Ian Kaiserman

4-19-2020

CSCD-300

HW2 Progress Report

HW2 Progress Report

Documentation for all logical ideas for methods in HW2

- void addLast(Object data)
    - Check if the size of the list is 0. If so, call the addFirst method with the data passed in
    - Else, Create Node reference cur assigned to the prev of head, and new Node nn with passed in data, cur for the prev reference, and the head for the next reference
        - Reassign the prev of head to nn
        - Reassign the next of cur to nn
        - Increase size by 1
- CDoublyLinkedList subListOfSmallerValues(Comparable data)
    - Create Node reference cur assigned to the next of head
    - Create new empty CDoublyLinkedList called res
    - While cur doesn’t equal head
        - Cast the data of cur to a Comparable variable curData
        - If curData compared to the passed in data is less than 0, have res call addLast passing in the data of cur
        - Reassign cur to the next of cur to continue while loop
    - Return res once the method has reached the end

- boolean removeStartingAtBack(Object dataToRemove)
    - Create Node references cur assigned to the prev of head, and next assigned to head
    - While cur doesn't equal head
        - Check if data of cur OR passed in data are null
            - If true, and if they're both null, assign the prev of next to the prev of cur, the next of prev of cur to next (push it in before cur), and return true (this process removes backwards)
            - If they're not both null, skip to next if statement
        - Else, if the data of cur is equal to the data passed in, complete the same switching process above to push the removed object out of between next and cur, and return true
        - Reassign next to cur and cur to the prev of cur (this iterates backwards)
    - Return false if no removal processes went through
- int lastIndexOf(Object o)
    - create integer index set to -1 (default if nothing gets pointed to)
    - Create Node reference cur assigned to next of head
    - For loop, continuing condition is cur doesn't equal head
        - Check if the data of cur OR the passed in object is null
            - If they're BOTH null, reassign index to the i value of the for loop
            - if they're not both null, skip to next if statement
        - else, if the data of cur equals the passed in object, reassign index to the i value of the for loop

        - reassign cur to the next of cur (iterate)
    - return the index integer
- boolean retainAll(CDoublyLinkedList other)
    - If the list passed in is null, throw a nullPointerException
    - Create new empty CDoublyLinkedList called res
    - Create Node reference cur assigned to next of head
        - If the passed in list calling contains passing in the data of cur returns true
            - Have res call addLast passing in the data of cur
        - Reassign cur to the next of cur
    - If the size of res and the size of the list calling this method are equal, meaning nothing was removed, return false
    - Else, reassign this head to the head of res, effectively replacing this entire list, and return true
- boolean contains(Object o) // HELPER METHOD TO retainAll
    - Create Node reference cur assigned to the next of head
    - While cur doesn't equal head
        - Check if neither the data of cur nor the data passed in are null
            - If true, and if the data of cur equals the data passed in, return true
        - If both the data of cur AND the data passed in are null, return true
        - Reassign cur to the next of cur
    - Return false if nothing else fired
- void insertionSort()

- **(Note: I used different variable names than shown in the lectures because I wanted to use what I'm used to when doing sort methods in previous classes, it makes more sense to me this way)**
- Create Nodes start and search (not initialized)
- Create Comparable object cur (not initialized)
- For loop, assigns start to the next of head, loops while start doesn't equal the prev of head (last item of list), reassigns start to the next of start each loop through
    - Assign cur to the data of the next of start, cast to Comparable
    - For loop, assigns search to start, loops while search doesn't equal head AND while the data of search (cast to Comparable) compared to cur returns greater than 0, meaning search is larger than start; reassign search to the prev of search
        - Reassign the data of next of search to the data of search itself (this loops to find the next largest piece after start)
    - After previous for loop, reassign the data of next of search to cur, replacing the lost object piece that needed to be sorted