Array List

CS 221 Programming Assignment

*“The human animal differs from the lesser primates in his passion for lists.”*  
– H. Allen Smith

# Objectives

* Create an array-based implementation of the IndexedUnsortedList interface, called IUArrayList.
* Expand the ListTester test suite to ensure correct IndexedUnsortedList functionality from IUArrayList.
* Create a fully functional Iterator for IUArrayList class that includes a functional remove() method.

# Tasks

* Create an array-based implementation of the IndexedUnsortedList interface, called IUArrayList.
* Create a fully functional iterator for IUArrayList as a private inner class that implements the Java [Iterator](http://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html) interface.
  + The Iterator interface (sadly) defines a default implementation of the remove() method, but you are required to override it and make it functional in your Iterator.
  + Iterator does not define the result of calling methods after a change has occurred to the list, because fail-fast behavior cannot be guaranteed for all Iterators. Your implementation, however, will be expected to throw a [ConcurrentModificationException](http://docs.oracle.com/javase/8/docs/api/java/util/ConcurrentModificationException.html) if any Iterator method is called after the list has been modified by any source other than the current Iterator.
* Add tests to ListTester to confirm correct IndexedUnsortedList functionality from IUArrayList.
  + Add at least 7 more of the change scenarios from your test plan (total of 14 scenarios in addition to given scenarios). Continue to follow the naming conventions for your tests outlined in the last homework.
  + Be sure to uncomment the lines in the newList method of the ListTester class, so the class can create IUArrayList objects.
  + Complete the set of tests for change scenarios resulting in a 3-element list (testThreeElementList()).
  + IUArrayList should pass all of the tests without any skips.
* Include tests for all list sizes in ListTester to confirm correct Iterator functionality from IUArrayList's iterator.

# Iterator Testing

In order to test the iterator for your IUArrayList, you have to consider not only the state of the list after a given change scenario, but also the state of the iterator itself.

For example, for change scenario A\_addToFrontB\_BA, the resulting state of the list is [B, A]. Given an iterator over this list, we want to run some tests for the three methods required by the Iterator interface: hasNext, next, and remove. But in order to run these tests, we have to know where the iterator is in the list, as well as what other iterator methods have been called to get to that location.

For instance, if we've just instantiated the iterator for this list, one test case would call hasNext and should return true, another test case (after instantiating the iterator again) would call next and should return B, and one last test case (after re-instantiating the iterator) would call remove and should throw an IllegalStateException because it wasn't preceded by a call to next.

Therefore, the methods in your iterator test classes will have to take into account the state of the iterator.

For change scenarios that result in an empty list:

* You should test one iterator state:
  + a newly initialized iterator

For change scenarios that result in a single-element list:

* You should test three iterator states:
  + a newly initialized iterator
  + an iterator after one call to the next() method
  + an iterator after one call to next() immediately followed by a call to remove()

For change scenarios that result in a two-element list:

* You should test six iterator states:
  + a newly initialized iterator
  + an iterator after one call to the next() method
  + an iterator after one call to next() immediately followed by a call to remove()
  + an iterator after two consecutive calls to next()
  + an iterator after two consecutive calls to next() immediately followed by a call to remove()
  + an iterator after one call to next(), a call to remove(), and finally, another call to next()

For change scenarios that result in a three-element list:

* You should test nine iterator states:
  + a newly initialized iterator
  + an iterator after one call to the next() method
  + an iterator after one call to next() immediately followed by a call to remove()
  + an iterator after one call to next(), a call to remove(), and finally, another call to next()
  + an iterator after two consecutive calls to next()
  + an iterator after two consecutive calls to next() immediately followed by a call to remove()
  + an iterator after two calls to next(), a call to remove(), and finally another call to next()
  + an iterator after three consecutive calls to next()
  + an iterator after three consecutive calls to next() followed by a call to remove()

These are tests added to existing change scenarios. For example, scenario A\_addToFrontB\_BA results in a 2-element list. In addition to the tests that confirm all IndexedUnsortedList methods work correctly, you need tests to confirm the list's Iterator operates correctly over the resulting 2-element list.

# New Change Scenarios

With the addition of Iterator's remove() method, we have a new way to modify a list. We need to test IndexedUnsortedList methods after the list has been modified by the Iterator. Changes to the list by an Iterator were not represented in the original 43 change scenarios, so consider adding several from this list:

* [A\_IterRemoveA\_empty](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/ExampleIterRemoveScenario.java)
* AB\_IterRemoveA\_B
* AB\_IterRemoveB\_A
* ABC\_IterRemoveA\_BC
* ABC\_IterRemoveB\_AC
* ABC\_IterRemoveC\_AB

Scenarios 44-49 in this [expanded set of change scenarios](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/ListTestingScenarios.txt) represent these new list change scenarios.

# Iterator Concurrent Modification Testing

Testing for ConcurrentModificationExceptions can be an especially challenging process. To keep your focus on the (many) other change scenarios and tests, a set of tests for concurrency is provided in ListTester and you should not need to write any additional tests for this functionality.

# Files

Build on files from the previous [list testing](https://docs.google.com/document/d/1zOTB8CkSVN4ntanyxlD3xPtcpnt6h6na_fMTDjRpSFM/edit?usp=sharing) assignments.

This [IUArrayList](https://raw.githubusercontent.com/BoiseState/CS221-Public/master/assignments/DataStructures/files/IUArrayList.java) includes the basic class structure and a couple sample methods.

# Grading

Points will be awarded according to the following breakdown:

| **Tasks** | **Points** |
| --- | --- |
| ListTester tests 7 more change scenarios (at least 14 scenarios added to given scenarios) and has tests for 3-element lists | 20 |
| IUArrayList class and Iterator functionality and quality | 20 |

# Required Files

Submit all source files necessary to compile and run ListTester with IUArrayList:

* IUArrayList.java
* ListTester.java
* IndexedUnsortedList.java
* *Any other 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.