CS 220 Software Design II | Spring 2022

In-class Exercises 06

University of Wisconsin - La Crosse | *Date: March 11*

1. Given the `SinglyLinkedList` class on the attached Reference Classes page, write the instance method `removeAll` described below (this method will be located in the `SinglyLinkedList` class). Do not use the remove methods you could otherwise assume are available for this class!

```
/**
 * Removes all instances of the given value from the list and returns true.
 * If the value is not in the list, the list will remain unchanged and should
 * return false.
 * @param value The value of the generic type E to remove
 * @return true if one or more values were removed, false if nothing was
 *         removed
 */
```

Solution:

```
public boolean removeAll(E value) {
    if(firstNode == null) {
        return false;
    }

    boolean flag = false;

    while(firstNode != null && firstNode.value.equals(value)) {
        firstNode = firstNode.nextNode;
        flag = true;
    }

    SingleLinkNode curNode = firstNode;
    while(curNode.nextNode != null) {
        if(curNode.nextNode.value.equals(value)) {
            curNode.nextNode = curNode.nextNode.nextNode;
            flag = true;
        }

        curNode = curNode.nextNode;
    }

    return flag;
}
```

2. Given the `SinglyLinkedList` class on the attached Reference Classes page, write the instance method `lastIndexOf` described below (this method will be located in the `SinglyLinkedList` class).

```
/**
 * Returns the last index of the target in the list, or -1 if not found.
 * @param target The value of type E to find
 * @return The last index (int) of the target in the list, or -1 if not found
 */
```

Solution:

```
public int lastIndexOf(int target) {
    SingleLinkNode pos = head.next;
    int index = -1;
    int i = 0;
    while (pos != null) {
        if (pos.data == target) {
            index = i;
        }
        ++i;
        pos = pos.next;
    }
    return index;
}
```

3. Implement a `public` method that has a return type of `SinglyLinkedList<E>` called `split` which takes in a given index and splits the list at that index, removing and returning the sublist created by starting at the index through the end of the list (somewhat similar to `substring(int)`). The new sublist must have at least one element in it; an empty sublist means an invalid index was given. Your method should modify the size attribute as appropriate for the current list, and should throw an `IndexOutOfBoundsException` if required. For example, consider an `SinglyLinkedList` storing the values `[1, 2, 3, null]`. Calling `split` at index 3 would be invalid, at index 2 would result in the original list looking like `[1, 2, null, null]` and returning `[3]`, and at index 0 would result in the original list looking like `[null, null, null, null]` and returning `[1, 2, 3]`. Note that the portrayed sublists returned might have additional `null` values depending on how the list grows.

```
/**
 * Splits and returns a new list starting at the given index through the end
   of the list.
 * @param index An int representing the index to split at
 * @return a new SinglyLinkedList<E>
 */
```

Solution:

```
public SinglyLinkedList<E> split(int index) {
    if(index < 0 || index >= size) {
        throw new IndexOutOfBoundsException();
    }

    SinglyLinkedList<E> toReturn = new SinglyLinkedList<>();

    SingleLinkNode curNode = firstNode.nextNode;
    for(int i = 0; i < index-1; ++i) {
        curNode = curNode.nextNode;
    }

    toReturn.firstNode.nextNode = curNode.nextNode;
    curNode.nextNode = null;
    toReturn.size = size-index;
    size = index;

    return toReturn;
}
```

Reference Classes

```
1  public class SinglyLinkedList<E> {
2      private int size;
3      private SingleListNode firstNode;
4
5      public SinglyLinkedList() {
6          size = 0;
7          // assumes the use of a sentinel node
8          firstNode = new SingleListNode(null);
9      }
10
11     private class SingleListNode {
12         private E data;
13         private SingleListNode nextNode;
14
15         public SingleListNode(E i) {
16             data = i;
17             nextNode = null;
18         }
19     }
20 }
```

4. Conceptually, why does it make sense to create the `ListNode` class as an inner class to the `LinkedList` class?

Solution: Nodes should only be used internally be the list, not externally by the programmer.

5. We often talk about data structures as the intersection of an *interface* and an *implementation*. Define these two terms.

Solution: *interface*: describes how the data structure behaves
implementation: how the data structure stores data

6. Implement the `public void clear` method for the `SinglyLinkedList` class on the previous page.

```
/**
 * Clears the data from the list.
 * @return void
 */
```

Solution:

```
public void clear() {
    firstNode = null;
    size = 0;
}
```