```
public class BadCLHLock implements Lock {
  AtomicReference<Qnode> tail = new AtomicReference<QNode>(new QNode());
  ThreadLocal<Qnode> myNode = new ThreadLocal<QNode> {
    protected QNode initialValue() {
      return new QNode();
    }
  };
  public void lock() {
    Qnode qnode = myNode.get();
    qnode.locked = true;     // I'm not done
    // Make me the new tail, and find my predecessor
    Qnode pred = tail.getAndSet(qnode);
    while (pred.locked) {}
  }
  public void unlock() {
    // reuse my node next time
    myNode.get().locked = false;
  }
  static class Qnode { // Queue node inner class
    volatile boolean locked = false;
  }
}
```

FIGURE 7.33 An incorrect attempt to implement a CLHLock.