Worksheet: Bounded Blocking Concurrent List using Semaphores

Use the semaphore acquire() and release() to ensure that all threads are able to fairly access the BoundBlocking Concurrent List in addFirst() and remove().

1. public class BoundedBlockingList {
2.     final int capacity;
3.     final ConcurrentLinkedList list = new ConcurrentLinkedList();
4.     final Semaphore sem;
5.     public BoundedBlockingList(int capacity) {
6.         this.capacity = capacity;
7.         sem = new Semaphore(capacity);
8.     }
9.     public void addFirst(Object x) throws InterruptedException {
10.        try { list.addFirst(x); }
11.        catch (Throwable t) { rethrow(t); } // only performed on exception
12.     }
13.     public boolean remove(Object x) {
14.        if (list.remove(x)) { return true; }
15.        return false;
16.     }
17. } // BoundedBlockingList
Worksheet solution: Bounded Blocking Concurrent List using Semaphores

Use the semaphore acquire() and release() to ensure that all threads are able to fairly access the BoundBlocking Concurrent List in addFirst() and remove().

1. public class BoundedBlockingList {
2.   final int capacity;
3.   final ConcurrentLinkedList list = new ConcurrentLinkedList();
4.   final Semaphore sem;
5.   public BoundedBlockingList(int capacity) {
6.       this.capacity = capacity;
7.       sem = new Semaphore(capacity);
8.   }
9.   public void addFirst(Object x) throws InterruptedException {
10.      sem.acquire(); // blocks until a permit is available
11.      try { list.addFirst(x); } catch (Throwable t) { sem.release(); rethrow(t); } // only performed on exception
12.   }
13.   public boolean remove(Object x) {
14.      if (list.remove(x)) { sem.release(); return true; }
15.      return false;
16.   }
17. } // BoundedBlockingList