```java
public class DualStack<T> {
  private class Slot {
    boolean full = false;
    volatile T value = null;
  }
  Slot[] stack;
  int capacity;
  private AtomicInteger top = new AtomicInteger(0); // array index
  public DualStack(int myCapacity) {
    capacity = myCapacity;
    stack = (Slot[]) new Object[capacity];
    for (int i = 0; i < capacity; i++) {
      stack[i] = new Slot();
    }
  }
  public void push(T value) throws FullException {
    while (true) {
      int i = top.getAndIncrement();
      if (i > capacity - 1) { // is stack full?
        top.getAndDecrement(); // restore index
        throw new FullException();
      } else if (i >= 0) { // i in range, slot reserved
        stack[i].value = value;
        stack[i].full = true; // push fulfilled
        return;
      }
    }
  }
  public T pop() throws EmptyException {
    while (true) {
      int i = top.getAndDecrement();
      if (i < 0) { // is stack empty?
        top.getAndDecrement() // restore index
        throw new EmptyException();
      } else if (i <= capacity - 1) {
        while (!stack[i].full){};
        T value = stack[i].value;
        stack[i].full = false;
        return value; // pop fulfilled
      }
    }
  }
}
```

FIGURE 11.10 Bob’s problematic dual stack.