T.C.
ANADOLU UNIVERSITY
ENGINEERING FACULTY

EEM 480 Algorithms and Complexity

Homework-4 Report

Student:

Name-Surname: NURİ ÖZBEY
ID Number : ---
Department : Electric-Electronic Engineering

25.12.2017 ESKİŞEHİR

1.Purpose

The purpose of the homework is build a simple Facebook-Like Environment. This data structure builded with hash structure. Our program work in the command line of NetBeans environment.

2.Background

Java is a computer programming language. It is easy to use that came from C++ and reliable to the design Object-Oriented Programming. Java was originally targeting mobile devices that would be exchanging data over networks, it was built to include a high level of security. Java is probably the most secure programming language because it's a virtual machine. It is compatible with Linux, Mac and Windows.

A Java Development Kit (JDK) is a program development environment for writing Java applets and applications. Also, developers have to install this kit because they need to compile, debug, and run applets and applications written in the Java language

NetBeans is a Java-based integrated development environment (IDE) which is designed to limit the coding errors. Although the NetBeans IDE is designed specifically for Java developers, it also supports C/C++, PHP, Groovy, and HTML5 in addition to Java, JavaScript and JavaFX. Tools and capabilities of the NetBeans IDE include a feature-rich text editor with refactoring tools and code templates, high level and granular views of applications, a drag and drop GUI design, and versioning using out-of-the-box integration with tools such as Git and Bitbucked. Shortly, NetBeans is a complier which is designed to correct codes and help to the devolopers.

Linked list is work like an Array. In array the length of array must be fixed and this type of usage sometimes good sometimes bad because if we define a huge number of array length which is never fulled, the memory filled by this array and we cannot do anything more. However in Link list we don't need to define length of list, how much element we want we can push on it. The other main difference is easy to add and delete item on list. We can push an item with complexity O(1) and we can take a selected item on list with complexity O(n).

Hash is a structure type which is the best for timing. The program Works with a hash function which is created by programmer. The String inputs quickly transformed to the shortest fixed length of an integer value. The value generally called Hash-Key. Hashing method and building key also used in encryption algorithms. Hash is like a huge amount of array which indexed with hash-key and contains the element.

This homework we Create a hash structure and the hash contains people on it. And do simple operations which are Insert a person, Friend two person, Delete the friendship of two people, Erase the person and all their friendships. List a person friends, check two people is friend or not, read a text file from the disk and implement all functions on it and exit the program. The main critical part is inserting a person, friending two person and checking two people are friend each other complexity must be O(1). And the other functions should be O(n) complexity not more.

3.Homework Algorithm

My hash table was builded with an array of length is 997 and every array elements points the link lists and every link lists also points the another link list to get rid of over hashing.

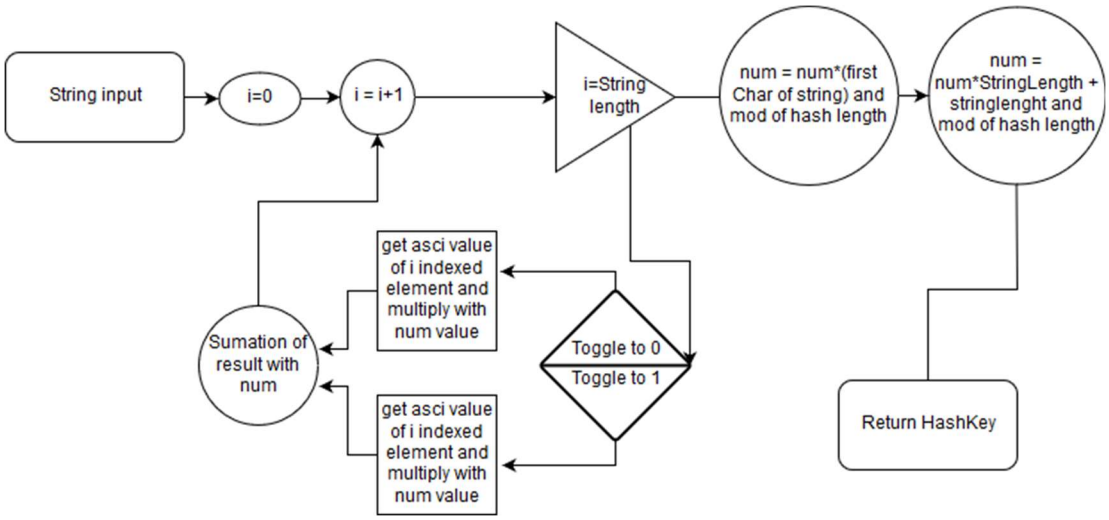

Figure-1: Hash Key of My Hash Structure

As it seen in the figure-1 the Hash key generated by manipulating the string char elements. The toqqle function implements for breaking the over hashing and insert different blank hash table place.

```
public interface HashInterface {
    public String Insert(String fName);                          //O(1)
    public void Friend(String fName,String sName);               //O(1)
    public void DeleteFriendship(String fName,String sName);     //O(n)
    public void Erase(String fName);                            //O(n)
    public void List(String fName);                              //O(n)
    public boolean ScheckF(String fName,String sName);           //O(1)
    public void Ofriends(String fName);                          //O(n)
}
```

Figure-2: Hash interface of Program

As it seen in the figure-2 the hash interface builded firstly then started to implement other functions. The interface show us which function must be which complexity.

```
28      public String Insert(String fName) {
29          int hashkey = getHashKey(fName, hashsize);
30          //myHashArr[hashkey];
31          int innersize = Myhasharr[hashkey].getInnersize();
32          Myhasharr[hashkey].InsertList(fName, innersize);
33          //myHashArr[hashkey].InsertList("ali", 0);
34          return fName;
        }
```

Figure-3: Insert a Person Case

As it seen in the figure-3 we take ınput string and get the hash key and write to the hash. The insertion of a person complexity is O(1).

```
38      public void Friend(String fName, String sName) {
39          if(FindPerson(fName)){
40              if(FindPerson(sName)){
41                  try {
42                      LNode2 asd = Myhasharr[getHashKey(fName, hashsize)].getNode();
43                      //System.out.println("1 Node:: " +asd.Element2);
44                      addfriend(asd,fName,sName);
45                      //Myhasharr[getHashKey(fName, hashsize)].Element2;
46                  } catch (Exception ex) {
47                      Logger.getLogger(MyhashF.class.getName()).log(Level.SEVERE, null, ex);
48                  }
49                  try {
50                      LNode2 asd = Myhasharr[getHashKey(sName, hashsize)].getNode();
51                      //System.out.println("1 Node:: " +asd.Element2);
52                      addfriend(asd,sName,fName);
53                      //Myhasharr[getHashKey(fName, hashsize)].Element2;
54                  } catch (Exception ex) {
55                      Logger.getLogger(MyhashF.class.getName()).log(Level.SEVERE, null, ex);
56                  }
57                  String name1 = fName+"@"+sName;
58                  Insert(name1);
59                  name1 = sName+"@"+fName;
60                  Insert(name1);
61              }else{
62                  System.out.println("[ERROR] "+fName+" Not found..!");
63              }
64          }else{
65              System.out.println("[ERROR] "+fName+" Not found..!");
66          }
        }
```

Figure-4: Friend two People Case

As it seen in the figure-4, firstly we check the person exist on the list or not if tow of them are exist we add the inner friend list of person and do same operation for other person. Then build two string with using two person name and add to the hash table to use the check friend operation. The friends of two people complexity is O(1).

```
178     public int addfriend(LNode2 mynode,String fName,String sName) throws Exception {
179         if(fName.equals(mynode.Element2)){
180             int size = mynode.InList.getsize();
181             mynode.InList.Insert(sName, size);
182             return 1;
183         }else{
184             addfriend(mynode.link2,fName,sName);
185 
186             return 0;
187         }
188     }
```

Figure-5: Friend two People Case with Recursive manner

As it seen in the figure-5, for addition a friend firstly checks array link list for the current person and after finding current person we add the friend name to the his or her inner link list.

```
121     public boolean ScheckF(String fName, String sName) {
122         String temp= fName+"@"+sName;
123         if(FindPerson(fName)){
124             if(FindPerson(sName)){
125             }else{
126                 System.out.println("[ERROR] "+sName+" Not found..!");
127             }
128         }else{
129             System.out.println("[ERROR] "+fName+" Not found..!");
130         }
131 
132         //int size = getHashKey(temp, hashsize);
133         //String temp2 = (String) Myhasharr[size].Element2;
134         return FindPerson(temp);
135     }
```

Figure-6: Are two People Friend Case

As it seen in the figure-6 , directly build a string and find hash address if it exist return true if not return false. The check two people are friends complexity is O(1).

```
143     public boolean FindPerson(String fName){
144         LNode2 Dummy ;
145 
146         Dummy = Myhasharr[getHashKey(fName, hashsize)].getNode();
147         if(null == Dummy){
148             return false;
149         }else{
150             while(null != Dummy){
151                 if(fName.equals(Dummy.Element2)){
152                     return true;
153                 }
154                 Dummy = Dummy.link2;
155             }
156         }
157         return false;
158     }
```

Figure-7: Are two People Friend Case Find Person in hash table

As it seen in the figure-7, general implementation of find person the finding person complexity is also O(1).

```
103     public void Erase(String fName) {
104       if(FindPerson(fName)){
105             LNode2 asd = Myhasharr[getHashKey(fName, hashsize)].getNode();
106             ErasePerson(asd,fName);
107         }
108     }
```

Figure-8: Delete a person case(implemented erase)

As it seen in the figure-8, to delete a person we called erase person function.

```
277     public int ErasePerson(LNode2 mynode,String fName)  {
278         if(fName.equals(mynode.Element2)){
279             LNode2 ret = Myhasharr[getHashKey(fName, hashsize)].getNode();
280             int listsize = ret.InList.getsize();
281             MyListOfLists blank = new MyListOfLists();
282             if(listsize == 0){
283                 Myhasharr[getHashKey(fName, hashsize)] = blank;
284             }else{
285                 for (int i = 0; i < listsize; i++) {
286                     try {
287                         //Person temp = new Person();
288                         String temp = ret.InList.getNodeData(0);
289                         DeleteFriendship(temp, fName);
290 //                          if(temp == null){
291 //                              Myhasharr[getHashKey(fName, hashsize)] = blank;
292 //                          }
293                     } catch (Exception ex) {
294                         Logger.getLogger(MyhashF.class.getName()).log(Level.SEVERE, null, ex);
295                     }
296                 }
297                 Myhasharr[getHashKey(fName, hashsize)] = blank;
298             }
299             return 1;
300         }else{
301             ErasePerson(mynode.link2,fName);
302             return 0;
303         }
304     }
```

Figure-9: Delete a person with using erase friendship Recursive manner

As it seen in the figure-9 the delete a person case repeated for all friends of the person. The function goes to his or her friends and delete name and return back after finishing the delete operations the person deleted from the array list with complexity of O (n).

```
public void DeleteFriendship(String fName, String sName) {
    if(FindPerson(fName)){
        if(FindPerson(sName)){
            try {
                LNode2 asd = Myhasharr[getHashKey(fName, hashsize)].getNode();
                //System.out.println("1 Node:: " +asd.Element2);

                deletefriend(asd,fName,sName,0);
            } catch (Exception ex) {
                Logger.getLogger(MyhashF.class.getName()).log(Level.SEVERE, null, ex);
            }
            try {
                LNode2 asd = Myhasharr[getHashKey(sName, hashsize)].getNode();
                //System.out.println("1 Node:: " +asd.Element2);
                deletefriend(asd,sName,fName,0);
                //Myhasharr[getHashKey(fName, hashsize)].Element2;
            } catch (Exception ex) {
                Logger.getLogger(MyhashF.class.getName()).log(Level.SEVERE, null, ex);
            }
            String name1 = fName+"@"+sName;
            Erase(name1);
            name1 = sName+"@"+fName;
            Erase(name1);
        }else{
            System.out.println("[ERROR] "+fName+" Not found..!");
        }
    }else{
        System.out.println("[ERROR] "+fName+" Not found..!");
    }
```

Figure-10: Erase the friendship between two person

As it seen in the figure-10 the erase two person friendship with complexity of O (n).

```
public int deletefriend(LNode2 mynode,String fName,String sName,int pos) throws Exception {
    if(fName.equals(mynode.Element2)){
        //DeleteInner(pos, mynode.InList);
        DeleteInlist(mynode.InList,sName,0);
        //mynode.InList.Delete(pos);
        return 1;
    }else{
        LNode2 innode = mynode.link2;
        deletefriend(innode,fName,sName,pos++);
        return 0;
    }
}
```

Figure-11: Delete a person with using erase friendship Recursive manner

As it seen in the figure-11 the recursive function used to detect person and delete friendliest element with complexity of O (n).

```
public void DeleteInlist(MyLinkedList list,String dName,int pos){

    int size = list.getsize();
    int delpos = 0;
    for (int i = 0; i < size; i++) {
        try {
            if(dName.equals(list.getNodeData(i))){
                delpos = i;
            }
        } catch (Exception ex) {
            Logger.getLogger(MyhashF.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    try {
        list.Delete(delpos);
    } catch (Exception ex) {
        Logger.getLogger(MyhashF.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Figure-12: Delete a person with using erase friendship Recursive manner

As it seen in the figure-12 the delete operation applied for friends of friendlist. with complexity of O (n).

```
111     public void List(String fName) {
112         if(FindPerson(fName)){
113             LNode2 asd = Myhasharr[getHashKey(fName, hashsize)].getNode();
114             listFriends(asd,fName);
115         }else{
116             System.out.println("[ERROR] "+fName+" Not found..!");
117         }
    }
```

Figure-13: List Friends of a person

As it seen in the figure-13 the list of friends of a person applied with complexity of O (n).

```
226     public int listFriends(LNode2 mynode,String fName)  {
227         if(fName.equals(mynode.Element2)){
228             LNode2 ret = Myhasharr[getHashKey(fName, hashsize)].getNode();
229             //ret.InList.Output();
230             int size = ret.InList.getsize();
231             if(size == 0){
232                 System.out.println(fName+" has no friend..!");
233             }
234             for (int i = 0; i < size; i++) {
235                 String geri = "";
236                 try {
237                      geri = ret.InList.getNodeData(i);
238                 } catch (Exception ex) {
239                     Logger.getLogger(MyhashF.class.getName()).log(Level.SEVERE, null, ex);
240                 }
241                 System.out.println(geri+" ");
242             }
243             return 1;
244         }else{
245             listFriends(mynode.link2,fName);
246             return 0;
247         }
248     }
```

Figure-14: List Friends of a person with recursive manner

```
138     public void Ofriends(String fName) {
139         if(FindPerson(fName)){
140             LNode2 asd = Myhasharr[getHashKey(fName, hashsize)].getNode();
141             outFriends(asd,fName);
142         }
    }
```

Figure-15: Out of Friends of Friends

As it seen in the figure-15 the out of friends of friends implemented and called outfriend function.

```
243     public int outFriends(LNode2 mynode,String fName)  {
244         if(fName.equals(mynode.Element2)){
245             LNode2 ret = Myhasharr[getHashKey(fName, hashsize)].getNode();
246             int size = ret.InList.getsize();
247             for (int i = 0; i < size; i++) {
248                 try {
249                     String arki = ret.InList.getNodeData(i);
250                     LNode2 arkiark = Myhasharr[getHashKey(arki, hashsize)].getNode();
251                     int innersize = arkiark.InList.getsize();
252                     if(innersize > 1 ){
253                         System.out.println(arki);
254                         for (int j = 0; j < innersize; j++) {
255                             String arkininarki = arkiark.InList.getNodeData(j);
256                             if(!ScheckF(fName,arkininarki)){
257                                 //System.out.println("True They are Friends Already");
258                                 if (!(fName.equals(arkininarki))) {
259                                     System.out.print("*");
260                                     System.out.println(arkininarki);
261                                 }
262                             }
263                         }
264                     }
265 
266                 } catch (Exception ex) {
267                     Logger.getLogger(MyhashF.class.getName()).log(Level.SEVERE, null, ex);
268                 }
269             }
270             //ret.InList.Output();
271             return 1;
272         }else{
273             outFriends(mynode.link2,fName);
274             return 0;
275         }
```

Figure-16: Out of Friends of Friends with recursive manner.

As it seen in the figure-16 the out friend function find persons and compare with itself and return and write with complexity of O (n).

Main Program Implementations

```
38     public static int myCommand(String myString) {
39         String[] parsedInput = myString.split(" ");
40         if((parsedInput[0].charAt(0) == 'X' || parsedInput[0].charAt(0) == 'x')&& parsedInput[0].length() <= 1){
41             System.out.println("Program Closing... ");
42             System.out.println("Have a good day :)");
43             return 1;
44         }else if((parsedInput[0].charAt(0) == 'R' || parsedInput[0].charAt(0) == 'r')&& parsedInput.length >= 2){
45             try {
46                 // Open the file that is the first
47                 // command line parameter
48                 //FileInputStream fstream = new FileInputStream("C:\\Users\\nurio\\Desktop\\asd.txt");
49                 FileInputStream fstream = new FileInputStream(parsedInput[1]);
50                 // Get the object of DataInputStream
51                 DataInputStream in = new DataInputStream(fstream);
52                 BufferedReader br = new BufferedReader(new InputStreamReader(in));
53                 String strLine;
54                 //Read File Line By Line
55                 while ((strLine = br.readLine()) != null) {
56                     // Print the content on the console
57                     //System.out.println(strLine);
58                     int res = myCommand(strLine);
59                     if (res == 1) {
60                         return 1;
61                     }
62                 }
63                 //Close the input stream
64             } catch (IOException e) {//Catch exception if any
65                 System.err.println("Error: " + e.getMessage());
66             }
```

Figure-17: Main loop and Read and Exit Statement implementations

```
67             }else if((parsedInput[0].charAt(0) == 'I' || parsedInput[0].charAt(0) == 'i')&& parsedInput.length >= 2){
68                 myhash.Insert(parsedInput[1]);
```

Figure-18: insert a person case

```
70          }else if((parsedInput[0].charAt(0) == 'F' || parsedInput[0].charAt(0) == 'f')&& parsedInput.length >= 2){
71              //ystem.out.println(parsedInput.length);
72              if(parsedInput.length > 1){
73                  if(parsedInput.length > 2){
74                      myhash.Friend(parsedInput[1],parsedInput[2]);
75                      //System.out.println("Person ["+parsedInput[1]+"] and ["+parsedInput[2]+"] are friends");//ka
76                  }else{
77                      System.out.println("[ERROR] Second person can not be blank..!");
78                  }
79              }else{
80                  System.out.println("[ERROR] First person can not be blank..!");
81              }
```

Figure-19: Friend two input person

```
92          }else if((parsedInput[0].charAt(0) == 'D' || parsedInput[0].charAt(0) == 'd')&& parsedInput.length >= 2){
93              myhash.Erase(parsedInput[1]);
94              System.out.println("Person ["+parsedInput[1]+"] deleted");
```

Figure-20: Delete a person and they all friendships

```
95          }else if((parsedInput[0].charAt(0) == 'E' || parsedInput[0].charAt(0) == 'e')&& parsedInput.length >= 2){
96
97              if(parsedInput.length > 1){
98                  if(parsedInput.length > 2){
99                      if(myhash.ScheckF(parsedInput[1],parsedInput[2])){
100                         myhash.DeleteFriendship(parsedInput[1],parsedInput[2]);
101                     }else{
102                         System.out.println(parsedInput[1]+" has No Friend..!");
103                     }
104                     //System.out.println("Person ["+parsedInput[1]+"] and ["+parsedInput[2]+"] are friends");//kal
105                 }else{
106                     System.out.println("[ERROR] Second person can not be blank..!");
107                 }
108             }else{
109                 System.out.println("[ERROR] First person can not be blank..!");
110             }
111             //System.out.println("person deleted");
```

Figure-21: Erase friendship of two person

```
89          }else if((parsedInput[0].charAt(0) == 'L' || parsedInput[0].charAt(0) == 'l')&& parsedInput.length >= 2){
90              myhash.List(parsedInput[1]);
91
```

Figure-22: List a person Friends

```
112         }else if((parsedInput[0].charAt(0) == 'O' || parsedInput[0].charAt(0) == 'o')&& parsedInput.length >= 2){
113             myhash.Ofriends(parsedInput[1]);
114             //System.out.println("Friends Out");
```

Figure-23: Friends of Friends show statement

3.Homework Project User Guide

- Step-1:Firstly, you have to choose open a Project java application.
- Step-2 Find the Project file on your disk.
- Step-3:Choose and click the open Project button.

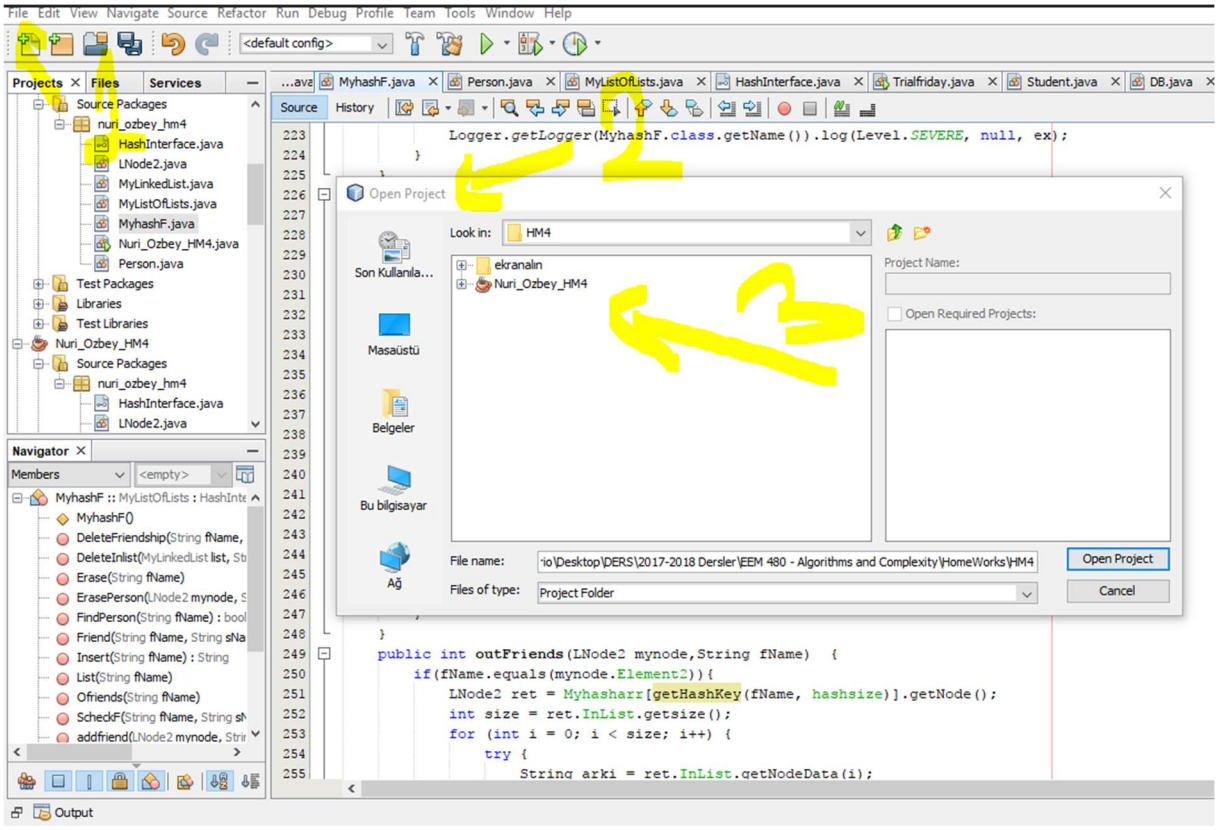

Figure-24: Step first to tree

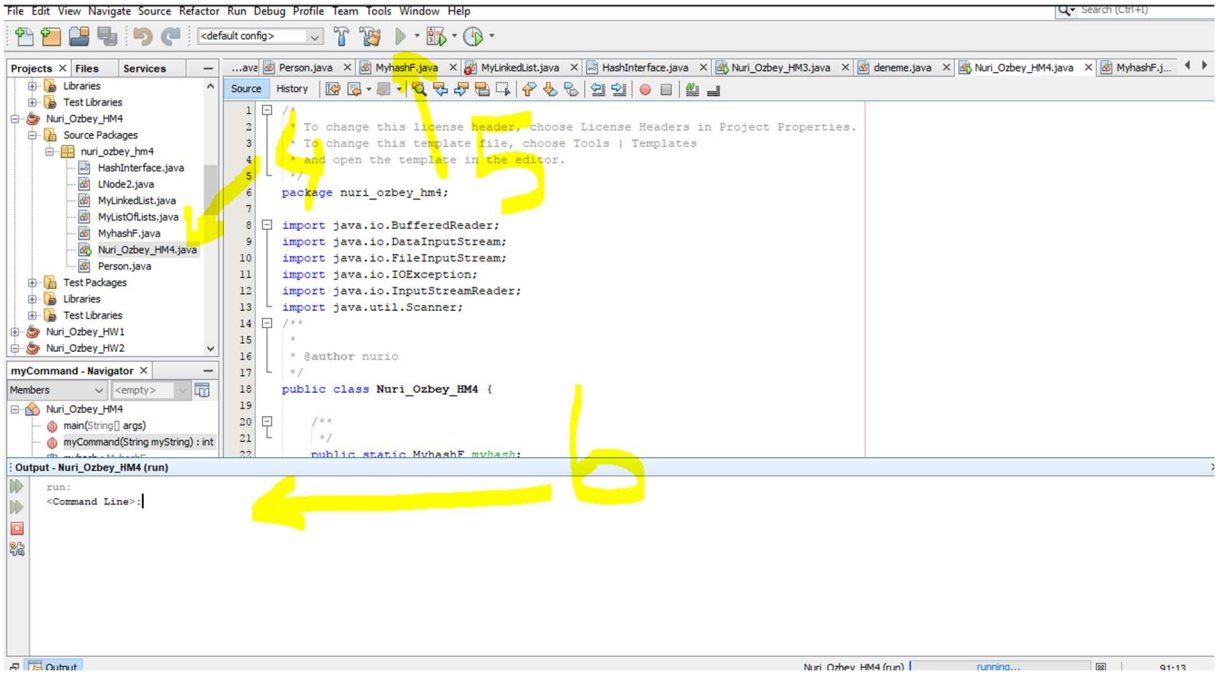

Figure-25: Step four to Step six

- Step-4: choose the runnable java class.
- Step-5: click the run button
- Step-6: Enter whatever you want

4.Conculusion

We have learned from this homework how to use hash tables and how to build a linked list of linked lists. And also learned how to make a uniform hash key function. Also another thing how can we solve the over hashing problem on the program and how can we manipulate the linked list elements with using hash functions.