GTU Department of Computer Engineering

CSE 222 - Homework 3 Report

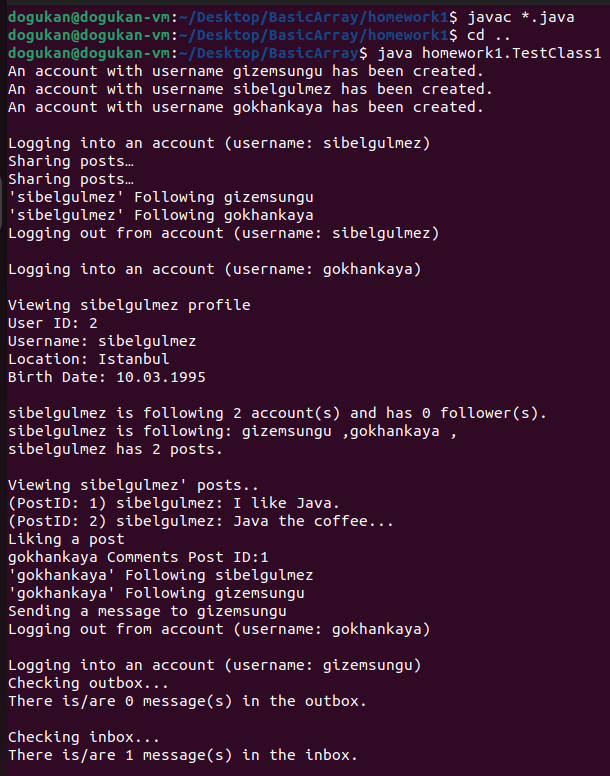
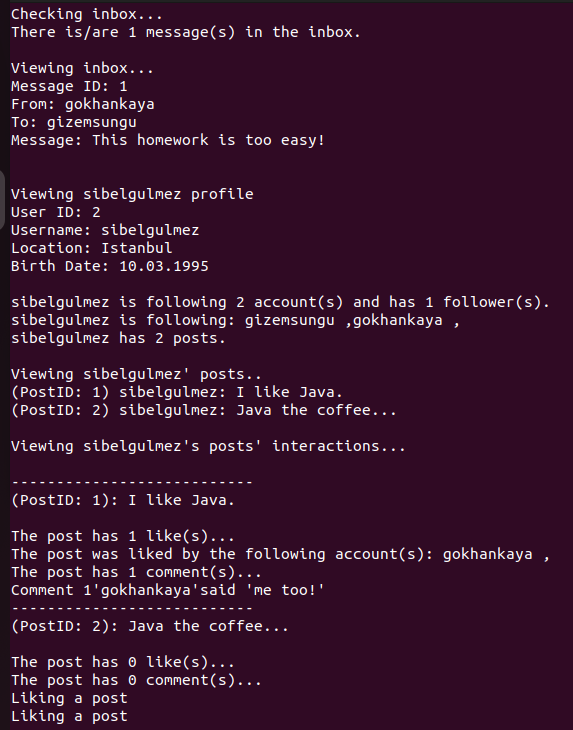
Doğukan Taştan 1901042627

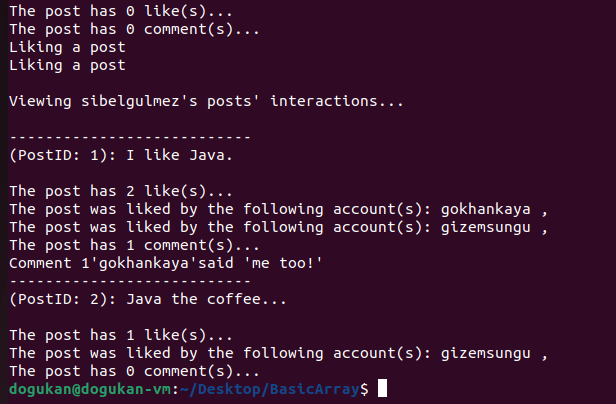
Running Command And Results

Operating System: *Ubuntu 22.04.2 LTS*

Java Version: *11.0.18*

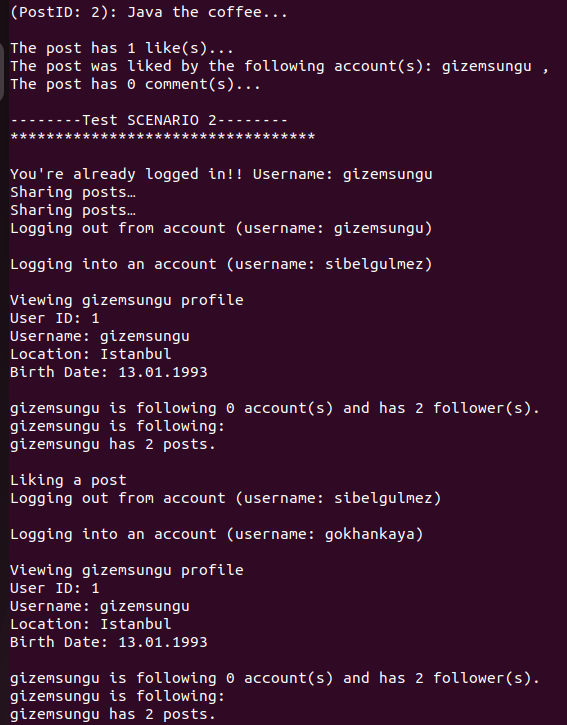
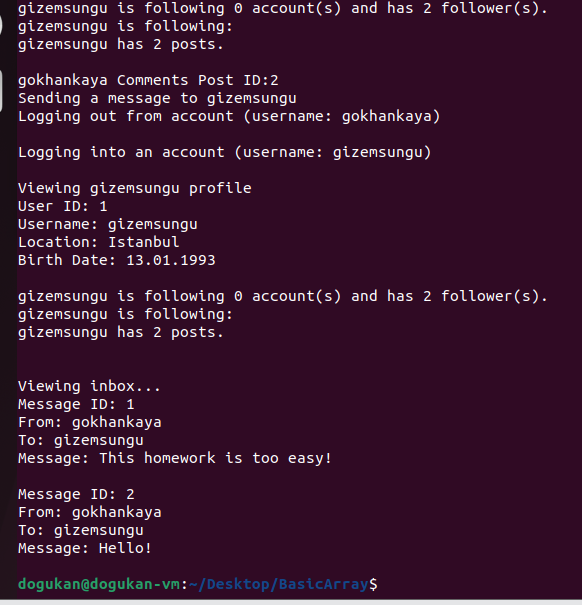
* 1. Basic Array Structure (HW1)
     1. Test Case 1

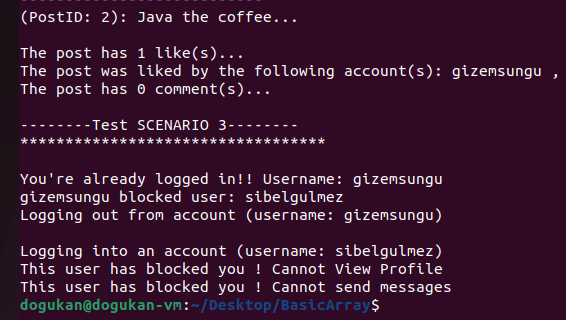


* + 1. Test Case 2

*The first part was not added again because it is the same as Test 1..*

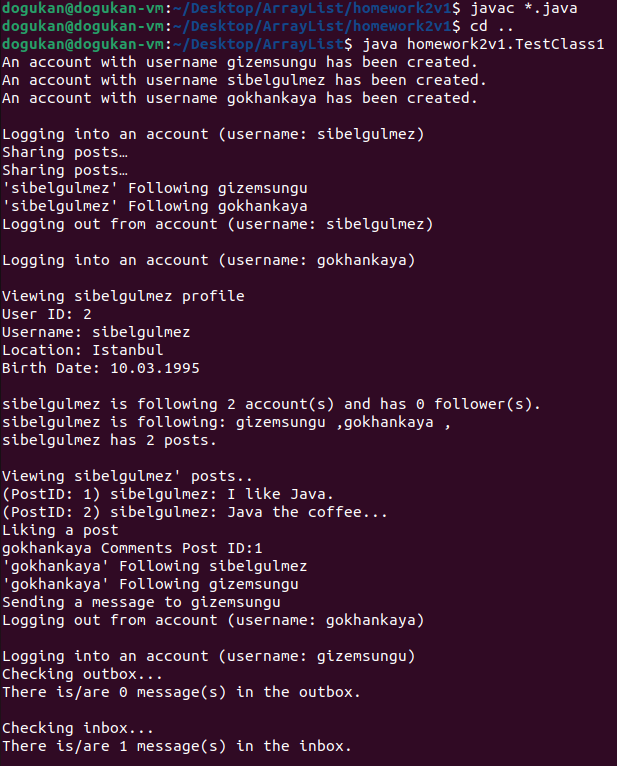
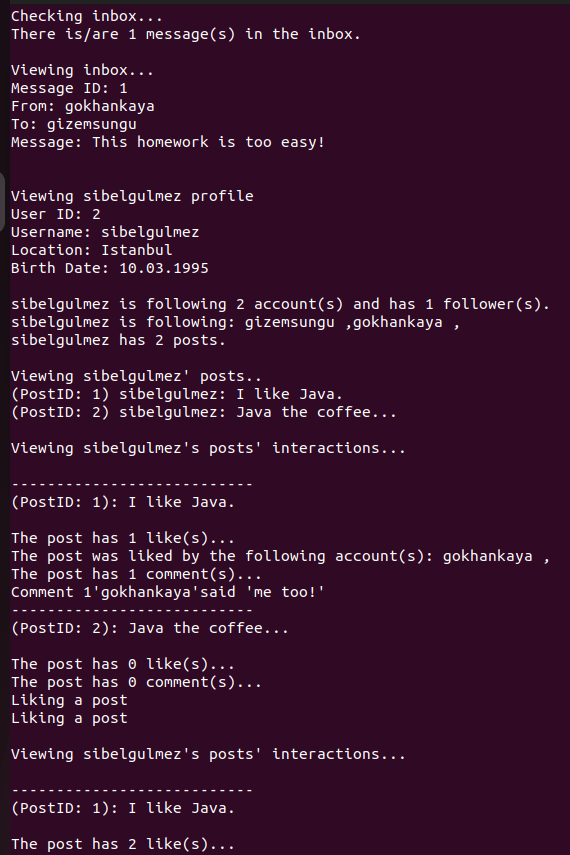
 

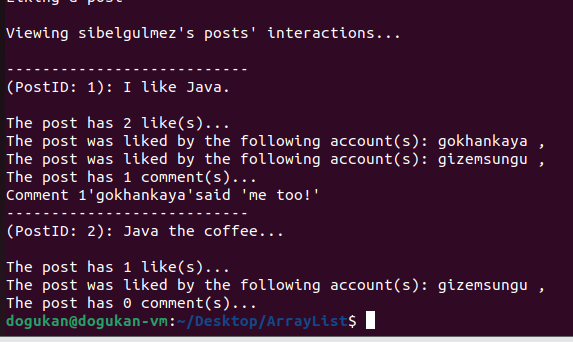
* + 1. Test Case 3



* 1. Array List Structure

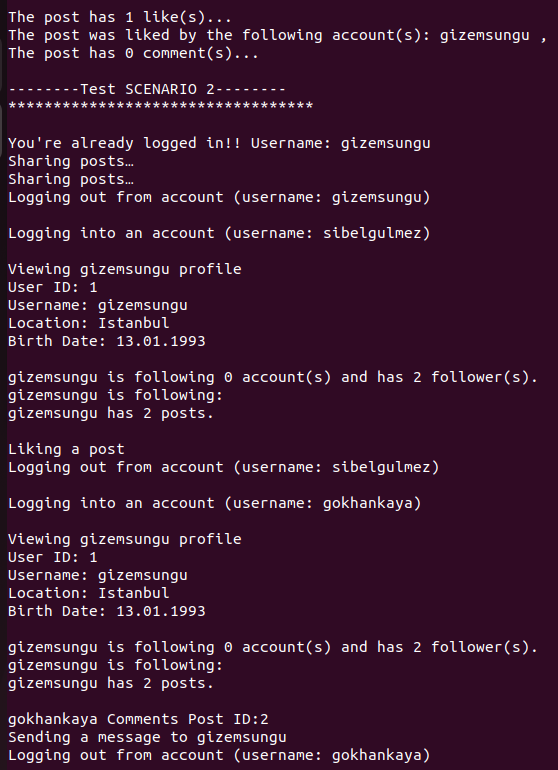
1.2.1 Test Case 1

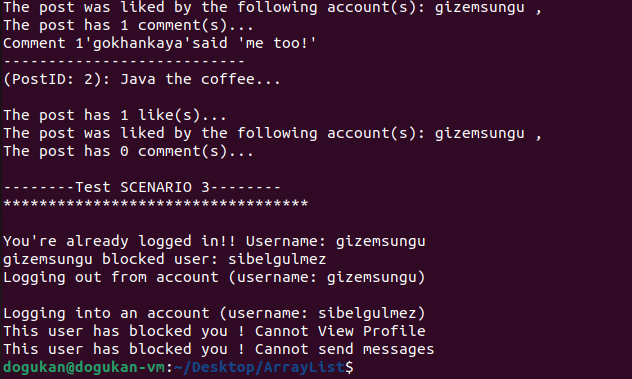


* + 1. Test Case 2

*The first part was not added again because it is the same as Test case 1..*

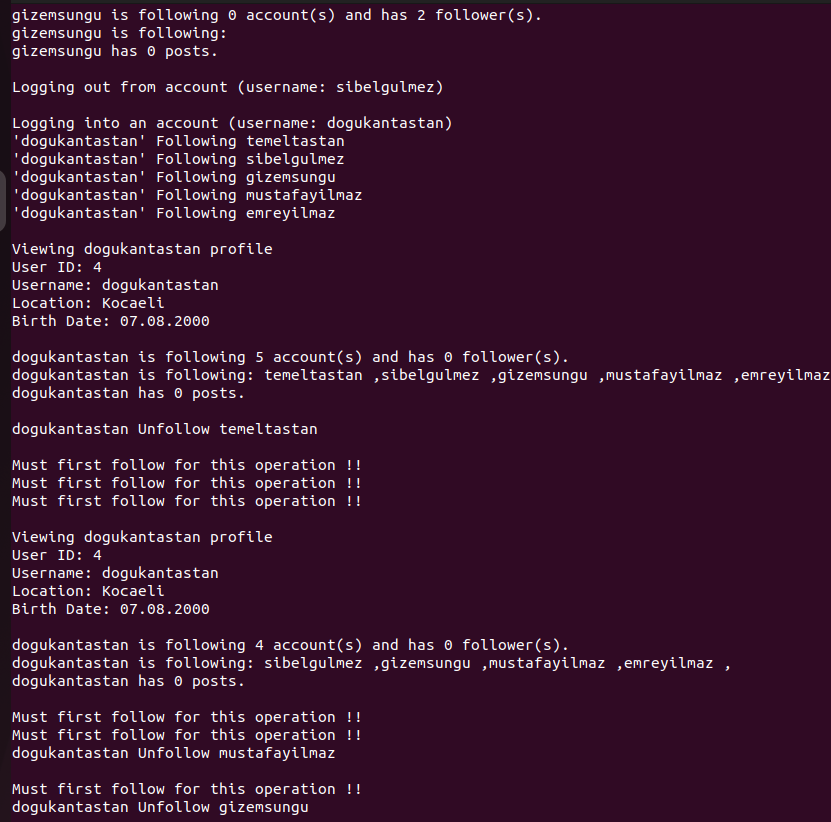
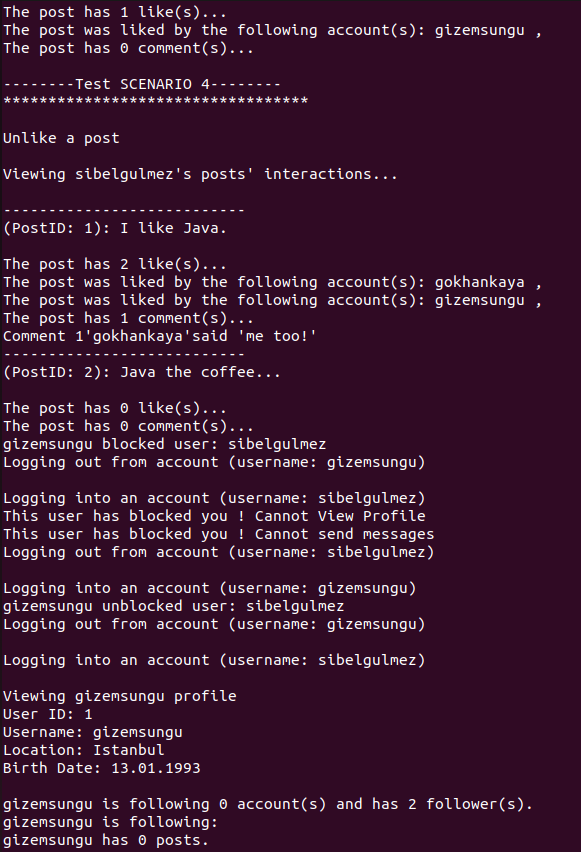
 

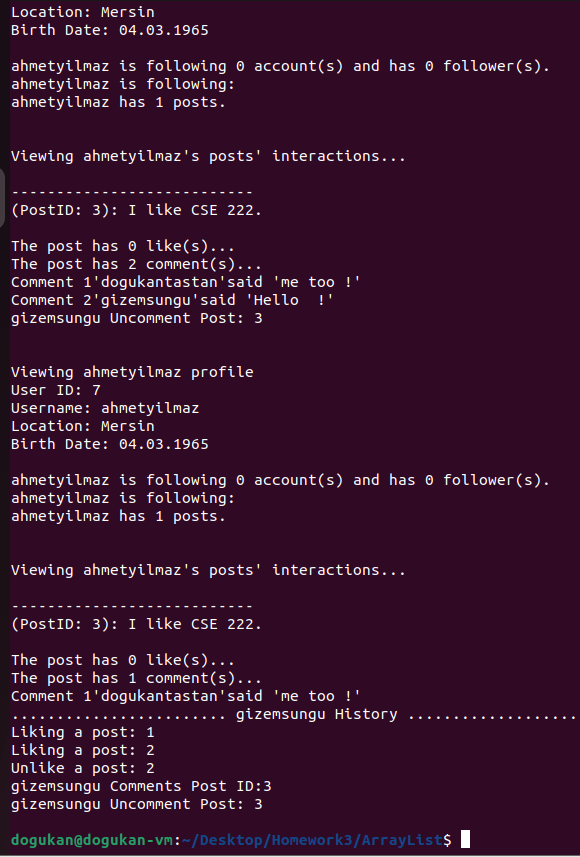
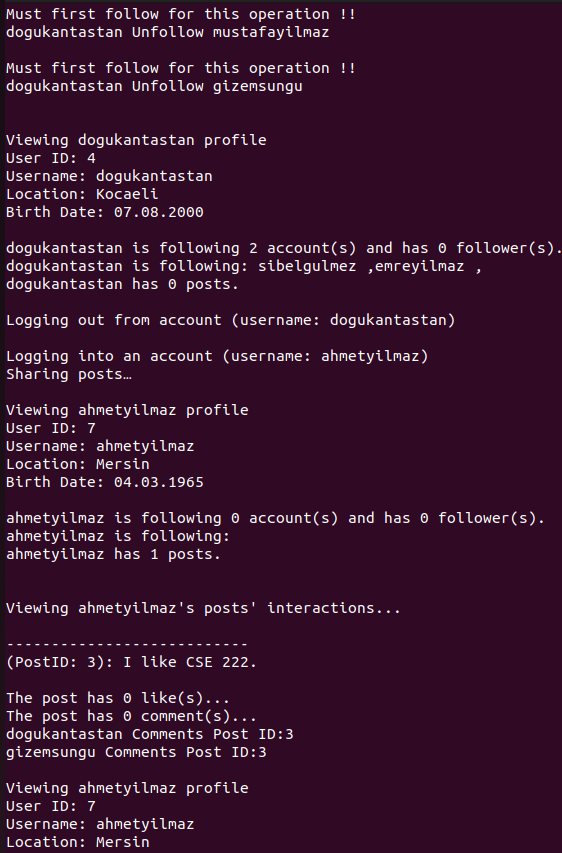
* + 1. Test Case 3



* + 1. Test Case 4

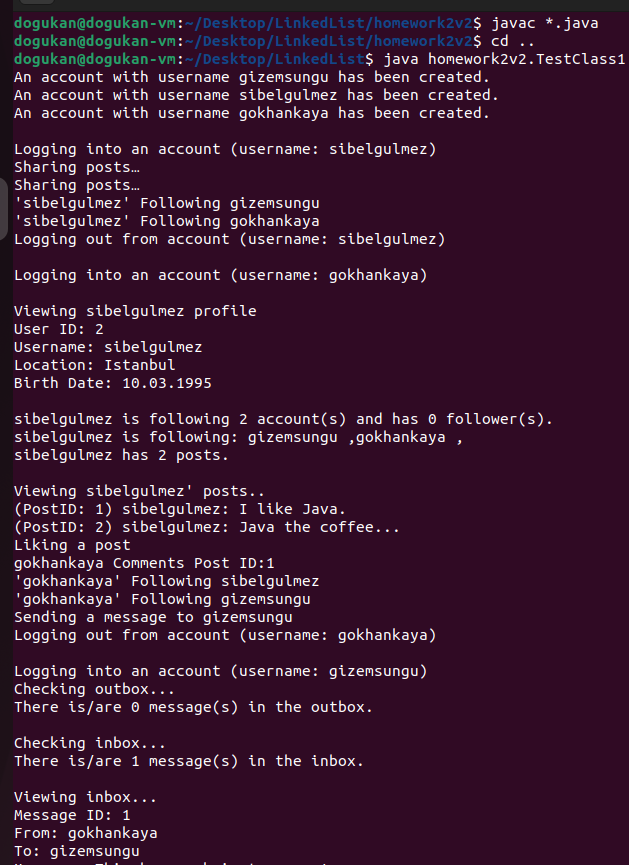
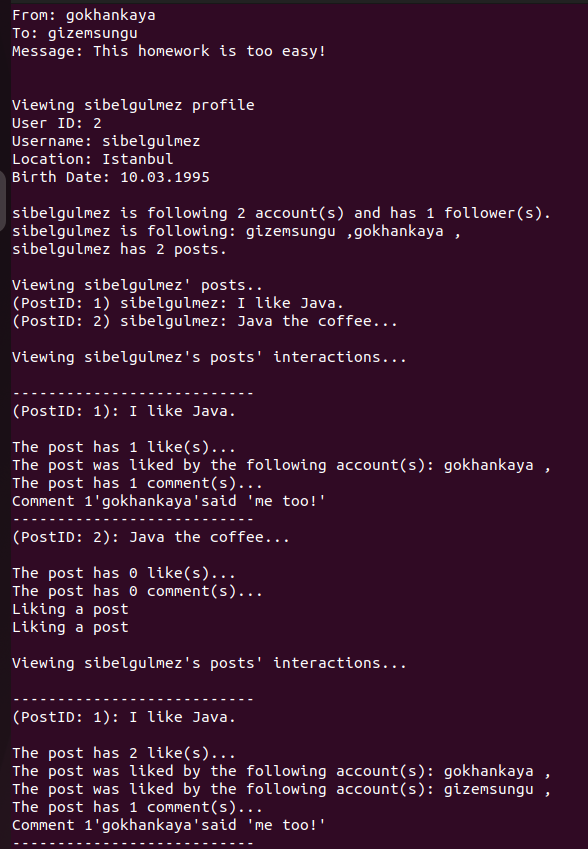
*The first part was not added again because it is the same as Test case 1..*

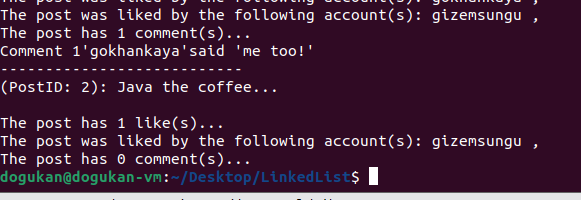




* 1. Linked List Structure

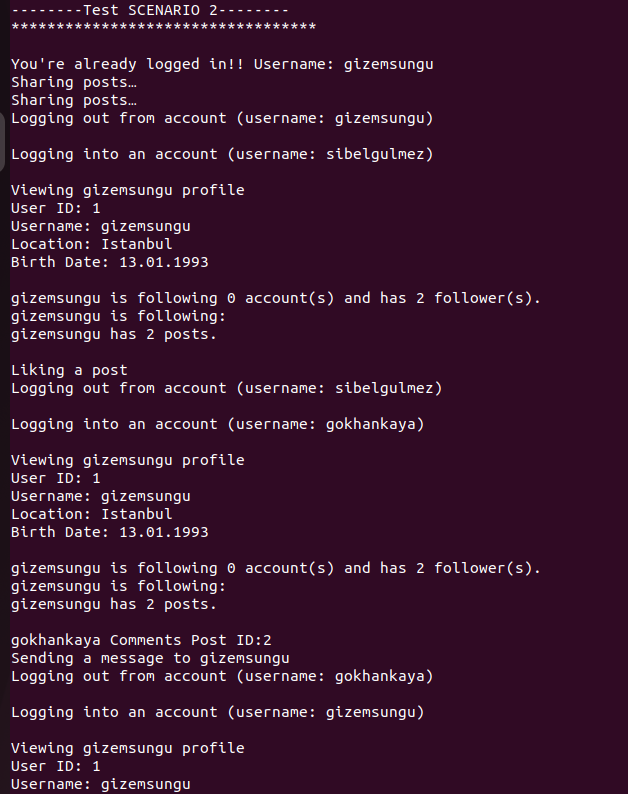
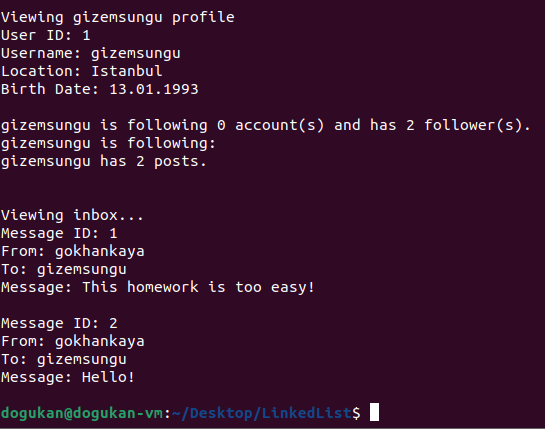
1.3.1 Test Case 1



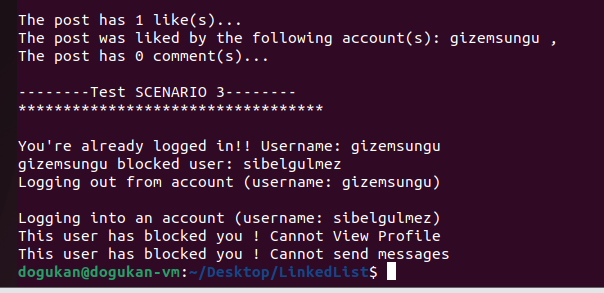
* + 1. Test Case 2

*The first part was not added again because it is the same as Test case 1..*

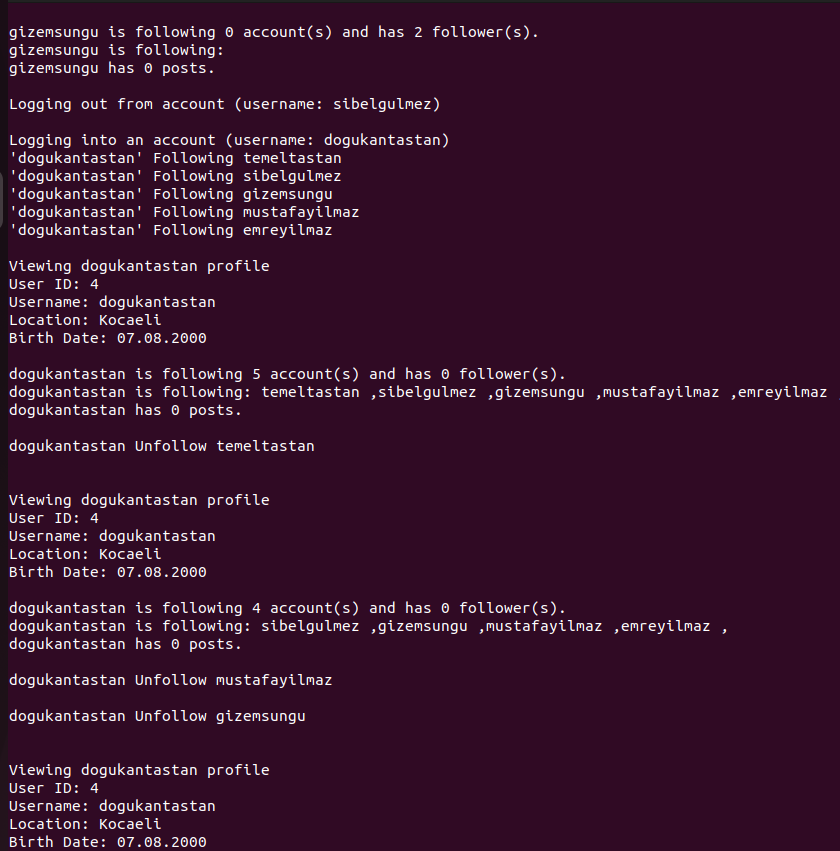
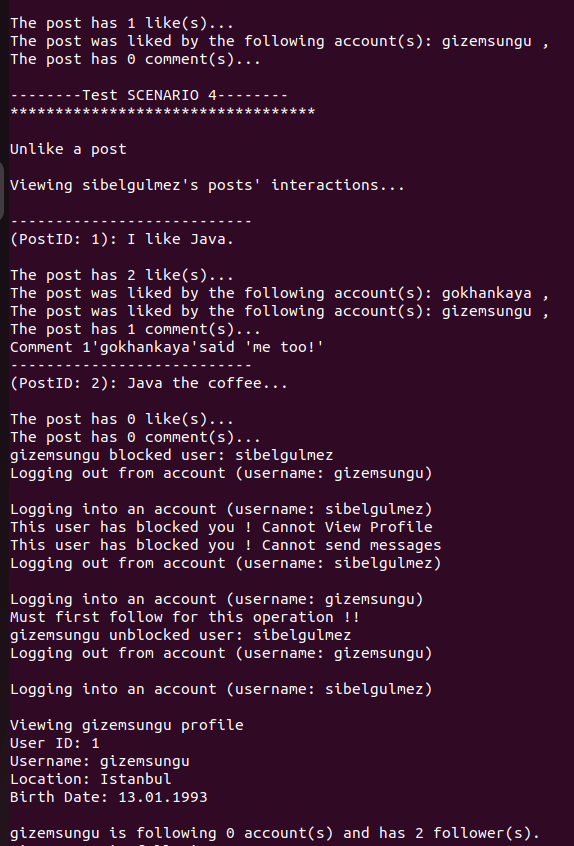
* + 1. Test Case 3

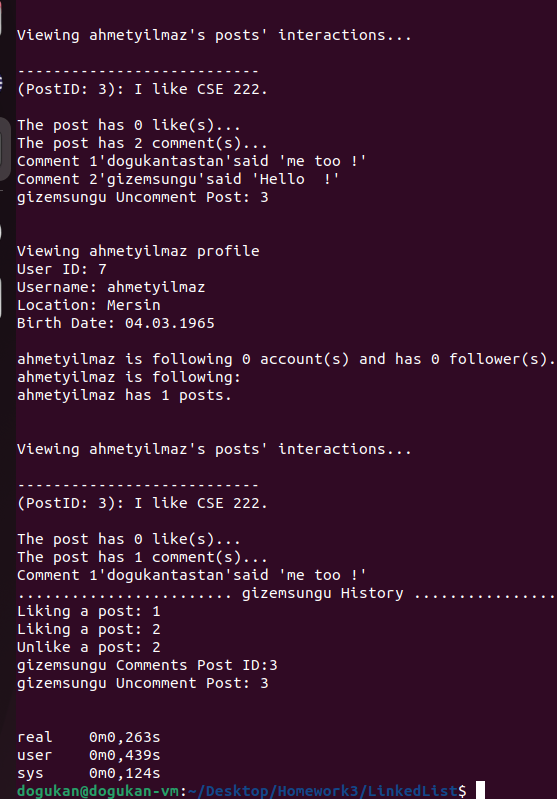
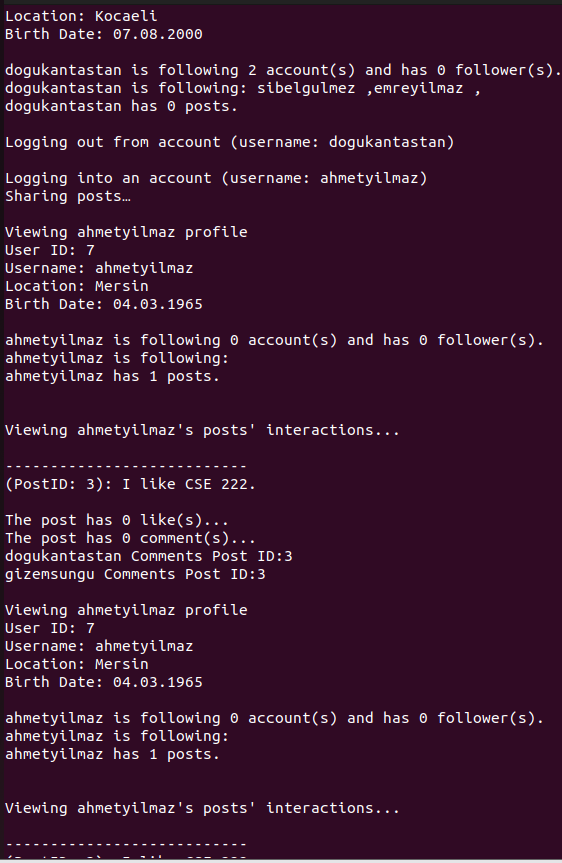
*The first part was not added again because it is the same as Test case 1..*



* + 1. Test Case 4

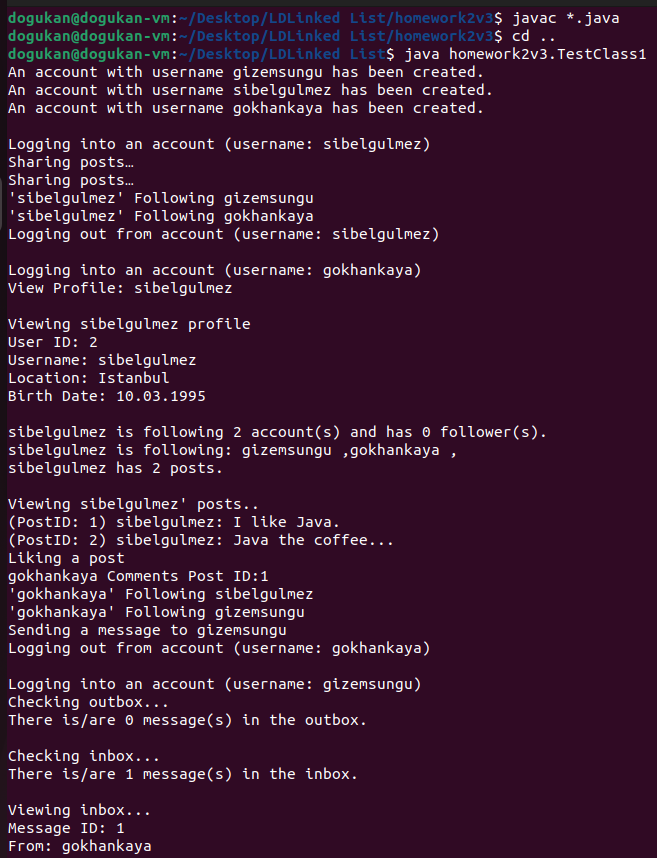
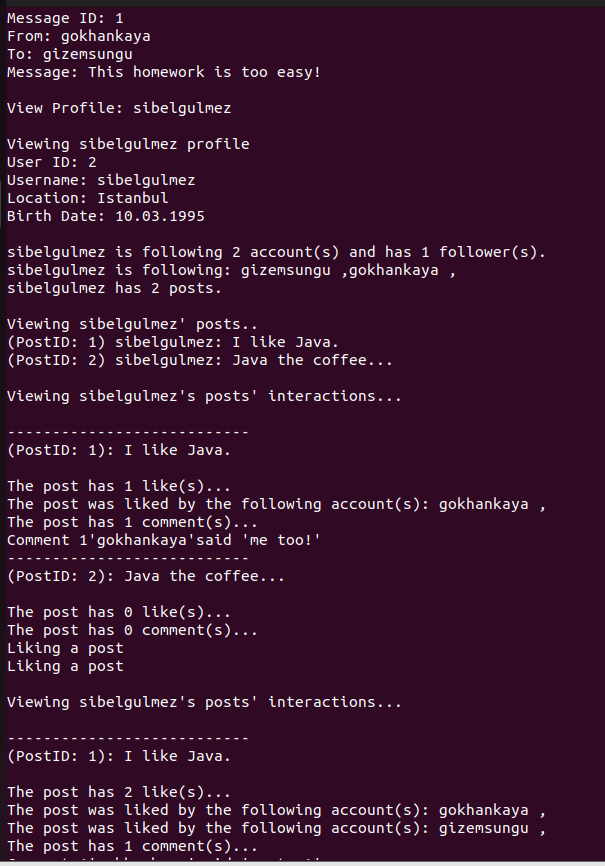
*The first part was not added again because it is the same as Test case 1..*

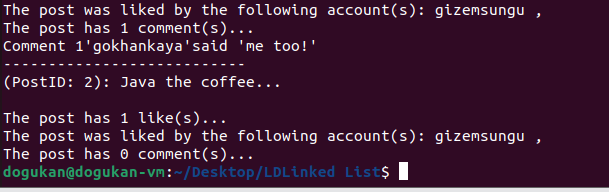




* 1. LD Linked List

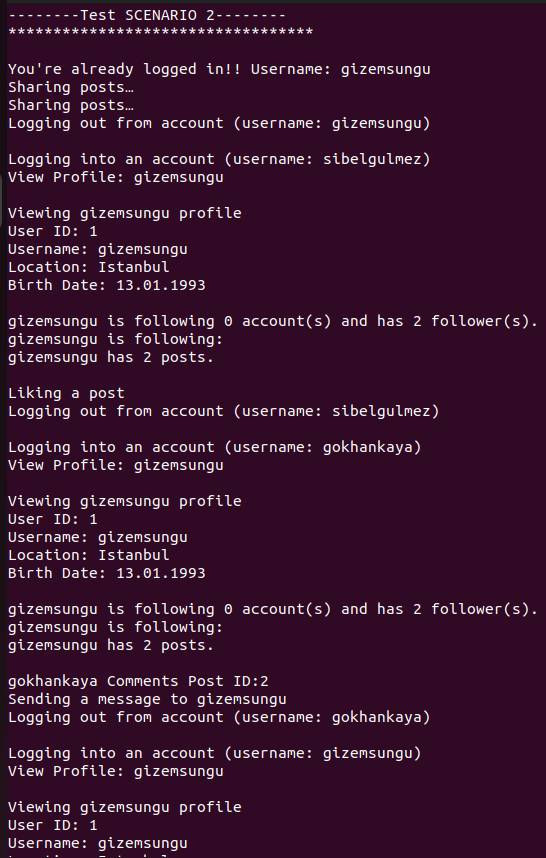
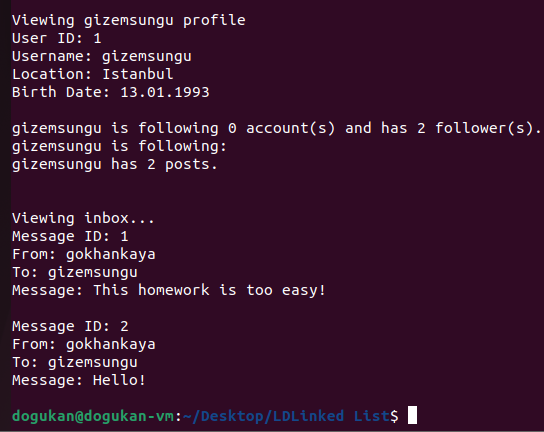
1.4.1 Test Case 1



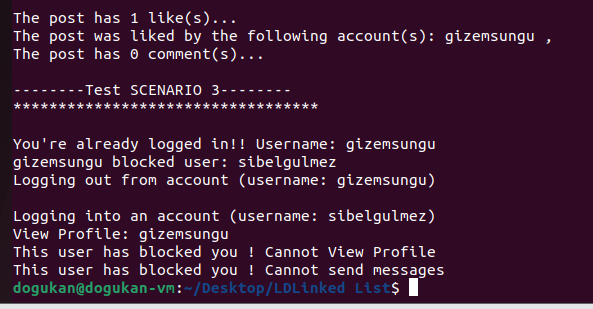
* + 1. Test Case 2

*The first part was not added again because it is the same as Test case 1..*

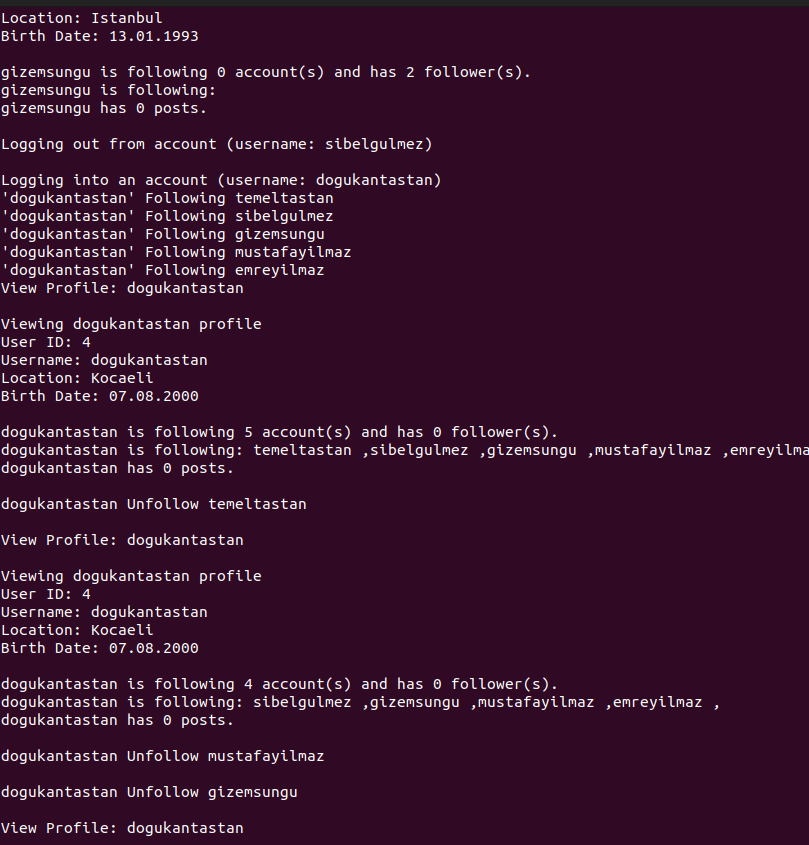
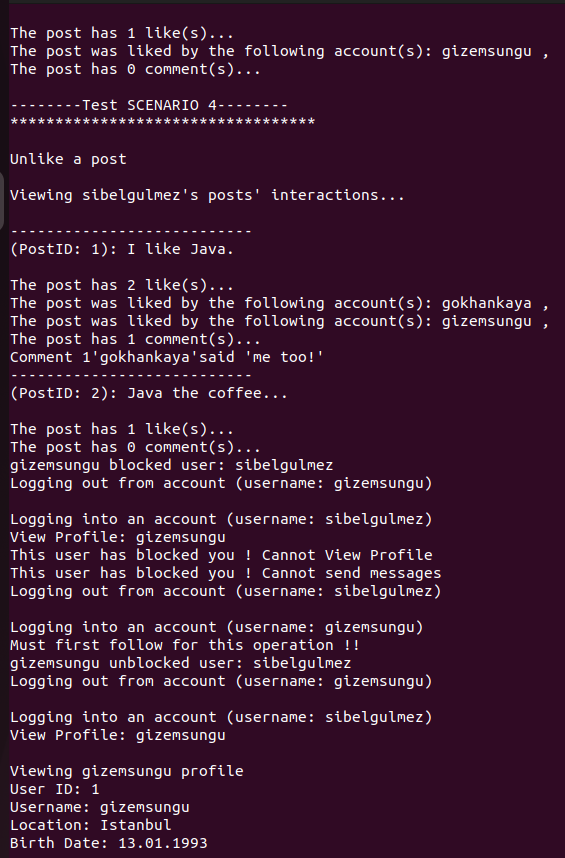
* + 1. Test Case 3

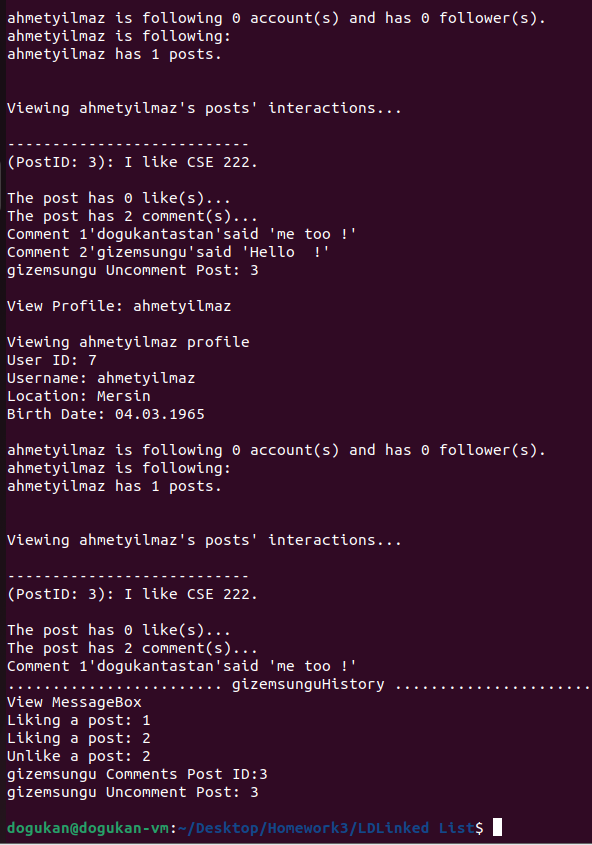
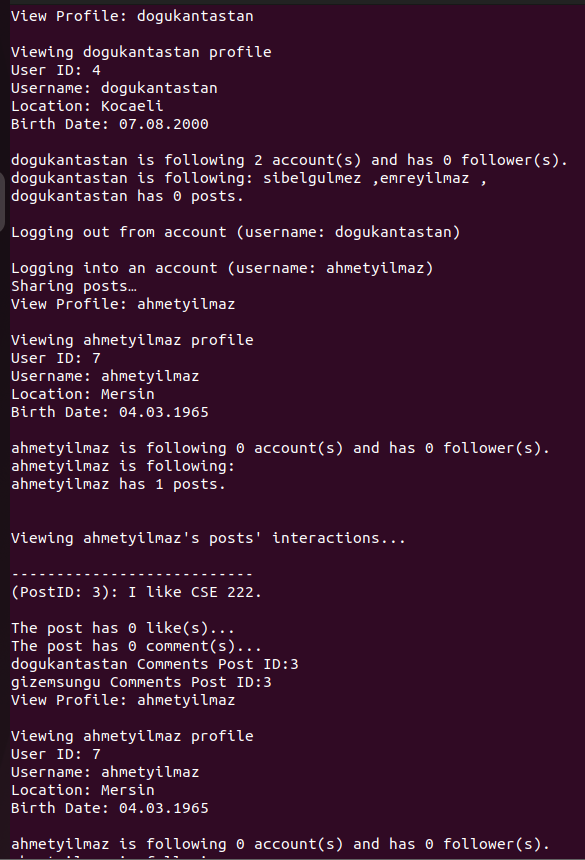
*The first part was not added again because it is the same as Test case 1..*



* + 1. Test Case 4

*The first part was not added again because it is the same as Test case 1..*





Time complexity analysis

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Implementation Type** | **Follow** | **Unfollow** | **Like** | **Unlike** | **Comment** | **Uncomment** | **Share post** | **Login** |
| Basic Array Structure | O(1) | **-** | O(1) | **-** | O(1) | **-** | O(1) | O(n) |
| Array List Structure | O(n) | O(n^2) | O(n) | O(n^2) | O(n) | O(n^2) | O(n) | O(n) |
| Linked List Structure | O(1) | O(n^2) | O(1) | O(n^2) | O(1) | O(n^2) | O(1) | O(n^2) |
| LD Linked List Structure | O(1) | O(n^2) | O(1) | O(n^2) | O(1) | O(n^2) | O(1) | O(n^2) |

Experimental running time analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Implementation Type | Scenario 1 | Scenario 2 | Scenario 3 | Scenario 4 |
| Basic Array Structure (HW1) | 0,404s | 0,404s | 0,411 | Not available |
| Array List Structure (a) | 0,405s | 0,413s | 0,415s | 0,460s |
| Linked List Structure (b) | 0,410s | 0,411s | 0,401s | 0,457s |
| LD Linked List Structure (c) | 0,406s | 0,423s | 0,434s | 0,466s |

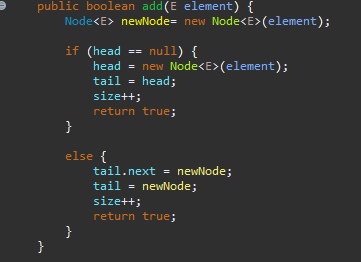
Compilations were made as “ time java homework1.TestClass1 ” and each result was repeated 5 times and the average was taken.

Problem solution approach for LDLinkedList

To implement the lazy deletion strategy, I designed the remove function as a management function. While the necessary checks and operations are done here, the actual deletion is done in the function named Myremove. When our Remove function is called, we first check whether a deletion has occurred before the current index with a while loop. If not, this item is only marked as isDeleted= True due to lazy deletion and the process ends. However, if an operation has been performed before, that is, if the true variable is found, we determine this with the help of the flag and send it to the My remove function to delete the oldIndex and the current index. Here there are conditions for the indexes not to be confused because when we delete a previous index, the index will change afterwards.  


Myremove function is the function where we do the actual deletion of our linkedlist. We said that the reason we use this function is to use the main remove function as a management function for lazy deletion. First of all, it checks if the list has a single element, if head==tail then the list is empty and it sets head and tail equal to null. Otherwise, if the element to be deleted is at the top, head is updated accordingly. head is first stored in the oldhead variable and head is now set to point next, then the next of oldhead is set to null.

index == size-1 represents deletion from the end and similar operations to head are done for tail. in case of else, deletion is done from the middle of the list.

In this version of the add function, it only adds to the end. The add leaf first checks if the list is empty, if head= null, it creates a new node object and shows it to head and tail. If not, tail.next shows the new node because it is added from the end and then tail itself shows the new node.

This is the general implementation of the LDLink class The getBool function returns the isDeleted value in the node, unlike the normal get function