COSC-211: Data Structures
HW3: Code testing
Due Sunday, Mar-14, 11:59 pm EDT

1 Code testing `AmhList` implementations

The `AmhList` interface is a simplified version of the standard Java `List` interface—one that nicely matches the basic *List* ADT.

Given implementations of that interface, how should you test whether they are correct? For this assignment, you will be given two **broken** implementations of `AmhList`. That is, they are whole, but they each have bugs. How do you find them?

You will also be given a *test generator* program, and a *tester* program. The former creates a sequence of *List* operations, randomly selected. These sequences are regular text files that you can, if you like, edit by hand.

The latter consumes that sequence, applying each operation to both a *reference list implementation*—a wrapper of the standard Java `LinkedList` class, which we assume to be correct—and a *test list implementation*, which is the class you are hoping to debug. It compares the results of each such operation, catching and outputting warnings on any mismatches between the behavior of the *reference* and *test* lists. As it runs, the *tester* can also display step-by-step information about the content of the lists.

With these tools, your goal will be to find and fix the bugs in these `AmhList` implementations. Of course, the **real** goal will be for you to get familiar with this kind of structured testing code and how to use it to track down subtle bugs.

2 Getting started

Create a new directory/project for yourself, and download the following ZIP archive:

`bit.ly/cosc-211-21s-hw3`

Extract/copy the files in this archive into your new directory/project. You will find the following files:

- `AmhList.java`

  The interface that matches the standard *List* ADT.

- `ListGenerator.java`

The program that creates a sequence of pseudo-randomly chosen *List* operations, along with random (but reasonable!) indices and values, as appropriate.

- `ListTester.java`

  The program that reads in a sequence of *List* operations and then applies them to both a *reference* list and a *test* list, comparing that the results from both are identical.

- `WrapperList.java`

  A wrapper around the standard Java `LinkedList` class, making it conform to the `AmhList` interface. This class serves as our reference standard that we assume to function correctly.

- `AmhArrayList.java`

  An array-based implementation of `AmhList`. It automatically doubles the array size when additional storage is needed. **This class contains bugs.**

- `AmhLinkedList.java`

  A linked-list-based implementation of `AmhList`. **This class contains bugs.**

Using the generator: In order to generate a sequence, you run it like so:[1]

```
$ java ListGenerator 100 42 sequence.txt
```

This command will run the generator program, asking it to create a list of `100` operations, using the arbitrary integer `42` as the *psuedorandom number generator seed*, and emitting the results into a file named `sequence.txt`. You can, of course, choose different values here to generate shorter or longer sequence, using different random seeds, and storing the results into files of whatever names you choose.

You should open `sequence.txt` (or whatever you chose to name it) and see the format of the operations generated by this program.

Using the tester: Once you have a sequence, you can test it like this:

```
$ java ListTester AmhArrayList sequence.txt
```

Here, you are asking it to read the sequence of operations from `sequence.txt`, and to apply those operations to an instance of the `AmhArrayList` class, as well as to an insance

[1] We assume, here, the ability to use *command-line arguments*. If you run Java from within an IDE, you may need to find how to pass these arguments within that environment.

of our reference class, `WrapperList.java`. As it runs, it will display copies of the list after each operation for your visual inspection, detecting any discrepencies. (See the method `ListTester.compare()` to see how that information is produced. You can comment out the call to this method in `ListTester.go()` if you want to see less output.

Additionally, if any operations cause the test and reference lists to produce different behaviors, the tester program will emit information about that difference.

3 Your assignment

Use the generator and tester to find the bugs in `AmhArrayList` and `AmhLinkedList`. Notice that not all of the bugs are ones that will manifest directly! That is, the failures that these classes will display, when tested, may reveal only indirectly the true cause of the error.

You should expect to add debugging output to these list classes, allow you to hone in on the true bugs. Use the repeatability of testing infrastructure to show yourself what is happening within the list code, and then reason about how to fix it!

IMPORTANT NOTE: Once you have found the bugs and corrected them, **you should remove your debugging output.** Always submit code that is ready to be used; don't leave it looking or functioning like a work in progress.

4 How to submit your work

Go to GradeScope for our course, where you can submit your work. It will be auto-tested, and you will see whether it *compiles* and *runs* successfully. Again, if the run fails, it won't tell you why; you need to go back and do more testing yourself. You may submit early and often!

Notice that **you should only submit** `AmhArrayList.java` and `AmhLinkedList.java`. The tester code should **not** be submitted; it exists only to help you debug the code that you are submitting.

This assignment is due on Sunday, Mar-14, 11:59 pm EST.