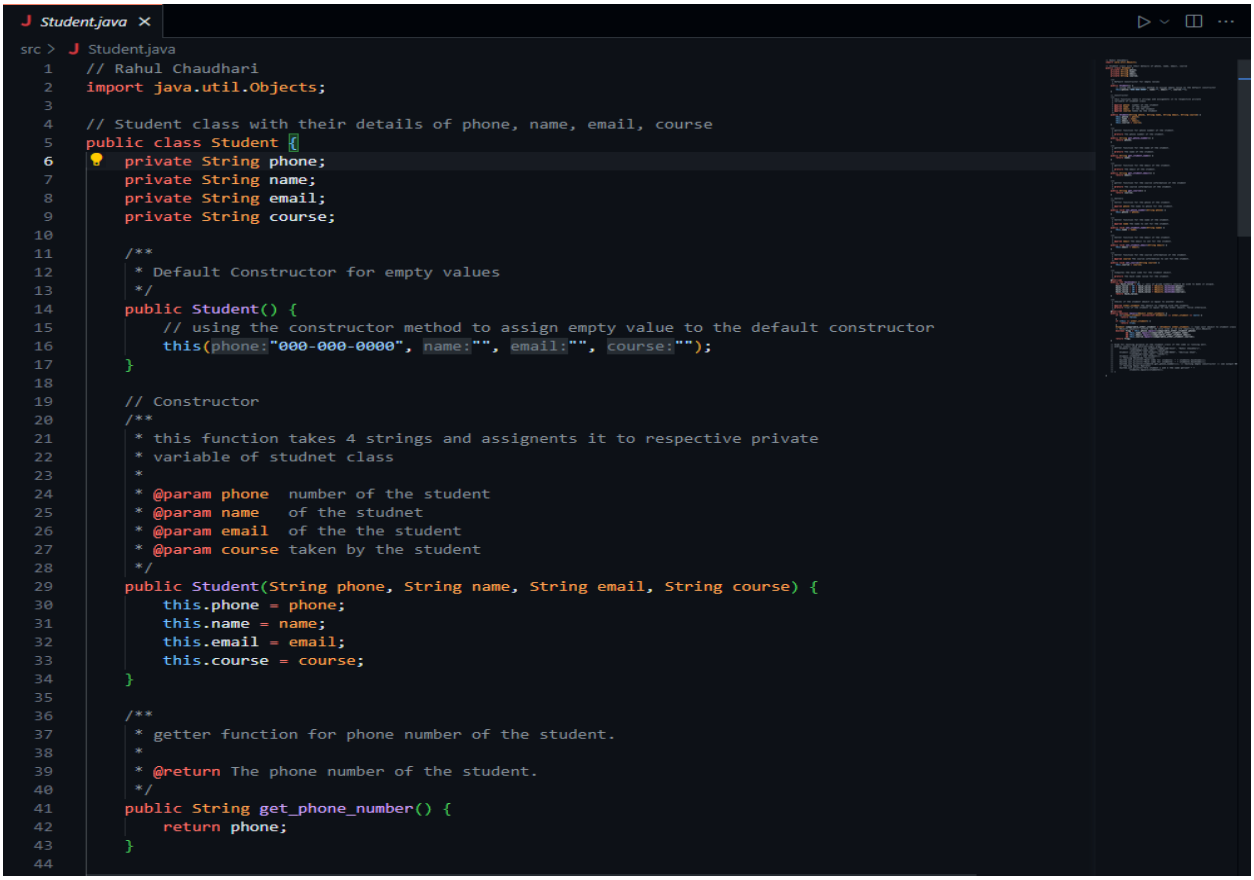
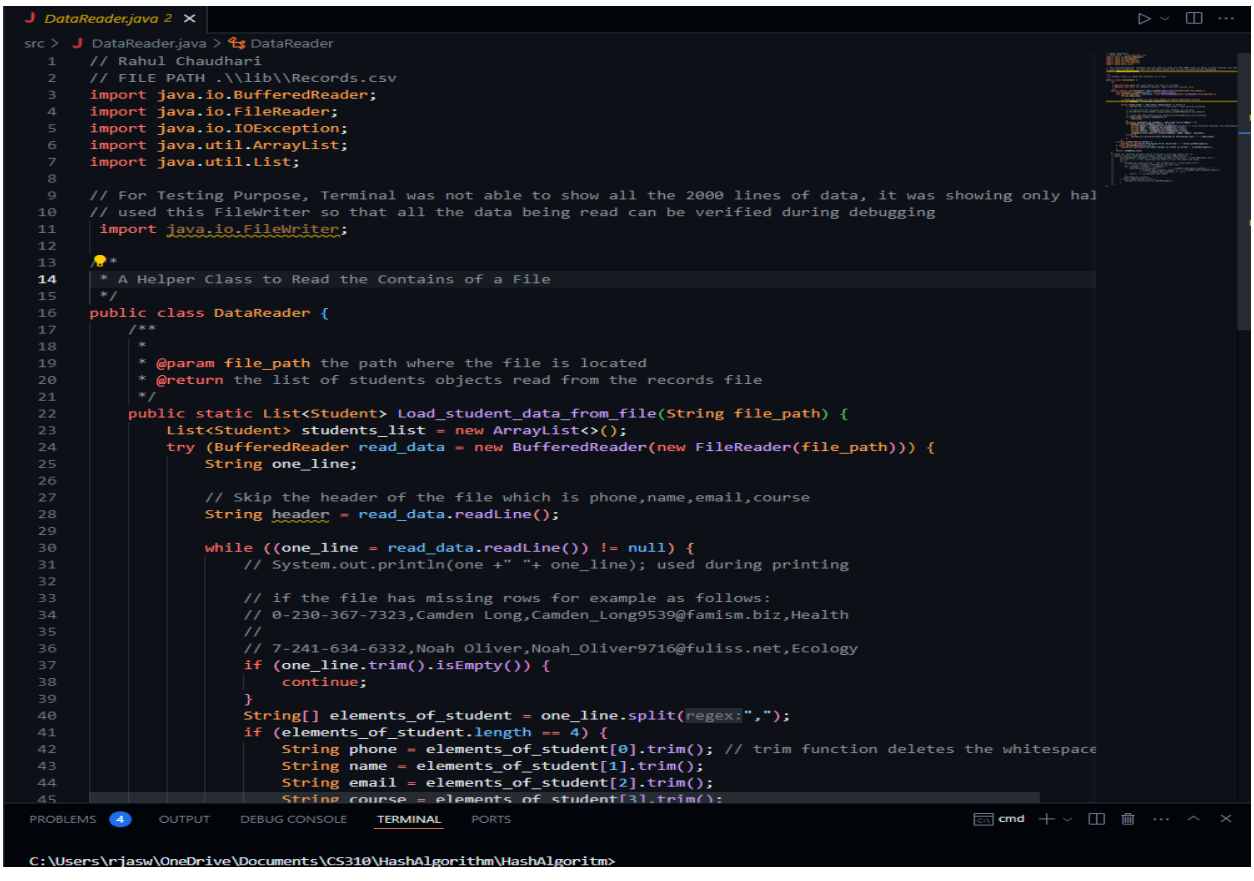
Deliverable 1: Student Class



```java
// Rahul Chaudhari
import java.util.Objects;

// Student class with their details of phone, name, email, course
public class Student {
    private String phone;
    private String name;
    private String email;
    private String course;

    /**
     * Default Constructor for empty values
     */
    public Student() {
        // using the constructor method to assign empty value to the default constructor
        this(phone:"000-000-0000", name:"", email:"", course:"");
    }

    // Constructor
    /**
     * this function takes 4 strings and assignents it to respective private
     * variable of studnet class
     *
     * @param phone  number of the student
     * @param name   of the studnet
     * @param email  of the the student
     * @param course taken by the student
     */
    public Student(String phone, String name, String email, String course) {
        this.phone = phone;
        this.name = name;
        this.email = email;
        this.course = course;
    }

    /**
     * getter function for phone number of the student.
     *
     * @return The phone number of the student.
     */
    public String get_phone_number() {
        return phone;
    }
```

Deliverable 2: Data Reader Class

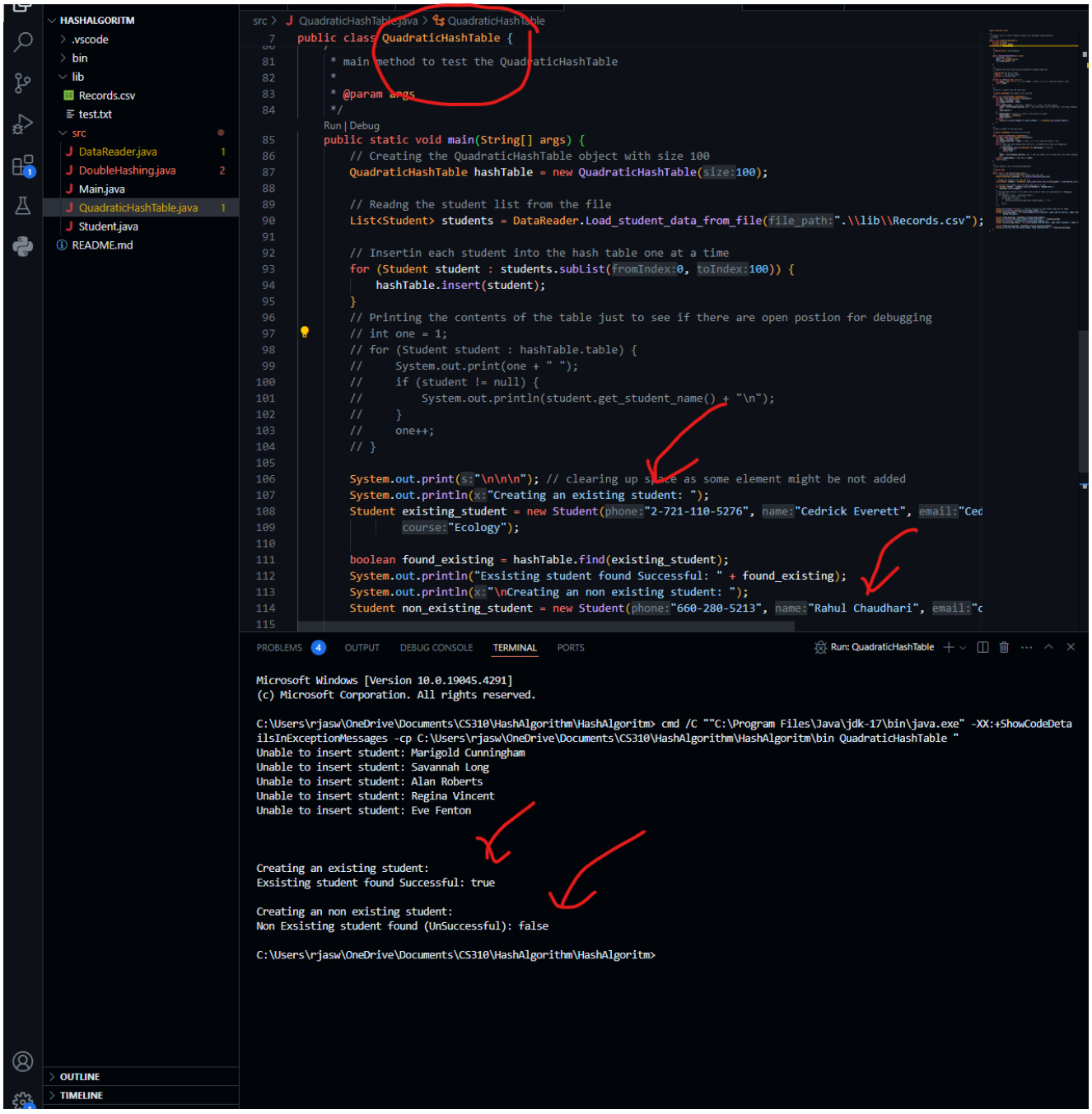


```java
// Rahul Chaudhari
// FILE PATH .\\lib\\Records.csv
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

// For Testing Purpose, Terminal was not able to show all the 2000 lines of data, it was showing only hal
// used this FileWriter so that all the data being read can be verified during debugging
import java.io.FileWriter;

/**
 * A Helper Class to Read the Contains of a File
 */
public class DataReader {
    /**
     *
     * @param file_path the path where the file is located
     * @return the list of students objects read from the records file
     */
    public static List<Student> Load_student_data_from_file(String file_path) {
        List<Student> students_list = new ArrayList<>();
        try (BufferedReader read_data = new BufferedReader(new FileReader(file_path))) {
            String one_line;

            // Skip the header of the file which is phone,name,email,course
            String header = read_data.readLine();

            while ((one_line = read_data.readLine()) != null) {
                // System.out.println(one +" "+ one_line); used during printing

                // if the file has missing rows for example as follows:
                // 0-230-367-7323,Camden Long,Camden_Long9539@famism.biz,Health
                //
                // 7-241-634-6332,Noah Oliver,Noah_Oliver9716@fuliss.net,Ecology
                if (one_line.trim().isEmpty()) {
                    continue;
                }
                String[] elements_of_student = one_line.split(regex:",");
                if (elements_of_student.length == 4) {
                    String phone = elements_of_student[0].trim(); // trim function deletes the whitespace
                    String name = elements_of_student[1].trim();
                    String email = elements_of_student[2].trim();
                    String course = elements_of_student[3].trim();
```

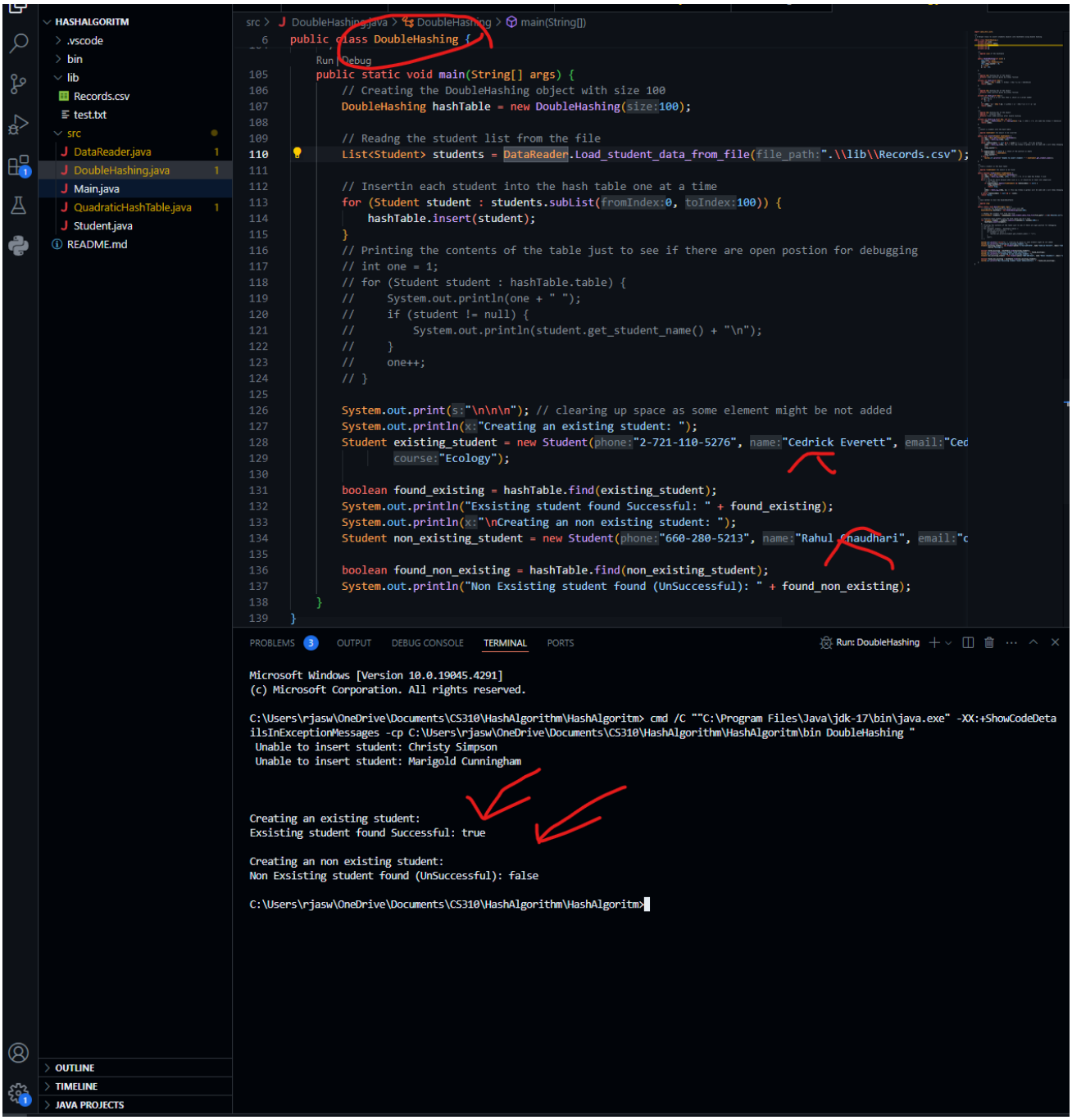
```
PROBLEMS 4  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
C:\Users\rjasw\OneDrive\Documents\CS310\HashAlgorithm\HashAlgoritm>
```

Deliverable 3 & 4:

1. **Quadratic Hashing Insert & Find Function**

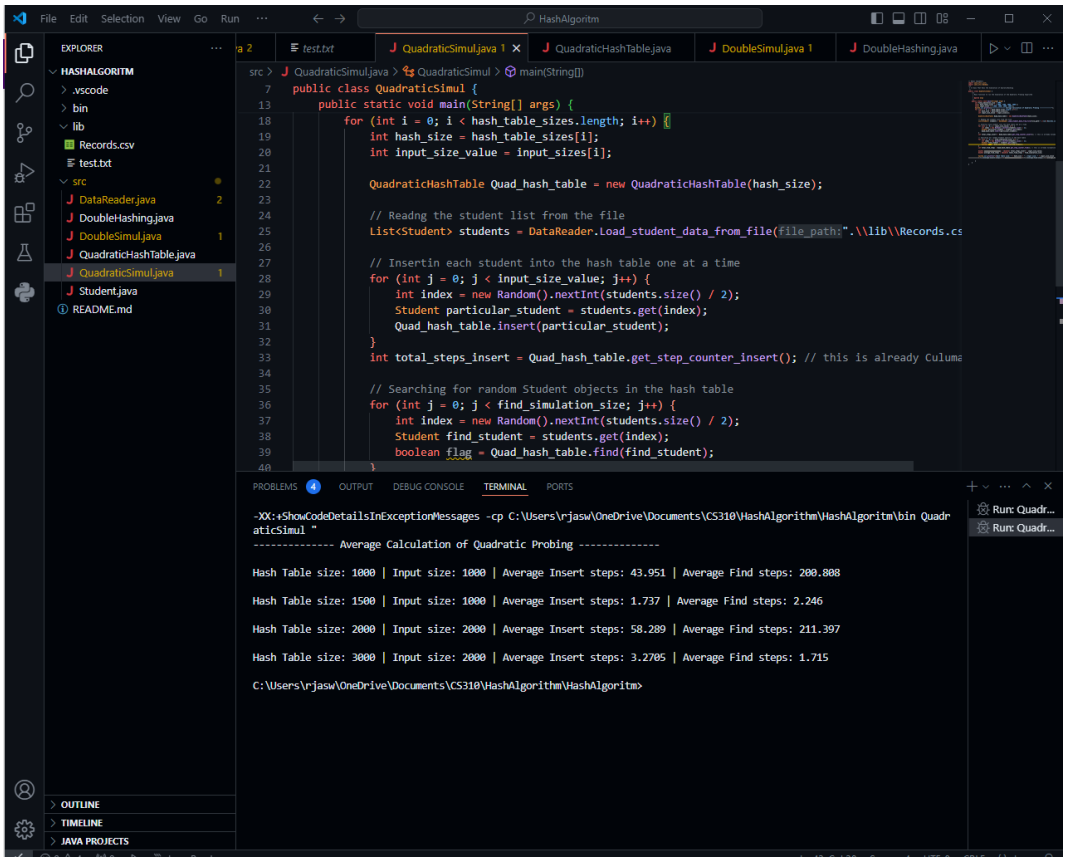


2. Double Hashing Insert & Find Function

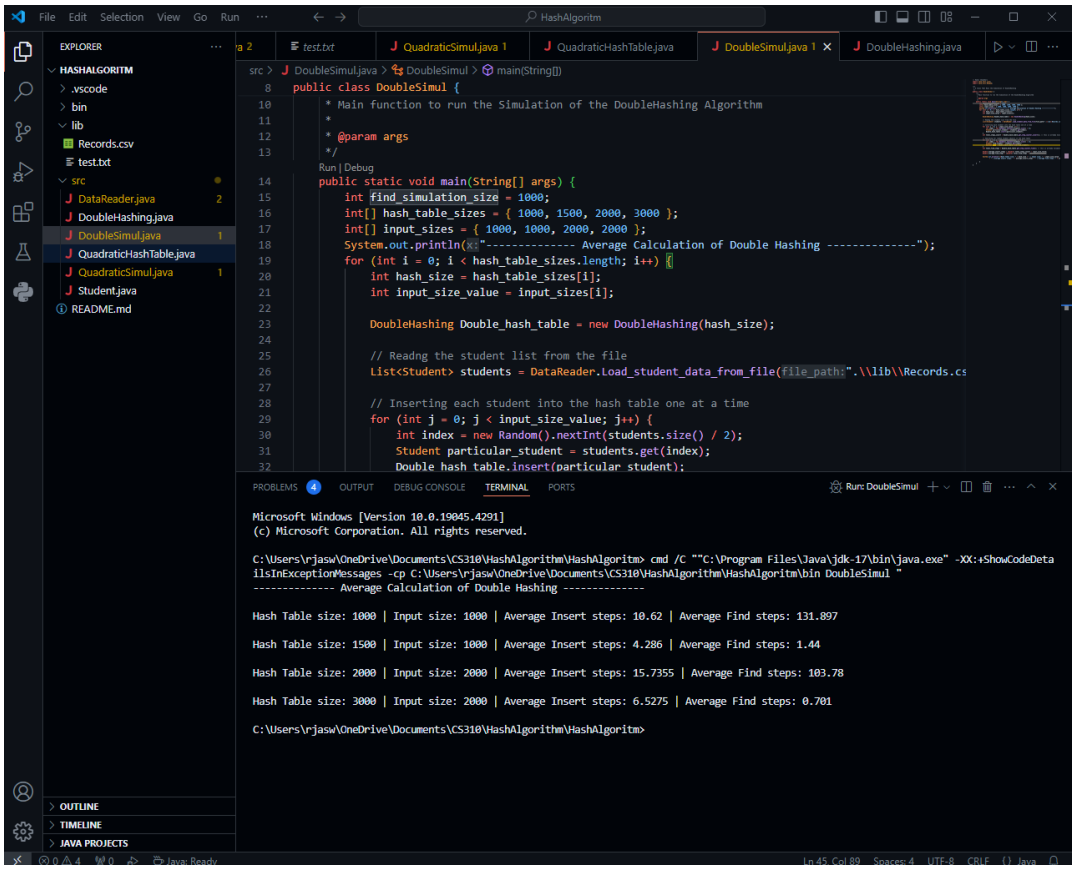


Deliverable 5:

1. Quadratic Hashing Simulation (Insert & Find)



2. Double Hashing Simulation (Insert & Find)

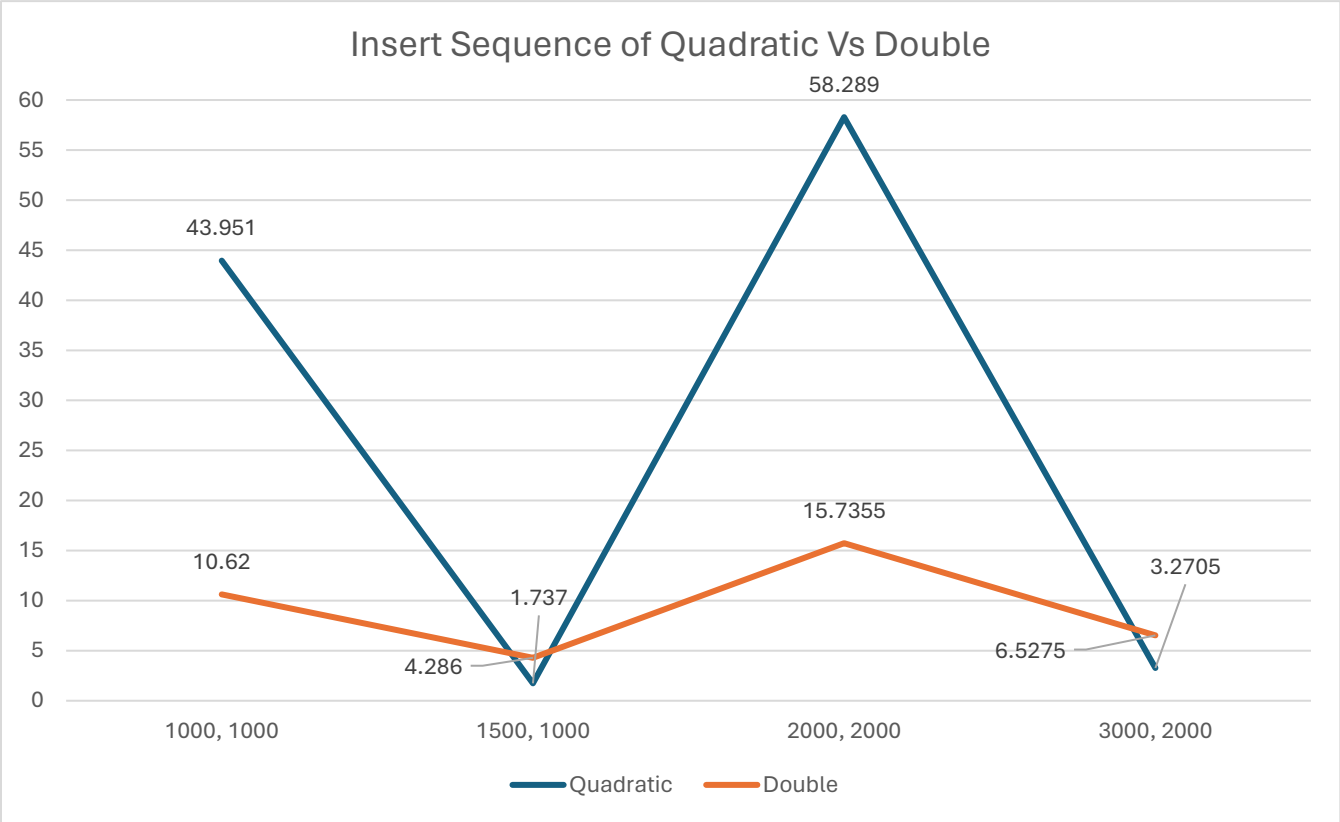


3. Data Table Report:

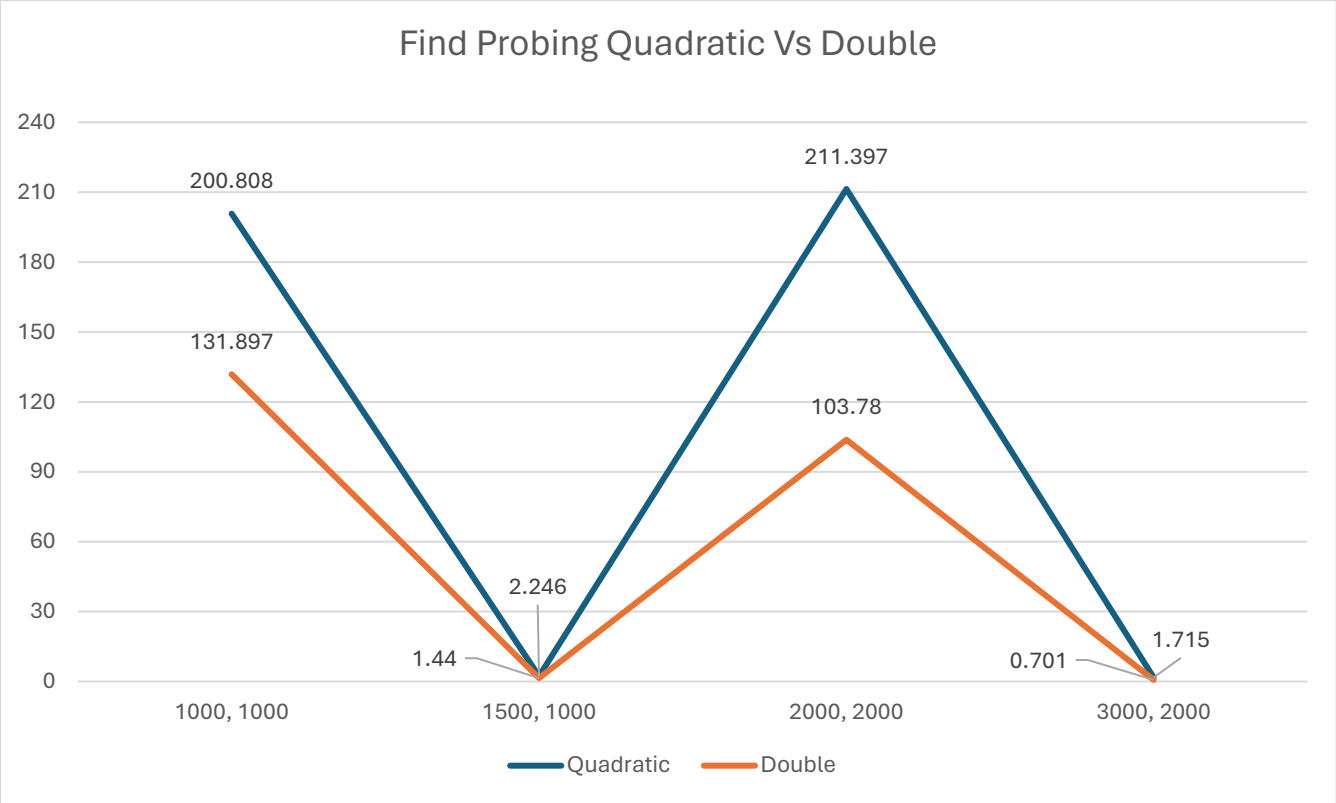
Algorithm	HashTable Size	Input Size	Insert Porbing Count	Find Probing Count	Average Step Count
Quadratic Hash Table	1000	1000	43.951	200.808	122.3795
	1500	1000	1.737	2.246	1.9915
	2000	2000	58.289	211.397	134.843
	3000	2000	3.2705	1.715	2.49275
Double Hash Table	1000	1000	10.62	131.897	71.2585
	1500	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.