Homework #4
due Monday, February 18, 2019, 10:00pm

In this assignment we continue our investigation of implementing "sequence" ADTs. This week, it will be re-implemented using a linked-list. Read Chapter 4 carefully and especially make sure that you read Section 4.5, pages 232–238 (225–231, 3rd ed.) which specifically say how to implement Sequence with linked lists.

1 Concerning New Implementation of HexTileSeq

The HexTileSeq class will have the same public declarations as the HexTileSeq you implemented before (except no way to specify an initial capacity), but the data structure (private fields) will be completely different. Declare a node class as a "`private static class`" inside the HexTileSeq class. Such a class (one declared inside another class) is called a "nested" class. The node class should have two fields: the data (of type `HexTile`) and the next node. It should have a constructor but no other methods. Despite what the textbook recommends, do not write methods in the "Node" class.

We will use the textbook's recommended design starting on page 232 (225, 3rd ed.) for the fields of the HexTileSeq class. Unlike in the past assignment, `clone` requires some substantial work for you to do: the list must be copied, cell by cell, and `precursor`, `cursor` and `tail` pointers of the clone made to point to the appropriate copied nodes.

1.1 Meaning of the fields

Please read the textbook's section "Revised Sequence ADT—Design Suggestion" for information on the fields. There is a slight difference with the "precursor" pointer. The "precursor" field records the node *before* the current point in the sequence. If we are at the *start* of the sequence, then the precursor is null, since there is no node before the first one. In later assignments, we will see how using a "dummy" node can simplify the logic, avoiding this special case. If there is no current element, the precursor points to the *last* node (unlike in the textbook where it points to null).

1.2 The Invariant

The invariant is more complex than in the previous implementation. It has the following parts:

1. The list may not include a cycle, where the `next` link of some node points back to some earlier node.
2. The field `manyNodes` should accurately represent the number of nodes in the list.
3. The `tail` pointer points to the last node in the list (if any).
4. The precursor field is either null (in which case, the cursor must be the head) or points to a node in the list (in which case, the cursor must be the next node, if any, in the list). It cannot point to a node no longer in the list—the node must be reachable from the head of the list.

We have implemented the first part for you; you should implement the other parts yourself. You should do this early on in developing the implementation—it will help you catch bugs as quickly as possible. We provide code to test the invariant checker.

Be very careful that you *never* change any fields in the invariant checker, `wellFormed`. It is *only* supposed to check the invariant and return true or false (with a report). It should never change things.

2 Files

The repository `homework4.git` includes the following files:

src/TestEfficiency.java Efficiency test (again). It should only take a few seconds.

src/TestHexTileSeq.java Updated test driver.

src/TestInvariantChecker.java Call out to the invariant checker tests.

src/edu/uwm/cs351/HexTileSeq.java Skeleton file.

lib/homework4.jar JAR file containing the other ADTs.

The random tester is available in the JAR file, as before.