EEM 480 Homework 2

Yunus Nihat Özpolat

```
public void Insert(int newElement, int pos) throws Exception {
```

I started my Insert method by first checking if my list is empty. If it is empty, I added the inserted value as the element of the first node of my list. If it is not empty and my entered pos value is 0, I have this new node added to the beginning and updated my head. For the remaining cases, I first calculated the size of my list with a while loop. I throw an exception if my pos value is greater than or equal to my size. Since my pos values start from 0, my maximum valid pos value is also (size-1). Afterwards, I reached the position entered with the while loop, and when I reached it, I connected a new node and wrote the value entered in the element of this node.

```
}
public int Delete(int pos) throws Exception {
```

In the Delete method, I first started by checking the fullness of my list. If it is empty, I throw an exception in the else part. If my list is not empty; If there is only one node in my list, I deleted it from the beginning. If there is more than one node in my list and my pos value is 0, I updated my head and deleted my first node. For the remaining possibilities, after counting the size of my list, if my pos value coincides with the last node, I updated my tail and deleted it from the end. If my pos value is an invalid value, I throw an exception and finally, if my pos value is somewhere in between, after I found that position, I connected the before and after of my node and performed my deletion.

```
}
public void LinkReverse() {
```

I created two nodes for the LinkReverse method and one of them points to the head and the other to the tail. Again, I counted the size of my list in a while loop. In my second while loop, I kept the element of my tail in the int k value I created, I threw the element of my head into my tail, and then I threw the element of my initial tail that I kept in k into the head. Finally, I updated my new head to the right of the head and my new tail to the left of my tail. I completed the inversion process by rotating this loop (size/2) times.

```
}
public void SacuraL() {
```

Initially I created 2 nodes named newNode and newNode1 and newNode shows head while newNode1 shows right of newNode.

In my first while loop, I count how many elements are in my LinkList and keep it in n (lines 138 to 141). My second while loop will loop up to my size. Inside this loop my first if will run if my newNode1 is not null and then my second if will run if the element of my newNode is not equal to the element of my newNode1. In my second if, I create a new node named tmpNode and connect my node between newNode and newNode1. Then I make the element of my tmpNode my counter. When exiting, I set the counter to 1 again. Because I want it to print the value 1 for other non-repeating numbers (between lines 148 and 15158). Else part I increase my counter by 1 if the elements of two consecutive nodes are equal because I will use this number when my iteration is over. As soon as I find a repeating number on lines 162 and

163, I delete the first repeating one. For example, if I have 9 9 9, for me to print 9 3, I need to get rid of the first two 9s and then print the count again with a new node to the right of the last 9. I update my newNode and newNode1 and exit. The else part of my first if (lines 168 to 173) means that my newNode1 is null. Here I am printing the iteration count of my last element using a temporary xNode.

```
}
```

```
public void OpacuraL() {
```

Again, I make the size of my list count and throw it to n. Between lines 190 and 195, I count the elements to the right of the newNode (i.e. the element of newNode1) 1 and assign it to c. I will use this value to determine how long my for loop will return.

As in SacuraL, I point to my newNode as head and newNode1 to the right of head. I open my for loop up to c and write 'newNode1.right != null' so that my first if in it goes to the end of the list and stops when it sees null. In my second if, if the element of newNode1 is 1, I connect the left and right of 1 to remove it. If the element of newNode1 is not 1 I am creating a temporary node. I connect my temporary node between newNode and newNode1 and write in its element the element of the repeating number newNode. I just update newNode1 and keep newNode1 constant as it is a repetitive operation. I also reduce the element of newNode1 by 1 because I want it to do this operation up to 1, and when it is 1, it does the operation in the if. I don't want to increase the value of i here because it is a repetitive process, so I say 'i = i - 1'. Then, since I just made a new update here, I check the newNode1 element again and if it is 1, I do the same as above (lines 207 to 224).

In the else on line 226 I update newNode1 if the right of my newNode1 equals null.

```
}
```

```
public void Output() {
```

I printed the element of the node I am, starting from the head. I updated my head to the right of head and hovered my whole list until null and printed it.

```
}
```

```
public void ReverseOutput() {
```

Starting from Tail, I printed the element of the node I was in. I updated my tail to left of tail and hovered my whole list until null and printed it. Here I just print the reverse of the list. The list is not updated.

```
}
```

```
public String toString() {
```

Here we convert our list to String.

```
}
```

```
public Exception LinkedListException() {
```

The Exceptions I throw in the Insert and Delete methods print our error here.

```
}
```