```java
import java.util.*;

// implementing hash tables as an array of linked lists

class Node{

    private int data;
    private Node nextNodePtr;


    public Node(){}

    public void setData(int d){
            data = d;
    }

    public  int getData(){
            return data;
    }

    public  void setNextNodePtr(Node nodePtr){
            nextNodePtr = nodePtr;
    }

    public  Node getNextNodePtr(){
        return nextNodePtr;
    }

}

class List{

    private Node headPtr;


    public  List(){
        headPtr = new Node();
        headPtr.setNextNodePtr(null);
    }


    public Node getHeadPtr(){
        return headPtr;
    }

    public  boolean isEmpty(){

        if (headPtr.getNextNodePtr() == null)
            return true;

        return false;
    }


    public  void insert(int data){

            Node currentNodePtr = headPtr.getNextNodePtr();
            Node prevNodePtr = headPtr;

            while (currentNodePtr != null){
                prevNodePtr = currentNodePtr;
                currentNodePtr = currentNodePtr.getNextNodePtr();
            }

```

```
65              Node newNodePtr = new Node();
66              newNodePtr.setData(data);
67              newNodePtr.setNextNodePtr(null);
68              prevNodePtr.setNextNodePtr(newNodePtr);
69
70      }
71
72      public  void insertAtIndex(int insertIndex, int data){
73
74              Node currentNodePtr = headPtr.getNextNodePtr();
75              Node prevNodePtr = headPtr;
76
77              int index = 0;
78
79              while (currentNodePtr != null){
80
81                  if (index == insertIndex)
82                      break;
83
84                  prevNodePtr = currentNodePtr;
85                  currentNodePtr = currentNodePtr.getNextNodePtr();
86                  index++;
87              }
88
89              Node newNodePtr = new Node();
90              newNodePtr.setData(data);
91              newNodePtr.setNextNodePtr(currentNodePtr);
92              prevNodePtr.setNextNodePtr(newNodePtr);
93
94      }
95
96
97      public  int read(int readIndex){
98
99              Node currentNodePtr = headPtr.getNextNodePtr();
100             Node prevNodePtr = headPtr;
101             int index = 0;
102
103             while (currentNodePtr != null){
104
105                 if (index == readIndex)
106                     return currentNodePtr.getData();
107
108                 prevNodePtr = currentNodePtr;
109                 currentNodePtr = currentNodePtr.getNextNodePtr();
110
111                 index++;
112
113             }
114
115         return -1; // an invalid value indicating
116                    // index is out of range
117
118     }
119
120     public  void modifyElement(int modifyIndex, int data){
121
122             Node currentNodePtr = headPtr.getNextNodePtr();
123             Node prevNodePtr = headPtr;
124             int index = 0;
125
126             while (currentNodePtr != null){
127
128                 if (index == modifyIndex){
```

```
                    currentNodePtr.setData(data);
                    return;
                }

                prevNodePtr = currentNodePtr;
                currentNodePtr = currentNodePtr.getNextNodePtr();

                index++;
            }


    }


    public  boolean deleteElement(int data){


            Node currentNodePtr = headPtr.getNextNodePtr();
            Node prevNodePtr = headPtr;
            Node nextNodePtr = headPtr;


            while (currentNodePtr != null){

                if (currentNodePtr.getData() == data){
                    nextNodePtr = currentNodePtr.getNextNodePtr();
                    prevNodePtr.setNextNodePtr(nextNodePtr);
                    return true;
                }

                prevNodePtr = currentNodePtr;
                currentNodePtr = currentNodePtr.getNextNodePtr();

            }

            return false;

    }

    public  int countList(){

            Node currentNodePtr = headPtr.getNextNodePtr();
            int numElements = 0;

            while (currentNodePtr != null){

                numElements++;
                currentNodePtr = currentNodePtr.getNextNodePtr();

            }

            return numElements;
    }


    public  void IterativePrint(){

            Node currentNodePtr = headPtr.getNextNodePtr();

            while (currentNodePtr != null){
                System.out.print(currentNodePtr.getData()+" ");
                currentNodePtr = currentNodePtr.getNextNodePtr();
            }

```

```
193             System.out.println();
194
195     }
196
197
198     public boolean containsElement(int data){
199
200         Node currentNodePtr = headPtr.getNextNodePtr();
201
202         while (currentNodePtr != null){
203
204             if (currentNodePtr.getData() == data)
205                 return true;
206
207             currentNodePtr = currentNodePtr.getNextNodePtr();
208         }
209
210         return false;
211
212     }
213
214
215 }
216
217
218 class Hashtable{
219
220     private List[] listArray;
221     private int tableSize;
222
223
224     public Hashtable(int size){
225         tableSize = size;
226         listArray = new List[size];
227         for (int index = 0; index < size; index++)
228             listArray[index] = new List();
229     }
230
231     public  int getTableSize(){
232         return tableSize;
233     }
234
235     public  void insert(int data){
236
237         int hashIndex = data % tableSize;
238         listArray[hashIndex].insert(data);
239
240     }
241
242     public void deleteElement(int data){
243
244         int hashIndex = data % tableSize;
245         while (listArray[hashIndex].deleteElement(data));
246
247     }
248
249     public  boolean hasElement(int data){
250
251         int hashIndex = data % tableSize;
252         return listArray[hashIndex].containsElement(data);
253
254     }
255
256     public  void printHashTable(){
```

```
257
258         for (int hashIndex = 0; hashIndex < tableSize; hashIndex++){
259                 System.out.print("Hash Index: " + hashIndex + " : ");
260                 listArray[hashIndex].IterativePrint();
261         }
262
263     }
264
265 }
266
267
268 class HashTableLinkedList{
269
270     public static void main(String[] args){
271
272     Scanner input = new Scanner(System.in);
273
274     int numElements;
275     System.out.print("Enter the number of elements you want to store in the hash table:
    ");
276     numElements = input.nextInt();
277
278     int maxValue;
279     System.out.print("Enter the maximum value for an element: ");
280     maxValue = input.nextInt();
281
282     int hashTableSize;
283     System.out.print("Enter the size of the hash table: ");
284     hashTableSize = input.nextInt();
285
286     Hashtable hashTable = new Hashtable(hashTableSize);
287
288     Random randGen = new Random(System.currentTimeMillis());
289
290     int array[] = new int[numElements];
291     System.out.print("Elements generated: ");
292     for (int index = 0; index < numElements; index++){
293         array[index] = randGen.nextInt(maxValue);
294         System.out.print(array[index] + " ");
295         hashTable.insert(array[index]);
296     }
297
298     System.out.println();
299
300     System.out.println("\nContents of the Hash Table ");
301     hashTable.printHashTable();
302
303
304     int searchElement;
305     System.out.print("Enter an element to search: ");
306     searchElement = input.nextInt();
307
308     if (hashTable.hasElement(searchElement))
309         System.out.println(searchElement + " is in the original array");
310     else
311         System.out.println(searchElement + " is not there!!");
312
313
314     int deleteElement;
315     System.out.print("Enter an element to delete: ");
316     deleteElement = input.nextInt();
317     hashTable.deleteElement(deleteElement);
318
319     System.out.println("\nContents of the Hash Table (after delete) ");
```

```
320         hashTable.printHashTable();
321
322         }
323
324     }
```

```
Enter the number of elements you want to store in the hash table: 10
Enter the maximum value for an element: 25
Enter the size of the hash table: 7
Elements generated: 3 2 2 19 24 17 9 23 24 14

Contents of the Hash Table
Hash Index: 0 : 14
Hash Index: 1 :
Hash Index: 2 : 2 2 9 23
Hash Index: 3 : 3 24 17 24
Hash Index: 4 :
Hash Index: 5 : 19
Hash Index: 6 :
Enter an element to search: 17
17 is in the original array
Enter an element to delete: 2

Contents of the Hash Table (after delete)
Hash Index: 0 : 14
Hash Index: 1 :
Hash Index: 2 : 9 23
Hash Index: 3 : 3 24 17 24
Hash Index: 4 :
Hash Index: 5 : 19
Hash Index: 6 :
```