CS1760: Multiprocessor Synchronization **Due: 01 November 2018, 12:59 PM**

Professor: Maurice Herlihy *HTA: Jonathan Lister*

Programming Assignment 3 - DupLists

Extend the `OptimisticList`, `LazyList`, and `LockFreeList` list algorithms described in pages 205, 208, and 213 of the book so that they represent multisets (lists that can contain duplicates). The stencil code you'll need to begin this problem can be found at `/course/cs1760/stencil/duplists`, and can be installed with

```
cs1760_install duplists
```

Since the point of this assignment is to introduce you to the sort of design patterns that arise when creating lock/wait-free implementations of an object's methods, the stencil code that we give you already consists of interface definitions. You are expected to provide the method definitions in the various `List` classes that implement the interfaces, and then fill in the tester class so that you can test your program.

Once you copy the stencil code into your directory, you'll be ready to go! The files you need to modify are:

```
LazyDupList.java
LockFreeDupList.java
OptimisticDupList.java
ListTester.java
```

Note that a portion of this assignment's total score is reserved for testing. Passing basic functionality tests, located in `BasicTestLists.java` and `BasicListTester.java`, will reward you with some points; however, we also expect you to write any additional tests you deem necessary.
Failing basic functionality tests does not necessarily imply that you will receive no credit for the assignment. At the same time, the course staff will be unable to conduct a rigorous inspection of non-functional code to award partial credit.

To hand in your code for this assignment, run

```
cs1760_handin duplists
```

Note: **the handin script expects your files to be found in `~/course/cs1760/duplists/`. If your code is not in that folder, you will have a blank handin and get 0 points.**

Here are some tips to get you started on the assignment:

1. Most of the code you will need for this assignment will be from the book, so feel free to use it!

2. You must use sentinels in your implementations of these lists. This is required to pass the minimal functionality tests and not including them will probably cause the TA test suite to break as well.

3. For clarity, here are some specification details. Remember that more than one instance of an item can exist in a multiset.
    a. `add(T item)` should increase the number of instances *n* of `item` in the list by one, returning `True`.
    b. `remove(T item)` should return `False` if *n* == 0. Otherwise (for *n* >= 1) decrease the number of instances of `item` in the list by one and return `True`.
    c. `contains(T item)` should return `True` for *n* >= 1, where *n* is the number of instances of `item` in the list. It should return `False` otherwise.
    d. You are not required to be able to count the number of instances of `item` in the list. You can support this feature if you really want to but the TAs won't give you any extra points.