

Deliverable 1: Student Class

```
J Student.java x
src > J Student.java
1 // Rahul Chaudhari
2 import java.util.Objects;
3
4 // Student class with their details of phone, name, email, course
5 public class Student {
6     private String phone;
7     private String name;
8     private String email;
9     private String course;
10
11     /**
12      * Default Constructor for empty values
13      */
14     public Student() {
15         // using the constructor method to assign empty value to the default constructor
16         this(phone:"000-000-0000", name:"", email:"", course:"");
17     }
18
19     // Constructor
20     /**
21      * this function takes 4 strings and assignments it to respective private
22      * variable of studnet class
23      *
24      * @param phone number of the student
25      * @param name of the studnet
26      * @param email of the the student
27      * @param course taken by the student
28      */
29     public Student(String phone, String name, String email, String course) {
30         this.phone = phone;
31         this.name = name;
32         this.email = email;
33         this.course = course;
34     }
35
36     /**
37      * getter function for phone number of the student.
38      *
39      * @return The phone number of the student.
40      */
41     public String get_phone_number() {
42         return phone;
43     }
44 }
```

Deliverable 2: Data Reader Class

```
J DataReader.java 2 x
src > J DataReader.java > DataReader
1 // Rahul Chaudhari
2 // FILE PATH .\\lib\\Records.csv
3 import java.io.BufferedReader;
4 import java.io.FileReader;
5 import java.io.IOException;
6 import java.util.ArrayList;
7 import java.util.List;
8
9 // For Testing Purpose, Terminal was not able to show all the 2000 lines of data, it was showing only hal
10 // used this FileWriter so that all the data being read can be verified during debugging
11 import java.io.FileWriter;
12
13 /**
14  * A Helper Class to Read the Contains of a File
15  */
16 public class DataReader {
17     /**
18      *
19      * @param file_path the path where the file is located
20      * @return the list of students objects read from the records file
21      */
22     public static List<Student> Load_student_data_from_file(String file_path) {
23         List<Student> students_list = new ArrayList<>();
24         try (BufferedReader read_data = new BufferedReader(new FileReader(file_path))) {
25             String one_line;
26
27             // Skip the header of the file which is phone,name,email,course
28             String header = read_data.readLine();
29
30             while ((one_line = read_data.readLine()) != null) {
31                 // System.out.println(one + " " + one_line); used during printing
32
33                 // if the file has missing rows for example as follows:
34                 // 0-230-367-7323,Camden Long,Camden_Long9539@famism.biz,Health
35                 //
36                 // 7-241-634-6332,Noah Oliver,Noah_Oliver9716@fuliss.net,Ecology
37                 if (one_line.trim().isEmpty()) {
38                     continue;
39                 }
40                 String[] elements_of_student = one_line.split(regex:"");
41                 if (elements_of_student.length == 4) {
42                     String phone = elements_of_student[0].trim(); // trim function deletes the whitespace
43                     String name = elements_of_student[1].trim();
44                     String email = elements_of_student[2].trim();
45                     String course = elements_of_student[3].trim();
46                 }
47             }
48         } catch (IOException e) {
49             e.printStackTrace();
50         }
51         return students_list;
52     }
53 }
```

C:\Users\rj\OneDrive\Documents\CS310\HashAlgorithm\HashAlgoritm>

Deliverable 3 & 4:

1. Quadratic Hashing Insert & Find Function

```
src > J QuadraticHashTable.java > QuadraticHashTable
7 public class QuadraticHashTable {
81
82 * main method to test the QuadraticHashTable
83 *
84 * @param args
85
Run | Debug
85 public static void main(String[] args) {
86 // Creating the QuadraticHashTable object with size 100
87 QuadraticHashTable hashTable = new QuadraticHashTable(size:100);
88
89 // Reading the student list from the file
90 List<Student> students = DataReader.Load_student_data_from_file(file_path:".\\lib\\Records.csv");
91
92 // Insertin each student into the hash table one at a time
93 for (Student student : students.subList(fromIndex:0, toIndex:100)) {
94 hashTable.insert(student);
95 }
96
97 // Printing the contents of the table just to see if there are open postion for debugging
98 // int one = 1;
99 // for (Student student : hashTable.table) {
100 // System.out.print(one + " ");
101 // if (student != null) {
102 // System.out.println(student.get_student_name() + "\n");
103 // }
104 // one++;
105 // }
106
107 System.out.print("\n\n"); // clearing up space as some element might be not added
108 System.out.println("Creating an existing student: ");
109 Student existing_student = new Student(phone:"2-721-110-5276", name:"Cedrick Everett", email:"Ced
110 course:"Ecology");
111
112 boolean found_existing = hashTable.find(existing_student);
113 System.out.println("Existing student found Successful: " + found_existing);
114 System.out.println("Creating an non existing student: ");
115 Student non_existing_student = new Student(phone:"660-280-5213", name:"Rahul Chaudhari", email:"c
```

Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rjask\OneDrive\Documents\CS310\HashAlgorithm\HashAlgoritm> cmd /C ""C:\Program Files\Java\jdk-17\bin\java.exe" -XX:+ShowCodeDetailsInExceptionMessages -cp C:\Users\rjask\OneDrive\Documents\CS310\HashAlgorithm\HashAlgorithm\bin QuadraticHashTable ""

Unable to insert student: Marigold Cunningham
Unable to insert student: Savannah Long
Unable to insert student: Alan Roberts
Unable to insert student: Regina Vincent
Unable to insert student: Eve Fenton

Creating an existing student:
Existing student found Successful: true

Creating an non existing student:
Non Existing student found (Unsuccessful): false

C:\Users\rjask\OneDrive\Documents\CS310\HashAlgorithm\HashAlgoritm>

2. Double Hashing Insert & Find Function

The screenshot displays a code editor with a Java file named `DoubleHashing.java`. The code implements a double hashing algorithm for inserting and finding students in a hash table. The `main` method reads student data from a CSV file, inserts it into a hash table, and then tests the `find` function with both existing and non-existing student records. Red annotations highlight the class definition, the `main` method, and specific student objects used for testing.

```
src > DoubleHashing.java > DoubleHashing > main(String[])
6 public class DoubleHashing {
    Run | Debug
    public static void main(String[] args) {
        // Creating the DoubleHashing object with size 100
        DoubleHashing hashTable = new DoubleHashing(size:100);

        // Reading the student list from the file
        List<Student> students = DataReader.load_student_data_from_file(file_path:"..\\lib\\Records.csv");

        // Insertin each student into the hash table one at a time
        for (Student student : students.subList(fromIndex:0, toIndex:100)) {
            hashTable.insert(student);
        }

        // Printing the contents of the table just to see if there are open postion for debugging
        // int one = 1;
        // for (Student student : hashTable.table) {
        //     System.out.println(one + " ");
        //     if (student != null) {
        //         System.out.println(student.get_student_name() + "\n");
        //     }
        //     one++;
        // }

        System.out.print("\n\n"); // clearing up space as some element might be not added
        System.out.println("\nCreating an existing student: ");
        Student existing_student = new Student(phone:"2-721-110-5276", name:"Cedrick Everett", email:"Cedrick.Everett@ecampus.utk.edu", course:"Ecology");

        boolean found_existing = hashTable.find(existing_student);
        System.out.println("Existing student found Successful: " + found_existing);
        System.out.println("\nCreating an non existing student: ");
        Student non_existing_student = new Student(phone:"660-280-5213", name:"Rahul Chaudhari", email:"Rahul.Chaudhari@ecampus.utk.edu", course:"Ecology");

        boolean found_non_existing = hashTable.find(non_existing_student);
        System.out.println("Non Existing student found (UnSuccessful): " + found_non_existing);
    }
}
```

The terminal output shows the execution of the program:

```
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rjaws\OneDrive\Documents\CS310\HashAlgorithm\HashAlgoritm> cmd /C ""C:\Program Files\Java\jdk-17\bin\java.exe" -XX:+ShowCodeDetailsInExceptionMessages -cp C:\Users\rjaws\OneDrive\Documents\CS310\HashAlgorithm\HashAlgorithm\bin DoubleHashing "
Unable to insert student: Christy Simpson
Unable to insert student: Marigold Cunningham

Creating an existing student:
Existing student found Successful: true

Creating an non existing student:
Non Existing student found (UnSuccessful): false

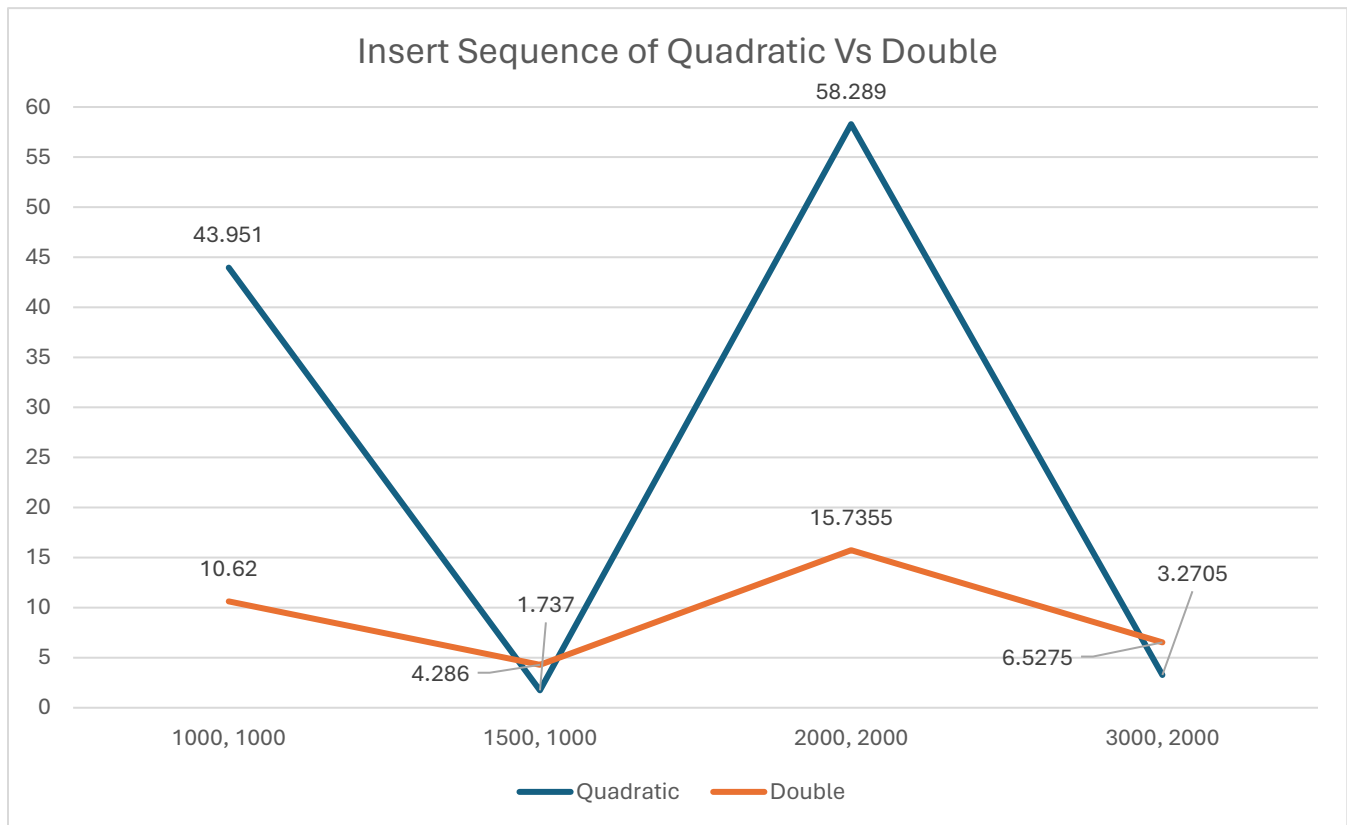
C:\Users\rjaws\OneDrive\Documents\CS310\HashAlgorithm\HashAlgoritm>
```


3. Data Table Report:

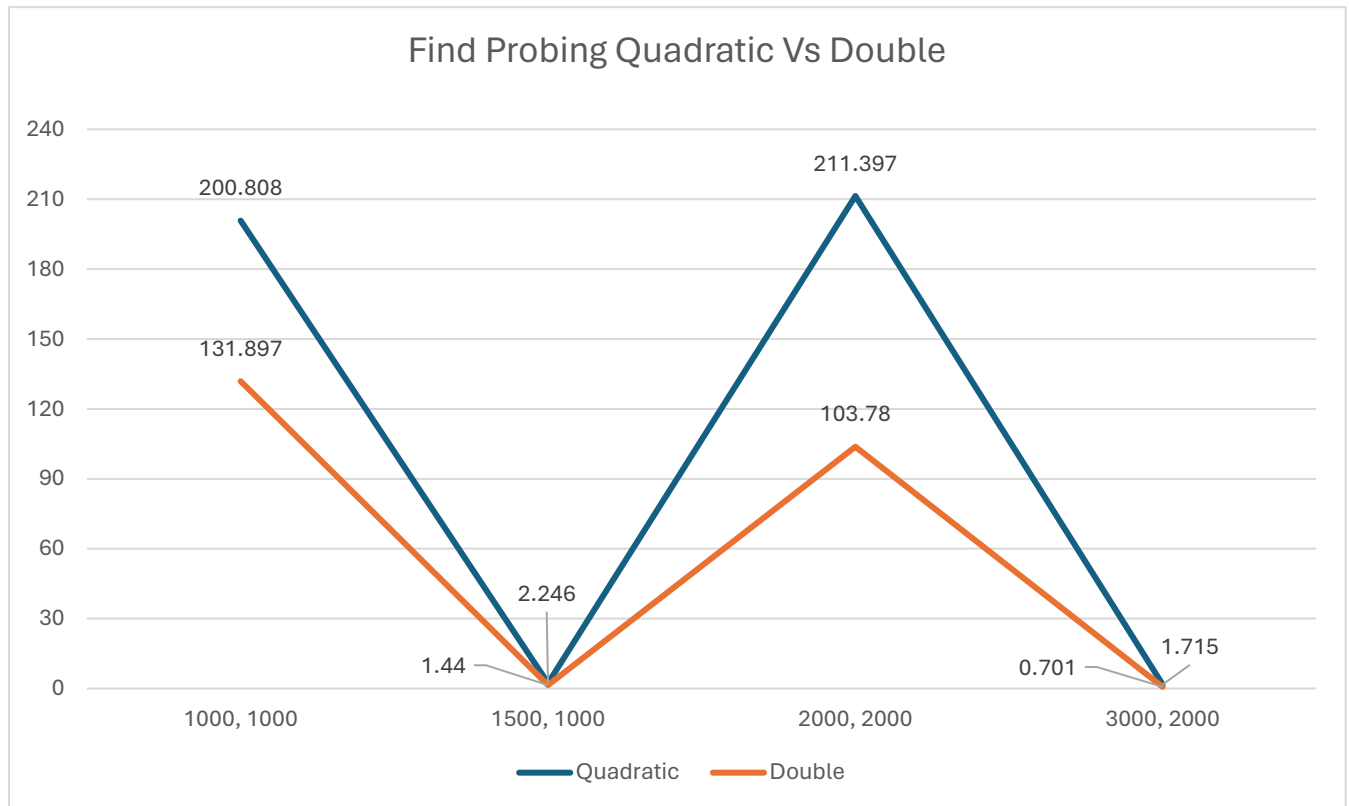
Algorithm	HashTable Size	Input Size	Insert Porbing Count	Find Probing Count	Average Step Count
Quadratic Hash Table	1000	1000	43.951	200.808	122.3795
	1500	1000	1.737	2.246	1.9915
	2000	2000	58.289	211.397	134.843
	3000	2000	3.2705	1.715	2.49275
Double Hash Table	1000	1000	10.62	131.897	71.2585
	1500	1000	4.286	1.44	2.863
	2000	2000	15.7355	103.78	59.75775
	3000	2000	6.5275	0.701	3.61425

Deliverable 6: Answering Questions

- a. Plot a graph showing the average probing sequences for the insert operation for the two algorithms.



- b. Similarly, plot a graph showing the average probing sequences of the find operation for the two algorithms.



- c. Which hash table algorithm requires the least number of probing sequences to store the student objects? Comment on why that is the case.

Comparing both algorithm and looking at the data from the above graph we can see that, Double Hash Algorithm performs a lot better than the Quadratic Hash Algorithm doesn't matter the case whether it is storing data or trying to find a data which is already in the hash table.

The reason behind this could be the nature of probing technique used by double hashing compared to quadratic. Quadratic Probing sometimes results in secondary clustering while double hash algorithm is efficient in resolving the collision.

- d. Compare the performance of both of the two hash table algorithms when the input size was 1000 vs Hash table size 1000 and 1500.

In both algorithms we notice that the algorithm worked a lot better than the table size was 1500 compared to when the table size was 1000, while the input size being constant to 1000.

While Double Hash Algorithm still is better than the Quadratic Hashing Algorithm both has a significance advantage in the performance while increasing the table size compared to the input size.

Larger table size results in fewer collisions, resulting in fewer probing sequences required.

