Homework 1 Progress

Luke Mattfeld
CSCD 300
04/10/2018

```
// Remove the first object in the list
public Object removeFirst() throws Exception {
    1) Create a node Cur and set it to the first item in the list (after the dummy node)
    2) Set this.head.next to cur.next, cutting out the first non-dummy element.
    3) Set cur.next to null, isolating the removed element.
    3) Decrease the lists size and return cur's data (the data of the element removed)
}

// Search the list for a specific object
public boolean contains(Object o) {
    1) Create a node cur, and assign it the first non-dummy element in the list
    2) Use a for loop to go through every element in the list by making cur = cur.next
    3) First check if each element is null, since the equals method won't work on null objects
    4) If the element is null and the searched for element is null, return true;
    5) If the element is not null, use the equals method to compare the current node to the given
        object
    6) If the two objects are equal return true. If not, advance cur.
    7) If the for loop exits, the list does not contain the given object, and the method returns false.
}

// Removes the first occurrence of the specified element o from this list.
public boolean remove(Object o) {
    1) Use the contains method to check if the list contains the object to be deleted. If the object is
        not in the list, return false;
    2) Create nodes cur (set to head.next) and prev(set to head);
    3) Iterate through the entire list with a loop by assigning prev to cur, and cur to cur.next,
        stopping at this.size;
    4) Check if each element is equal to the passed in object.
    4a) If the element is equal, set prev.next to cur.next (cutting out the element), reduce the list
        size by one, and return true
    4b) If the element is not equal, advance cur & prev
}

// Removes all copies of o from this linked list.
public boolean removeAllCopies( Object o ) { //passed test
    1) Create a while loops that runs as long as the list contains object o, which is checked using
        the contains method;
    2) Every time the while loop runs, it uses the remove(object) method to remove the first
        occurance of object o;
    3) Use a counter variable to keep track of if an object was removed;
}
```

```
    // Insert data elements from linkedlist A and B alternately into
public static MyLinkedList interleave(MyLinkedList A, MyLinkedList B) {
    1) Create an int called Big to hold the size of the bigger list. Use a if/else statement to set this
        value;
    2) Create a new LinkedList called ret that will hold the interleaved values of A and B;
    3) Create a for loop that starts at 0 and ends at the value of the integer big created above;
    4) Every time the loop runs, it checks if the loop variable is within the size of lists A and B. If it
        is, it will add that element of each list to the new list ret. This way, if one list is longer
        than another, once the loop variable gets larger than the small list, it will only add the
        elements of the longer list;
    5) Return the new interleaved list ret;
}

    // Inserts the specified element at the specified position in this list.
public void add(int index, Object o) {
    1) Create Nodes cur (set to this.head.next) and prev (set to this.head);
    2) Create a for loop that advances prev and cur as long as the loop variable is smaller than the
        passed in index;
    3) Once the for loop exits, cur is on the index we want to add. Set prev.next to a new node
        containing the passed in object.
    4) Set prev to prev.next (the newly created node).
    5) Set prev.next to cur (linking the new node to the rest of the list).
    6) Increase the list size by one.
}

    // Returns the element at the specified index in this list.
public Object get(int index) throws IndexOutOfBoundsException{
    1) Check to ensure the index is within range of the list;
    2) Create a node called cur set to this.head.next;
    3) Create a for loop which advances cur. It starts at 0 and ends when the loop variable is equal
        to the index;
    4) Return cur.data (since cur is on the element of the given index);
}

    // Removes (cuts out) the list node at the specified index in this list.
public Object remove(int index) throws IndexOutOfBoundsException {
    1) Check to ensure the index is within range of the list size;
    2) Create nodes cur (set to this.head.next) and prev (set to this.head);
    3) Create a for loop which advences prev and cur until the loop variable equals the index. Now
        cur is on the element to remove.
    4) Set prev.next to cur.next, cutting out cur;
    5) Decrease the list size variable by one;
    6) Return cur.data;
}

    //Add the object e to the end of this list.
public boolean add(Object e) {
    1) Check if the size is zero. If so, use the addFirst method and return true.
    2) If the size is greater than zero, create a new ListNode called cur assigned to this.head.next
    3) Create a while loop that advances cur down the list as long as cur.next is not null.
    4) Once the while loop exits, we are at the end of the list. Now we set cur.next to a new node
        with object e as a parameter. Then, return true.
 }
```