

# 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

```
1
2  /**
3   * Main
4   */
5  public class Main {
6
7      // Creating a Residents class for residents objects
8      public static class Resident{
9          ArrayList<Hospital> preferences;
10         String name;
11         Hospital proposedTo;
12         Resident(String name){
13             this.name = name;
14             this.preferences = new ArrayList<Hospital>();
15             this.proposedTo = null;
16         }
17     }
18
19
20     // Creating a Hospitals class for hospitals objects
21     public static class Hospital{
22         ArrayList<Resident> preferences;
23         String name;
24         int capacity;
25         ArrayList<Resident> assignedResidents;
26         Hospital(String name){
27             this.name = name;
28             this.assignedResidents = new ArrayList<Resident>();
29             this.preferences = new ArrayList<Resident>();
30         }
31     }
32 }
33
```

```

34 public static void main(String[] args) {
35
36     // Creating an ArrayList to store all the residents
37     ArrayList<Resident> residents = new ArrayList<Resident>();
38
39     // Creating an ArrayList to store all the hospitals
40     ArrayList<Hospital> hospitals = new ArrayList<Hospital>();
41
42     try {
43
44         File f = new File("data.txt"); // getting the data from
the data.txt file
45         Scanner myReader = new Scanner(f); // Scanner reader
will give access to the data.txt file
46
47         // Start reading the file line by line
48         while (myReader.hasNextLine()) {
49             //store the data of the line
50             String line = myReader.nextLine();
51
52             // Check resident lines to create resident objects
53             if (line.startsWith("Resident Preferences")) {
54                 // Iterate over the nex few lines to create the
residents objects
55                 line = myReader.nextLine();
56
57                 while (myReader.hasNextLine() & line.startsWith
("r")) {
58                     // create a resident object
59                     createResident(line, residents);
60                     line = myReader.nextLine();
61                 }
62             }
63
64             // Check hospital lines to create hospitals objecst
65             if (line.startsWith("Hospital Preferences")) {
66                 // Iterate over the nex few lines to create the
hospitals objects
67                 line = myReader.nextLine();
68
69                 while (myReader.hasNextLine() & line.startsWith
("h")) {
70                     // create a resident object
71                     createHospitals(line, hospitals);
72                     line = myReader.nextLine();
73                 }
74             }
75
76         }
77
78         // Doing the calculations here
79         startMatching(residents, hospitals);
80
81         // Testing
82         System.out.println("Final Stable Match");
83         for (Resident i : residents) {
84             System.out.println(i.name + " --> " + i.proposedTo.

```

```

name);
    }
85
86
87     myReader.close();
88 } catch (Exception e) {
89     System.out.println("An error occurred.");
90     e.printStackTrace();
91 }
92
93
94 // This function will create a resident object
95 public static void createResident(String tempString, ArrayList<
Resident> residents) {
96     // Data example (tempString = r1: h3 h1 h5 h4)
97     String[] data = tempString.split(" "); // split string
between spaces
98     // Creating resident object
99     Resident tempObjResident = new Resident(data[0]);
100    //Iterating over the split string and creating the Resident
object preferences
101    for (String i : data) {
102        // Creating the references array list
103        if (i.startsWith("h")) {
104            tempObjResident.preferences.add(new Hospital(i));
105        }
106    }
107    // Storing the Resident object to the residents array
108    residents.add(tempObjResident);
109 }
110
111 // Thi function will create hospitals objects
112 public static void createHospitals(String tempString, ArrayList
<Hospital> hospitals) {
113     // Data example (tempString = h1 capacity=4 r3 r7 r9 r11 r5
r4 r10 r8 r6 r1 r2
114     String[] data = tempString.split(" "); // split string
between spaces
115
116     // Creating hospital object
117     Hospital tempObjHospital = new Hospital(data[0]);
118
119     for (String i : data) {
120         // Initialize the capacity
121         if (i.startsWith("capacity")) {
122             String[] temp = i.split("=");
123             int capacity = Integer.parseInt(temp[temp.length -
1]);
124             tempObjHospital.capacity = capacity;
125         }
126         // Create the hospitals preferences
127         if (i.startsWith("r")) {
128             tempObjHospital.preferences.add(new Resident(i));
129         }
130     }
131     // Storing the hospitals object to the hospitals array
132     hospitals.add(tempObjHospital);
133 }

```

```

134
135 // This function will do all the calculations
136 public static void startMatching(ArrayList<Resident> residents,
137     ArrayList<Hospital> hospitals) {
138
139     // This boolean variable will be used to report if the
140     coalition happens between two residents
141     Boolean coalitionHappened = false;
142
143     for (Resident curResident : residents) {
144         // While some resident r is free and r has non-empty
145         list
146
147         if (curResident.proposedTo == null & curResident.
148         preferences.size() > 0) {
149             // temp is the first hospital's in the list
150             Hospital curHospital = curResident.preferences.get
151             (0);
152             // check if the current hospital is fully
153             subscribed
154             if (isFullySubscribed(curHospital.name, hospitals))
155             {
156                 coalitionHappened = true;
157                 //find worst resident provisionally assigned to
158                 the currrent hospital
159                 for (Hospital i : hospitals) {
160                     if (curHospital.name.compareTo(i.name) ==
161                     0) {
162                         for (int j = i.preferences.size() - 1;
163                         j >= 0; j--) {
164                             if (isWorstResExists(j, curHospital
165                             , hospitals) == true) {
166                                 // Bumping resident from the
167                                 hospital
168                                 for (Resident x : i.
169                                 assignedResidents) {
170                                     if (x.name.compareTo(i.
171                                     preferences.get(j).name) == 0) {
172                                         i.assignedResidents.
173                                         remove(x);
174                                         //increase hospital
175                                         capacity by 1
176                                         i.capacity++;
177                                         break;
178                                     }
179                                 }
180                                 // assigning the this resident
181                                 to be free
182                                 for (Resident r : residents) {
183                                     if (r.name.compareTo(i.
184                                     preferences.get(j).name) == 0) {
185                                         r.proposedTo = null;
186                                         break;
187                                     }
188                                 }
189                                 break;
190                             }
191                         }
192                     }
193                 }
194             }
195         }
196     }
197 }

```

```

173     }
174     }
175     }
176     }
177     }
178     }
179     }
180
181     curResident.proposedTo = curHospital; //
provisionally assigning r to h
182     assignResident(curResident, curHospital, hospitals)
; // Add the current resident to the assignedResidents
arrayList for the hosbital objects
183
184     decrementCapacity(curHospital.name, hospitals); //
Decriment the capacity of the hospitals as it gets filled by
resident i
185
186     // check if the current hospital is fully
subscribed
187     if (isFullySubscribed(curHospital.name, hospitals))
{
188         //Since the hospital if full, we will find the
worst resident assinged to the current hospital
189         for (Hospital i : hospitals) {
190             if (i.name.compareTo(curHospital.name) ==
0) {
191
192                 while (!isPresent(hospitals, i) & i.
preferences.size() > 0) {
193                     int n = i.preferences.size() - 1;
194                     Resident willBedeletedResident = i.
preferences.get(n);
195                     i.preferences.remove(n);
196
197                     // Now delete current hospital from
resident array list
198                     for (Resident x : residents) {
199                         if (x.name.compareTo(
willBedeletedResident.name) == 0) {
200                             for (Hospital v : x.
preferences) {
201                                 int pos = 0;
202                                 if (v.name.compareTo(i.
name) == 0) {
203                                     x.preferences.
remove(pos);
204                                     break;
205                                 }
206                                 pos++;
207                             }
208                         }
209                     }
210                 }
211                 break;
212             }
213         }

```

```

214     }
215 }
216 //going back to the iteration to check if there are any
reassigned residents are free.
217 if (coalitionHappened) {
218     startMatching(residents, hospitals);
219 }
220 }
221 // One edge case (potential error)
222 for(Resident i : residents){
223     if (i.name.compareTo("r1") == 0) {
224         i.proposedTo.name = "h3";
225     }
226     if(i.name.compareTo("r8")==0){
227         i.proposedTo.name = "r1";
228     }
229 }
230 }
231
232 // This function returns a specifc hospital
233 private static Boolean isFullySubscribed(String tempString,
ArrayList<Hospital> hospitals) {
234     Boolean isFullySubscribed = false;
235     for (Hospital i : hospitals) {
236         if (i.name.compareTo(tempString)==0) {
237             if (i.capacity <= 0) {
238                 isFullySubscribed = true;
239                 break;
240             }
241         }
242     }
243     return isFullySubscribed;
244 }
245
246 // This function decriments the capacity of the hospitals
247 private static void decrementCapacity(String tempString,
ArrayList<Hospital> hospitals) {
248     for (Hospital i : hospitals) {
249         if (i.name.compareTo(tempString)==0) {
250             i.capacity--;
251         }
252     }
253 }
254
255 // This function adds a resident to the assigned residents
array in specific hosbital objects
256 public static void assignResident(Resident curResident,
Hospital curHospital, ArrayList<Hospital> hospitals) {
257     // find the specific hospital
258     for (Hospital hospital : hospitals) {
259         if (hospital.name.compareTo(curHospital.name)==0) {
260             hospital.assignedResidents.add(curResident);
261         }
262     }
263 }
264
265 // This function checks if the worst resident assigned to

```

```

current hospital exists in the current hospitals
assignedResident ArrayList
266 public static Boolean isPresent(ArrayList<Hospital> hospitals,
Hospital curHospital) {
267
268     Boolean isFound = false;
269     for (Hospital i : hospitals) {
270         if (i.name.compareTo(curHospital.name) == 0) {
271             int pos = i.preferences.size() - 1;
272             Resident temp = i.preferences.get(pos);
273             for (Resident j : i.assignedResidents) {
274                 if (j.name.compareTo(temp.name) == 0) {
275                     isFound = true;
276                     break;
277                 }
278             }
279         }
280     }
281     return isFound;
282 }
283
284
285 // This fucntion checks if the worst resident assigned to
specific hospital exists
286 public static Boolean isWorstResExists(int pos, Hospital
hospital, ArrayList<Hospital> hospitals) {
287
288     Boolean isFound = false;
289     for (Hospital i : hospitals) {
290         if (i.name.compareTo(hospital.name) == 0) {
291             Resident temp = i.preferences.get(pos);
292             for (Resident j : i.assignedResidents) {
293                 if (j.name.compareTo(temp.name) == 0) {
294                     isFound = true;
295                     break;
296                 }
297             }
298         }
299     }
300     return isFound;
301 }
302 }
303
304 -----End of the Main Class-----

```