type
39
         public GenderLock(int i) {
41
           type = i;
42
         }
43
44
         // 1. Acquire common lock.
45
         // 2. Wait until there is no other gender.
46
         // 3. Increment my gender count.
         // 4. Release common lock.
         @Override
48
49
         public void lock() {
50
           lock.lock(); // 1
51
           try {
52
             while (!oneGender()) // 2
                condition.await(); // 2
54
             counts[type]++; // 3
           }
56
           catch (InterruptedException e) {}
57
           finally { lock.unlock(); } // 4
58
         }
59
60
         // 1. Acquire common lock.
61
         // 2. Decrement my gender count.
         // 3. If my gender cleared, signal others.
62
         // 4. Release common lock.
63
         @Override
64
65
         public void unlock() {
           lock.lock(); // 1
           counts[type]--; // 2
67
68
           if (counts[type] == 0) // 3
             condition.signalAll(); // 3
           lock.unlock(); // 4
70
         }
71
72
73
         private boolean oneGender() {
74
           for (int i=0; i<counts.length; i++)</pre>
75
             if (i!=type && counts[i]>0) return false;
```

```
76 | return true;
77 | }
78 | }
79 | }
```



```
log("M"+id+": saw "+f+" females");
28
           }
29
30
31
           males.decrementAndGet();
32
           if (safe) lock.genderLock(0).unlock();
33
         });
34
       }
37
       static Thread female(int id, boolean safe) {
38
         return new Thread(() -> {
39
           if (safe) lock.genderLock(1).lock();
40
           females.incrementAndGet();
41
42
           for (int i=0; i<CS; i++) {</pre>
43
              sleep(1);
44
              int m = males.get();
              if (m == 0) continue;
45
46
              clashes.incrementAndGet();
              log("F"+id+": saw "+m+" males");
           }
48
49
50
           females.decrementAndGet();
           if (safe) lock.genderLock(1).unlock();
51
52
         });
53
       }
54
56
       // Tests to see if males and females entered
57
       // bathroom separately.
58
       static void testThreads(boolean safe) {
59
         String type = safe? "safe" : "unsafe";
         log("Starting "+ML+" "+type+" males ...");
60
61
         log("Starting "+FM+" "+type+" females ...");
62
63
         males.set(0);
         females.set(0);
64
65
         clashes.set(0);
         Thread[] t = new Thread[ML+FM];
67
68
         for (int i=0; i<ML+FM; i++) {</pre>
           t[i] = i<ML? male(i, safe) : female(i, safe);
70
           t[i].start();
         }
71
72
73
         try {
         for (int i=0; i<ML+FM; i++)</pre>
74
75
           t[i].join();
```

```
76
 77
          catch(InterruptedException e) {}
          log("Clashes occurred: "+clashes.get()+"\n");
 78
 79
        }
 80
 81
 82
        public static void main(String[] args) {
 83
          lock = new BathroomLock(2);
          males = new AtomicInteger(0);
 84
          females = new AtomicInteger(0);
 85
          clashes = new AtomicInteger(0);
 86
 87
          testThreads(false);
 88
          testThreads(true);
 89
 90
        }
 91
 92
 93
        static void sleep(long ms) {
 94
          try { Thread.sleep(ms); }
 95
          catch (InterruptedException e) {}
 96
        }
97
        static void log(String x) {
98
99
          System.out.println(x);
100
        }
101
      }
```