Homework Turnin

Account:	6G_06 (rgalanos@fcps.edu)
Section:	6G
Course:	TJHSST APCS 2016-17
Assignment:	08-01
Receipt ID:	d04e6b30cff688d985b8957adc80f2c9

Turnin Successful!

The following file(s) were received:

Hashing.java (6843 bytes)

```
1.  //name:    date:
2.  /*  Assignment:  This hashing program results in collisions.
3.      You are to implement three different collision schemes: linear
4.      probing, rehashing, and chaining.  Then implement a search
5.      algorithm that is appropriate for each collision scheme.
6.  */
7.  import java.util.*;
8.  import javax.swing.*;
9.  public class Hashing
10. {
11.    public static void main(String[] args)
12.    {
13.       int arrayLength = Integer.parseInt(JOptionPane.showInputDialog(
14.                         "Hashing!\n"+
15.                         "Enter the size of the array:  "));//20
16.
17.       int numItems = Integer.parseInt(JOptionPane.showInputDialog(
18.                         "Add n items:  "));               //15
19.
20.       int scheme = Integer.parseInt(JOptionPane.showInputDialog(
21.            "The Load Factor is " + (double)numItems/arrayLength +
22.            "\nWhich collision scheme?\n"+
23.            "1. Linear Probing\n" +
24.            "2. Rehashing\n"+
25.            "3. Chaining"));
26.       Hashtable table = null;
27.       switch( scheme )
28.       {
29.          case 1:
30.             table = new HashtableLinearProbe(arrayLength);
31.             break;
32.          case 2:
33.             table = new HashtableRehash(arrayLength);
34.             break;
35.          case 3:
36.             table = new HashtableChaining(arrayLength);
37.             break;
38.          default:  System.exit(0);
39.       }
40.       for(int i = 0; i < numItems; i++)
41.          table.add("Item" + i);
42.       int itemNumber = Integer.parseInt(JOptionPane.showInputDialog(
43.                         "Search for:  Item0" + " to "+ "Item"+(numItems-1)));
44.       while( itemNumber != -1 )
45.       {
46.          String key = "Item" + itemNumber;
47.          int index = table.indexOf(key);
48.          if( index >= 0)    //found it
49.             System.out.println(key + " found  at index " + index);
50.          else
```

```
51.            System.out.println(key + " not found!");
52.         itemNumber = Integer.parseInt(JOptionPane.showInputDialog(
53.                         "Search for:  Item0" + " to "+ "Item"+(numItems-1)));
54.      }
55.      System.exit(0);
56.   }
57. }
58. /*********************************************/
59. interface Hashtable
60. {
61.   void add(Object obj);
62.   int indexOf(Object obj);
63. }
64. /*************************************************/
65. class HashtableLinearProbe implements Hashtable
66. {
67.   private Object[] array;
68.   public HashtableLinearProbe(int size)
69.   {
70.      array = new Object[size];              //constructor
71.   }
72.   public void add(Object obj)
73.   {
74.      int code = obj.hashCode();
75.      int index = Math.abs(code % array.length);
76.      if(array[index]==null)  //empty
77.      {
78.         array[index]=obj;   //insert it
79.         System.out.println(obj + "\t" + code + "\t" + index);
80.      }
81.      else //collision
82.      {
83.         System.out.println(obj + "\t" + code + "\tCollision at "+ index);
84.         index = linearProbe(index);
85.         array[index] = obj;
86.         System.out.println(obj + "\t" + code + "\t" + index);
87.      }
88.   }
89.   public int linearProbe(int index)
90.   {
91.      //be sure to wrap around the array.
92.      while(array[index]!=null)
93.      {
94.         if(index==array.length-1)
95.            index = 0;
96.         else
97.            index++;
98.      }
99.
100.     return index;
101.  }
102.  public int indexOf(Object obj)
103.  {
104.     int index = Math.abs(obj.hashCode() % array.length);
105.     while(array[index] != null)
106.     {
107.        if(array[index].equals(obj))  //found it
108.        {
109.           return index;
110.        }
111.        else //search for it in a linear probe manner
112.        {
113.           System.out.println("Looking at index " + index);
114.           index++;
115.        }
116.     }
117.     return -1;     //not found
118.  }
119. }
120. /*****************************************************/
121. class HashtableRehash implements Hashtable
122. {
123.   private Object[] array;
124.   private int constant = 2;
125.   public HashtableRehash(int size)
126.   {
127.      array = new Object[size];
128.      while(gcd(constant,size)!=1)
129.         constant++;               //constructor
130.                           //find a constant that is relatively prime to the size of the array
131.   }
```

```
132.    private int gcd(int x, int y)
133.    {
134.       int temp;
135.       while(y!=0)
136.       {
137.          temp = x;
138.          x = y;
139.          y = temp%y;
140.       }
141.       return x;
142.    }
143.    public void add(Object obj)
144.    {
145.       int code = obj.hashCode();
146.       int index = Math.abs(code % array.length);
147.       if( array[index]==null )  //empty
148.       {
149.          array[index] = obj;  //insert it
150.          System.out.println(obj + "\t" + code + "\t" + index);
151.       }
152.       else //collision
153.       {
154.          System.out.println(obj + "\t" + code + "\tCollision at "+ index);
155.          while(array[index]!=null)
156.             index = rehash(index);
157.          array[index] = obj;
158.          System.out.println(obj + "\t" + code + "\t" + index);
159.       }
160.    }
161.    public int rehash(int index)
162.    {
163.       int x = (index+constant) % array.length;
164.       return x;
165.    }
166.    public  int indexOf(Object obj)
167.    {
168.       int index = Math.abs(obj.hashCode() % array.length);
169.       while(array[index] != null)
170.       {
171.          if( array[index].equals(obj) )  //found it
172.          {
173.             return index;
174.          }
175.          else //search for it in a rehashing manner
176.          {
177.             index = rehash(index);
178.             System.out.println("Looking at index " + index);
179.          }
180.       }
181.       return -1;  //not found
182.    }
183. }
184. /*********************************************************/
185. class HashtableChaining implements Hashtable
186. {
187.    private LinkedList[] array;
188.    public HashtableChaining(int size)
189.    {
190.       array = new LinkedList[size];         //instantiate the array
191.       for(int i=0;i<array.length;i++)                    //instantiate the LinkedLists
192.          array[i] = new LinkedList();
193.    }
194.    public void add(Object obj)
195.    {
196.       int code = obj.hashCode();
197.       int index = Math.abs(code % array.length);
198.       array[index].addFirst(obj);
199.       System.out.println(obj + "\t" + code + " " + " at " +index + ": "+ array[index]);
200.    }
201.    public int indexOf(Object obj)
202.    {
203.       int index = Math.abs(obj.hashCode() % array.length);
204.       if( !array[index].isEmpty() )
205.       {
206.          if(array[index].getFirst().equals(obj))  //found it
207.          {
208.             return index;
209.          }
210.          else //search for it in a chaining manner
211.          {
212.             int x = 1;
```

```
213.             while(x<array[index].size())
214.             {
215.                if(array[index].get(x).equals(obj))
216.                   return index;
217.                else
218.                   x++;
219.             }
220.          }
221.       }
222.       return -1;     //not found
223.    }
224. }
```