List Testing (Part 2)

CS 221 Programming Assignment

*“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”*  
– Martin Golding

# Objectives

* Implement a test suite for classes that implement the IndexedUnsortedList interface

# Tasks

In [part 1](https://docs.google.com/document/d/10LZeFle3DBsyoOnUMzv-RsgESrz14IFB1iYs00_NHOA/edit?usp=sharing) of this homework, you created a test plan for classes that implement the [IndexedUnsortedList](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/IndexedUnsortedList.java) interface. For part 2, you will implement at least one third of your plan. This means you will add at least seven of the change scenarios outlined in your test plan (in addition to those given as examples) to class [ListTester](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/ListTester.java). Recommended scenarios to test first are those resulting in empty lists and single-element lists.

You are strongly advised to follow the scenario and test naming conventions demonstrated in the given examples.

* Scenarios are named as StartingState\_Change\_ResultingState. For example, Scenario 6, which begins with list [A] and contains [B,A] after *addToFront(B)*, is identified as A\_addToFrontB\_BA.
* Test names begin with the associated scenario followed by the method being tested. For example, the test for *size()* after scenario 6 is named A\_addToFrontB\_BA\_testSize.

Tests have already been written for change scenarios resulting in empty and single-element lists. You will need to complete the set of tests for scenarios resulting in a two-element list (testTwoElementList()). Notice that there will be more tests for a larger list than for a smaller list. Follow the pattern demonstrated in the given tests, but make sure you are testing all appropriate positions in larger lists.

# Running Your Test Class

Because this is your first test class, it is helpful to have something to test right away (although this is rarely possible in real-world situations). Two implementations of IndexedUnsortedList are provided to illustrate how your test class will handle implementations of different quality. Do not modify either of these classes. They do exactly what they are supposed to do.

* [GoodList](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/GoodList.java) wraps the Java API LinkedList class in the IndexedUnsortedList interface and all of its list methods work correctly. However, the Iterators in the Java API lists do not throw ConcurrentModificationExceptions in all situations that will be required for your IndexedUnsortedLists. When an implementation of an ADT does not pass tests for ADT-defined behaviors, do not modify tests to make the list look good. Tests should always reflect the ADT's defined behavior. It is the burden of the list to satisfy the test.  
  GoodList, therefore, fails some of the tests for Iterator concurrency. We aren't going to edit Java API library code, so we'll have to live with those failures, for now.
* [BadList](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/BadList.java) has stubs - methods containing only enough code to compile and run - for each ADT method. We *expect* it to fail all tests. When you test it with the given ListTester, however, you will see that it passes a surprising number of tests. This example serves as a reminder that individual tests are unlikely to reveal if a method is really working. It is the cumulative results of all tests passing when they should and failing when they should that builds confidence in correct functionality.

# Files

To complete the homework, you will need these files, but you should not modify them:

* [GoodList.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/GoodList.java)
* [BadList.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/BadList.java)
* [IndexedUnsortedList.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/IndexedUnsortedList.java)

You will also need this file, which you will modify to add scenarios and related tests

* [ListTester.java](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/ListTester.java)

All of these files are bundled in this [zip file](https://github.com/BoiseState/CS221-Public/raw/master/assignments/DataStructures/files/lists2.zip).

# Grading

Points will be awarded according to the following breakdown:

| **Tasks** | **Points** |
| --- | --- |
| ListTester implements tests for lists up to 2 elements and adds at least 7 change scenarios from your test plan to the given scenarios | 10 |
| GoodList passes all IndexedUnsortedList tests (though not Iterator concurrency tests) | 5 |
| BadList fails most tests | 5 |

# Required Files

You should submit the following:

* ListTester.java
* *All additional source files needed to compile and run ListTester*

# Submission

Submit all files from the same directory. Do not include any unnecessary files.

Use the submission command given on your section's class web page from the directory containing your files.