|  |
| --- |
| C:\Users\mabarkat\Documents\KSU_LOGO.BMPKing Saud University  College of Computer and Information Sciences  Department of Computer Science  CSC 212 Data Structures Project Report – 1st Semester 2016-2017  Developing a Ratings Query Application |

**Authors**

|  |  |  |
| --- | --- | --- |
| Name | ID | Section |
| Areej Almalki | **435202033** | **41196** |
| Haya Alwahhabi | **435202510** | **41196** |
| Tarfah Alghofili | **435200281** | **41196** |

# Introduction

The hole idea of this project is how to store data using the right data structure in an appropriate manner and in a way to avoid memory consumption also to retrieve this data in an efficient way

this report will contain the ADTs specification format and how we designed this project ,also we will clarify some  of the methods that need to be clarified

a summary of the project and the source cod

# Specification

**Elements:** The elements are of generic type <Type> (The elements are placed in nodes for linked list implementation).

**Structure:** the elements are linearly arranged. The first element is called head, there is a element called current.

**Domain:** the number of elements in the list is bounded therefore the domain is finite. Type name of elements in the domain: List

**Operations:** We assume all operations operate on a list L.

**Method** FindFirst ( )

**requires:** list L is not empty. **input:** none

**results:** first element set as the current element. **output:** none.

**Method** FindNext ( )

**requires**: list L is not empty. Cur is not last. **input**: none

**results**: element following the current element is made the current element.

**output**: none.

**Method** Retrieve (Type e)

**requires**: list L is not empty. **input**: none

**results**: current element is copied into e. **output**: element e.

**Method** Update (Type e).

**requires**: list L is not empty. **input**: e.

**results**: the element e is copied into the current node.

**output**: none.

**Method** Insert (Type e).

**requires**: list L is not full. **input**: e.

**results**: a new node containing element e is created and inserted after the current element in the list. The new element e is made the current element. If the list is empty e is also made the head element. **output**: none.

**Method** Remove ( )

**requires**: list L is not empty. **input**: none

**results**: the current element is removed from the list. If the resulting list is empty current is set to NULL. If successor of the deleted element exists it is made the new current element otherwise first element is made the new current element. **output**: none.

**Method** Full (boolean flag)

**input**: none. **returns**: if the number of elements in L has reached the maximum number allowed then flag is set to true otherwise false. **output**: flag.

**Method** Empty (boolean flag).

**input**: none. **results**: if the number of elements in L is zero, then flag is set to true otherwise false.

**Output**: flag.

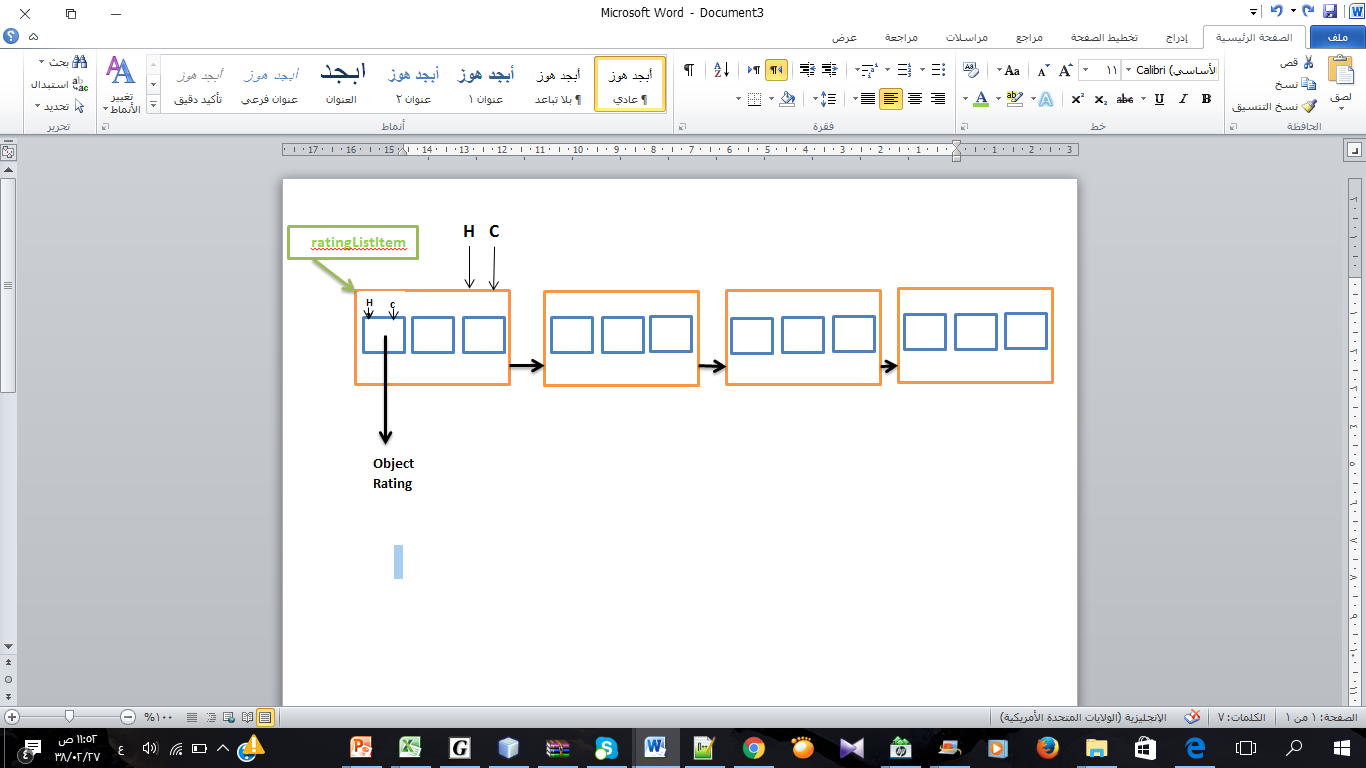
**Method** Last (boolean flag).

**input**: none. **requires**: L ist not empty. **Results**: if the last element is the current element then flag is set to true otherwise false. **Output**: flag

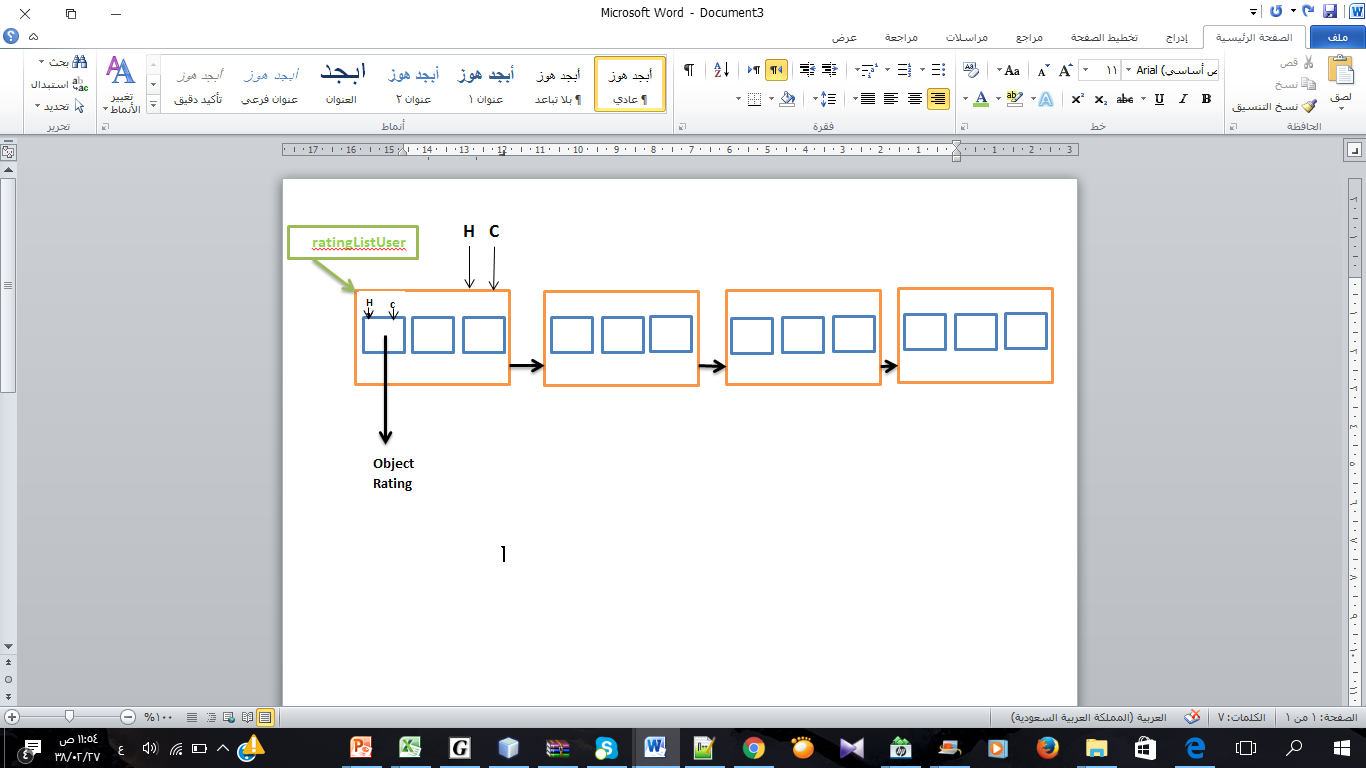
# Design

In this project we have two linked lists ,one for the user and one for the item ,each one has a linked list of ratings ,for Item linked list the the item id in each rating is the same ,for user linked list the user id in each rating is the same .

* **Item linked list**



* **User linked list**



# Implementation

Public RatingManager ()

We declared 2 linked lists of ratings one for the user and one for the item

And we declared two counters for the user and item .

Public static RatingManger read (String fileName)

We read the rating from the file using scanner and we saved the rating in an object of type Rating ,then we added this object to the item llist and user list

Public void addRating (Rating rating)

1-we devided the method into two sections …add rating item and add rating user

2-in add rating item the ratings in each node has the same item id

3-in add rating user the ratings in each node hase the same user is

private void addRatingUser(Rating rating)

1-In this method I receive a user number

2- we make a new list that should contain the rating of the user 3-if the user is found thin I add its ratings to the list and return it ,or I return null.

Public Linkedlist<Rating> getUserRatings(int i)

1-In this method I receive a user number

2- we make a new list that should contain the rating of the user 3-if the user is found thin I add its ratings to the list and return it ,or I return null.

Public Linkedlist<Rating> getItemRatings(int i)

1-In this method I receive an Item number

2- we make a new list that should contain the rating of the Item 3-if the Item is found thin I add its ratings to the list and return it ,or I return null.

public double getAverageItemRating(int i)

1- In this method we receive an item number

2- We make a new list that will contain all item ratings

3- If the item is found it will be inserted in the list and increase the count by 1 and add to the sum

4- Finally if the count is zero then -1 is returned ,else,the average will be calculated

Average =sum/count

public double getAverageUserRating(int i)\*\*\*\*

1- In this method we receive an user number

2- We make a new list that will contain all user ratings

3- If the user is found it will be inserted in the list and increase the count by 1 and add to the sum

4- Finally if the count is zero then -1 is returned ,else,the average will be calculated

Average =sum/count

Public linkedList<Integer> getHighestRatingItem()

In this method we will return a linked list that contain the item that have the highest average rating

# Conclusion

This program adds ratings and each rating have an Item ID,User ID,and the value of the rating.

Also this program calculate the average ratings for each items and users ,also calculate the highest average ratings and return all items that have the highest average .

# Source Code

///////////////////////////\*\* LinkedList and Node \*\*//////////////////////////////////////

1   
 2 class Node<T> {  
 3   
 4 public T data;  
 5 public Node<T> next;  
 6   
 7 public Node(T val) {  
 8 data = val;  
 9 next = null;  
10 }  
11 }  
12   
13 public class LinkedList<T> {  
14   
15 private Node<T> head;  
16 private Node<T> current;  
17   
18 public LinkedList() {  
19 head = current = null;  
20 }  
21   
22 public boolean empty() {  
23 return head == null;  
24 }  
25   
26 public boolean last() {  
27 return current.next == null;  
28 }  
29   
30 public boolean full() {  
31 return false;  
32 }  
33   
34 public void findFirst() {  
35 current = head;  
36 }  
37   
38 public void findNext() {  
39 current = current.next;  
40 }  
41   
42 public T retrieve() {  
43 return current.data;  
44 }  
45   
46 public void update(T val) {  
47 current.data = val;  
48 }  
49   
50 public void insert(T val) {  
51 Node<T> tmp;  
52 if (empty()) {  
53 current = head = new Node<T>(val);  
54 } else {  
55 tmp = current.next;  
56 current.next = new Node<T>(val);  
57 current = current.next;  
58 current.next = tmp;  
59 }  
60 }  
61   
62 public void remove() {  
63 if (current == head) {  
64 head = head.next;  
65 } else {  
66 Node<T> tmp = head;  
67 while (tmp.next != current) {  
68 tmp = tmp.next;  
69 }  
70 tmp.next = current.next;  
71 }  
72 if (current.next == null) {  
73 current = head;  
74 } else {  
75 current = current.next;  
76 }  
77 }  
78   
79 public void display() {  
80 Node<T> tmp = head;  
81 while (tmp != null) {  
82 System.out.print(tmp.data);  
83 tmp = tmp.next;  
84   
85 }  
86 System.out.println();  
87   
88 }  
89   
90 }  
///////////////////////\*\*class Rating\*\*///////////////////////

1 public class Rating {  
 2 private int userId;  
 3 private int itemId;  
 4 private int value; // The value of the rating  
 5   
 6 // Constructor  
 7 public Rating(int userId, int itemId, int value){  
 8 this.itemId=itemId;  
 9 this.userId=userId;  
10 this.value=value;  
11 }  
12 public int getUserId(){  
13 return userId;  
14 }  
15 public int getItemId() {  
16 return itemId;   
17 }  
18 public int getValue(){  
19 return value;   
20 }  
21 public String toString(){  
22 return userId+" "+itemId+" "+value;  
23 }  
24 }  
/////////////////////\*\*class RatingManager\*\*/////////////////

1 import java.io.File;  
 2 import java.io.FileNotFoundException;  
 3 import java.util.NoSuchElementException;  
 4 import java.util.Scanner;  
 5   
 6 public class RatingManager {  
 7   
 8 private int sizeItem;  
 9 private int sizeUser;  
 10 private LinkedList<LinkedList<Rating>> ratingListItem;  
 11 private LinkedList<LinkedList<Rating>> ratingListUser;  
 12 // Constructor  
 13   
 14 public RatingManager() {  
 15 sizeItem = 0;  
 16 sizeUser = 0;  
 17 ratingListItem = new LinkedList<>();  
 18 ratingListUser = new LinkedList<>();  
 19 }  
 20   
 21 // Read ratings from a file and create a RatingManager object that stores these ratings  
 22 public static RatingManager read(String fileName) {  
 23 RatingManager result = new RatingManager();  
 24 Scanner scan=null;  
 25 try {  
 26 scan = new Scanner(new File(fileName));  
 27 while (scan.hasNextLine()) {  
 28   
 29 int userID = scan.nextInt();  
 30 int itemID = scan.nextInt();  
 31 int value = scan.nextInt();  
 32 scan.next();  
 33 Rating rating = new Rating(userID, itemID, value);  
 34 result.addRating(rating);  
 35   
 36 }  
 37 scan.close();  
 38 } catch (FileNotFoundException ex) {  
 39 ex.printStackTrace();  
 40 return null;  
 41 }  
 42 catch (NoSuchElementException ex) {  
 43 // ex.printStackTrace();  
 44 scan.close();  
 45 return result;  
 46 }  
 47 return result;  
 48 }  
 49   
 50 // Add a rating  
 51 public void addRating(Rating rating) {  
 52 addRatingItem(rating);  
 53 addRatingUser(rating);  
 54   
 55 }  
 56   
 57 private void addRatingItem(Rating rating) {  
 58 LinkedList<Rating> list = null;  
 59 if (ratingListItem.empty()) {  
 60 list = new LinkedList<>();  
 61 ratingListItem.insert(list);  
 62 sizeItem++;  
 63 } else {  
 64 ratingListItem.findFirst();  
 65 while (!ratingListItem.last()) {  
 66   
 67 list = ratingListItem.retrieve();  
 68 if (list.retrieve().getItemId() == rating.getItemId()) {  
 69 break;  
 70 } else {  
 71 list = null;  
 72 }  
 73 ratingListItem.findNext();  
 74 }//end while   
 75   
 76 if (list == null) {  
 77 list = ratingListItem.retrieve();  
 78 if (list.retrieve().getItemId() != rating.getItemId()) {  
 79 list = null;  
 80 }  
 81 }  
 82   
 83 }  
 84 if (list == null) {  
 85 list = new LinkedList<>();  
 86 ratingListItem.insert(list);  
 87 sizeItem++;  
 88 }  
 89 list.insert(rating);  
 90   
 91   
 92 }  
 93   
 94 private void addRatingUser(Rating rating) {  
 95 LinkedList<Rating> list = null;  
 96 if (ratingListUser.empty()) {  
 97 list = new LinkedList<>();  
 98 ratingListUser.insert(list);  
 99 sizeUser++;  
100 } else {  
101 ratingListUser.findFirst();  
102 while (!ratingListUser.last()) {  
103   
104 list = ratingListUser.retrieve();  
105 if (list.retrieve().getUserId() == rating.getUserId()) {  
106 break;  
107 } else {  
108 list = null;  
109 }  
110 ratingListUser.findNext();  
111 }  
112 if (list == null) {  
113 list = ratingListUser.retrieve();  
114 if (list.retrieve().getUserId() != rating.getUserId()) {  
115 list = null;  
116 }  
117 }  
118   
119 }  
120 if (list == null) {  
121 list = new LinkedList<>();  
122 ratingListUser.insert(list);  
123 sizeUser++;  
124 }  
125 list.insert(rating);  
126   
127   
128 }  
129   
130 // Return all ratings given by user i. Search should be efficient.   
131 public LinkedList<Rating> getUserRatings(int i) {  
132 LinkedList<Rating> result = null;  
133 ratingListUser.findFirst();  
134 for (int j = 0; j < sizeUser; j++) {  
135 Rating r = ratingListUser.retrieve().retrieve();  
136 if (r.getUserId() == i) {  
137 result = ratingListUser.retrieve();  
138 return result;  
139 }  
140 ratingListUser.findNext();  
141   
142 }  
143 return null;  
144 }  
145   
146 // Return all ratings given to item j. Search should be efficient.  
147 public LinkedList<Rating> getItemRatings(int i) {  
148 LinkedList<Rating> result = null;  
149 ratingListItem.findFirst();  
150 for (int j = 0; j < sizeItem; j++) {  
151 Rating r = ratingListItem.retrieve().retrieve();  
152 if (r.getItemId() == i) {  
153 result = ratingListItem.retrieve();  
154 return result;  
155 }  
156 ratingListItem.findNext();  
157   
158 }  
159 return null;  
160 }  
161   
162 // Return the average rating of item j. If j has no ratings, -1 is returned  
163 public double getAverageItemRating(int i) {  
164 double avg = 0;  
165 double sum = 0;  
166 int count = 0;  
167 LinkedList<Rating> list = this.getItemRatings(i);  
168 if(list==null)  
169 return -1;  
170 list.findFirst();  
171 while (!list.last()) {  
172 Rating r = list.retrieve();  
173   
174 if (r.getItemId() == i) {  
175 count++;  
176 sum += r.getValue();  
177 }  
178 list.findNext();  
179 }  
180 Rating r = list.retrieve();  
181   
182 if (r.getItemId() == i) {  
183 count++;  
184 sum += r.getValue();  
185 }  
186   
187 if (count == 0) {  
188 return -1;  
189 }  
190 avg = sum / count;  
191 return avg;  
192 }  
193   
194 // Return the average rating given by user i. If i has no ratings, -1 is returned  
195 public double getAverageUserRating(int i) {  
196 double avg = 0;  
197 double sum = 0;  
198 int count = 0;  
199 LinkedList<Rating> list = this.getUserRatings(i);  
200 if(list==null)  
201 return -1;  
202 list.findFirst();  
203 while (!list.last()) {  
204 Rating r = list.retrieve();  
205   
206 if (r.getUserId() == i) {  
207 count++;  
208 sum += r.getValue();  
209 }  
210 list.findNext();  
211 }  
212 Rating r = list.retrieve();  
213   
214 if (r.getUserId() == i) {  
215 count++;  
216 sum += r.getValue();  
217 }  
218   
219 if (count == 0) {  
220 return -1;  
221 }  
222 avg = sum / count;  
223 return avg;  
224 }  
225   
226 // Return the list of all items having the highest average rating (for example if the highest average rating is 4.9, the method should return all items with average rating 4.9)  
227 public LinkedList<Integer> getHighestRatedItems() {  
228 // this.ratingListItem.display();  
229 LinkedList<Integer> result = new LinkedList<>();  
230 double[] avg = new double[sizeItem];  
231 double maxAvg = 0;  
232 ratingListItem.findFirst();  
233 for (int i = 0; i < sizeItem; i++) {  
234 Rating r = ratingListItem.retrieve().retrieve();  
235 avg[i] = this.getAverageItemRating(r.getItemId());  
236 if (maxAvg < avg[i]) {  
237 maxAvg = avg[i];  
238 result = new LinkedList<>();  
239 result.insert(r.getItemId());  
240 } else if (maxAvg == avg[i]) {  
241 result.insert(r.getItemId());  
242 }  
243   
244 ratingListItem.findNext();  
245 }//  
246 System.out.println("max avg: "+ maxAvg);  
247 return result;  
248   
249 }  
250   
251 }  
252