YSC4231: Parallel, Concurrent and Distributed Programming

Programming Assignment 4

In this assignment, you will be enhancing two implementations of concurrent lists that we studied recently: `OptimisticList` and `LazyList`. Please, extend them so that

1. they can represent multisets (lists that can contain duplicates), and
2. can correctly handle cases when different objects have the same hash code (so-called hash-collisions). Remember that the implementations from the lectures assumed that hashes are *always distinct*.

The template repository for this assignment is available on Canvas. Once you clone the template repository, you'll be ready to go! The files you need to modify are:

- `OptimisticDupList.scala`
- `LazyDupList.scala`
- `MultiSetTests.scala`
- `HashCollistionTests.scala`

As before, portion of this assignment's total score is reserved for additional testing (templates for tests are provided). Passing basic functionality tests, invoked in `*Tests.scala` files, will reward you with some points; however you are also expected to write any additional tests you deem necessary. For instance, at the moment there are no tests that validate the multi-set functionality and the behaviour in the presence of has collisions. To hand in your code for this assignment, submit the link to the tagged release in your repository on GitHub. Feel free to provide additional comments about your implementation and discoveries to the `README.md` file in the root of the project.

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

- Most of the code you will need for this assignment is available in the main repository for the class, so feel free to reuse it.
- You must use sentinels in your implementations of these lists. This is required to pass the minimal functionality tests.
- Here are some specification details. Remember that more than one instance of an item can exist in a multiset.

    1. `add(item)` should increase the number of instances `n` of item in the list by one.
    2. `remove(item)` should return `false` if `n == 0`. Otherwise decrease the number of instances of itemin the list by one and return `true`.
    3. `contains(item)` should return `true` for `n >=1`, where `n` is the number of instances of item in the list. It should return `false` otherwise.
    4. `count(item)` should return the number of instances of `item` currently in the list.

- It is not trivial to come up with two different objects that have the same hashcode. Therefore, in order to test the collision-freedom in `HashCollistionTests`, please, implement your own class `MyClass`, which defines its own way to compute hash code of an object. Since this is only for the purpose of testing, you don't have to ensure Java's contract between `equals` nad `hashCode` in this case.[1]

[1] https://dzone.com/articles/working-with-hashcode-and-equals-in-java