CS151:Spring 2023

Lab 6

Lab exercises due by Assignment 5 due date. Note that this lab will count towards a portion of A5 credit because it is more substantial

1. Download `List.java` and `ArrayList.java` from `~dxu/handouts/labs/06`. As we covered in class, the inner class `ArrayIterator` lets us create iterators over elements of the `ArrayList`. Specifically, the method `iterator()` creates and returns an `Iterator`. Study this class before you continue. Write code to test the iterator.

2. Copy `MyIterator.java` and add a second iterator class within the `ArrayList` class called `MyListIterator` that implements `MyIterator`. This iterator will be used to traverse the list in both forward and backward direction. `next()`, `hasNext()` methods are the same as in `ArrayIterator` and `remove()` is very similiar. But it has additional methods as specified below:

   - `boolean hasNext()`: Returns true if this list iterator has more elements when traversing the list in the forward direction. (In other words, returns `true` if `next()` would return an element rather than throwing an exception.)
   - `E next()`: Returns the next element in the list and advances the cursor position. It throws `NoSuchElementException` if the iteration has no next element. This method may be called repeatedly to iterate through the list, or intermixed with calls to `previous()` to go back and forth. (Note that alternating calls to `next` and `previous` will return the same element repeatedly.)
   - `boolean hasPrevious()`: Returns true if this list iterator has more elements when traversing the list in the backward direction. (In other words, returns true if `previous()` would return an element rather than throwing an exception.)
   - `E previous()`: Returns the previous element in the list and moves the cursor position backwards. It throws `NoSuchElementException` if the iteration has no previous element. This method may be called repeatedly to iterate through the list backwards, or intermixed with calls to `next()` to go back and forth. (Note that alternating calls to `next` and `previous` will return the same element repeatedly.)
   - `void remove()`: Removes from the list the last element that was returned by `next()` or `previous()`. This call can only be made once per call to `next` or `previous`. This method throws `IllegalStateException` if neither `next` nor `previous` have been called, or `remove` has already been called after the last call to `next` or `previous`. Refer to the `remove()` method in `ArrayIterator` which is very similiar.
   - `void set(E e)`: Replaces the last element returned by `next()` or `previous()` with the specified element. This call can be made only if `remove()` has not been called after the last call to `next` or `previous`. This method throws `IllegalStateException` if neither `next` nor `previous` have been called, or `remove` has been called after the last call to `next` or `previous`.

3. Add in your `ArrayList` class two methods `myListIterator()` and `myListIterator(int i)` that will make an instance of `MyListIterator` and position it at the beginning of the list, and position it right before the i-th index, respectively. These methods are similar to the `iterator()` method. Note that indices start with 0.

4. Write a driver class which creates an `ArrayList` of Integers, and creates a `MyListIterator` for it. Test ALL the methods in `MyListIterator`.

5. Now, in your driver, create two `ArrayList<Integer>` objects, called `L` and `P`. Populate these lists with integers, `L` need not be sorted but `P` should contain integers sorted in ascending order (just initialize it that way, no need to sort). Now implement a method `removePositions(ArrayList L, ArrayList P)` which removes the elements in `L` that are in positions specified by `P`. Positions start with 0. For instance, if `L` contains $[3, 10, 8, 5, 12, 67, 25, 22]$ and `P` contains $[1, 3, 4, 6]$, the elements at index 1, 3, 4 and 6 in `L` should be removed, resulting in `L`=$[3, 8, 67, 22]$. You must use your `MyListIterator` to implement `removePositions` - this is not because of purely pedagogical reasons. Removing multiple items from an `ArrayList` is tricky because of the index shifting. Using an iterator will make your life a lot easier. You may assume `P` only contains valid indices of `L`, sorted in ascending order.