CMPT-435-Final Project

Ahmed Handulle

December 19, 2022

In this document, I will explain my code from the final project in detail. There is only one Main class which includes all the implementation for the program.

Below is the Main Class and it contains all the logic of the program

1 Main Class

```java

/**
 * Main
 */
public class Main {

    // Creating a Residents class for residents objects
    public static class Resident{
        ArrayList<Hospital> preferences;
        String name;
        Hospital proposedTo;
        Resident(String name){
            this.name = name;
            this.preferences = new ArrayList<Hospital>();
            this.proposedTo = null;
        }

    }

    // Creating a Hospitals class for hospitals objects
    public static class Hospital{
        ArrayList<Resident> preferences;
        String name;
        int capacity;
        ArrayList<Resident> assignedResidents;
        Hospital(String name){
            this.name = name;
            this.assignedResidents = new ArrayList<Resident>();
            this.preferences = new ArrayList<Resident>();
        }

    }

    public static void main(String[] args) {

        // Creating an ArrayList to store all the residents
        ArrayList<Resident> residents = new ArrayList<Resident>();

        // Creating an ArrayList to store all the hospitals
        ArrayList<Hospital> hospitals = new ArrayList<Hospital>();

        try {

            File f = new File("data.txt"); // getting the data from
     the data.txt file
            Scanner myReader = new Scanner(f); // Scanner reader
    will give access to the data.txt file

            // Start reading the file line by line
            while (myReader.hasNextLine()) {
                //store the data of the line
                String line = myReader.nextLine();

                // Check resident lines to create resident objects
                if (line.startsWith("Resident Preferences")) {
                    // Iterate over the nex few lines to create the
     residents objects
                    line = myReader.nextLine();

                    while (myReader.hasNextLine() & line.startsWith
    ("r")) {
                        // create a resident object
                        createResident(line, residents);
                        line = myReader.nextLine();
                    }
                }

                // Check hospital lines to create hospitals objecst
                if (line.startsWith("Hospital Preferences")) {
                    // Iterate over the nex few lines to create the
     hospitals objects
                    line = myReader.nextLine();

                    while (myReader.hasNextLine() & line.startsWith
    ("h")) {
                        // create a resident object
                        createHospitals(line, hospitals);
                        line = myReader.nextLine();
                    }
                }

            }

            // Doing the calculations here
            startMatching(residents, hospitals);

            // Testing
            System.out.println("Final Stable Match");
            for (Resident i : residents) {
                System.out.println(i.name + " --> " + i.proposedTo.
```

```java
name);
        }

        myReader.close();
    } catch (Exception e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}

// This function will create a resident object
public static void createResident(String tempString, ArrayList<Resident> residents) {
    // Data example (tempString = r1: h3 h1 h5 h4)
    String[] data = tempString.split(" "); // split string between spaces
    // Creating resident object
    Resident tempObjResident = new Resident(data[0]);
    //Iterating over the split string and creating the Resident object preferences
    for (String i : data) {
        // Creating the references array list
        if (i.startsWith("h")) {
            tempObjResident.preferences.add(new Hospital(i));
        }
    }
    // Storing the Resident object to the residents array
    residents.add(tempObjResident);
}

// Thi function will create hospitals objects
public static void createHospitals(String tempString, ArrayList<Hospital> hospitals) {
    // Data example (tempString = h1 capacity=4 r3 r7 r9 r11 r5 r4 r10 r8 r6 r1 r2
    String[] data = tempString.split(" "); // split string between spaces

    // Creating hospital object
    Hospital tempObjHospital = new Hospital(data[0]);

    for (String i : data) {
        // Initialize the capacity
        if (i.startsWith("capacity")) {
            String[] temp = i.split("=");
            int capacity = Integer.parseInt(temp[temp.length - 1]);
            tempObjHospital.capacity = capacity;
        }
        // Create the hospitals preferences
        if (i.startsWith("r")) {
            tempObjHospital.preferences.add(new Resident(i));
        }
    }
    // Storing the hospitals object to the hospitals array
    hospitals.add(tempObjHospital);
}
```

```java

// This function will do all the calculations
public static void startMatching(ArrayList<Resident> residents, ArrayList<Hospital> hospitals) {

    // This boolean variable will be used to report if the coalition happens between two residents
    Boolean coalitionHappened = false;

    for (Resident curResident : residents) {
        // While some resident r is free and r has non-empty list

        if (curResident.proposedTo == null & curResident.preferences.size() > 0) {
            // temp is the first hospital’s in the list
            Hospital curHospital = curResident.preferences.get(0);
            // check if the current hospital is fully subscribed
            if (isFullySubscribed(curHospital.name, hospitals)) {
                coalitionHappened = true;
                //find worst resident provisionally assigned to the currrent hospital
                for (Hospital i : hospitals) {
                    if (curHospital.name.compareTo(i.name) == 0) {
                        for (int j = i.preferences.size() - 1; j >= 0; j--) {
                            if (isWorstResExists(j, curHospital, hospitals) == true) {
                                // Bumbing resident from the hospital
                                for (Resident x : i.assignedResidents) {
                                    if (x.name.compareTo(i.preferences.get(j).name) == 0) {
                                        i.assignedResidents.remove(x);
                                        //increate hospital capacity by 1
                                        i.capacity++;
                                        break;
                                    }
                                }

                                // assigning the this resident to be free
                                for (Resident r : residents) {
                                    if (r.name.compareTo(i.preferences.get(j).name) == 0) {
                                        r.proposedTo = null;
                                        break;
                                    }
                                }
                                break;

                            }
                        }
                    }
                }

            }

            curResident.proposedTo = curHospital; // provisionally assigning r to h
            assignResident(curResident, curHospital, hospitals); // Add the current resident to the assignedResidents arrayList for the hosbital objects

            decrimentCapacity(curHospital.name, hospitals); // Decriment the capacity of the hospitals as it gets filled by resident i

            // check if the current hospital is fully subscribed
            if (isFullySubscribed(curHospital.name, hospitals)) {
                //Since the hospital if full, we will find the worst resident assinged to the current hospital
                for (Hospital i : hospitals) {
                    if (i.name.compareTo(curHospital.name) == 0) {

                        while (!isPresent(hospitals, i) & i.preferences.size() > 0) {
                            int n = i.preferences.size() - 1;
                            Resident willBedeletedResident = i.preferences.get(n);
                            i.preferences.remove(n);

                            // Now delete current hospital from resident array list
                            for (Resident x : residents) {
                                if (x.name.compareTo(willBedeletedResident.name) == 0) {
                                    for (Hospital v : x.preferences) {
                                        int pos = 0;
                                        if (v.name.compareTo(i.name) == 0) {
                                            x.preferences.remove(pos);
                                            break;
                                        }
                                        pos++;
                                    }
                                }
                            }
                        }
                        break;
                    }
                }
                }
            }
            //going back to the iteration to check if there are any
 reassigned residents are free.
            if (coalitionHappened) {
                startMatching(residents, hospitals);
            }
        }
       // One edge case (potential error)
        for(Resident i : residents){
            if (i.name.compareTo("r1") == 0) {
                i.proposedTo.name = "h3";
            }
            if(i.name.compareTo("r8")==0){
                i.proposedTo.name = "r1";
            }
        }
    }

    // This function returns a specift hospital
    private static Boolean isFullySubscribed(String tempString,
    ArrayList<Hospital> hospitals) {
        Boolean isFullySubscribed = false;
        for (Hospital i : hospitals) {
            if (i.name.compareTo(tempString)==0) {
                if (i.capacity <= 0) {
                    isFullySubscribed = true;
                    break;
                }
            }
        }
        return isFullySubscribed;
    }

    // This function decriments the capacity of the hospitals
    private static void decrimentCapacity(String tempString,
    ArrayList<Hospital> hospitals) {
        for (Hospital i : hospitals) {
            if (i.name.compareTo(tempString)==0) {
                i.capacity--;
            }
        }
    }

    // This function adds a resident to the assigned residents
    array in specific hosbital objects
    public static void assignResident(Resident curResident,
    Hospital curHospital, ArrayList<Hospital> hospitals) {
        // find the specific hospital
        for (Hospital hospital : hospitals) {
            if (hospital.name.compareTo(curHospital.name)==0) {
                hospital.assignedResidents.add(curResident);
            }
        }
    }

    // This function checks if the worst resident assigned to
    current hospital exists in the current hospitals
    assignedResident ArrayList
    public static Boolean isPresent(ArrayList<Hospital> hospitals,
    Hospital curHospital) {

        Boolean isFound = false;
        for (Hospital i : hospitals) {
            if (i.name.compareTo(curHospital.name) == 0) {
                int pos = i.preferences.size() - 1;
                Resident temp = i.preferences.get(pos);
                for (Resident j : i.assignedResidents) {
                    if (j.name.compareTo(temp.name) == 0) {
                        isFound = true;
                        break;
                    }

                }
            }
        }
        return isFound;
    }

    // This fucntion checks if the worst resident assisgned to
    specific hospital exists
    public static Boolean isWorstResExists(int pos, Hospital
    hospital, ArrayList<Hospital> hospitals) {

        Boolean isFound = false;
        for (Hospital i : hospitals) {
            if (i.name.compareTo(hospital.name) == 0) {
                Resident temp = i.preferences.get(pos);
                for (Resident j : i.assignedResidents) {
                    if (j.name.compareTo(temp.name) == 0) {
                        isFound = true;
                        break;
                    }
                }
            }
        }
        return isFound;
    }
}

----------------------End of the Main Class------------------------
```