Homework # 6
due Monday, October 14, 10:00 PM

This assignment will focus on the concepts of generics, dummy nodes, and more collection methods.

1 The Data Structure

For this homework, you will implement a *generic* implementation of the standard collection interface using a *cyclic doubly-linked list with a dummy node*. The list contains a *dummy* node even when it represents an empty collection. This dummy node has an uninteresting data value (a reference to itself). At the start, when the collection is empty, the `next` and `prev` fields of the dummy node will point back to the dummy node. After elements are added, the dummy node's `next` pointer refers to the first (non-dummy) node in the list and the `prev` pointer refers to the last node in the list.

The main list class is required to have a pointer which points to the dummy node, as well as a field tracking the number of items in the list. You will have to write the invariant checker. The dummy node will always be the next of the last node and the prev of the first.

Using a dummy node means that our data structure will not need to use null pointers (except when there are null elements), which makes the implementation simpler. In particular, the precursor for the iterator (see below) will never be null, nor will any `next` or `prev` links.

2 Generics

In some earlier programs, you may have *used* generic classes. This week you will be *creating* your own generic class, LinkedCollection. The purpose of this is to familiarize you with making classes that are not limited to a specific element type. Inside the generic class, you can use type parameter `E` everywhere where an explicit type was used (e.g., `Painting` in Homework # 4). Furthermore, since the `Node` class is also generic, you must create instances of Node using this parameter as in `new Node<E>(...)`.

3 What You Need To Do

You need to implement `LinkedCollection`, a generic cyclic doubly-linked list with a dummy node implementation of Collection *without* extending `AbstractCollection`. You will have to override every required method required by `Collection`, except for two methods which we have given you, which are not supported and simply throw an exception.

Because we are not extending `AbstractCollection` this time, you will not be able to use any default implementations. You may still use your iterator in many of your methods. This is what `AbstractCollection` does, and you may do the same! There are also some new methods you have not implemented before:

addAll(Collection<? extends E> c)) Add all elements in c to this collection. Return true if anything was added.

containsAll(Collection<?> c) Return true if every element in c is found in this collection, and false otherwise.

removeAll(Collection<?> c) Remove every element from this collection which is found in c. Return true if anything is removed.

toArray() Return an array of `Objects` containing the contents of this collection.

Your class should use the same style as in the previous homework assignments: it should assert the correctness of the invariant at the beginning and end of any method that could change the state of the data structure, and at the beginning of every public method. The invariant should also be checked at the end of the constructor and when the iterator invariant is checked.

Remember to update version numbers whenever the data structure is actually changed.

4 Design of the Iterator

There are many possible designs for the iterator. Here is the one to be used in this homework assignment: the iterator keeps one pointer `precursor` and a boolean `hasCurrent`. You will also need a copy of the version. Do *not* add any more fields. When there is a current element, the precursor will point to its previous node. When there is no current element, the precursor's next will be the node holding the next element that needs to be iterated over. So when the iterator is first created, the precursor's next should be pointing to the first node. (So which node is the precursor pointing to?) If the next node is the dummy, there are no more elements.

5 Files

The directory `homework6.git` repository contains the following files:

src/TestCollection.java This driver tests the collection operations.

src/TestInternals.java Test driver for invariants of list and iterator.

src/TestEfficiency.java Test driver that runs with larger data sets.

src/edu/uwm/cs351/LinkedCollection.java This is the skeleton file you should work with.