GIT Department of Computer Engineering
CSE 222/505 - Spring 2023
Homework 3 Report

Mehmet Mete Şamlıoğlu
200104004093

1-) SYSTEM REQUIREMENTS

The desired system is basically a social media application that facilitates users to interact with each other throughout the system. The system requires basic operations such as sharing a post, sending a message to another user, interacting with a post of an account, viewing other accounts' profiles, follow or blocking an account. There are some restrictions to follow to perform each operation properly. For instance, to send a message to another account it must be checked whether this account has been followed or not. This is the only restriction that is related to the following operations in the system. When it comes to the restrictions that are related to login( ) operations, basically to perform any operation you have to log into your account. The system is not allowing account owners to login into two accounts at once, only one account could be active at once. This exception will be controlled by the system during the runtime, whenever another account tries to log in while there was an active account, an error message will be displayed. There are some other restrictions that are controlled by the system during run time, such as instantiating an object that has the same ID or username as another account that has already been created. It is not allowed to instantiate another account that has a similar ID or username with any account that is already registered to the system, if this restriction is not obeyed, the constructor of the class will throw an Exception and terminate the system. There are two types of errors in the system, in the first type the error message will be displayed only but the system allows you to continue. The second type of operation is handled by an exception-handling mechanism, and this type terminates the system whenever it occurs. As It was mentioned before, if you instantiate an object that has a similar ID to another object the system will terminate because it is not allowed to create that.

2-) USE CASE

Implementation Type	Scenario 1	Scenario 2	Scenario 3	Scenario 4
Basic Array	**0.0460**	**0.0590**	**0.0550**	
ArrayList	**0.0510**	**0.0660**	**0.0500**	**0.0870**
LinkedList	**0.0510**	**0.0630**	**0.0530**	**0.0860**
LDLinkedList	**0.0550**	**0.0730**	**0.0550**	**0.0890**

Experimental running time analysis

I examined the results and calculated the average, so the performances as following;

Basic Array > Array List > Linked List > LDLinkedList. The reason for that is, in Basic Array structure, the data container is fixed size array, not a dynmaic one. Each array is fixed size which has 100 capacity. Thanks to that, whenever we add an element, or remove an element most of the time it happens in O(1) time. There is no need to expand the array or reduce its size, it did not happen in this program so every operation has been performed at worst in O(n) time (Except some printing functions). Since in the test cases there are no objects more than 100, it worked without increasing the size. However, if system needs more than 100 accounts, I must be modified and turned into dynamic array. And If I make it dynamic, the running time results will approach to result of ArrayList and LinkedList.

When it comes to the difference between ArrayList and LinkedList, in the program the only Iterator methods that I used is, get( ), add( ) and remove( ). In some cases I used contains( ) and indexOf but these were not comman. The method that I used the most was get method and when we look at the time complexity of get( ) method, we will see that for LinkedList time complexity of get( ) is O(n), in ArrayList on the other hand it is O(1). However, when it comes to add function the difference is in favor of LinkedList. Linked List's add method's time complexity is O(1), but Array List's add is O(n) (Since it needs to slide all elements). The second most method that I used after get was add( ), so that LinkedList's performance was better whenever an addition operation occurs. For remove( ) method their time complexity is the same. So, as I mentioned in the beginnig the most common method was get( ) and this method's efficiency is better in Linked List so that Linked List had a little bit slower than ArrayList.

LDLinkedList structer had the slowet running time, because of the deletion operation difference between LinkedList and LDLinkedList. LDLinkedList uses lazy delete strategy. In

this method whenever a node is deleted it is marked as deleted and kept waiting until another node is deleted. So it bring inefficiency to LDLinkedList. The other methods have the same implementation with LinkedList and we know that LinkedList's perfomance was worse than arrayList. Therefore, LDLinkedList has the worst performance among others.

4-) PROBLEM-SOLUTION APPROACH FOR LDLINKEDLIST

The only difference between LDLinkedList and LinkedList was the way of removing nodes. As I mentioned in the experimental running time analysis, LDLinkedList is using a lazy deletion strategy. In this strategy, nodes are marked as deleted but they are not removed physically. We just ignore them in other operations of LDLinkedList. However, when another node is deleted, all deleted methods are removed at once physically. In this way whenever a removal operation occurs time complexity of first remove operation becomes O(1) so it is better than original LinkedList implementation. However, since we are not deleting these node phsically whenever a search is needed in LinkedList the path we are searching is increasing so it brings inefficiency, and also removing all the nodes at once takes more time than removing only one node. So if the frequency of remove operation is lot it also brings inefficiency as you can see from the experimental running time analysis results.

Normally, lazy deletion strategy also improves efficiency of adding operation but in my code it doesn't help it(since it was not mentioned in pdf). Therefore, In my LDLinkedList implementation it only marks nodes as deleted until they are deleted physically. In my implementation I deleted nodes physically whenever two deleted node occurences detected. I added a boolean variable to private inner Node<E> class, by the help of this method I marked the deleted nodes as true. So, in other operations I controlled this situation for example in search whenever boolean isDeleted value is true, the searching method ignores that not and continues searching. I kept track of the deleted nodes and whenever two nodes are deleted, I removed them phsically and continued doing it. To perform that I implemented another function which is getDeletedIndex( ) it return the index of deleted node and after that I sent them to remove function and perform delete operation. The reason why LDLinkedList had the worst running time is whenever is delete is needed I called remove operation two times and getDeleteIndex operation one time both operation has time complexity of O(n).

In conculusion, this way of implementing LinkedList helped me whenever only one node is deleted. However in general this feature did not work in my favor and I had not a good result in terms of efficiency.

5-) TEST CASES

```java
/* Step 1 */
Account Admin = new Account( );
Account sibelgulmez = new Account(13, "sibelgulmez", "06-05-1995","Istanbul",Admin);
Account gokhankaya = new Account (12, "gokhankaya",  "08-09-1990", "Ankara",Admin);
Account gizemsungu = new Account (14, "gizemsungu",  "05-04-1995", "Izmir",Admin);
/* ------------------------------------------------------------------------*/
sibelgulmez.login( ); /* Step2 */
Post post1 = new Post(1, "I like Java.");
Post post2 = new Post(2, "Java the coffee...");
/* Step 3 */
sibelgulmez.addPost(post1);
sibelgulmez.addPost(post2);
/* ------------------------*/
/* Step 4 */
sibelgulmez.follow(gizemsungu);
sibelgulmez.follow(gokhankaya);
/*------------------------*/
sibelgulmez.logout( ); /* Step 5 */
gokhankaya.login( ); /*Step 6 */
gokhankaya.viewProfile(sibelgulmez); /* Step 7 */
gokhankaya.viewPosts(sibelgulmez); /* Step 8 */
Like like1 = new Like(1, gokhankaya, post1); /*Step 9 */
Comment comment1 = new Comment(2, gokhankaya, post1,"me too!"); /*Step 10 */
gokhankaya.follow(sibelgulmez); /* Step 11 */
gokhankaya.follow(gizemsungu);
Message message1 = new Message(1, gokhankaya, gizemsungu, "Hello!"); /* Step 12 */
gokhankaya.logout( ); /* Step 13 */
gizemsungu.login( ); /* Step 14 */
gizemsungu.checkOutbox( ); /* Step 15 */
gizemsungu.checkInbox( ); /* Step 16 */
gizemsungu.viewInbox( ); /* Step 17 */
gizemsungu.viewProfile(sibelgulmez); /* Step 18 */
gizemsungu.viewPosts(sibelgulmez); /* Step 19 */
gizemsungu.viewPostInteractions(post1); /* Step 20 */
Like like2 = new Like(2, sibelgulmez, post1); /* Step 21 */
gizemsungu.viewPostInteractions(1, sibelgulmez); /* Step 22 */  /* There are two ways to invoke method viewPostInteractions(), it is overloaded*/
```

```java
/* ------------------------- Scenario 2 Starts -------------------------/ */
System.out.printf("-----------Scenario 2 Starts-----------\n");
/* ------------Step1 -----------*/
gizemsungu.login( );
Post post1 = new Post(1,"Today is a good day!");
Post post2 = new Post(2,"I'm not gonna work today.");
gizemsungu.addPost(post1);
gizemsungu.addPost(post2);
gizemsungu.logout( );
/*----------Step2--------------*/
sibelgulmez.login( );
sibelgulmez.viewProfile(gizemsungu);
sibelgulmez.viewPosts(gizemsungu);
Like like3 = new Like(3, sibelgulmez, post1);
sibelgulmez.logout( );
/*-------------------------------*/
/*-----------Step3--------------*/
gokhankaya.login( );
gokhankaya.viewProfile(gizemsungu);
Comment comment2 = new Comment(4, gokhankaya, post2, "Nice!");
Message message2 = new Message(5, gokhankaya, gizemsungu,"Hello!");
gokhankaya.logout( );
/*-------------------------------*/
/*-------------Step4--------------*/
gizemsungu.login( );
gizemsungu.viewProfile(gizemsungu);
gizemsungu.viewPosts(gizemsungu);
gizemsungu.viewPostInteractions(post1);
System.out.printf("\n");
gizemsungu.viewPostInteractions(post2);
gizemsungu.viewInbox( );
gizemsungu.logout( );
```

```java
/*----------------- SCENARIO 3 (Bonus Part) Starts--------------*/
System.out.printf("------------- SCENARIO 3 (Bonus Part) Starts--------------\n");
gizemsungu.login( );
gizemsungu.block(sibelgulmez);
gizemsungu.logout( );
sibelgulmez.login( );
sibelgulmez.viewProfile(gizemsungu);
Message message4 = new Message(4, sibelgulmez, gizemsungu, "Hello Gizem!");
sibelgulmez.logout( );
```

6-) RUNNING AND RESULTS

```
Array TestClass1 Execution time: 0.0460 seconds
```

```
Array TestClass2 Execution time: 0.0590 seconds
```

```
Array TestClass3 Execution time: 0.0550 seconds
```

```
ArrayList TestClass1 Execution time: 0.0510 seconds
```

```
ArrayList TestClass2 Execution time: 0.0660 seconds
```

```
ArrayList TestClass3 Execution time: 0.0500 seconds
```

```
ArrayList TestClass4 Execution time: 0.0870 seconds
```

```
LDLinkedList TestClass1 Execution time: 0.0550 seconds
```

```
LDLinkedList TestClass2 Execution time: 0.0730 seconds
```

```
LDLinkedList TestClass3 Execution time: 0.0550 seconds
```

```
LDLinkedList TestClass4 Execution time: 0.0890 seconds
```

```
LinkedList TestClass1 Execution time: 0.0510 seconds
```

```
LinkedList TestClass2 Execution time: 0.0630 seconds
```

```
LinkedList TestClass3 Execution time: 0.0530 seconds
```

```
LinkedList TestClass4 Execution time: 0.0860 seconds
```