Final Project

Kylie Wasserman

Kylie.Wasserman1@Marist.edu

December 14, 2022

1 Code Listings

My code is split into 9 files: Element.java, Hospital.java, Resident.java, Partition.java, Matching.java, Solve.java, SolveV2.java, Main.java, and MainV2.java.

1.1 Element Class

The Element Class is a simple class that has one item in it called getName.

1.2 Hospital Class

The Hospital Class keeps track of the name and capacity of a hospital. It has getters and setters as well as a compareTo method to compare one hospital to another. It also has a equals function.

1.3 Resident Class

The Resident Class keeps track of the name of a resident. It has getters and setters as well as a compareTo method to compare one hospital to another. It also has a equals function.

1.4 Partition Class

The Partition Class keeps track of all the residents and hospitals that are inputted. It has getters and setters.

1.5 Matching Class

The Matching Class keeps track of the matches of one element to another in a map. It has an addPair method that is similar to a set method.

1.6 Solve Class

The Solve Class uses the Partition Class to get the preferences of both residents and hospitals. It also has a method called solve where it goes through every hospital to match it to a resident. It does this by checking the capacity of the spots left for the hospitals and seeing where the resident ranked the hospital and where the hospital ranked the resident to determine if it is a good match. If it is then it is added to a Matching object and eventually returned in the end.

1.7 SolveV2 Class

SolveV2 is used for the second part of the assignment where the hospitals do not rank the residents, therefore there are some edits made to this file. The Solve Class uses the Partition Class to get the preferences of the residents. The solve method in this case loops through each resident and matches it with their first choice of a hospital. Luckily with this given test file only one resident has to be matched with their second choice, but with other test cases this method might not work.

1.8 Main Class

The main class starts off by having a method to parse through the file. Each of these items gets added to an arraylist of type integer. The hospital preferences and resident preferences then get added. I subtract 1 from each as I am using it for the index, which starts at 0, not 1. Eventually the parse method is used to get the capacities of each hospital. The matches get solved and printed.

1.9 MainV2 Class

The main class in Version 2 is very similar to the other main class minus the hospital preferences. The main class starts off by having a method to parse through the file. Each of these items gets added to an arraylist of type integer. The hospital preferences and resident preferences then get added. I subtract 1 from each as I am using it for the index, which starts at 0, not 1. Eventually the parse method is used to get the capacities of each hospital. The matches get solved and printed.

2 Version 2

Stability in this context means that each resident gets paired with a hospital that is on its preferences list. The hospital that gets chosen is not necessarily at the top of the list, but that is preferred. It starts moving down the preferences list if the hospital is out of spots. My method could be improved by first randomizing the order.

3 Appendix

3.1 Element Class

```
package FinalProject;

public interface Element {
    String getName();
}
```

3.2 Hospital Class

```
package FinalProject;

import java.util.Objects;

// class for all of the hospitals
// keeps track of the names and the capacity of each hospital

public class Hospital implements Comparable<Hospital>, Element {
    private String name;
    private int capacity;

    public Hospital() {

    }

    public Hospital(String name) {
        this.name = name;
    }

    public Hospital(String name, int capacity) {
        this.name = name;
        this.capacity = capacity;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getCapacity() {
        return capacity;
    }

    public void setCapacity(int capacity) {
        this.capacity = capacity;
    }

    @Override
    public String toString() {
        return name + "(c:" + capacity + ")";
    }

    @Override
    public int compareTo(Hospital hospital) {
        return this.getName().compareTo(hospital.getName());
    }

    @Override
```

```
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Hospital hospital = (Hospital) o;
        return Objects.equals(name, hospital.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }
}
```

3.3 Resident Class

```
package FinalProject;

import java.util.Objects;

// class for all of the residents
// keeps track of the names of each resident

public class Resident implements Comparable<Resident>, Element {
    private String name;

    public Resident() {
    }

    public Resident(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return name;
    }

    @Override
    public int compareTo(Resident resident) {
        return this.getName().compareTo(resident.getName());
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Resident resident = (Resident) o;
        return Objects.equals(name, resident.name);
    }

    @Override
    public int hashCode() {
        return Objects.hash(name);
    }
}
```

3.4 Partition Class

```
package FinalProject;

import java.util.Set;

public class Partition {
    private Set<Resident> residentSet;
    private Set<Hospital> hospitalSet;

    public Partition(Set<Resident> residentSet, Set<Hospital> hospitalSet) {
        this.residentSet = residentSet;
        this.hospitalSet = hospitalSet;
    }

    public Set<Resident> getResidentSet() {
        return residentSet;
    }

    public void setResidentSet(Set<Resident> residentSet) {
        this.residentSet = residentSet;
    }

    public Set<Hospital> getHospitalSet() {
        return hospitalSet;
    }

    public void setHospitalSet(Set<Hospital> hospitalSet) {
        this.hospitalSet = hospitalSet;
    }
}
```

3.5 Matching Class

```
package FinalProject;

import java.util.*;

// matching class
public class Matching {
    private Map<Resident, Hospital> matchingList = new HashMap<>();

    public Matching() {
    }

    // once a pair is made, use this function to add it to a list to eventually print
    public void addPair(Resident firstElement, Hospital secondElement) {
        matchingList.put(firstElement, secondElement);
    }

    @Override
    public String toString() {
        return "Matching: " + matchingList ;
    }
}
```

3.6 Solve Class

```
package FinalProject;

import java.util.*;
```

```
4
5  // class where they are actually matched
6  public class Solve {
7      private Partition partition;
8      private Map<Resident, List<Hospital>> residentsPreferences;
9      private Map<Hospital, List<Resident>> hospitalsPreferences;
10
11     public Solve() {
12
13     }
14
15     public Solve(Set<Resident> residents, Set<Hospital> hospitals,
16     Map<Resident, List<Hospital>> residentsPreferences,
17     Map<Hospital, List<Resident>> hospitalsPreferences) {
18         partition = new Partition(residents, hospitals);
19         this.residentsPreferences = residentsPreferences;
20         this.hospitalsPreferences = hospitalsPreferences;
21     }
22
23     public Partition getPartition() {
24         return partition;
25     }
26
27     public void setPartition(Partition partition) {
28         this.partition = partition;
29     }
30
31     public Map<Resident, List<Hospital>> getResidentsPreferences() {
32         return residentsPreferences;
33     }
34
35     public void setResidentsPreferences(Map<Resident,
36     List<Hospital>> residentsPreferences) {
37         this.residentsPreferences = residentsPreferences;
38     }
39
40     public Map<Hospital, List<Resident>> getHospitalsPreferences() {
41         return hospitalsPreferences;
42     }
43
44     public void setHospitalsPreferences(Map<Hospital,
45     List<Resident>> hospitalsPreferences) {
46         this.hospitalsPreferences = hospitalsPreferences;
47     }
48
49     // method where they are actually matched
50     public Matching solve() {
51
52         // empty match that you will add to
53         Matching finalMatching = new Matching();
54
55         // get the sets
56         Set<Hospital> hospitalSet = new HashSet<>(partition.getHospitalSet());
57         Set<Resident> residentSet = new HashSet<>(partition.getResidentSet());
58
59         //the max number of preferred residents from all hospitals sets
60         int maxNumberOfResidentsPreferences = partition.getHospitalSet().stream().
61         mapToInt(hospital -> hospitalsPreferences.get(hospital).size()).max().getAsInt();
62
63         int indexInResidentSet = 0;
64         int lengthOfResidentSet = residentSet.size();
65
66         while (indexInResidentSet <= maxNumberOfResidentsPreferences &&
67         lengthOfResidentSet > 0) {
68
```

```
            // do this for every hospital
            for (Hospital hospital : hospitalSet) {
                if (hospital.getCapacity() > 0 &&
                indexInResidentSet < hospitalsPreferences.get(hospital).size()) {

                    // the first preference
                    Resident hospitalPref = hospitalsPreferences.get(hospital).
                    get(indexInResidentSet);

                    // if it is available
                    if (residentsPreferences.get(hospitalPref).contains(hospital) &&
                    residentSet.contains(hospitalPref)) {
                        finalMatching.addPair(hospitalPref, hospital);
                        residentSet.remove(hospitalPref);
                        hospital.setCapacity(hospital.getCapacity() - 1);
                        lengthOfResidentSet--;
                    }
                }
            }
            indexInResidentSet++;
        }
        return finalMatching;
    }
}
```

3.7 SolveV2 Class

```

package FinalProject;

import java.util.*;

// class where they are actually matched
public class SolveV2 {
    private Partition partition;
    private Map<Resident, List<Hospital>> residentsPreferences;

    public SolveV2() {

    }

    public SolveV2(Set<Resident> residents, Set<Hospital> hospitals,
    Map<Resident, List<Hospital>> residentsPreferences) {
        partition = new Partition(residents, hospitals);
        this.residentsPreferences = residentsPreferences;
    }

    public Partition getPartition() {
        return partition;
    }

    public void setPartition(Partition partition) {
        this.partition = partition;
    }

    public Map<Resident, List<Hospital>> getResidentsPreferences() {
        return residentsPreferences;
    }

    public void setResidentsPreferences(Map<Resident,
    List<Hospital>> residentsPreferences) {
        this.residentsPreferences = residentsPreferences;
    }
```

```
37
38      // method where they are actually matched
39      public Matching solve() {
40
41          // empty match that you will add to
42          Matching finalMatching = new Matching();
43
44          // get the sets
45          Set<Hospital> hospitalSet = new HashSet<>(partition.getHospitalSet());
46          Set<Resident> residentSet = new HashSet<>(partition.getResidentSet());
47
48          int indexInResidentSet = 0;
49
50          // for each resident if the first hospital in thier list is available,
51          make that match
52          for( Resident res : residentSet){
53              Hospital reChoice = residentsPreferences.get(res).get(indexInResidentSet);
54              if (reChoice.getCapacity() != 0){
55                  finalMatching.addPair(res, reChoice);
56                  reChoice.setCapacity(reChoice.getCapacity() - 1);
57
58              }
59          }
60          indexInResidentSet++;
61
62          return finalMatching;
63      }
64
65  }
```

3.8 Main Class

```
1   package FinalProject;
2
3   import java.util.*;
4   import java.util.stream.Collectors;
5   import java.util.stream.IntStream;
6   import java.io.BufferedReader;
7   import java.io.FileReader;
8   import java.io.IOException;
9   import java.util.ArrayList;
10  import java.util.Arrays;
11  import java.util.List;
12
13  public class Main {
14
15      // items for the parse file method
16      public static ArrayList<Integer> allResidents = new ArrayList<Integer>();
17      public static ArrayList<Integer> allHospitals = new ArrayList<Integer>();
18
19      public static ArrayList<Integer> resPrefItems = new ArrayList<Integer>();
20      public static ArrayList<Integer> hospPrefItems = new ArrayList<Integer>();
21      public static ArrayList<ArrayList<Integer>> allResPrefItems =
22      new ArrayList<ArrayList<Integer>>();
23      public static ArrayList<ArrayList<Integer>> allHospPrefItems =
24      new ArrayList<ArrayList<Integer>>();
25
26      public static ArrayList<Integer> hospSpots = new ArrayList<Integer>();
27
28      // method to get an integer arraylist from a string arraylist
29      private static ArrayList<Integer> getIntegerArray(ArrayList<String> stringArray) {
30          ArrayList<Integer> result = new ArrayList<Integer>();
31          for(String stringValue : stringArray) {
```

```
            try {
                result.add(Integer.parseInt(stringValue));
            } catch(NumberFormatException nfe) {

            }
        }
        return result;
    }

    // method to parse through the given txt file
    public static void parseFile() throws IOException{
        // load data from file
        BufferedReader in2 = new BufferedReader(new FileReader("HopsRes.txt"));
        String str2;

        // create array list
        List<String> list2 = new ArrayList<String>();

        // add each line to a new item in the arraylist
        while((str2 = in2.readLine()) != null)
        {
            // make each item lowercase
            str2 = str2.toLowerCase();
            list2.add(str2);
        }

        // close bufferedReader object
        in2.close();

        // storing the data in arraylist to an array
        String[] arrayLines =  list2.toArray(new String[0]);

        // for each line in the file
        for(String line : arrayLines){

            if(!line.equals("")){

                // line for declaring the total number of residents and hospitals
                if(line.charAt(0) == 'c') {

                    String resNumString = line.substring(7,10);
                    String hospNumString = line.substring(10,12);

                    resNumString = resNumString.replaceAll("\\s", "");
                    hospNumString = hospNumString.replaceAll("\\s", "");

                    int resNumInt = Integer.valueOf(resNumString);
                    int hospNumInt = Integer.valueOf(hospNumString);

                }

                // line for a specific resident (with prefered hospital list)
                else if (line.charAt(0) == 'r') {

                    // some are single digit, others are double, remove : for that reason
                    String resIdString = line.substring(0, 3);
                    resIdString = resIdString.replaceAll(":", "");
                    resIdString = resIdString.replaceAll("\\s", "");
                    resIdString = resIdString.replaceAll("r", "");
                    int resIDInt = Integer.parseInt(resIdString);

                    // add the id to the list
                    // -1 bc we are using it for index, which starts at 0, not 1
                    allResidents.add(resIDInt-1);

```

```
                    // for preferred hospitals lists
                    int beginIndex = 3;
                    int endIndex = 7;

                    for(int i = 3; i < line.length(); i++){

                        if (line.length() > endIndex){
                            String resPrefItem = line.substring(beginIndex, endIndex);
                            resPrefItem = resPrefItem.replaceAll("\\s", "");
                            resPrefItem = resPrefItem.replaceAll(":", "");
                            resPrefItem = resPrefItem.replaceAll("h", "");

                            int resPrefItemInt = Integer.parseInt(resPrefItem);

                            beginIndex += 3;
                            endIndex += 3;

                            resPrefItems.add(resPrefItemInt-1);
                        }
                        else{
                            break;
                        }
                    }

                    // you know which list goes with which resident by index
                    (they have the same index)
                    allResPrefItems.add(resPrefItems);
                }

                // line for a specific hospital (with prefered resident list and capacity)
                else if (line.charAt(0) == 'h') {

                    // some are single digit, others are double, remove : for that reason
                    String hospIdString = line.substring(0, 2);
                    hospIdString = hospIdString.replaceAll(":", "");
                    hospIdString = hospIdString.replaceAll("h", "");
                    int hospIDInt = Integer.parseInt(hospIdString);

                    // add the id to the list (-1 bc we are using it for the index)
                    allHospitals.add(hospIDInt-1);

                    String hospSpotsLeft = line.substring(4, 5);
                    hospSpotsLeft.replaceAll("\\s", "");
                    hospSpotsLeft.replaceAll(":", "");
                    hospSpotsLeft.replaceAll("-", "");
                    int hospSpotsLeftNum = Integer.parseInt(hospSpotsLeft);

                    hospSpots.add(hospSpotsLeftNum);

                    String hospPrefItemAll = line.substring(8);
                    String[] hospPrefItemArray = hospPrefItemAll.split("\\s+");
                    for(int i = 0; i < hospPrefItemArray.length; i++){
                        hospPrefItemArray[i].replaceAll(" ", "");
                        hospPrefItemArray[i].replaceAll(":", "");
                    }
                    ArrayList<String> hospPrefItemList = new ArrayList<String>();
                    hospPrefItemList.addAll(Arrays.asList(hospPrefItemArray));

                    ArrayList<Integer> hospPrefItemListInt =
                    getIntegerArray(hospPrefItemList);

                    hospPrefItems.addAll(hospPrefItemListInt);

                    // you know which list goes with which hospital by index
                    (they have the same index)
```

```
                allHospPrefItems.add(hospPrefItems);
            }
        }
    }
}

// put in all of the hospitals and the preferred list
public static Map<Hospital, List<Resident>> createMapHospitalsResidents(Hospital[] h,
Resident[] r) throws IOException{
    Map<Hospital, List<Resident>> hospitalPref = new TreeMap<>();

    hospitalPref.put(h[1-1], Arrays.asList(r[3-1], r[7-1], r[9-1], r[11-1],
    r[5-1], r[4-1], r[10-1], r[8-1], r[6-1], r[1-1], r[2-1]));
    hospitalPref.put(h[2-1], Arrays.asList(r[5-1], r[7-1], r[10-1], r[6-1],
    r[8-1], r[2-1], r[3-1], r[11-1]));
    hospitalPref.put(h[3-1], Arrays.asList(r[11-1], r[6-1], r[8-1], r[3-1],
    r[2-1], r[4-1], r[7-1], r[1-1], r[10-1]));
    hospitalPref.put(h[4-1], Arrays.asList(r[10-1], r[1-1], r[2-1], r[11-1],
    r[4-1], r[9-1], r[5-1], r[3-1], r[6-1], r[8-1]));
    hospitalPref.put(h[5-1], Arrays.asList(r[2-1], r[4-1], r[10-1], r[7-1],
    r[6-1], r[1-1], r[8-1], r[3-1], r[11-1], r[9-1]));
    return hospitalPref;
}

// put in all of the residents and the preferred list
public static Map<Resident, List<Hospital>> createMapResidentsHospitals(Resident[] r,
Hospital[] h) throws IOException{
    Map<Resident, List<Hospital>> residentPref = new HashMap<>();

    residentPref.put(r[1-1], Arrays.asList(h[3-1], h[1-1], h[5-1], h[4-1]));
    residentPref.put(r[2-1], Arrays.asList(h[1-1], h[3-1], h[4-1], h[2-1], h[5-1]));
    residentPref.put(r[3-1], Arrays.asList(h[4-1], h[5-1], h[3-1], h[1-1], h[2-1]));
    residentPref.put(r[4-1], Arrays.asList(h[3-1], h[4-1], h[1-1], h[5-1]));
    residentPref.put(r[5-1], Arrays.asList(h[1-1], h[4-1], h[2-1]));
    residentPref.put(r[6-1], Arrays.asList(h[4-1], h[3-1], h[2-1], h[1-1], h[5-1]));
    residentPref.put(r[7-1], Arrays.asList(h[2-1], h[5-1], h[1-1], h[3-1]));
    residentPref.put(r[8-1], Arrays.asList(h[1-1], h[3-1], h[2-1], h[5-1], h[4-1]));
    residentPref.put(r[9-1], Arrays.asList(h[4-1], h[1-1], h[5-1]));
    residentPref.put(r[10-1], Arrays.asList(h[3-1], h[1-1], h[5-1], h[2-1], h[4-1]));
    residentPref.put(r[11-1], Arrays.asList(h[5-1], h[4-1], h[1-1], h[3-1], h[2-1]));
    return residentPref;
}

// queries that display the residents who find acceptable some given hospitals
public static List<Resident> getPrefResidents(List<Resident> residentList,
List<Hospital> prefHospitals, Map<Resident, List<Hospital>> resPrefMap) {
    return residentList.stream().filter(resident ->
    resPrefMap.get(resident).containsAll(prefHospitals)).collect(Collectors.toList());
}

// queries that display the hospitals that have a given resident as their top preference
public static List<Hospital> getPrefHospitals(Set<Hospital> hospitalList,
Map<Hospital, List<Resident>> hospitalsPreferences, Resident preferredResident) {
    return hospitalList.stream().filter(hospital ->
    hospitalsPreferences.get(hospital).get(0).equals(preferredResident)).
    collect(Collectors.toList());
}

// main method
public static void main(String[] args) throws IOException{

    // call the parse method
    parseFile();

    // set all the residents
```

```
        Resident[] residents = IntStream.rangeClosed(0, 10).mapToObj(i ->
        new Resident("R" + i)).toArray(Resident[]::new);

        // print the residents
        System.out.print("Residents (by index (so -1 to all)):");
        for (Resident resident : residents) {
            if (!resident.equals(residents[residents.length - 1])) {
                System.out.print(resident + ", ");
            } else {
                System.out.println(resident);
            }

        }

        // set all of the hospitals
        Hospital[] hospitals = IntStream.rangeClosed(0, 4).mapToObj(i ->
        new Hospital("H" + i)).toArray(Hospital[]::new);

        // set the capacity of the hospitals with parsing the file
        for(int i = 0; i < allHospitals.size(); i++){
            hospitals[i].setCapacity(hospSpots.get(i));
        }

        // print the hospitals
        System.out.print("Hospitals (by index (so -1 to all)): ");
        for (Hospital hospital : hospitals) {
            if (!hospital.equals(hospitals[hospitals.length - 1])) {
                System.out.print(hospital + ", ");
            } else {
                System.out.println(hospital);
            }

        }
        System.out.println();

        // create a list of residents
        List<Resident> residentsList = new ArrayList<>(Arrays.asList(residents));

        // sort the residents using a comparator
        residentsList.sort(Comparator.comparing(Resident::getName));

        // create a set of hospitals
        Set<Hospital> hospitalsList = new TreeSet<>(Arrays.asList(hospitals));

        // create a map of hospital prefereneces
        Map<Hospital, List<Resident>> hospitalPreferences =
        createMapHospitalsResidents(hospitals, residents);

        // create a map of resident prefereneces
        Map<Resident, List<Hospital>> residentPreferences =
        createMapResidentsHospitals(residents, hospitals);

        Solve problem = new Solve(new HashSet<>(residentsList),
        hospitalsList, residentPreferences, hospitalPreferences);
        Matching matching = problem.solve();
        System.out.println(matching);
    }
}
```

3.9 MainV2 Class

```
package FinalProject;

```

```
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.IntStream;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class MainV2 {

    // items for the parse file method
    public static ArrayList<Integer> allResidents = new ArrayList<Integer>();
    public static ArrayList<Integer> allHospitals = new ArrayList<Integer>();

    public static ArrayList<Integer> resPrefItems = new ArrayList<Integer>();
    public static ArrayList<Integer> hospPrefItems = new ArrayList<Integer>();
    public static ArrayList<ArrayList<Integer>> allResPrefItems =
    new ArrayList<ArrayList<Integer>>();
    public static ArrayList<ArrayList<Integer>> allHospPrefItems =
    new ArrayList<ArrayList<Integer>>();

    public static ArrayList<Integer> hospSpots = new ArrayList<Integer>();

    // method to get an integer arraylist from a string arraylist
    private static ArrayList<Integer> getIntegerArray(ArrayList<String> stringArray) {
        ArrayList<Integer> result = new ArrayList<Integer>();
        for(String stringValue : stringArray) {
            try {
                //Convert String to Integer, and store it into integer array list.
                result.add(Integer.parseInt(stringValue));
            } catch(NumberFormatException nfe) {

            }
        }
        return result;
    }

    // method to parse through the given txt file
    public static void parseFile() throws IOException{
        // load data from file
        BufferedReader in2 = new BufferedReader(new FileReader("HopsRes.txt"));
        String str2;

        // create array list
        List<String> list2 = new ArrayList<String>();

        // add each line to a new item in the arraylist
        while((str2 = in2.readLine()) != null)
        {
            // make each item lowercase
            str2 = str2.toLowerCase();
            list2.add(str2);
        }

        // close bufferedReader object
        in2.close();

        // storing the data in arraylist to an array
        String[] arrayLines =  list2.toArray(new String[0]);

        // for each line in the file
        for(String line : arrayLines){

```

```
68              if(!line.equals("")){
69
70                  // line for declaring the total number of residents and hospitals
71                  if(line.charAt(0) == 'c') {
72
73                      String resNumString = line.substring(7,10);
74                      String hospNumString = line.substring(10,12);
75
76                      resNumString = resNumString.replaceAll("\\s", "");
77                      hospNumString = hospNumString.replaceAll("\\s", "");
78
79                      int resNumInt = Integer.valueOf(resNumString);
80                      int hospNumInt = Integer.valueOf(hospNumString);
81
82                  }
83
84                  // line for a specific resident (with prefered hospital list)
85                  else if (line.charAt(0) == 'r') {
86
87                      // some are single digit, others are double, remove : for that reason
88                      String resIdString = line.substring(0, 3);
89                      resIdString = resIdString.replaceAll(":", "");
90                      resIdString = resIdString.replaceAll("\\s", "");
91                      resIdString = resIdString.replaceAll("r", "");
92                      int resIDInt = Integer.parseInt(resIdString);
93
94                      // add the id to the list
95                      // -1 bc we are using it for index, which starts at 0, not 1
96                      allResidents.add(resIDInt-1);
97
98                      // for preferred hospitals lists
99                      int beginIndex = 3;
100                     int endIndex = 7;
101
102                     for(int i = 3; i < line.length(); i++){
103
104                         if (line.length() > endIndex){
105                             String resPrefItem = line.substring(beginIndex, endIndex);
106                             resPrefItem = resPrefItem.replaceAll("\\s", "");
107                             resPrefItem = resPrefItem.replaceAll(":", "");
108                             resPrefItem = resPrefItem.replaceAll("h", "");
109
110                             int resPrefItemInt = Integer.parseInt(resPrefItem);
111
112                             beginIndex += 3;
113                             endIndex += 3;
114
115                             resPrefItems.add(resPrefItemInt-1);
116                         }
117                         else{
118                             break;
119                         }
120                     }
121
122                     // you know which list goes with which resident by index
123                     (they have the same index)
124                     allResPrefItems.add(resPrefItems);
125                 }
126
127                 // line for a specific hospital (with prefered resident list and capacity)
128                 else if (line.charAt(0) == 'h') {
129
130                     // some are single digit, others are double, remove : for that reason
131                     String hospIdString = line.substring(0, 2);
132                     hospIdString = hospIdString.replaceAll(":", "");
```

```
                    hospIdString = hospIdString.replaceAll("h", "");
                    int hospIDInt = Integer.parseInt(hospIdString);

                    // add the id to the list (-1 bc we are using it for the index)
                    allHospitals.add(hospIDInt-1);

                    String hospSpotsLeft = line.substring(4, 5);
                    hospSpotsLeft.replaceAll("\\s", "");
                    hospSpotsLeft.replaceAll(":", "");
                    hospSpotsLeft.replaceAll("-", "");
                    int hospSpotsLeftNum = Integer.parseInt(hospSpotsLeft);

                    hospSpots.add(hospSpotsLeftNum);

                    String hospPrefItemAll = line.substring(8);
                    String[] hospPrefItemArray = hospPrefItemAll.split("\\s+");
                    for(int i = 0; i < hospPrefItemArray.length; i++){
                        hospPrefItemArray[i].replaceAll(" ", "");
                        hospPrefItemArray[i].replaceAll(":", "");
                    }
                    ArrayList<String> hospPrefItemList = new ArrayList<String>();
                    hospPrefItemList.addAll(Arrays.asList(hospPrefItemArray));

                    ArrayList<Integer> hospPrefItemListInt =
                    getIntegerArray(hospPrefItemList);

                    hospPrefItems.addAll(hospPrefItemListInt);

                    // you know which list goes with which hospital by index
                    (they have the same index)
                    allHospPrefItems.add(hospPrefItems);
                }
            }
        }
    }

    // put in all of the hospitals
    public static List<Hospital> createMapHospitalsResidents(Hospital[] h)
    throws IOException{
        List<Hospital> hospitalPref = new ArrayList<>();

        hospitalPref.add(h[1-1]);
        hospitalPref.add(h[2-1]);
        hospitalPref.add(h[3-1]);
        hospitalPref.add(h[4-1]);
        hospitalPref.add(h[5-1]);
        return hospitalPref;
    }

    // put in all of the residents and the preferred list
    public static Map<Resident, List<Hospital>> createMapResidentsHospitals(Resident[] r,
    Hospital[] h) throws IOException{
        Map<Resident, List<Hospital>> residentPref = new HashMap<>();

        residentPref.put(r[1-1], Arrays.asList(h[3-1], h[1-1], h[5-1], h[4-1]));
        residentPref.put(r[2-1], Arrays.asList(h[1-1], h[3-1], h[4-1], h[2-1], h[5-1]));
        residentPref.put(r[3-1], Arrays.asList(h[4-1], h[5-1], h[3-1], h[1-1], h[2-1]));
        residentPref.put(r[4-1], Arrays.asList(h[3-1], h[4-1], h[1-1], h[5-1]));
        residentPref.put(r[5-1], Arrays.asList(h[1-1], h[4-1], h[2-1]));
        residentPref.put(r[6-1], Arrays.asList(h[4-1], h[3-1], h[2-1], h[1-1], h[5-1]));
        residentPref.put(r[7-1], Arrays.asList(h[2-1], h[5-1], h[1-1], h[3-1]));
        residentPref.put(r[8-1], Arrays.asList(h[1-1], h[3-1], h[2-1], h[5-1], h[4-1]));
        residentPref.put(r[9-1], Arrays.asList(h[4-1], h[1-1], h[5-1]));
        residentPref.put(r[10-1], Arrays.asList(h[3-1], h[1-1], h[5-1], h[2-1], h[4-1]));
        residentPref.put(r[11-1], Arrays.asList(h[5-1], h[4-1], h[1-1], h[3-1], h[2-1]));
```

```
198         return residentPref;
199     }
200 
201     // queries that display the residents who find acceptable some given hospitals
202     public static List<Resident> getPrefResidents(List<Resident> residentList,
203     List<Hospital> prefHospitals, Map<Resident, List<Hospital>> resPrefMap) {
204         return residentList.stream().filter(resident ->
205         resPrefMap.get(resident).containsAll(prefHospitals)).collect(Collectors.toList());
206     }
207 
208     // main method
209     public static void main(String[] args) throws IOException{
210 
211         // call the parse method
212         parseFile();
213 
214         // set all the residents
215         Resident[] residents = IntStream.rangeClosed(0, 10).mapToObj(i ->
216         new Resident("R" + i)).toArray(Resident[]::new);
217 
218         // print the residents
219         System.out.print("Residents (by index (so -1 to all)):");
220         for (Resident resident : residents) {
221             if (!resident.equals(residents[residents.length - 1])) {
222                 System.out.print(resident + ", ");
223             } else {
224                 System.out.println(resident);
225             }
226 
227         }
228 
229         // set all of the hospitals
230         Hospital[] hospitals = IntStream.rangeClosed(0, 4).mapToObj(i ->
231         new Hospital("H" + i)).toArray(Hospital[]::new);
232 
233         // set the capacity of the hospitals with parsing the file
234         for(int i = 0; i < allHospitals.size(); i++){
235             hospitals[i].setCapacity(hospSpots.get(i));
236         }
237 
238         // print the hospitals
239         System.out.print("Hospitals (by index (so -1 to all)): ");
240         for (Hospital hospital : hospitals) {
241             if (!hospital.equals(hospitals[hospitals.length - 1])) {
242                 System.out.print(hospital + ", ");
243             } else {
244                 System.out.println(hospital);
245             }
246 
247         }
248         System.out.println();
249 
250         // create a list of residents
251         List<Resident> residentsList = new ArrayList<>(Arrays.asList(residents));
252 
253         // sort the residents using a comparator
254         residentsList.sort(Comparator.comparing(Resident::getName));
255 
256         // create a set of hospitals
257         Set<Hospital> hospitalsList = new TreeSet<>(Arrays.asList(hospitals));
258 
259         // create a map of resident prefereneces
260         Map<Resident, List<Hospital>> residentPreferences =
261         createMapResidentsHospitals(residents, hospitals);
262 
```

```
263         SolveV2 problem = new SolveV2(new HashSet<>(residentsList),
264         hospitalsList, residentPreferences);
265         Matching matching = problem.solve();
266         System.out.println(matching);
267     }
268 }
```