CollectionTest

In dieser Aufgabe untersuchen Sie das Verhalten der Klasse `java.util.LinkedList` wenn mehrere Threads parallel Elemente einfügen. Den Code dazu finden Sie in der Klasse `CollectionTest` im Paket `worksheet`.

```
private void testCollection(final Collection<Integer> col, final int nThreads) {
    /* 1. Create threads. */
    List<Thread> threads = new ArrayList<Thread>(nThreads);
    for (int n = 0; n < nThreads; n++) {
        Thread inserter = new Thread(){
            public void run() {
                for(int i = 0; i < N_INSERTS / nThreads; i++) {
                    col.add(i);
                }
            }
        };
        threads.add(inserter);
    }
    /* 2. Start threads. */
    for (Thread t : threads) { t.start(); }
    /* 3. Wait for termination. */
    for (Thread t : threads) { t.join(); }
    /* 4. Check inserts. */
    assertEquals(N_INSERTS, col.size());
}
```

Die obige Methode erzeugt `nThreads` Threads, die je `N_INSERTS/nThreads` Elemente in die übergebene Collection einfügen. Am Schluss wird geprüft, ob alle Elemente in der Collection vorhanden sind. Im gegeben Test wird als Collection eine `java.util.LinkedList` verwendet.

Aufgaben:

1. Lassen Sie den Test laufen. Sie werden feststellen, dass nicht alle Elemente in der Collection enthalten sind wenn mehrere Threads gleichzeitig Elemente einfügen.

2. Identifizieren Sie das Problem. Untersuchen Sie dazu die `add` Methode der Klasse LinkedList (unten finden Sie einen Auszug). Notieren Sie genau was schief gehen kann, wenn mehrere Threads diese Methode gleichzeitig aufrufen.

```
public class LinkedList<E> ...
    Node<E> first;
    Node<E> last;

    public boolean add(E e) {
        linkLast(e);
        return true;
    }

    void linkLast(E e) {
        final Node<E> l = last;
        final Node<E> newNode = new Node<>(l, e, null);
        last = newNode;
        if (l == null)
            first = newNode;
        else
            l.next = newNode;
        size++;
        modCount++;
    }
}
```