```java
import java.util.*;

// implementing hash tables as an array of linked lists
// and using it to check whether two sequencs are permutations of each other

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

```
129                 if (index == modifyIndex){
130                     currentNodePtr.setData(data);
131                     return;
132                 }
133 
134                 prevNodePtr = currentNodePtr;
135                 currentNodePtr = currentNodePtr.getNextNodePtr();
136 
137                 index++;
138             }
139 
140 
141     }
142 
143 
144     public  boolean deleteElement(int data){
145 
146 
147             Node currentNodePtr = headPtr.getNextNodePtr();
148             Node prevNodePtr = headPtr;
149             Node nextNodePtr = headPtr;
150 
151 
152             while (currentNodePtr != null){
153 
154                 if (currentNodePtr.getData() == data){
155                     nextNodePtr = currentNodePtr.getNextNodePtr();
156                     prevNodePtr.setNextNodePtr(nextNodePtr);
157                     return true;
158                 }
159 
160                 prevNodePtr = currentNodePtr;
161                 currentNodePtr = currentNodePtr.getNextNodePtr();
162 
163             }
164 
165             return false;
166 
167     }
168 
169     public  int countList(){
170 
171             Node currentNodePtr = headPtr.getNextNodePtr();
172             int numElements = 0;
173 
174             while (currentNodePtr != null){
175 
176                 numElements++;
177                 currentNodePtr = currentNodePtr.getNextNodePtr();
178 
179             }
180 
181             return numElements;
182     }
183 
184 
185     public  void IterativePrint(){
186 
187             Node currentNodePtr = headPtr.getNextNodePtr();
188 
189             while (currentNodePtr != null){
190                 System.out.print(currentNodePtr.getData()+" ");
191                 currentNodePtr = currentNodePtr.getNextNodePtr();
192             }
```

```
193
194             System.out.println();
195
196     }
197
198
199     public boolean containsElement(int data){
200
201         Node currentNodePtr = headPtr.getNextNodePtr();
202
203         while (currentNodePtr != null){
204
205             if (currentNodePtr.getData() == data)
206                 return true;
207
208             currentNodePtr = currentNodePtr.getNextNodePtr();
209         }
210
211         return false;
212
213     }
214
215
216 }
217
218
219 class Hashtable{
220
221     private List[] listArray;
222     private int tableSize;
223
224
225     public Hashtable(int size){
226         tableSize = size;
227         listArray = new List[size];
228         for (int index = 0; index < size; index++)
229             listArray[index] = new List();
230     }
231
232     public  int getTableSize(){
233         return tableSize;
234     }
235
236     public  void insert(int data){
237
238         int hashIndex = data % tableSize;
239         listArray[hashIndex].insert(data);
240
241     }
242
243     public void deleteElement(int data){
244
245         int hashIndex = data % tableSize;
246         listArray[hashIndex].deleteElement(data);
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     public  boolean isEmpty(){
267
268         for (int hashIndex = 0; hashIndex < tableSize; hashIndex++){
269
270             if (!listArray[hashIndex].isEmpty())
271                 return false;
272         }
273
274         return true;
275
276     }
277
278 }
279
280
281 class HashTableLinkedList{
282
283     public static void main(String[] args){
284
285     Scanner input = new Scanner(System.in);
286
287     String integerSequence;
288     System.out.print("Enter the integer sequence: ");
289     integerSequence = input.nextLine();
290
291     String testSequence;
292     System.out.print("Enter the test sequence for permutation: ");
293     testSequence = input.nextLine();
294
295     int hashTableSize;
296     System.out.print("Enter the size of the hash table: ");
297     hashTableSize = input.nextInt();
298     Hashtable hashTable = new Hashtable(hashTableSize);
299
300
301     StringTokenizer stk = new StringTokenizer(integerSequence, ", ");
302     while (stk.hasMoreTokens()){
303         int value = Integer.parseInt(stk.nextToken());
304         hashTable.insert(value);
305     }
306
307     System.out.println();
308
309     hashTable.printHashTable();
310
311
312     stk = new StringTokenizer(testSequence, ", ");
313     while (stk.hasMoreTokens()){
314         int testValue = Integer.parseInt(stk.nextToken());
315         if (hashTable.hasElement(testValue))
316             hashTable.deleteElement(testValue);
317         else{
318             System.out.println(testSequence + " is not a permuted sequence of " +
                integerSequence);
319             return;
```

```
320             }
321
322         }
323
324         hashTable.printHashTable();
325
326         if (hashTable.isEmpty())
327             System.out.println(testSequence +" is a permuted sequence of " + integerSequence
                );
328         else
329             System.out.println(testSequence +" is not a permuted sequence of " +
                integerSequence);
330
331         }
332
333     }
```