Assignment 3: Coding Question

● Graded

Student

Tyler NGUYEN

Total Points

100 / 100 pts

Autograder Score

24.0 / 24.0

Passed Tests

Test randomly generated elements (4/4)
Test member() with delete() (1/1)
Test small hash table { 2, 1, 3 } (1/1)
Whitebox test (16/16)
Test tiny hash table { 1 } (1/1)
Test member() (1/1)

Question 2

Manual Grading

76 / 76 pts

✔ **– 0 pts** Correct

– 76 pts Late submission

– 76 pts Incorrect or cannot be compiled

+ 5 pts Implementation of full

+ 5 pts Implementation of member

+ 5 pts Implementation of insert

+ 5 pts Implementation of delete

+ 5 pts Implementation of toString

+ 5 pts Provide a constructor that does not take any arguments and initialize thehash table to an empty one.

– 5 pts No constructor that does not take any arguments and initialize the hash table to an empty one.

– 2 pts No RuntimeException when inserting into a full table

– 12 pts Passed 20/24 tests

– 66 pts Passed 3/24 tests

– 63 pts Passed 4/24 tests

– 50 pts Passed 8/24 tests

– 69 pts Passed 2/24 tests

Autograder Results

Test randomly generated elements (4/4)

Test member() with delete() (1/1)

Test small hash table { 2, 1, 3 } (1/1)

Whitebox test (16/16)

Test tiny hash table { 1 } (1/1)

Test member() (1/1)

Submitted Files

HashTable.java Download

```java

/*
author: Tyer Nguyen UCID: 30158563
date: November 22, 2023
description: HashTable class that implements Dictionary and its methods
 */


package ca.ucalgary.cpsc331.a3;

public class HashTable implements Dictionary{
    /**
     * the maximum capacity of the hash table
     */
    private final static int TABLE_SIZE = 17;
    /**
     * the array to store keys
     */
    private String[] table;

    /**
     * constructs a new and empty hash table
     */
    public HashTable() {
        table = new String[TABLE_SIZE];
    }

    /**
     * will compute the hash code for a given key
     *
     * @param key to hash
     * @return index in the hash table
     */
    private int hash(String key) {
        return Math.abs(key.hashCode()) % TABLE_SIZE;
    }

    /**
     * will check if the hash table is full or not
     *
     * @return true if the hash table is full and false if it is not
     */
    public boolean full() {
        for (String key : table) { //iterate over each slot in the hash table
            if (key == null || key.equals("DELETED")) {
                //if the slot is empty or marked as DELETED then the table is not full
                return false;
            }
        }
```

```
50          return true;
51      }
52
53      /**
54       * will check if a key is a member of the hash table or not
55       *
56       * @param key to check
57       * @return true if the key is in the hash table and false if not
58       */
59      @Override
60      public boolean member(String key) {
61          int i = hash(key); //initial index for the key
62          int ogi = i; //original index used to determine if we have looped through the entire table
63          do {
64              if (table[i] == null) { //key is not in the table
65                  return false;
66              }
67              if (table[i].equals(key)) { //key is found
68                  return true;
69              }
70              i = (i + 1) % TABLE_SIZE; //move to the next index
71          } while (i != ogi);
72          return false;
73      }
74
75      /**
76       * will insert a key into the hash table
77       *
78       * @param key the key to insert
79       * @return true if the key was inserted and false if not
80       * @throws RuntimeException if the hash table is full or the key already exists
81       */
82      @Override
83      public boolean insert(String key) {
84          if (member(key) || full()) { //check if the key already exists or if the table is full
85              throw new RuntimeException("HashTable is full or key already exists");
86          }
87          int i = hash(key); //initial index for the key
88          while (table[i] != null && !table[i].equals("DELETED")) {
89              //linear probing used to find the next available index
90              i = (i + 1) % TABLE_SIZE;
91          }
92          table[i] = key; //insert the key into the found index
93          return true;
94      }
95
96      /**
97       * will delete a key from the hash table
98       *
99       * @param key to delete
100      * @return true if the key was deleted and false if not
101      */
```

```
    @Override
    public boolean delete(String key) {
        int i = hash(key); //compute initial index for key
        int ogi = i; //original index
        do {
            if (table[i] == null) {
                return false;
            }
            if (table[i].equals(key)) {
                table[i] = "DELETED"; //if found then we need to mark the slot as DELETED
                return true;
            }
            i = (i + 1) % TABLE_SIZE;
        } while (i != ogi);
        return false; //key was not found after searching the whole table
    }

    /**
     * will return a string representation of the hash table
     *
     * @return a string with each non-empty slot in the hash table
     */
    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < TABLE_SIZE; i++) { //iterate through all slots in the table
            if (table[i] != null) {
                sb.append(i).append(":");
                if (table[i].equals("DELETED")) {
                    //checking if the slot is marked as DELETED or contains an actual key
                    sb.append("deleted");
                } else {
                    sb.append("\"").append(table[i]).append("\"");
                }
                sb.append("\n");
            }
        }
        return sb.toString();
    }

}

```