```
import java.util.concurrent.atomic.AtomicInteger;

public class IQueue<T> {
    private final AtomicInteger head = new AtomicInteger(0);
    private final AtomicInteger tail = new AtomicInteger(0);
    private final Object[] items;

    public IQueue(int capacity) {
        items = new Object[capacity];
    }

    public void enq(T x) throws FullException {
        int slot;
        do {
            slot = tail.get();
            if (slot - head.get() >= items.length) {
                throw new FullException("Queue is full");
            }
        } while (!tail.compareAndSet(slot, slot + 1));
        items[slot % items.length] = x;
    }

    @SuppressWarnings("unchecked")
    public T deq() throws EmptyException {
        T value;
        int slot;
        do {
            slot = head.get();
            if (tail.get() - slot <= 0) {
                throw new EmptyException("Queue is empty");
            }
            value = (T) items[slot % items.length];
        } while (!head.compareAndSet(slot, slot + 1));
        return value;
    }

    public static class FullException extends Exception {
        public FullException(String message) {
            super(message);
        }
    }

    public static class EmptyException extends Exception {
        public EmptyException(String message) {
```

```
        super(message);
      }
    }
  }
```

```
1  class IQueue<T> {
2    AtomicInteger head = new AtomicInteger(0);
3    AtomicInteger tail = new AtomicInteger(0);
4    T[] items = (T[]) new Object[Integer.MAX_VALUE];
5    public void enq(T x) {
6      int slot;
7      do {
8        slot = tail.get();
9      } while (!tail.compareAndSet(slot, slot+1));
10     items[slot] = x;
11   }
12   public T deq() throws EmptyException {
13     T value;
14     int slot;
15     do {
16       slot = head.get();
17       value = items[slot];
18       if (value == null)
19         throw new EmptyException();
20     } while (!head.compareAndSet(slot, slot+1));
21     return value;
22   }
23 }
```

IQueue.java → Not Linearizable