Homework 2 Progress

Luke Mattfeld
CSCD300
04/20/18

```
public void addLast(Object data) {
    1) Create a new Node with the passed in data, this.head.prev as it's predecessor,
       and this.head as it's successor;
    2) Set this.head.prev.next to the new node
    3) Set this.head.prev to the new node.
    4) Increase the size of the list by one.
}
```

```
public CDoublyLinkedList subListOfSmallerValues(Comparable data) {
    1) Create a CDLL called small to hold the smaller values
    2) Create a node cur set to this.head.next to walk down the list
    3) Create a while loop that exits when cur = this.head (cur has gone through the
       entire list)
    4) In the while loop, cast cur.data to comparable and use the compareTo() method
       check weather the element is smaller than the passed in data. If it is smaller, add
       it to the end of the small CDLL. No matter the data comparison, cur is advanced
    5) Once out of the loop the CDLL small has all of the smaller data elements, and is
       then returned.
}
```

```
public boolean removeStartingAtBack(Object dataToRemove) {
    1) Create a node cur to walk the list, and set it to this.head.prev (since it's starting at
       the "end" of the list)
    2) Create a while loop that terminates once cur = this.head (gone through the entire
       list).
    3) Inside the while loop, check to see if cur's data is equal to the passed in
       dataToRemove. If it is, cur will be cut out (cur.prev.next = cur.next; cur.next.prev
       = cur.prev), the size of the list reduced by one, and the method will return true. If
       the data is not equal, cur will continue towards the beginning of the list
    4) If the while loop exits without returning true, the method will return false, as the
       list does not contain the given data.
}
```

Homework 2 Progress

Luke Mattfeld
CSCD300
04/20/18

public int lastIndexOf(Object o) {

1) Create a node cur set to this.head.prev (since we're going backwards). Create an integer index set to this.size - 1 (since arrays/lists start at 0 ;)
2) Create a while loop that terminates if cur = this.head. In the while loop:
    a) Check if cur.data equals the object passed in. If so, return the integer index.
    b) If cur's data does not match, cur is advanced and index is decremented
3) If the while loop exits, the list does not contain the object, so -1 is returned

}

public boolean retainAll(CDoublyLinkedList other) throws NullPointerException {

1) Create a Node cur = this.head.next to search the list. Also create a boolean value to return if any deletes have taken place
2) Create a while loop that runs as long as cur != this.head. In the loop:
    a) Check if the passed in list (other) contains the current object. It will be helpful to create a private contains(Object o) helper method to check this. Briefly, the helper method would:
        i) Advance cur down the list, and if it happens upon the given object in the list, it will return true. If it reaches this.head again, it returns false.
    b) If the list does not contain the object, the node is removed (cur.prev.next = cur.next; cur.next.prev = cur.prev;) and if the removed boolean has not been set to true, it is done now.
    c) Regardless of the previous if statement, cur is advanced
3) When the while loop exits, only the items also contained in other remain. The "check-for-removed" boolean is returned.

}

Homework 2 Progress

Luke Mattfeld
CSCD300
04/20/18

```
public void insertionSort() {
    1) Create a node cur to traverse the list forward, and a node curBack back to traverse backwards.
    2) Set curBack to this.head.next and cur to curBack.next (since the first element in the list is already “sorted”).
    3) Create a while loop that exits when cur = this.head (so that the loop ends after cur compares the last element of the list).
    4) Inside the while loop:
        a) Create a nested while loop that moves curBack towards the beginning as long as curBack != this.head and as long as curBack’s data is greater than that of cur.
        b) Once the nested while loop exits, curBack is on the element of the list before where we want to insert cur (either the node with data smaller than cur, or this.head).
        c) Create a new node nn with cur’s data, curBack as it’s predecessor, and curBack.next as it’s successor.
        d) Then set curBack.next.prev =nn and cur.next = nn; (wiring in the new node).
        e) Cut out the old cur by setting cur.prev.next = cur.next && cur.next.prev = cur.prev.
        f) Set cur to cur.next, and set curBack to cur.prev.
    5) Once the loop ends, the list is sorted
}
```